Table of Contents

1 Welcome to NinjaTrader

The NinjaTrader Help Guide is your reference to product features descriptions and detailed instructional content on their use. Instructional content is delivered via text, images and video where applicable. This Help Guide also serves as a reference to NinjaScript used in the development of automated trading systems (strategies) and custom indicators.

In addition to this Help Guide, NinjaTrader hosts multiple live on-line training sessions per week on various aspects of our product.

Additional information and a schedule of upcoming training events.

Thank you for choosing NinjaTrader.

Good trading,
NinjaTrader Customer Service

2 Video Library

Within the Help Guide are numerous videos providing a step by step tour through the NinjaTrader Platform. Select your area of interest below to view an expanded list of all available topics within each category.

Order Entry

Trade Controls Overview

The Trade Controls Overview video provides a walkthrough of various trade management features which can be accessed in a variety of order-entry windows. Several order-entry windows are shown in the video to show the commonalities in how trade controls operate in NinjaTrader.

Chart Trader Overview

Chart Trader allows orders and positions to be entered and managed directly within a chart window. Advanced Trade Management strategies can be employed directly in the Chart Trader window. Orders and positions on multiple instruments can be managed within a single chart window, as well. This video covers the basics of enabling and using Chart Trader, including a visual method of placing Limit and Stop orders.

SuperDOM Order Submission Overview

This video covers submitting new orders in the SuperDOM.

SuperDOM Position Management Overview

The SuperDOM Position Management Overview video covers scaling in, scaling out, and closing positions directly in the SuperDOM.

SuperDOM Order Modification Overview

This video shows the ways in which resting orders can be modified directly in the SuperDOM.

Attach to Indicator Overview

The Attach to Indicator feature allows resting orders to be attached to indicator plots, automatically updating an order's price as the indicator value changes. This feature can be used to partially automate entries, exits, stop losses, and profit targets. The Attach to Indicator Overview video provides working examples of using this feature in Chart Trader and the SuperDOM.

Overview of the Basic Entry and FX Pro Windows

The Basic Entry and FX Pro windows conveniently group order-entry and market analysis features into compact windows which function in similar ways. This video explores the layout and basic features of both window.

Advanced Trade Management (ATM) Strategies

Advanced Trade Management Overview

Advanced Trade Management Strategies, also referred to as ATM Strategies, provide a layer of discretionary automation to manage a position's exit orders without the need to make continual manual modifications. The Advanced Trade Management Overview video introduces and defines ATM Strategies, while demonstrating a simple ATM setup.

ATM Stop Strategies

ATM Stop Strategies provide additional functionality for the stop losses placed by an ATM Strategy, including auto-breakeven, auto-trail, and **Simulated Stop** orders.

ATM Additional Options and Strategy Selection Modes

Additional options for ATM Strategies include Auto-Reverse and Auto-Chase features, and the ability to specific the order type used for profit targets and stop losses. ATM Strategy Selection Modes determine the behavior of the ATM Strategy Control List after placing an order.

Advanced Trade Management Examples

In the final Advanced Trade Management video, several real-world examples are created and saved as templates for later use, showing many core ATM features in use in a live market.

SuperDOM

SuperDOM Display Overview

The SuperDOM displays five levels of market depth on a price ladder, and allows for the entry and management of orders and positions, as well as the use of Advanced Trade Management Strategies. The SuperDOM Display Overview video covers the layout and basic functionality of the SuperDOM.

Static vs. Dynamic Price Ladder

Two versions of the SuperDOM are available: Static and Dynamic. This video covers the difference between the two.

Working with Indicators on the SuperDOM

The Working with Indicators video provides an overview of configuring technical indicators on the SuperDOM. Just like charts, the SuperDOM can display a wide range of price- and volume-based indicators, and allows resting orders to be attached to indicator plots moving in real-time.

SuperDOM Columns Overview

The SuperDOM Columns Overview shows how additional columns can be added to a SuperDOM window to display profit and loss, volume, notes, or any information configured in a custom column created via NinjaScript.

Control Center

Control Center Overview

The Control Center acts as the primary window in NinjaTrader, providing access to all trading windows, performance reporting, and other features of the platform. This video provides an overview of the Control Center's layout and menus.

Control Center Tabbed Display and Account Data

The Control Center's tabbed layout provides quick access to Orders, Positions, Accounts, Strategies, and Executions. The Control Center Tabbed Display and Account Data video covers navigating Control Center tabs, as well as managing and editing connected brokerage accounts.

Market Analyzer

Market Analyzer Display Overview

The Market Analyzer is NinjaTrader's answer to the traditional quote sheet, adding a wide range of functionality to extend the features of traditional quote sheets, such as the ability to view indicator values, create alerts, and link to charts and order-entry windows for instant instrument switching. The Market Analyzer Display video covers these features in detail.

Market Analyzer Columns and Indicators

The Market Analyzer can be configured with a wide range of pre-built and custom columns and indicators. This video demonstrates applying and configuring these items.

Alerts

Overview of Alerts

Alerts can be configured on Charts or Market Analyzer windows, allowing you to set custom actions to take when predefined conditions are met in the market, including automatically placing orders or sharing messages via social media. This video covers configuring and testing alerts using the Simulated Data Feed.

Alerts Examples

In this video, a few real-world examples of alerts are set up in a Chart and Market Analyzer to show the Alerts feature in action.

Charts

Creating Charts Overview

NinjaTrader charts feature a wide range of advanced features and options, which are covered in several videos. The Creating Charts Overview provides a walkthrough of creating a new chart or duplicating an existing chart.

Navigating Charts Overview

The Navigating Charts Overview video picks up where the Creating Charts video left off, showing you how to manage instruments, navigate chart windows, and manipulate the viewable area of charts.

Working With Indicators on Charts

Technical indicators plot mathematical derivatives of price action graphical on charts. Over 100 indicators come preloaded with NinjaTrader and can be applied right away. Additionally, custom indicators can be developed via NinjaScript or obtained through third-party vendors for an even greater array of indicator selections.

Chart Panels and Objects Overview

Charts can contain numerous objects, including bars series, indicator plots, **Drawing Objects**, and execution plots. The Chart Panels and Objects Overview video shows how to manage, drag and drop, or copy and paste chart objects.

Chart Drawing Objects

Drawing Objects allow you to mark any area of a chart panel in a variety of ways. Numerous **Drawing Objects** are available for use right away, including several Fibonacci tools, and additional **Drawing Objects** can be created via NinjaScript or obtained from third-party vendors.

Working with Price Data on Charts

Charts allow you to view price data in a wide variety of formats, including different Chart Styles, Bar Types, and intervals. This video provides an overview of setting up price data to your liking on a chart.

▼ Market Data Windows

FX Board Display

The FX Board allows forex traders to view a wide range of forex instruments at a glance, using an advanced interface to quick enter, exit, and manage trades on numerous instruments from within a single window. The FX Board Display video covers the layout and primary functions of the FX Board.

Level II Window Overview

The Level II window presents a complete view of market depth events for an instrument, displaying all 10 levels of depth, including the price, size, volume, and spread of each order. This video covers the basics of opening, populating, and reading the Level II window.

Time & Sales Window Overview

The Time & Sales window can be used to view granular details about all orders being filled at the exchange for a particular instrument. This video provides an overview of the layout and basic operation of the Time & Sales window.

Miscellaneous

Window Management Overview

NinjaTrader windows include common features to increase workflow and workspace efficiency. The Window Management Overview video covers such topics as creating and editing tabs within windows, and duplicating existing content in new windows.

Hot List Analyzer Overview

The Hot List Analyzer screens equity instruments based on a variety of criteria with over 30 filters. For example, this window can be used to spot the most active, highest gaining, or highest losing stocks of the day on an individual exchange. This video shows how to set up and populate the Hot List Analyzer.

Share and Print Overview

Sharing content such as positions and chart screenshots is an integral feature in NinjaTrader. The Share and Print Overview video covers linking social media accounts to NinjaTrader and sharing content from a variety of windows within the platform.

Trade Performance Overview

The Trade Performance window provides robust reporting on the performance of completed trades, including a number of graphs covering popular performance metrics. The Trade Performance Overview video provides a high-level overview of using the Trade Performance window and it's various filters to view meaningful performance reports.

Playback Connection

The Playback connection allows you to use Market Replay data, or historical data, to play back market action from previous days. This video provides an overview of downloading Market Replay data, setting up the Playback connection, and playing data back at different speeds.

Strategy Builder Overview

The Strategy Builder is used to generate NinjaScript based strategies for automated systems trading. This video provides an overview of creating a strategy.

Strategy Backtesting and Optimization

Strategy Analyzer Overview

NinjaScript strategies can be backtested and optimized to test theoretical strategy performance on historical data within the Strategy Analyzer. The Strategy Analyzer Overview covers the basic layout of the Strategy Analyzer, and individual test types will be covered in greater detail in future videos.

Backtesting Strategies

The Backtesting Strategies video walks through the process of configuring, running, and analyzing the results of a standard strategy backtest in the Strategy Analyzer. All configurable backtest properties are covered in this video.

Optimizing Strategies

Strategy optimizations allow you to iterate over a pre-defined range of strategy input values to determine the combination of property values which score highest on a chosen performance metric. The Optimizing Strategies video covers the aspects of optimizations which differ from standard backtests.

Understanding Walk-Forward Optimization

Walk-Forward Optimization combines the features of optimizations and standard backtests. This Backtest Type performs an optimization over a pre-defined date range, then applies the optimal parameter combinations to a standard backtest over another pre-defined date range.

Understanding Multi-Objective Optimization

Multi-Objective Optimization uses Pareto Analysis to find a set of possible input-value combinations which score higher or lower on individual metrics (of which there can be many), but for which there are no obviously superior alternatives on all metrics tested. This video introduces the goals of Multi-Objective Optimization and explains the concept of the Pareto Frontier.

Understanding the Genetic Algorithm

The Genetic Algorithm is an optional optimization engine which leverages evolutionary theory to find optimal combinations of strategy input parameters through multi-generational crossover and mutation, focusing on the fittest individuals in each generation.

3 Release Notes

NinjaTrader release notes can be found below, if you have any questions on a specific release please contact platformsupport@ninjatrader.com

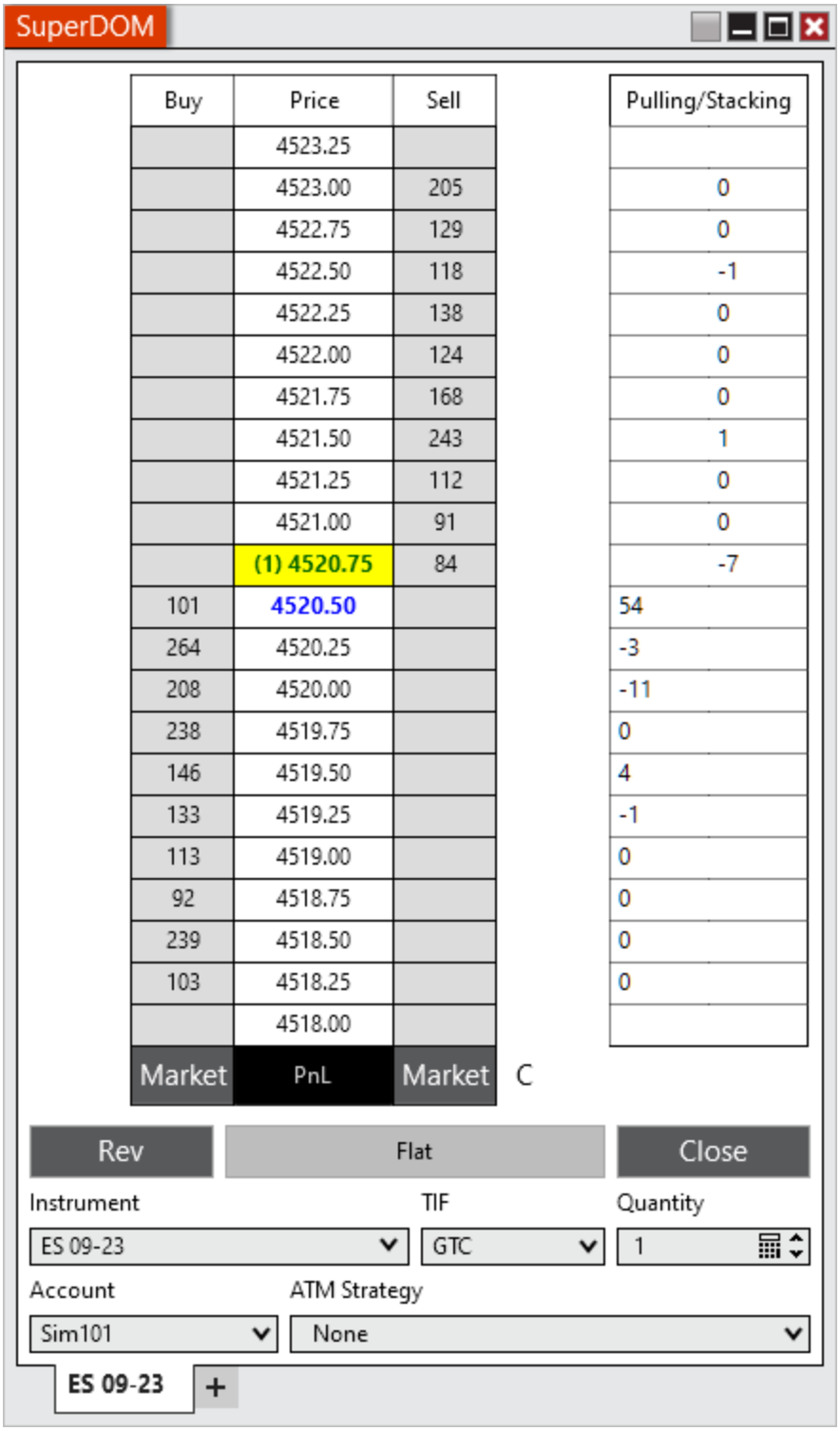
Version	Released
8.1.2.0	October 25, 2023
8.1.1.7	March 4, 2023

3.1 8.1.2.0

8.1.2.0 Release Date

October 25, 2023

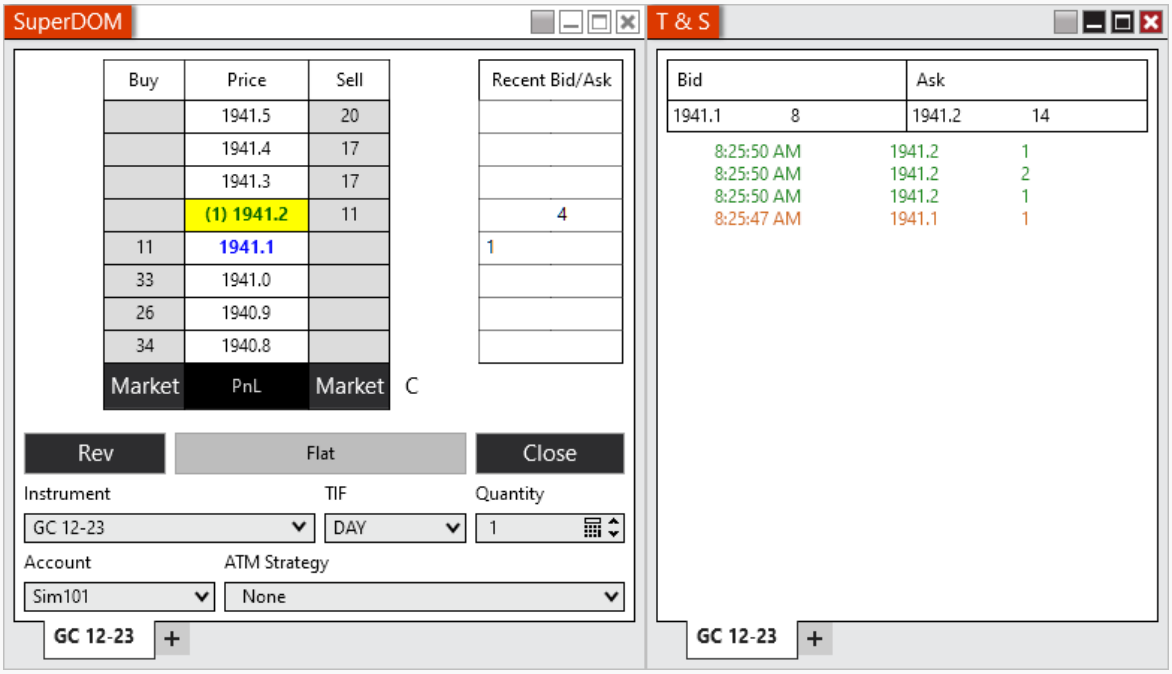
Features
SuperDOM Pulling/Stacking column Feature #2873



The Pulling/Stacking column is a customizable display that indicates the changes in the market depth based on user settings or if a reset notification is received. For example, in

the screenshot above the sell depth at 4520.75 was initially at 91, but dropped to 84 resulting in a display of -7.
Within the columns settings you can change the reset to occur on bid/ask change or when no longer receiving depth data at that level. Additionally, you can adjust reset tolerance and color settings.

SuperDOM Recent column
Feature #2874

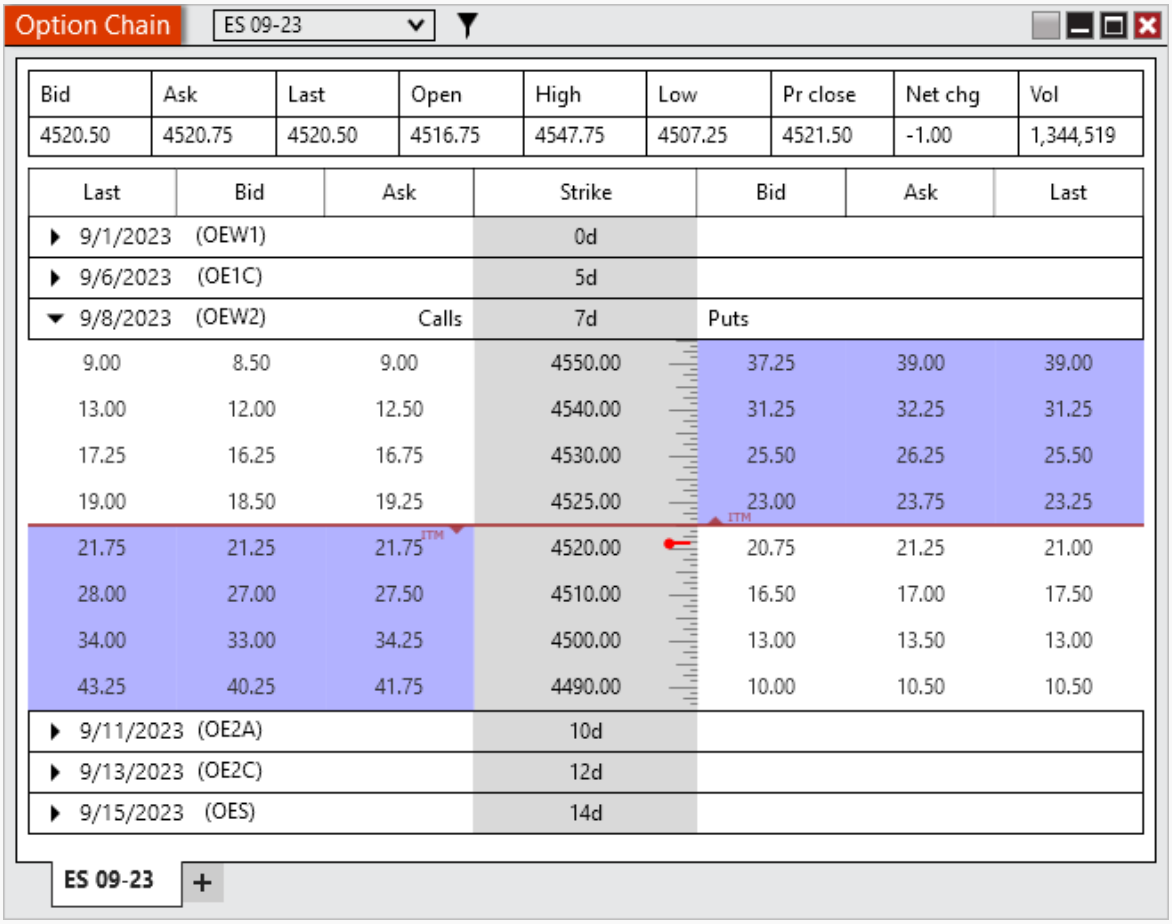


The Recent column is a customizable display that indicates the recent volume that occurred at the bid or ask prices. For example, in the above screenshot we can see a volume of 1 occurred at the bid price and a volume of 4 occurred at the ask.
Within the column's settings you can change the reset to occur when the bid/ask change or when the price returns. Additionally, you can adjust the reset tollerance and color settings.

Selected indicators label displays in bold/italic
Feature #2894

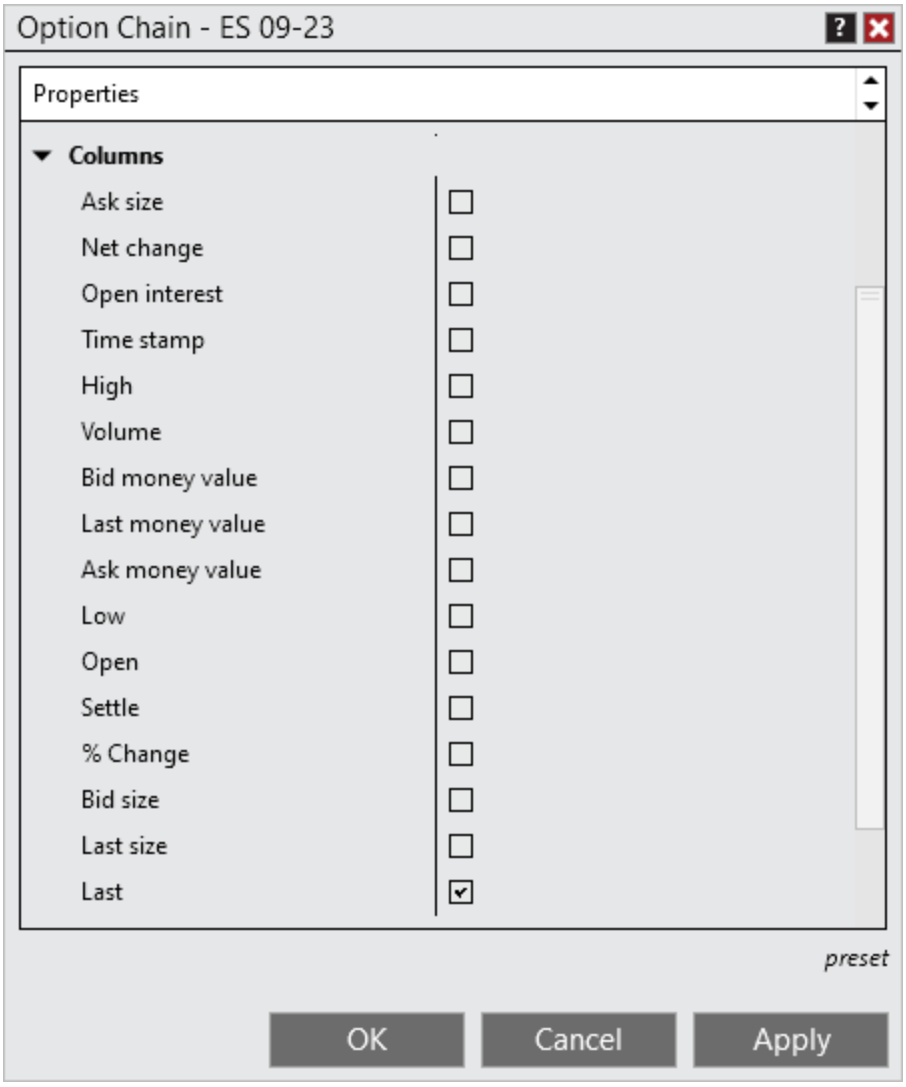
Often times multiple indicators can be loaded on a chart and it can be difficult to know what plot is for what indicator and settings. Now you can select the indicator plot and it's label will display in bold and italic to easily and quickly identify it.

Options on Futures data available with NinjaTrader connection
Feature #1633



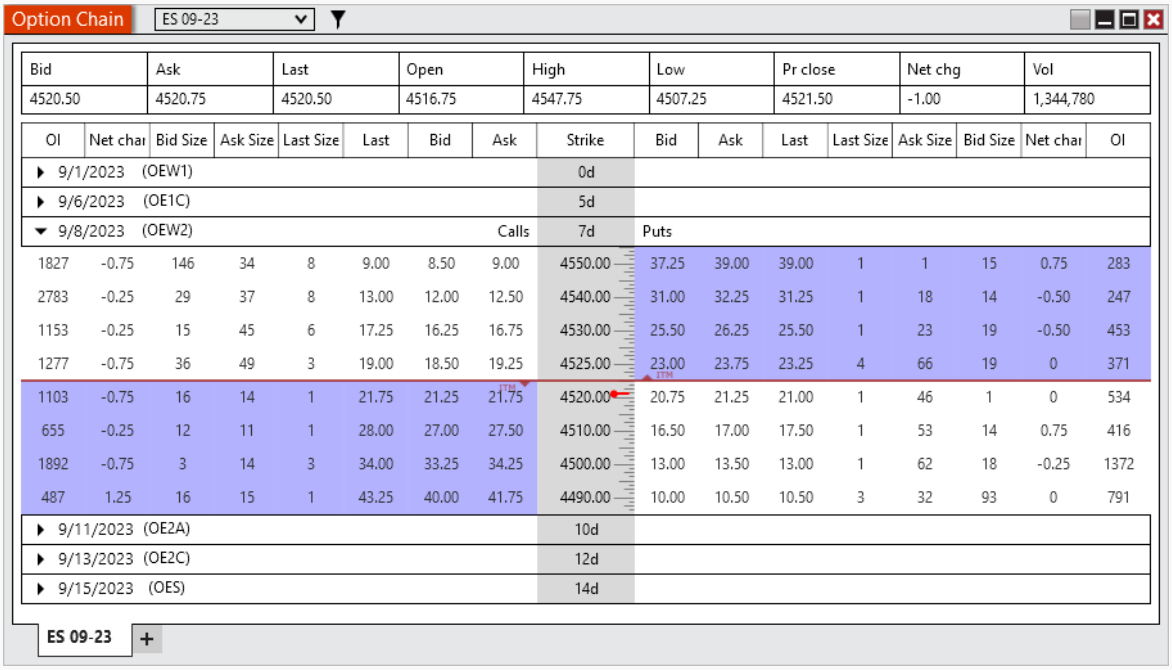
To match Web Trader, NinjaTrader Desktop can now also display option on futures data with the NinjaTrader connection.

Additional columns available with Option Chain
Feature #2882



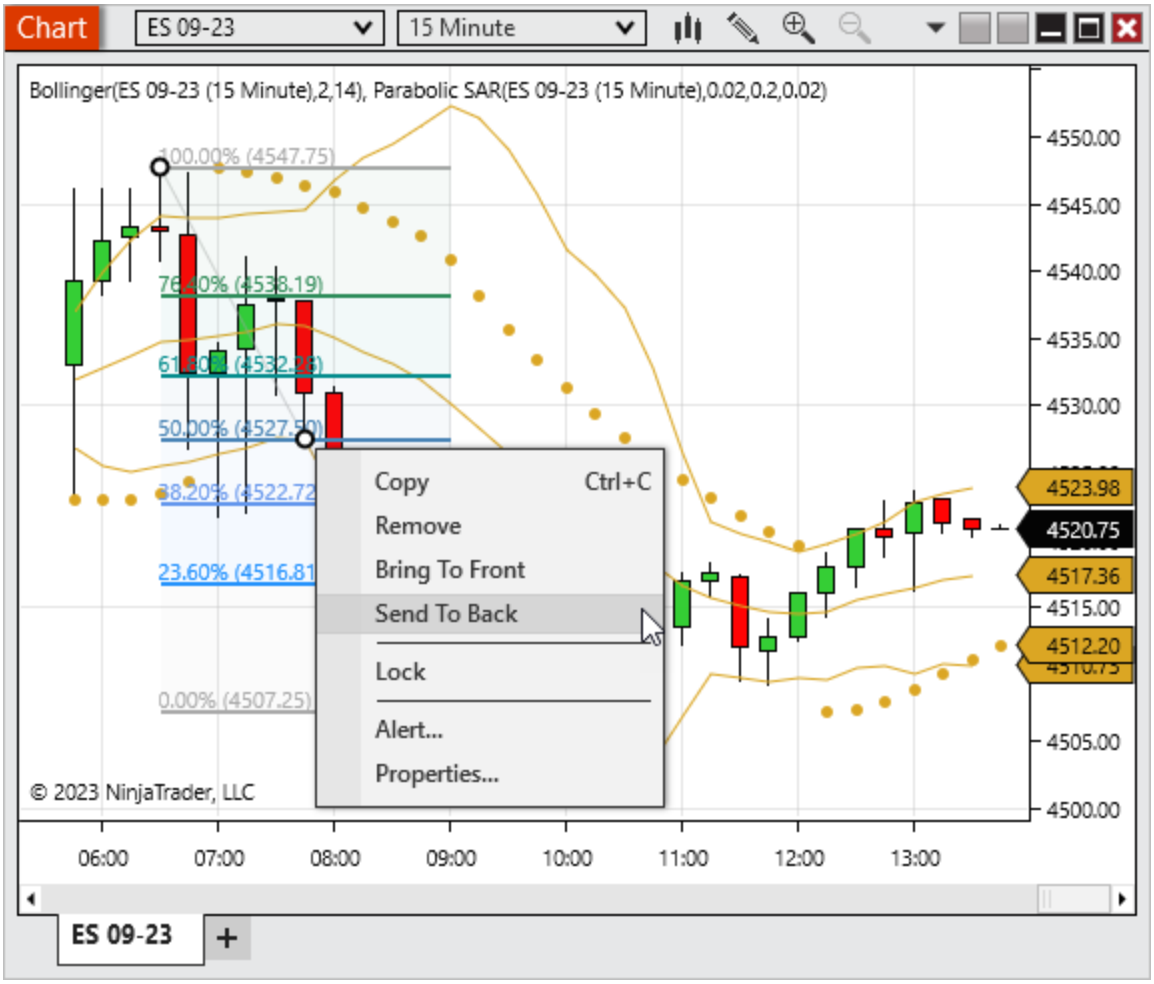
Additional columns were added to the option chain for additional analysis and to match what's available on Web Trader.

Unlimited number of columns can be added to the Option Chain
Feature #6010



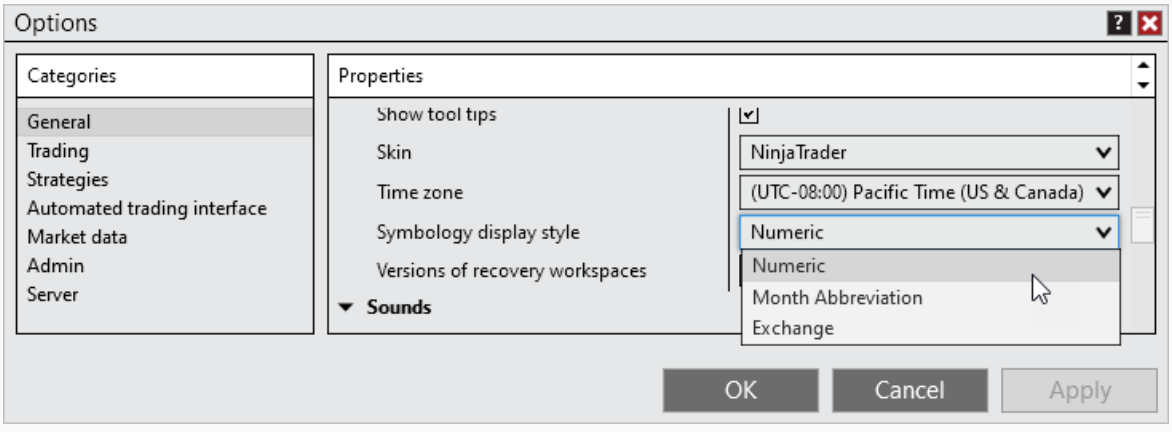
Previously only 4 columns could be added to the Option Chain. Now it is possible to enable all available columns to display at the same time.

Chart objects can be sent to back/front
Feature #197



Selecting and right clicking on a chart object now gives the ability to quickly send the object to the front or back of the chart.

Symbology Display Style can be managed within Properties
Feature #6579



You can now make Symbology Display Style changes directly within the platform, eliminating the need to access the Client Dashboard for this task. This can be done under Tools> Options> General.

Updated compiler to use Roslyn
Feature #1642

This enhancement now grants you the flexibility to leverage C# features up to version 8, providing even more versatility in your scripting endeavors.

Workspace loading indicator
Feature #4451

An animated NinjaTrader icon now displays while workspaces are loading at startup to help reassure everything is in motion and processing smoothly.

Warning: Some custom scripts that reference old versions of Newtonsoft will need the references updated to Newtonsoft 13

Issue #	Status	Category	Comments
6018	Fixed	Alerts	After an alert sent by a NinjaScript was cleared from the output window it will re-appear in new Alert windows
3341	Fixed	Backup & Restore	Custom instruments were not saved in backup file
3719	Fixed	Backup & Restore	Backup reminder prompt did not open the backup window
4438	Fixed	Backup &	Database was not properly backing up/restoring

		Restore	
247	Fixed	Chart	Resolved a scenario where the data series window could open unexpectedly
322	Fixed	Chart	Print chart function wasn't working
4111	Fixed	Chart	Mini data had display issue with a large list of indicators
7481	Fixed	Coinbase	Level 2 data stopped working
6560	Fixed	Continuum/CQG, Orders	Resolved a scenario where an order that was correctly sent got rejected with an incorrect tick value
363	Fixed	Control Center	Client Dashboard did not load with some set-ups
2253	Fixed	Control Center	Intermittently the Watch button wasn't detecting if a stream was live
2792	Fixed	Control Center	WebView Task Bar icon was missing
2897	Fixed	Control Center	Client Dashboard could require multiple attempts to load

2898	Fixed	Control Center	Client Dashboard didn't have a header title
2965	Fixed	Control Center	Exception in Vimeo interactions when no data is received
4590	Fixed	Control Center	In product announcements were not received if no multi-provider connections were created
4658	Fixed	Control Center	Gross Realized PnL calculated incorrectly when a lost connection restored
5012	Fixed	Control Center	WebView instances stayed running after closing NinjaTrader
7161	Fixed	Control Center	Multiple part fills resulted in unexpected gross realized PnL values
4570	Fixed	Database	Error occurred with new database if server side ATMs were used
3435	Fixed	Drawing Tool	Line drawing tool 45 degree snap could place second anchor in wrong location
4636	Fixed	Drawing Tool	Global drawing objects disappeared from charts when a template was applied

5773	Fixed	Drawing Tool, Workspaces	Resolved a scenario where a global drawing tool and a multi-data series chart resulted in a lock up
7359	Fixed	Forex.com	Forex.com demo accounts could not be connected to
174	Fixed	Forex.com, Orders	Order updates for DFT instruments, which are not supported, were not being ignored
2257	Fixed	Historical Data Window	Always showed in numerical format
170	Fixed	Historical Data Window, Charts	Excluded historical data showed in charts and added Changed rows in Historical Data window
6115	Fixed	Indicator	Indicators were not able to plot on first bar when only one bar of data was loaded
2948	Fixed	Installer	"Repair" installation ignores custom folder and breaks the installation
209	Changed	Instruments	Symbology display styles updated for Options on Futures and Index instruments
2258	Fixed	Instrume	Instruments at times were not rolling over

		nts	
6819	Changed	Instruments	Rollover window text updated and link included for more information
3437	Fixed	Interactive Brokers	Resolved a scenario that prevented some non-US users to not be able to load data
1644	Fixed	Kinticeck, Option Chain	Some expirations didn't show any strike prices
4696	Changed	Log In	When reCaptcha is needed it now shows immediately and wait time to log in has been reduced to 15 seconds
2868	Changed	Log In	Added icon to show entered password
1375	Fixed	Log In	After log in attempts with incorrect credentials, first log in with correct credentials showed an error still
1838	Changed	Log In	Updated Google sign in authorization
2254	Fixed	Log In	Recaptcha window was too small when displayed images to verify
4670	Fixed	Log In	Closing Recaptcha window prevented ability to log in
4961	Fixed	Log In	Resolved a scenario where log in failed as task canceled

5188	Fixed	Log In	Style fixes to log in window
4955	Changed	Market Analyzer, Workspaces	Add SPK to Market Analyzer in default workspaces
2881	Fixed	Market Analyzer, NinjaScript	Resolved s scenario where a custom script didn't properly display with blank values
320	Fixed	NinjaScript	Resolved a scenario where a 'Message sent successfully' log wasn't displayed
2265	Changed	NinjaScript	NTDirect.dll was no longer working. It has been removed and support has been dropped
3429	Fixed	NinjaScript	Calling a barsAgo index of 1 when CurrentBar is 0 displayed an unexpected bar
3802	Fixed	NinjaScript	Scripts referencing HLCCalculationMode enum could not be exported/imported as an assembly
2957	Changed	NinjaScript	Added additional precision to cryptocurrency volume
4729	Fixed	NinjaScript	AddDataSeries on pre-rollover instrument using custom trading hours failed with incorrect error message

8753	Changed	NinjaScript	Updated Newtonsoft to version 13
4287	Changed	NinjaScript Editor	Only Visual Studio 2019 or 2022 are now supported
4009	Fixed	NinjaScript Editor	Some system indicators could not properly be copied
3431	Fixed	NinjaScript, Drawing Tool	There was inconsistency with start/end time errors with @Lines and @Shapes
362	Fixed	Order	TIF selector was not preventing using of invalid dates
326	Fixed	Order Flow +	Order Flow Volume Profile drawing tool plotted above price bars if drawn before the first Split date
3805	Fixed	Order Flow +	There was an incorrect blank cell on Order Flow + Volumetric Bars when using Diagonal Imbalance
4698	Fixed	Orders	A token not being able to renew prevented order/position updates
4700	Fixed	Orders	A part filled order that was then canceled still showed as pending cancel

5006	Fixed	Orders, Server Side ATM	Attempting to use locally simulated orders with server side ATMs didn't show an error
169	Fixed	Playback	Playing back with historical data and switching instruments caused time to skip
6970	Changed	Playback	Playback account's settings can now be independently changed from the local simulation account
6552	Fixed	Playback, Chart	Daily charts displayed the session high/low prior to reaching the levels
5800	Fixed	Playback, Indicator	Resolved a scenario where an error occurred with the Price Line indicator and switching between Playback and a live connection
329	Fixed	Playback, NinjaScript	Incorrect prints of bar indexes could occur when disabling and re-enabling strategy via the Strategy tab that was generated in a chart
406	Changed	Regionalizetion	Buy/sell buttons locations are now regionalized for Korea and can be changed under Options> Trading
704	Changed	Regionalizetion	Primary and secondary buy/up & sell/down colors have been regionalized for Korea
5408	Fixed	Server	Resolved a scenario where order updates were delayed

		Side ATM	
298	Fixed	Share Adapter	Could not send email while using a Hotmail account
1643	Fixed	Share Adapter	Connect Button for Gmail Share Service is Collapsed in NinjaTrader Desktop
6287	Changed	Share Adapter	Removed Twitter/X share service
933	Fixed	Strategies	Orders using isLiveUntilCancelled false couldn't be cancelled with CancelOrder()
2255	Fixed	Strategy	Copying and pasting a multi-data-series strategy it could change to the secondary series
3427	Fixed	Strategy	There was inconsistency between sound files played for orders filled via SetStopLoss() and SetProfitTarget()
5426	Fixed	Strategy	When clicking 'View code' in Strategy Builder, SMA() did not show when assigning it to a custom Series using a different custom series as input for SMA()
6587	Fixed	Strategy	Resolved a scenario where SetParabolicStop could move an order further away
168	Fixed	Strategy Analyzer	A Walk Forward optimization with Break At EOD off duplicated results from Friday into the weekend

178	Fixed	Strategy Analyzer	Some inputs could not be modified after an optimization if Optimize Data Series was set to True
2259	Fixed	Strategy Analyzer	Genetic optimization with a generation size of 2 got stuck running and did not complete
4052	Fixed	Strategy Analyzer	Clicking 'View' in the View Strategy column of an AI Generate optimization with an aggregated basket test resulted in an error
4530	Fixed	Strategy Analyzer	BarsSinceNewTradingDay could be negative 10,000+ when running Genetic Optimization with IsInstantiatedOnEachOptimizationIteration=false
1645	Fixed	SuperDOM	PnL Freezes when hovering mouse over the Dynamic SuperDOM
6051	Fixed	TD Ameritrade	Resolved an 'object reference not set to an instance of an object' error when connecting
4641	Fixed	Tick Replay, Indicator	Swing indicator OnPriceChange showed different results than OnBarClose when Tick Replay was enabled
5345	Fixed	Trade Performance	Some regions/timezone settings prevented reports from running

5421	Fixed	Trade Performance	In some scenarios trades could show with the wrong timezone
5568	Fixed	Trade Performance	PC Region settings set to Poland prevented reports from running
6026	Fixed	Trade Performance	Resolved a scenario where historical MIT orders caused an error
6107	Fixed	Trade Performance	Resetting a template did not reset AI Generate properties
2246	Changed	Workspaces	Removed Eurex instruments from the default workspaces

3.2 8.1.1.7

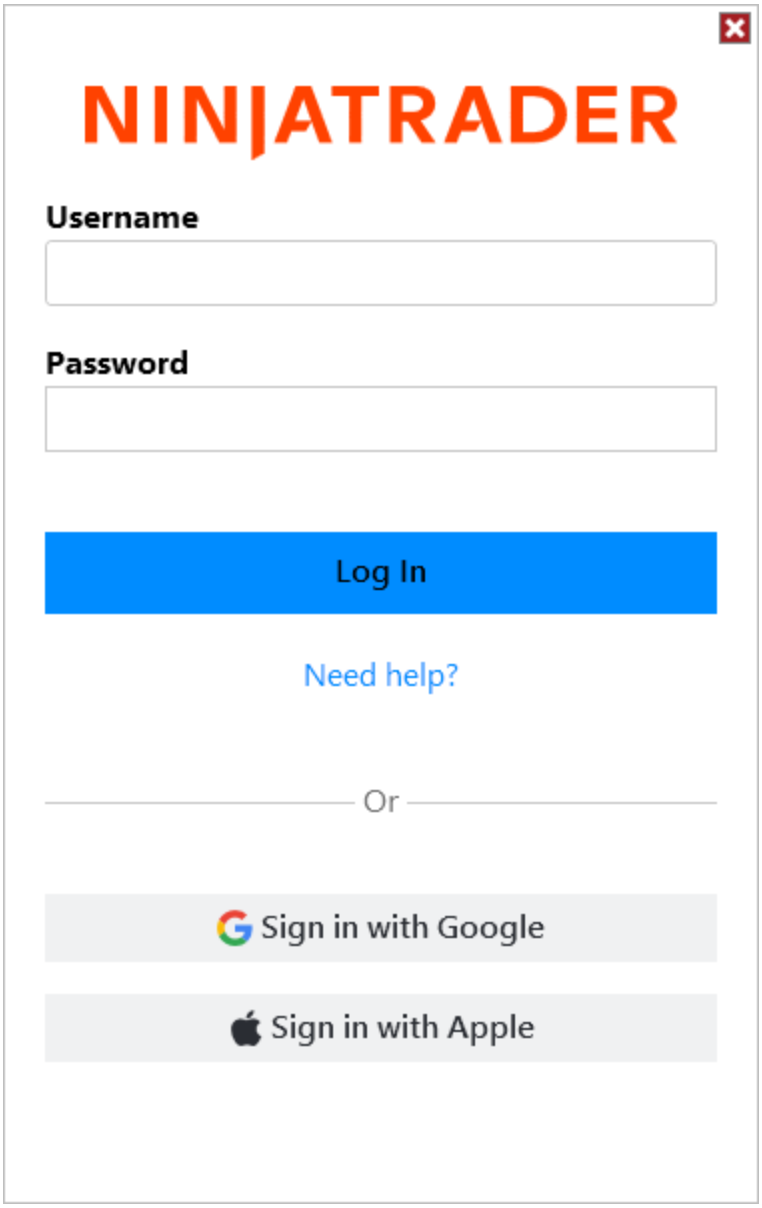
See additional patch notes at the bottom

8.1.1.0 Release Date

March 4, 2023

Features
User based log in with associated plans and add-ons The new Log In screen adds a layer of security while removing the need for license keys. Account plans and power-ups enabled within the Client Dashboard or imported will be

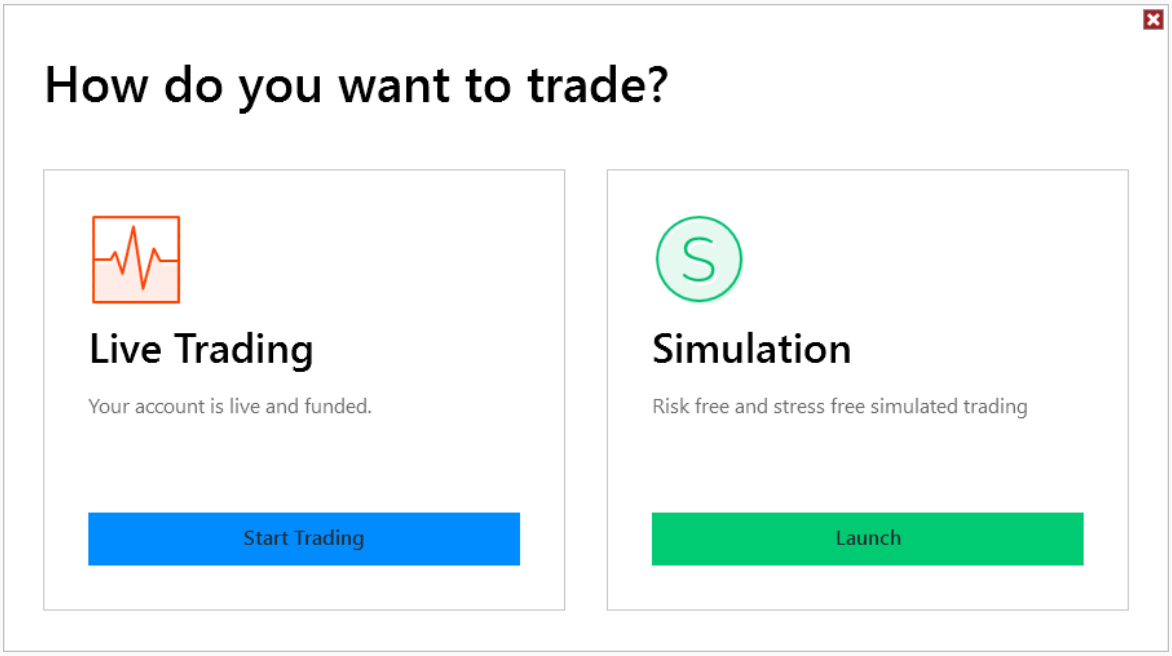
associated with the username. There no longer is a need for a license key. Upgrading NinjaTrader will import entitlements from your old license or you can import the license within the Client Dashboard.
Users who signed up for a demo account with the Sign in with Google/Apple feature can quickly connect by selecting those options. Live accounts require a Username and Password.



Warning: The log in credentials are only to provide access to your account, entitlements, and server side ATM templates. Any multi-provider connections,

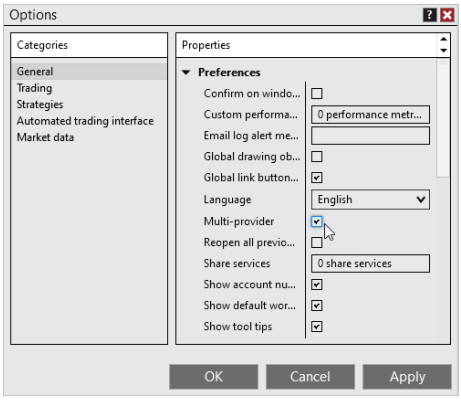
NinjaScripts, workspaces, etc saved to your local computer will be available to any log in to your local computer.

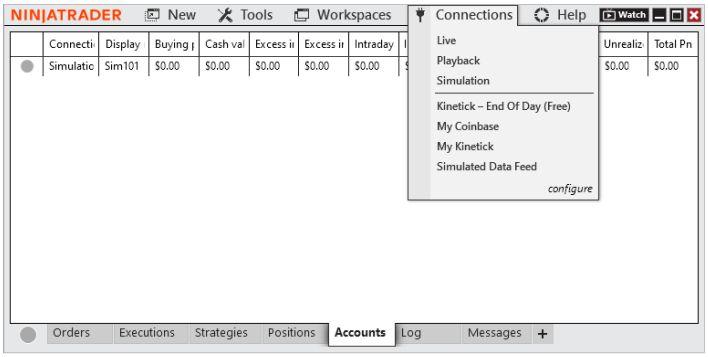
Trading Mode window
After logging in, you will be presented with a Trading Mode screen. The status of your live or simulation account will be display here and you can select how you would like to continue. You can now connect to Live and only have your live account available, preventing trading to the simulation account in error.
The Trading Mode screen will not display when Multi-provider mode is enabled.



Multi-provider mode
Multi-provider mode can be enabled or disabled under Tools> Options> General. When enabled, you will have the ability to configure and connect to other providers. Additionally, the local simulation accounts will be available to trade. The Trading Mode screen will be skipped so you can manage what connection(s) you want to connect to under the Connections menu.

When disabled, you can opt-in to use server side ATMs (beta). The local simulation account will be disabled, so you can connect to the server side simulation account when you desire.

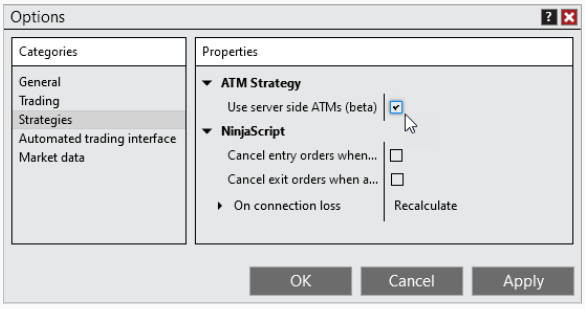


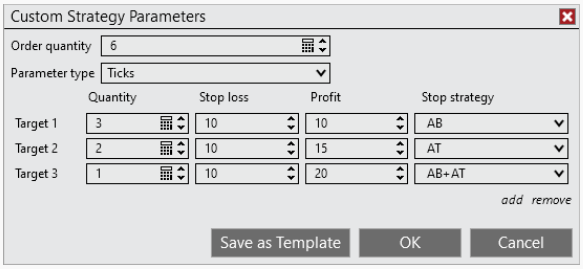


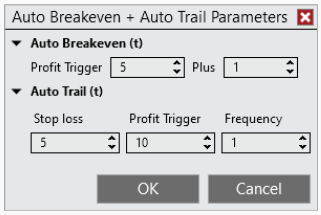
Warning: Multi-provider connections are available to any user that logs in to your local computer. If other NinjaTrader users are connecting from your computer, it is recommended to configure your connections to **Ask password on connect**.

Server side ATMs (beta)

Under Tools> Options> Strategies, you can opt in to use server side ATMs with the NinjaTrader connection. They are only available with the NinjaTrader connection and are disabled when Multi-provider is enabled. Server side ATMs will continue to function, even when you are not connected to NinjaTrader. Templates are saved to the server and can be used on any device you log in to. There are differences in functionality between the new server side ATMs and the previous local ATMs. **Ensure to review the Server Side vs Local ATMs section of the help guide to ensure your orders will function as you intend.**



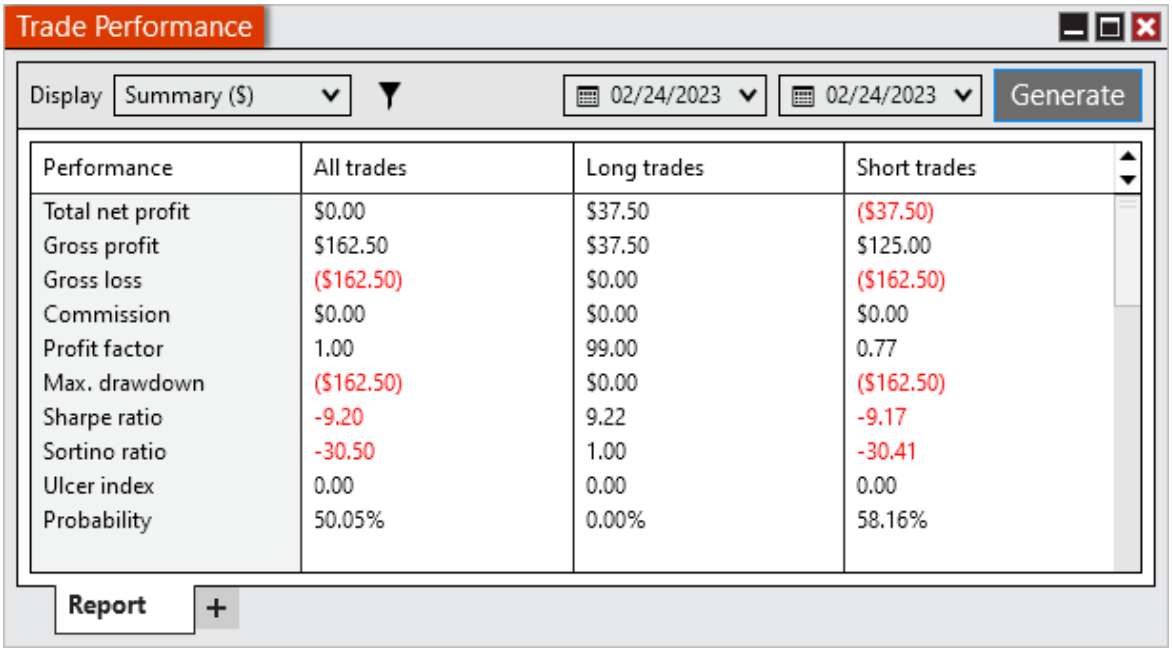




Warning: When an entry order is submitted with a server side ATM the stop loss and target will display as Suspended, indicating they will not become active until the entry is filled. The orders can be modified before the entry is filled. Any scripts that check order state should be tested to ensure the new order state doesn't result in unintended results.

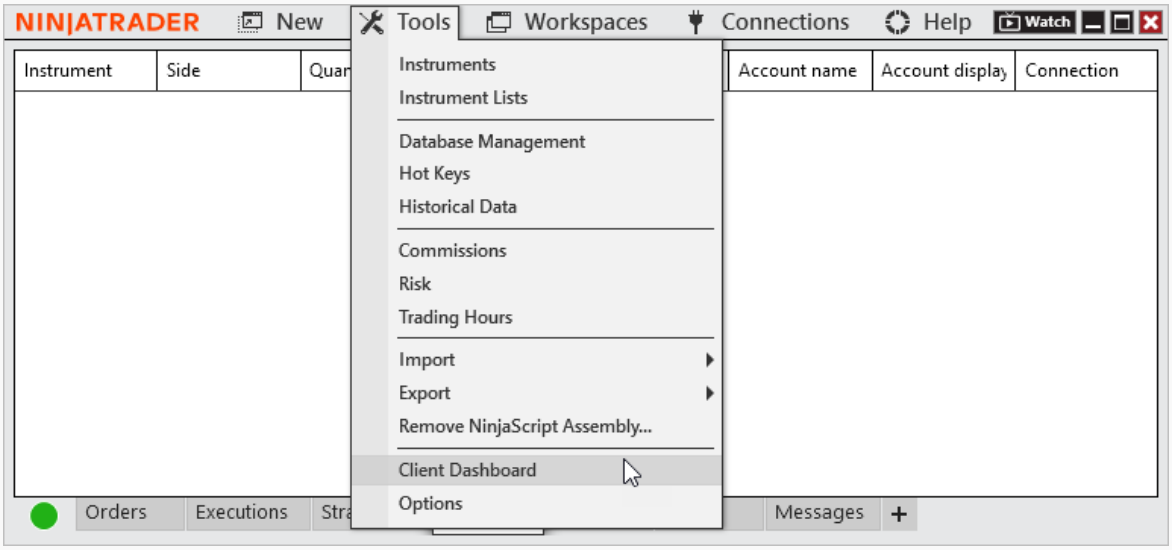
Server side Trade Performance reports

Trade Performance reports for the NinjaTrader connection are pulled from the server. NinjaTrader will no longer require to be connected as orders are filled to give accurate reports.



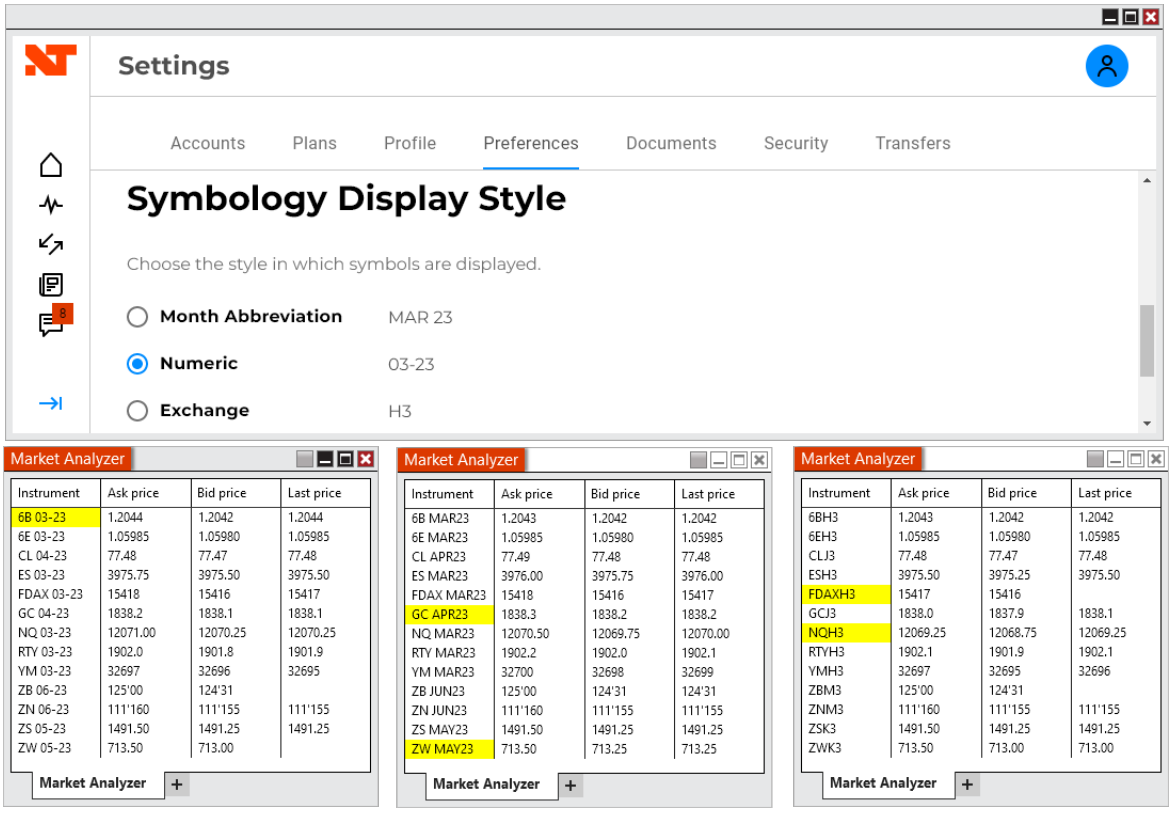
Client Dashboard access

Under Tools there is a link to the Client Dashboard. The Client Dashboard can be used to change your plan, add plan power-ups, perform transfers, view your statements, etc.



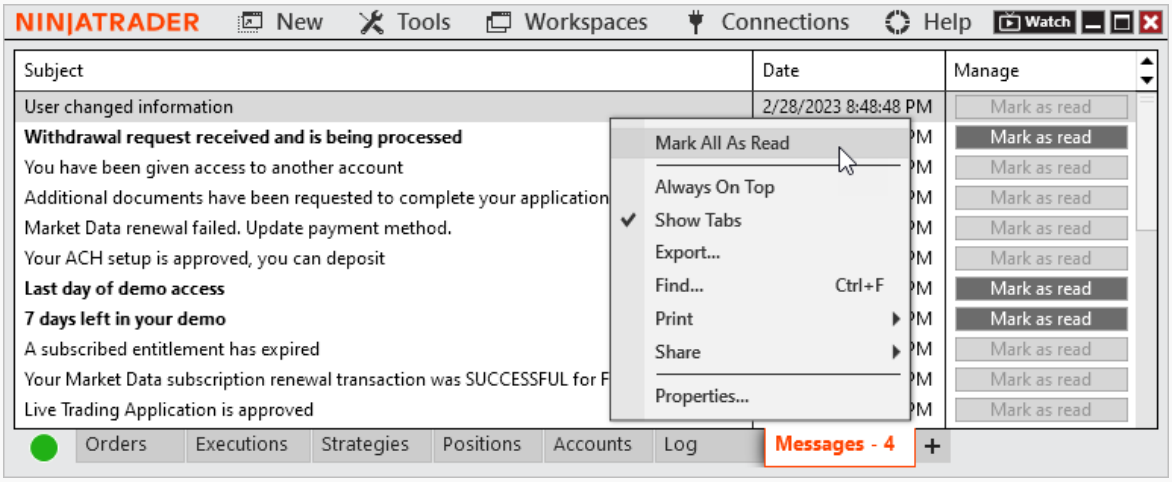
Customizable contract month display format

NinjaTrader can now display contract months in a format of your choosing. The Symbology Display Style can be selected within the Client Dashboard and will be applied to the Desktop, Web, and Mobile platforms.



Messages tab

The Messages tab will display important messages to your account. An indication of how many unread messages there are will be displayed within the tab.



Issue #	Status	Category	Comments
15450	Done	Alerts	Alerts window did not allow numeric value to be compared with numeric value
15387	Fixed	Alerts, Strategy Builder	Disable BarsAgo & Offset for CrossAbove / Below in Strategy Builder and Alerts as only the series is ever compared
15555	Fixed	Order Flow+	Volumetric Bars buy imbalance boxes could appear outside candle box
14760	Changed	Continuum/CQG	Web API now uses oAuth based authentication

15349	Fixed	Interactive Brokers, Chart	Multiple Change Order requests on paper account caused chart window to freeze
15309	Fixed	NinjaScript Editor, Strategy	Renaming strategy that changes parameter names caused compile errors
15416	Fixed	NinjaScript Editor	Resolved a scenario where clicking on a compile error resulted in an error
15440	Fixed	NinjaScript Editor, Workspaces	Tabs could scroll unexpectedly when switching between them from a saved workspace
15537	Fixed	ShareAdapter	iCloud share service could not send messages
15418	Fixed	Strategy Analyzer	Backtest type reverted when selecting a strategy
15457	Fixed	Workspaces	Closing default workspace on new install could prevent proper creation of new savable workspace

15561	Fixed	Workspaces	Warning of closing default workspaces did not function as expected

8.1.1.1 Release Date

March 8, 2023

Issues #	Status	Category	Comments
617	Fixed	Charts	Setting Symbology Display Style to Exchange resulted in daily charts being slow to load
761	Fixed	Data	Historical data was saving based on Symbology Display Style
390	Fixed	Log In	Google and Apple log ins could fail
366	Changed	Log In	Log In window now saves username by default
399	Fixed	Log In	Log in error was not removed when attempting to log in again
626	Changed	Log In	Changed "Need help?" To "Forgot Username? | Forgot Password?" on Log In window
650	Changed	Log In	Optimized the log in experience and added a loading indicator
389	Fixed	NinjaScript	NinjaScript Add-Ons would fail to load if it utilized NinjaTrader.Cbi.License.MachineId

401	Fixed	NinjaScript	NinjaScript Add-Ons would fail to load if it utilized Newtonsoft.Json

8.1.1.2 Release Date

March 22, 2023

Issues #	Status	Category	Comments
1426	Fixed	ATMs	Deleting an ATM template set the ATM dropdown to None on all windows
153	Added	Log In	A captcha was added to allow log in after too many failed log ins from IP address
409	Fixed	NinjaTrader Connection	Resolved a scenario where users with a live account couldn't access a simulation account
1391	Fixed	NinjaTrader Connection	Resolved a scenario where Account Type was locked to Simulation
1444	Fixed	NinjaTrader Connection	After many connections drops the application could crash
1217	Fixed	NinjaTrader Connection	Configured connection name could change
1406	Fixed	NinjaTrader Connect	Resolved a scenario where position and order updates received an error that they were for an unknown symbol

		ion, Orders	
1641	Fixed	NinjaTrader Continuum, SuperDOM	Resolved a scenario where some accounts that should be available in the SuperDOM were disabled

8.1.1.3 Release Date

March 28, 2023

Issues #	Status	Category	Comments
1941	Fixed	Logs	Security fix for local tracing
1846	Fixed	NinjaTrader Connection	Last trade events could be marked with incorrect bid/ask prices

8.1.1.4 Release Date

April 27, 2023

Issues #	Status	Category	Comments
2073	Changed	Continuum, CQG	Updated a CQG Server Certificate. All users connecting via Continuum or CQG are required to update to this version on or before June 23th, 2023.
2252	Fixed	NinjaTrader	In some scenarios account name didn't display as expected

		Connection, Multi-provider	
1355	Fixed	Trade Performance	Resolved multiple scenarios that could cause no results to be returned

8.1.1.5 Release Date
May 1, 2023

Issues #	Status	Category	Comments
3319	Fixed	NinjaTrader Connection	Resolved a scenario where stop limit orders would get an error

8.1.1.6 Release Date
May 16, 2023

Issues #	Status	Category	Comments
3484	Fixed	ATMs	Resolved a scenario that applied a previously selected ATM to a stop limit order
3299	Fixed	Trade Performance	Resolved additional scenarios that could cause no results to be returned

8.1.1.7 Release Date
June 6, 2023

Issues #	Status	Category	Comments
4600	Fixed	ATMs, Chart Trader	ATM strategies attached to stop limit orders did not deploy on Chart Trader
2262	Fixed	Connections	Resolved a scenario where a connection would not save
3729	Fixed	Control Center	Watch button didn't show when live at times and could error when selecting
4113	Fixed	NinjaTrader Connection	Preemptive property updates to prevent connection issues
4605	Fixed	Trade Performance	Multiple fixes to reports, report values, and errors

3.3 8.0

NinjaTrader release notes can be found below, if you have any questions on a specific release please contact platformsupport@ninjatrader.com

Version	Released
8.0.28.0	April 27, 2023
8.0.27.1	November 29, 2022
8.0.26.1	April 27, 2022
8.0.25.0	October 19, 2021
8.0.24.3	March 10, 2021
8.0.23.2	November 16, 2020

8.0.22.2	June 2, 2020
8.0.21.1	February 25, 2020
8.0.20.1	December 5, 2019
8.0.19.1	September 23, 2019
8.0.18.1	May 6, 2019
8.0.17.2	January 28, 2019
8.0.16.3	October 16, 2018
8.0.15.1	July 30, 2018
8.0.14.2	May 24, 2018
8.0.13.1	March 26, 2018
8.0.12.0	January 30, 2018
8.0.11.1	December 6, 2017
8.0.10.0	November 7, 2017
8.0.9.0	September 12, 2017
8.0.8.0	July 26, 2017
8.0.7.1	June 6, 2017
8.0.6.1	April 17, 2017
8.0.5.2	March 6, 2017
8.0.4.0	January 31, 2017
8.0.3.0	January 9, 2017
8.0.2.0	December 5, 2016

8.0.1.0	November 14, 2016

3.3.1 8.0.28.0

8.0.28.0 Release Date

April 27, 2023

Issue #	Status	Category	Comments
2072	Changed	Continuum, CQG	Updated a CQG Server Certificate. All users connecting via Continuum or CQG are required to update to this version on or before June 30th, 2023.

3.3.2 8.0.27.1

Attention 32-Bit Users:

- 64-bit has become the standard for applications in the Windows environment
- As we continue to modernize our desktop platform and clearing up performance related confusions around 32 vs 64 bit NinjaTrader Desktop usage
- **NinjaTrader 8.0.27.0 will be solely a 64-bit Windows application**, this means the installation directory will change **from C:\Program Files (x86) to C:\Program Files**
- To initiate the update simply follow the installer and restart the PC.

8.0.27.1 Release Date

December 19, 2022

Issue #	Status	Category	Comments
15439	Fixed	Control Center	Email to support could fail
15466	Fixed	Options Chain	Options did not display

15526	Fixed	Interactive Brokers, Charts	There was a gap of data on charts from when connecting
15527	Fixed	NinjaScript	Resolved a scenario where importing a script resulted in an error

8.0.27.0 Release Date

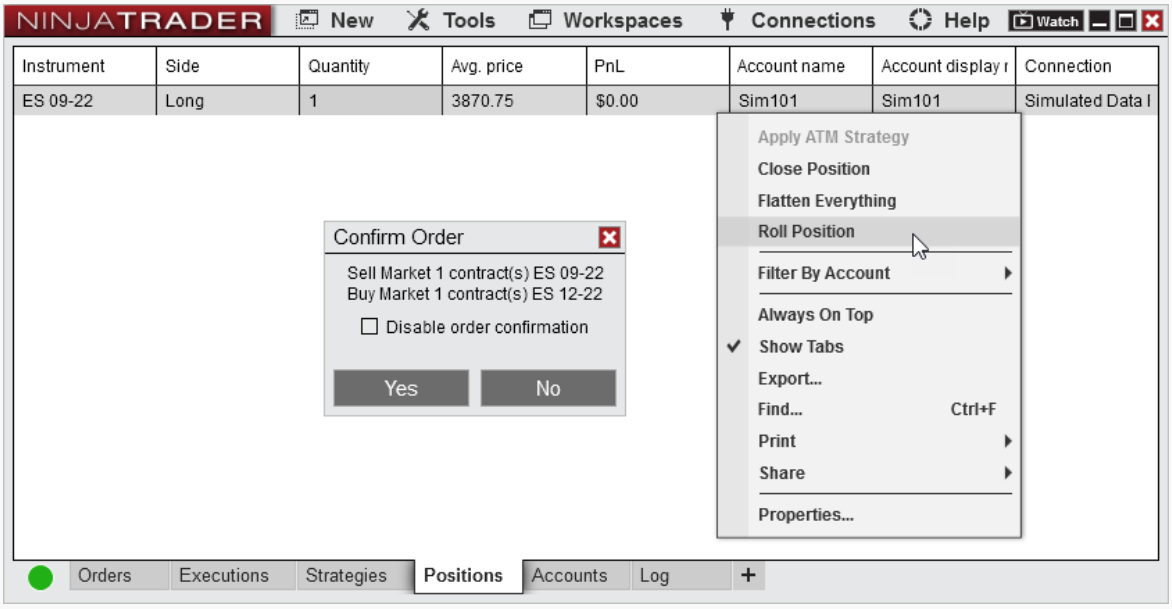
November 29, 2022

Features

SQLite Database technology
Database
Feature #15258

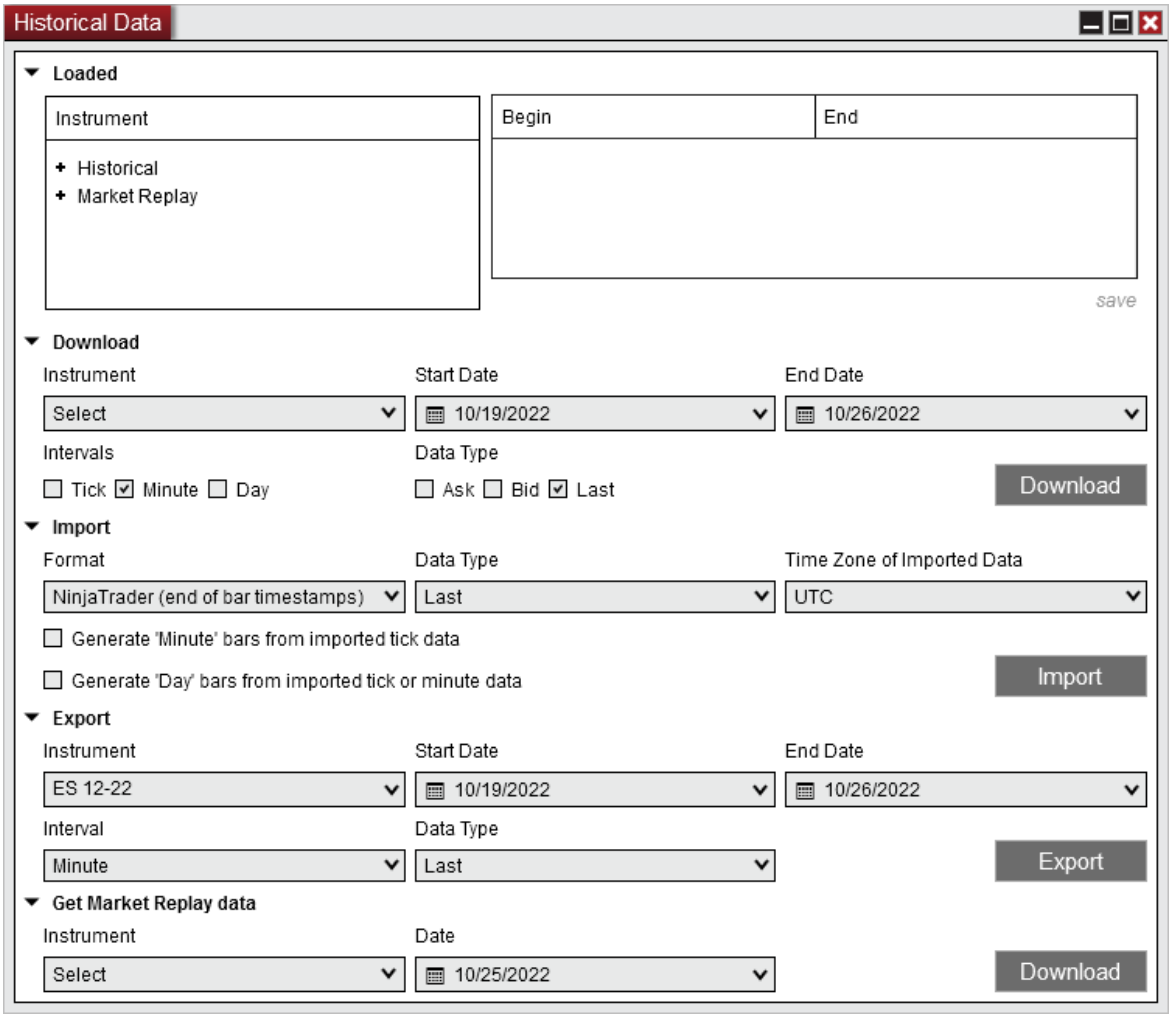
We have updated our core internal database technology from the deprecated Microsoft SQL CE to SQLite. Supported / documented NinjaScript code will **not** be impacted by this change; however if you have specifically developed your Add-On to call or interact with the Microsoft SQL CE database you should review and update code for compatibility.

Added ability to roll positions from Control Center position grid
Orders, Position Display
Feature #14350



On the Positions tab of the Control Center you can now right click on Futures position and select Roll Position. This will send a Market order to exit the position in the current contract month and send another Market order in the next contract month to roll your position.

Historical Data window tabs unified and displays prior to connecting to Playback
Historical Data Window, Playback
Feature #15305



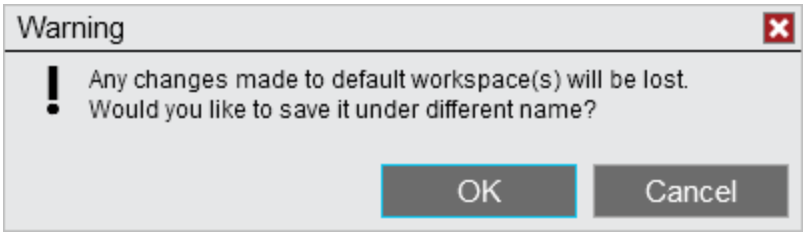
The Historical Data Window has now been unified into a single tab for simplicity. Now you can view what data is already loaded while deciding what data to download.
Additionally, the Historical Data Window will now appear before connecting to Playback. This will enable you to view what Market Replay data is loaded and download desired data before continuing.

Place Lines at 45 Degree Increments
DrawingTool

Feature #15263

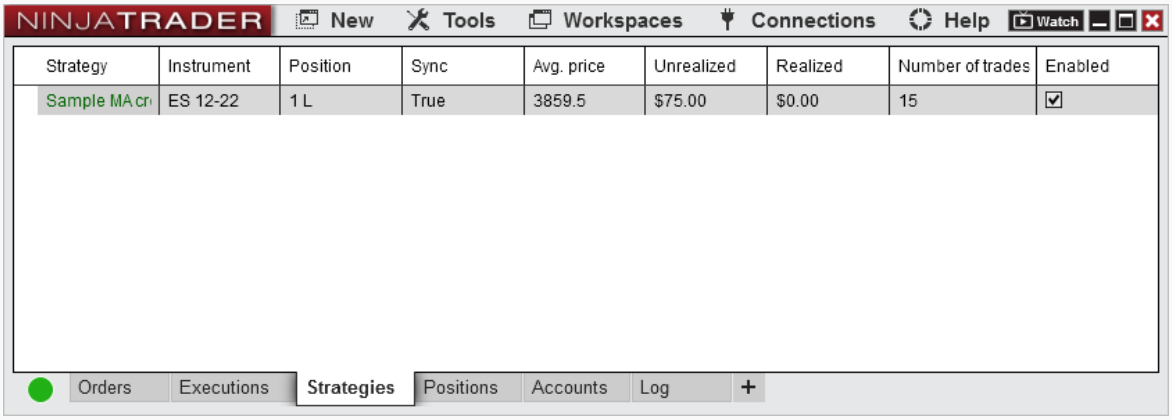
With the line drawing tools, after placing the first anchor, you can hold SHIFT on the keyboard and move the mouse around to adjust the line in 45 degree increments. This is based on chart scaling at the time the line is placed.

Added Save-As Option When Closing Default Workspace
Workspaces
Feature #15222



Now when modifying a default workspace, which cannot be save, you will be asked if you would like to save it as a different name

Added Property to Show Number of Trades on Strategies Tab
Control Center, Strategy
Feature #15351



The Number of Trades column in the Strategies tab will display the number of Real-time & Historical trades that occurred with the enabled strategy. This column is disabled by default, but can be enabled by right clicking on the Strategies tab of the Control Center and selecting Properties.

All System NinjaScript types can be exported
NinjaScript Editor
Feature #14665

Previously only indicators from System NinjaScript could be exported. Now all System NinjaScript types can be exported to allow successful exports. For example an indicator that references a share service could now successfully be exported.

Issue #	Status	Category	Comments
14914	Fixed	Alerts	German translation was missing in the Alerts window
15277	Fixed	Alerts, Chart	Vertical lines incorrectly showed as available for alerts on a chart
15239	Fixed	ATI	OIF files could generate unhandled exception when sending several OIFs
15245	Fixed	ATI	Orders submitted via external application using the DLL interface

			could incorrectly labeled as OIF files
15214	Fixed	ATM Strategies	Add target feature added target to first target price even after first target was already filled
15320	Fixed	ATM Strategies	Auto Breakeven could use entry price rather than new avg entry price as trigger when scaling into ATM
15357	Fixed	ATM Strategies	ATM and Stop Strategy templates saved with only a blank character cause the platform to malfunction
15338	Fixed	Bars	Futures instrument bars request could get stuck when time range requested outside of valid contracts
15061	Fixed	Chart	Repeated Instrument Quick Searches could slow down Quick Search display
15278	Fixed	Chart	Plot executions could show up in multiple languages when switching to Russian
15292	Fixed	Chart	Unhandled exception could be thrown when loading up workspace and connecting to data feed
15371	Fixed	Chart	Indicator dialog modal window could display out of screen boundary
15280	Fixed	Chart Trader	Instrument selected on Chart Trader does not remain in place on a chart with multiple tabs that have a secondary data series

15355	Changed	Chart Trader, Orders	Default Quantity for Stocks changed from 100 to 1
15215	Changed	Chart, Drawing	Fibonacci drawing objects selection improvements
15303	Fixed	Chart, Drawing	Drawing tools do not allow you to use scroll wheel to continue plotting
15326	Fixed	Chart, NinjaScript	DrawingTools GetAttachedToChartBars() method returns previous instrument switched from chart after returning to the attached Instrument
15401	Fixed	Control Center	Collapsed Control Center showing just icons could show drop-down text after restart
15337	Fixed	Control Center, NinjaScript	Opening strategy dialog and connection dialog could cause deadlock
15218	Changed	Control Center, Strategy Analyzer	Removed tool-tip timeout on Strategy Parameters
15166	Fixed	CQG	Live MIT orders could not be changed by NinjaScript
15408	Fixed	CQG, Continuum	Unexpected duplicate order update message could be received
15078	Fixed	CQG, NinjaScript	WebAPI connections could miss strategy OnPositionUpdate() iterations when entering a position

15261	Fixed	CQG, NinjaScript	SetTrailStop() could see errors stating the price was the same on live/paper accounts
15189	Fixed	Database, Strategy	Instrument rollover does not switch strategy instruments on strategies tab
15283	Fixed	Drawing, DrawingTool	Only some Global Drawing Objects reappeared on a recreated chart prior to a platform restart
15289	Fixed	Drawing, Rithmic	NullReferenceException could be seen restoring workspace with drawing objects on Rithmic connection
15098	Fixed	Forex.com	Forex.com executions from several days prior could be repeated
15175	Fixed	Forex.com	Current daily bar could be incorrect for Forex.com
15350	Changed	Forex.com	Order URL updated
15354	Fixed	FXCM	Unknown order type could be seen from FXCM when trade was closed in TradingStation
15272	Fixed	Indicator	BOP indicator did not have a ZeroLine
15246	Changed	Indicator	CurrentDayOHL / PriorDayOHLC Price Marker formatting improvement for treasury futures
14730	Fixed	Indicator, Log	If the Indicator label is removed, errors in the log now show indicator class name

15008	Fixed	Indicator	HMA did not plot as expected during high volatility on certain bar types
15206	Fixed	Indicator	Parabolic SAR Input series change was not reflected
15390	Fixed	Indicator	Correlation plot contained gaps when both series have same time frame with same trading hours
15368	Removed	Installer	Removed 32-bit installer
15027	Fixed	Instruments	Exception thrown when opening a chart of auto added instrument
15036	Fixed	Instruments	Selecting multiple instruments to add to a list and hitting enter only added the first instrument
15213	Fixed	Instruments	Instrument selector search issue for multi-strings
15229	Fixed	Instruments	Editing auto added futures instrument forces user to define month
15399	Changed	Interactive Brokers	Updated required version of Trader Workstation & IB Gateway to 10.19-1c
15365	Fixed	Interactive Brokers	Option position updates missing
15380	Fixed	Interactive Brokers, Strategy	OnOrderUpdate() called twice for same order
15299	Fixed	Kinetick, Data	Real-time tick timestamps could be different than historical in some scenarios

15242	Fixed	NinjaScript	Modifying order quantity to 0 could result in stuck order instead of UnableToChangeOrder error
15282	Fixed	NinjaScript	Strategy containing TimeEditorKey property could result in unhandled exception
15347	Fixed	NinjaScript	Draw.Triangle does not have auto-scale applied when drawn on chart if auto-scale is set to true in NS
15216	Fixed	NinjaScript Editor	Debug warning could trigger when it should not
15306	Fixed	NinjaScript Editor	Compiling could lock up editor
15358	Fixed	NinjaScript Editor	Excluding Strategy throws unhandled exception
15309	Fixed	NinjaScript Editor, Strategy	Renaming strategy could cause compile errors on parameter name changes
15265	Added	NinjaScript, Property Grids	Browsable attribute on indicator having class scope
15362	Fixed	NinjaScript, Strategy Analyzer	IsInStrategyAnalyzer is false for Optimizations
15300	Fixed	Order Flow +, Chart	Order Flow Volume Profile plots on wrong bar for current session when using the CME RTH Trading Hours
15391	Fixed	Order Flow +, NinjaScript	AddVolumetric() could trigger outside template TradingHours

15284	Fixed	Playback, Chart	Tick chart could build a wrong bar between merge of historical and Playback data
15264	Fixed	Rithmic, Position Display	Unexpected position updates could be received
15343	Fixed	Rithmic, Position Display	Position updates could be missed
15318	Fixed	Rithmic, ATM Strategies	Too many order change/cancel requests could be sent
15122	Changed	ShareAdapter	Updated SMTP client for sending emails
15247	Changed	ShareAdapter	Implemented SMTP changes for Google
15288	Changed	ShareAdapter	Outlook SMTP config updated
14994	Fixed	Strategy, Playback	Strategy with 52 added data series caused playback controller to stop
15219	Fixed	Strategy Analyzer	Strategy Analyzer Optimization displays double values for Increment on integer inputs if typed in the main field
15322	Fixed	Strategy Analyzer	Trades on chart did not always show
15259	Fixed	Strategy Builder	Gui.CategoryOrder not applying properly to indicators
15394	Fixed	Strategy Builder	Adding draw ruler action would not compile

15400	Changed	Strategy Builder, Alerts	Disabled lookback period of 0 on CrossAbove and CrossBelow as it is unusable
15329	Fixed	Strategy, Templates	Strategy templates saving could erroneously include contract
15345	Fixed	Strategy, Templates	Strategy could be enabled on invalid Instrument
15116	Fixed	SuperDOM	Fast mouse scroll actions could be ignored
14993	Fixed	TD Ameritrade	Synchronize Account Strategy order could be rejected for Sell Short and TIF = GTC
15317	Fixed	TD Ameritrade	Strategy and ATM submitting stop and target could leave an erroneous Stop Loss order in Initialized state
15366	Fixed	TD Ameritrade	Market order submitted to stock which is halted did not reflect that the order was rejected
15367	Fixed	TD Ameritrade	Closing an opening position with an offsetting order did not cause position update to be reflected
15274	Fixed	Trade Performance	Filter options couldn't be unchecked
15375	Fixed	Vendor licensing	Vendor license timed out when fetching a big number of licenses
15037	Fixed	Workspaces	Window size could not be increased if workspace was opened prior to additional monitor setup

3.3.3 8.0.26.1

8.0.26.1 Release Date

June 22, 2022

Features
NinjaTrader Watch icon added Feature #15285 In the top right corner of the Control Center a watch icon was added to connect more directly with our daily live events. When events are streamed live, this will be indicated via green coloring.

Issue #	Status	Category	Comments
15267	Fixed	Drawing	Drawing objects removed with RemoveDrawObject() could remain in the Drawing Objects list until object is redrawn
15249	Fixed	NinjaScript	Trendlines indicator could display message after disconnect/reconnect that was targeted only for Strategy Analyzer
15294	Added	Rithmic	Added Apex connect points
15264	Fixed	Rithmic	Out of sequence position updates could occur
15271	Fixed	Workspaces	Unexpected behavior deleting _Workspaces.xml file
15298	Fixed	Workspaces, Strategy	Strategies would not be shown in new workspace unless Strategies tab was duplicated

8.0.26.0 Release Date

April 27, 2022

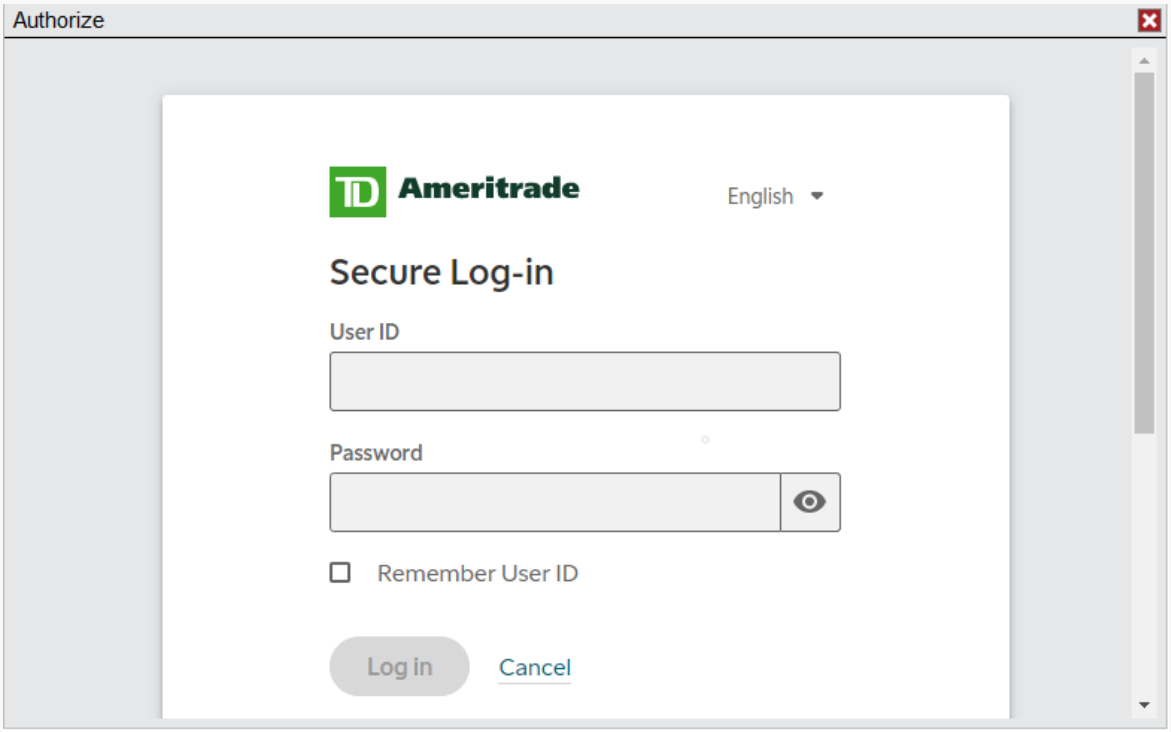
Features

Support FairX Exchange
Feature #15190

Support for the FairX Exchange was added, available products are TEC, LTEC, B5, LB5 and OIL.

TDA login authorization over WebView2
TDA Adapter
Feature #15072

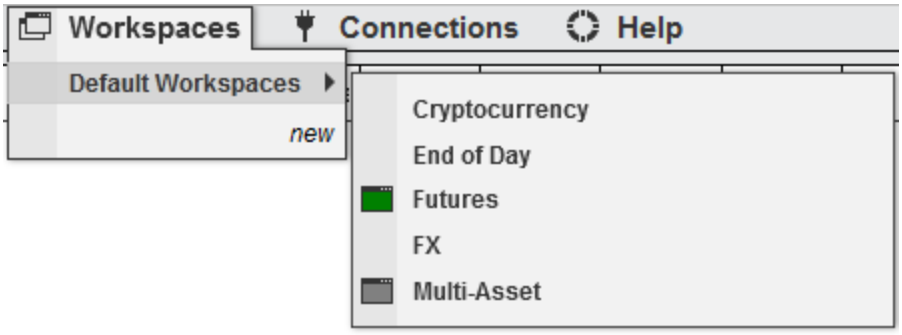
TDA users can now login and authorize their accounts over WebView2, this includes Windows 11 users.



Default workspaces are now accessible in own subfolder Control Center Workspaces menu
Workspaces

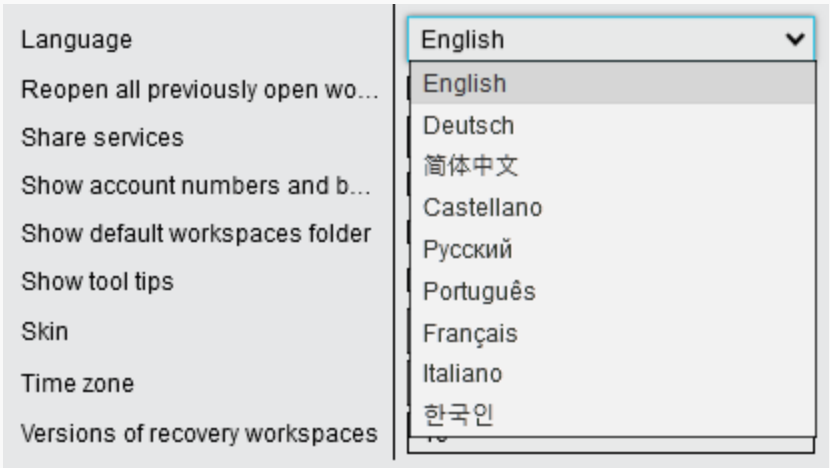
Feature # 15104

With Tools > Options > General > Preferences > Show Default Workspaces folder checked, the default workspaces shipped with NinjaTrader Desktop will always be accessible in a dedicated folder under Workspaces.



Language menu improved to show new target language non translated
Control Center
Feature # 15181

When switching NinjaTrader Desktop to a new language, the new target language is now non-translated making it more intuitive to select.



Korean language support added (Beta)
Localization
Feature # 15044

Support for Korean was added, please report any improvement suggestions via platformsupport@ninjatrader.com

Issue #	Status	Category	Comments
15053	Fixed	Adapter, Coinbase	ADAUSD is Missing historical data, throws Cast Error when opening Historical Data Manager
14826	Fixed	Adapter, CQG	CQG WebApi with Sim License Key: " Order Received order for unknown account" '16984730'
15047	Fixed	Adapter, cTrader	Modifying working order in cTrader is not reflected in NT
15121	Fixed	Adapter, Forex.com	Close Operation could time out with large number of orders
14953	Fixed	Adapter, FXCM	MIT orders could be cancelled after connection loss and reconnect on FXCM Demos only
14954	Fixed	Adapter, FXCM	Market Data mapping issue when Trading CFDs with FXCM in UK Time Zone
15160	Fixed	Adapter, Gain	Gain CFD resolution could be failing due to backend changes
15049	Fixed	Adapter, Interactive Brokers	NinjaTrader would not update position on IB Liquidation
15063	Fixed	Adapter, Interactive Brokers	Interactive Brokers Auto Logon has issues when the password contains certain non-alphanumeric characters
15101	Fixed	Adapter, Interactive Brokers	Interactive Brokers CZK denominations do not show in platform

15109	Fixed	Adapter, Interactive Brokers	Interactive Brokers: Auto Logon will not populate credentials in Gateway 981.3g
15048	Added	Adapter, Rithmic	Updated Rithmic API to version 11.3.0.0
14981	Changed	Adapter, TD Ameritrade	User who is able to place short entries through TDA (TOS) cannot in NinjaTrader
14923	Fixed	Adapter, TD Ameritrade	Every tick reported as daily candle with Hong Kong SAR regional settings
15178	Fixed	Alerts	Alert always triggers when opening Workspace, if alert is applied to background chart tab
14658	Fixed	ATI	Moving OIF files to incoming folder results in no action
15040	Fixed	ATM Strategies	Hovering over ATM Strategy Info would not reflect modification
15042	Fixed	ATM Strategies	Reversal Order is affected by 'Chase' option and may cause multiple ATM entries
14813	Fixed	ATM Strategies, Order Entry	Order entry windows would not show working ATM's if custom ATM was selected then canceled
15076	Fixed	Backup & Restore	Backup could fail if the backup folder is set to a mobile drive's root directory

14999	Fixed	Chart	Time and Date Entries in the Mini Data Box could be duplicated
15006	Fixed	Chart	Chart would not relocate properly using Windows + Shift + Cursor
14951	Fixed	Chart	White grid lines show incorrect color with Dark Skin applied.
14983	Fixed	Chart	Chart stuck 'loading' when chart tab is switched prior to historical reload completion
15148	Fixed	Chart	Unable to change instruments in multiple panel charts
15195	Fixed	Chart	NullReferenceException in rendering when running strategy
14942	Fixed	Chart, Bars	Weekly, monthly and yearly charts didn't respect trading hours definition on intraday realtime/playback data
14864	Fixed	Chart, Drawing	Extended Lines and Rays on non-equidistant charts will 'slide' over the chart when scrolling past today
15179	Fixed	Chart, Drawing Tool, NinjaScript	Polygon and Path Draw Objects could disappear on reloading NinjaScript
14988	Fixed	Chart, NinjaScript	A strategy ZOrder property is 0 by default, when the default should be 10001

15153	Fixed	Chart, NinjaScript	Andrew's Pitchfork disappears when anchors are out of view
14936	Fixed	Chart, NinjaScript	Setting plot values to double.MinValue on some visible bars causes d2d error
15184	Fixed	Chart, NinjaScript	COT indicator updated mappings
14840	Fixed	Chart, Strategy	Strategy with AddChartIndicator, puts indicator on wrong panel when changing instruments
15106	Fixed	Chart, SuperDOM	SuperDOM/Charts could become unresponsive when duplicating SuperDOM
15170	Fixed	Connections, FXCM	Configuring FXCM connection while platform language is Portuguese could result in error
15073	Added	Control Center	Direct video guides link in Help menu
14995	Fixed	Control Center	Window maximized on a non-primary screen will incorrectly expand onto an additional screen
15124	Changed	Control Center	Improved error messaging on attempting to Import Non-NT8 zip file
15127	Fixed	Control Center	Orders Tab Filter by Instrument would not uncheck
15168	Fixed	Control Center	Orders Tab Filter prevents any orders shown on restart

15177	Added	Control Center	Help menu link to live events
14937	Fixed	Control Center	Minimizing modal forms could cause undesirable behavior
14959	Fixed	Control Center	Tab key does not switch between 'Max Order Size' and 'Max Position Size'
15097	Fixed	Core	Print orientation default should be Landscape
14935	Fixed	Database, Instruments	Changing Instrument Properties of instrument applied to chart causes NT to become unresponsive
15173	Fixed	DrawingTool	Global Drawing Objects could be duplicated when using the 'Save As' to save a workspace
15103	Fixed	DrawingTool	Reload NinjaScript can temporarily hide drawing objects attached to "all charts"
15089	Added	Installer	Safe mode entry made available to Start menu
14934	Fixed	Instruments	Typing ^ or @, into chart, correctly selects list but excludes those instruments
15207	Fixed	Instruments	Instrument Selector Displays placeholder after deleting instrument from list
14903	Fixed	Localization	IBKR and BarChart adapters report incorrect historical data and executions if Windows number format is non-default

15167	Fixed	Market Analyzer	Unhandled exception when removing Blank rows in Market Analyzer
15183	Fixed	Market Analyzer	Market Analyzer would not add previously added master instrument
15079	Changed	NinjaScript	Remove Requirement for all VS's needing to be closed, to launch VS from NT
14972	Fixed	NinjaScript	Unmanaged Orders on primary Series Canceled, Secondary are not, on start up Multi Time Frame
14998	Fixed	NinjaScript	TimeEditorKey uses Text editing cursor when mouse over arrow buttons
15000	Fixed	NinjaScript	Path and Polygon drawing tools could have error after recompile/reload
15002	Fixed	NinjaScript	Volume Up Down indicator would not display when Language is set to Italian
15016	Fixed	NinjaScript	When StopCancelClose takes place, call the order "StopCancelClose"
15045	Fixed	NinjaScript	Strategy with 1 tick added series could cancel stoploss when Tick Replay is used in Strategy Analyzer
15058	Fixed	NinjaScript	No .cs file is generated in the StrategyAnalyzerLogs folder when backtesting a strategy in a subfolder

15060	Fixed	NinjaScript	Error exporting strategy that includes indicator in sub-folder/sub-namespace cannot export
15080	Fixed	NinjaScript	Strategy using Collection Editor cannot be reloaded after recompiling
15081	Changed	NinjaScript	Managed Approach Internal Handling Rules log reporting improved
15117	Fixed	NinjaScript	COT indicator not plotting new week's data after holiday/New Year
15157	Fixed	NinjaScript	TrendLines indicator throws error in OnCalculateMinMax when viewed in Strategy Analyzer chart
14882	Fixed	NinjaScript	AddOn that adds an image to ControlCenter crashes on second import
15038	Fixed	NinjaScript	@Pivots.cs causes invalid index error on blank chart
15046	Fixed	NinjaScript	Tick counter shows repeating decimals when shown as a percentage
15050	Fixed	NinjaScript	Backups with large paths can be created, but cannot be restored
15055	Fixed	NinjaScript	MA Envelopes indicator generates error when Period is set to 1

15064	Fixed	NinjaScript	Bar Timer does not work with Visibility check box as expected.
14977	Fixed	NinjaScript Editor	NinjaScript editor unsaved changes message box on exit could not be getting proper focus
15107	Changed	NinjaScript Editor	There is an "Exclude from Compilation" button in the NinjaScript Editor's subfolders that does no operation
15108	Fixed	NinjaScript Editor	NinjaScript Editor throws an error message regarding code that is commented out
14900	Fixed	NinjaScript Editor	Moving script with enum to new folder throws object reference not set to instance of object error
15119	Fixed	NinjaScript Editor, Visual Studio Integration	Editor could fail to find Visual Studio 2022 installation
14869	Fixed	NinjaScript, DrawingTool	Regression Channel with either Extend Left or Extend Right selected could disappear from chart if end points are outside of the viewable chart window
15033	Fixed	NinjaScript, Strategy	Strategy instace could be missing from strategies tab but applied on chart
15128	Fixed	NinjaScript, Strategy	Changing indicator panels while using AddChartIndicator could behave differently when

			enabled from a Chart vs. Control Center
14883	Fixed	NinjaScript, Strategy	Wait until flat not consistently submitting working historical orders that reach real-time
15174	Fixed	NinjaScript, Strategy	Issues enabling AdoptAccountPositionAware strategies
15034	Fixed	NinjaScript, Strategy Analyzer	Browsable / NinjaScriptProperty Attributes could throw error in SA Optimization
15186	Fixed	NinjaScript, Strategy Analyzer	Strategy Analyzer view model could have issue accessing indicator found in two obfuscated assemblies
15096	Fixed	Order Entry	Position Close algorithm could be triggered than once on same position
15067	Fixed	Order Flow +	OrderFlow Cumulative Delta indicator could have incorrect values when applied to 5 day chart
15056	Fixed	Order Flow +	OrderFlow Volume Profile set to Price causes charting freeze when zooming in
15069	Fixed	Playback	Market replay data replays slow on maximum speed since 10-18-2021
15068	Fixed	Playback, Strategy Analyzer	Strategy Analyzer 'Run' should be disabled in Playback mode

15043	Changed	Strategy	StopTargetHandling default should .PerEntryExecution
15099	Fixed	Strategy Analyzer	Strategy Analyzer 'Optimize On' property changes when changing strategies
14891	Fixed	Strategy Analyzer	Include Commission and Default Quantity do not persist in Strategy Analyzer after WS reopen
14962	Fixed	Strategy Analyzer	Strategy Analyzer does not use default template and preset using Open in New Strategy Analyzer from Log grid
15111	Fixed	Strategy Analyzer	A strategy that adds indicators that draw objects to the chart is throwing a NullReferenceException in the strategy analyzer.
15125	Added	Strategy Builder	IsFirstTickOfBar added to Strategy Builder
15165	Fixed	Strategy Builder	Strategy Builder could be unable to repopulate saved To / Subject fields from a share service
14816	Fixed	Strategy Builder	Strategy Builder: Changing an input then clicking Cancel, does not cancel change
14912	Fixed	Strategy Builder	Passing a secondary series to MACD in the strategy builder results in bars ago error
14990	Fixed	Strategy Builder	Strategy Builder could throw an error when opening a

			Condition Builder with platform in Italian
15113	Fixed	Strategy Builder	Strategy Builder Trailing Stop(s) can select irrelevant calculation modes of Price and Currency.
15075	Fixed	Strategy, Kinetick	Unable to connect to Kinetick with strategies on strategies tab
15150	Fixed	Trading Hours	Trading Hours holiday segment duplicate holidays reporting improved
14782	Fixed	UI	On certain setups indicator selector starts in wrong position
15149	Fixed	Workspaces	After switching to a workspace using the Alert Log, Shift-F3 causes both workspaces to appear as one

3.3.4 8.0.25.0

8.0.25.0 Release Date
October 19, 2021

Attention TD Ameritrade Users:
- **The legacy account authorization mode cannot be supported on Windows 11, as such TDA users are advised to continue using Windows 10 until further notice.**

Issue #	Status	Category	Comments
15020	Added	Adapter, Authorization	Implemented OAuth using the default browser for cTrader connection and Twitter share

			adapters to support Windows 11.
14858	Fixed	ATM Strategies	Scaling into a position twice with ATM Strategy using Auto Breakeven between entries could result in stops not respecting ATM Strategy Selection Mode
14894	Fixed	ATM Strategies	After an ATM strategy closes, the quantity field could not be reloaded from the template
14849	Fixed	Bars, Market Analyzer, Playback	AddDataSeries in Indicator MA column would cause playback to hang on connect
14888	Fixed	Basic Entry, FX Pro, Order Ticket, SuperDOM	Order entry windows would not always retain the selected account after restarting
14867	Fixed	Kinetick, Options	Loading an option chain on a stock symbol could cause connection losses
14941	Fixed	Chart, ChartTrader	Escape key could not cancel order modification if Instrument overlay displays while modifying an order
14853	Fixed	Chart, Chart Trader	Order could alternate between multiple instances of the same indicator when using 'Attach To Indicator' feature
14964	Fixed	Chart, DrawingTool	Text drawing tool does not initially show in all charts

14889	Fixed	Chart, DrawingTool	Ruler in Currency mode doesn't show negative
14968	Fixed	Chart, DrawingTool, Alerts	Trend Channel plots could not be selected in Alerts Condition Builder
14904	Fixed	Chart, NinjaScript	Closing Chart with Indicator could throw an unhandled exception
14943	Fixed	Chart, Workspaces	Connecting to data provider while Workspace is loading could re-size panels
14851	Fixed	Control Center	Control Center and Account Data window would not respect column sorting on workspace load
14945	Fixed	CQG, Orders	MIT order could be triggered, then showed it successfully canceled, but later could report filled
14905	Fixed	Depth Chart	Depth Chart price could not be rounded to instrument tick-size
14970	Fixed	Indicator	Enhanced COT defaults and plot names
14948	Done	Installer	NinjaTrader Desktop icon and splash-screen updated to remove version mention
14878	Changed	Instruments, AutoRollover	Non workspace used instruments from instrument lists should be updated silently in AutoRollover window

14938	Fixed	Interactive Brokers	Various 09-21 future expiries could be reported as 12-99
14896	Added	Interactive Brokers	Updated supported version to Trader Workstation 985.1g & Gateway 981.3c
14887	Fixed	Interactive Brokers	Initial Margin would not populate
14865	Fixed	Interactive Brokers	NinjaTrader would not reconnect to Gateway after Auto-Restart occurs.
14846	Fixed	Interactive Brokers	Unfilled Market Orders submitted before the Open could throw an error on reconnect
14793	Fixed	Interactive Brokers	Reconnect could fail when using multiple Gateway connections
14867	Fixed	Kinetick, Options	Loading an option chain on a stock symbol could cause connection losses
15012	Fixed	Licensing	Vendor Licensing could be unable to delete or modify licenses
14815	Fixed	Market Analyzer, NinjaScript, TickReplay	Market Analyzer Indicator column calling hosted indicator with added series could produce incorrect values
14824	Fixed	NinjaScript	BarsRequest Open could not match chart for current bar when Update is not subscribed

14758	Fixed	NinjaScript	Calling SetState from an Indicator or Strategy allows OBU to run in terminated
14984	Fixed	NinjaScript	Alert Log Entries from Candlestick Pattern Indicator could be hard to read
14979	Fixed	NinjaScript	Removed debug prints from Darvas indicator
14910	Fixed	NinjaScript, Drawing Tool	OnRender() could be processed for DrawingObjects not present on chart after NS was reloaded
14956	Fixed	NinjaScript, ShareAdapter	NinjaTrader.Core.Globals.SendMail() cc parameter is not sending the email.
14916	Fixed	NinjaScript, Strategy	Using SetTrailStop with OrderFillResolution could incorrectly throw an error about an invalid stop price
15003	Changed	NinjaScript, Strategy	Changed SampleMultiTimeFrame and SampleMultiInstrument to follow best practices with indicator instantiation
14974	Fixed	NinjaScript, Strategy	Submitting unmanaged orders in State.Realtime could cause freezes
14866	Fixed	NinjaScript, Tick Replay	Failed to call 'Add' method could result in NullReferenceException
14952	Fixed	Order Flow +	Order Flow Volume Price Profile could not print letters on

			open of 1440 minute bar
14881	Fixed	Playback	Market Replay Data could be missing Market Depth Data
14783	Fixed	Playback	Replay bar could be misaligned with physical bar index
14821	Fixed	Playback, Strategy	Playback connection could freeze while running a Multi Series strategy
15018	Removed	ShareAdapter	StockTwits share service support on hold
15005	Fixed	Simulator	Simulator to roll accounts at new CME session end time
14925	Fixed	Strategy	Adding a strategy to a custom Range Chart then editing Strategy from Control Center tab, could modify the strategy data series
14920	Fixed	Strategy Builder	Attempting to use a user variable as an offset to a price could not initially show the selected variable
14831	Fixed	Strategy Builder	Strategy Builder folders were expanded by default when should be collapsed.
14967	Fixed	Strategy Builder, DrawingTool	Strategy Builder: Y values for drawing tools do not maintain their value when edited
14922	Fixed	Strategy, Workspaces	Saving Workspace loses disabled strategy when connected to IQFeed

14839	Fixed	Strategy, Workspaces	Strategy could be enabled on chart but not the strategies tab when switching workspace

3.3.5 8.0.24.3

8.0.24.3 Release Date

June 15, 2021

Issue #	Status	Category	Comments
14908	Fixed	FXCM	Updated to latest ForexConnect API to support https connect points
14946	Fixed	Workspaces	Overwriting existing workspace could be incomplete

8.0.24.2 Release Date

April 1, 2021

Issue #	Status	Category	Comments
14836	Added	Rithmic	Added the ability to connect with futures.de
14870	Fixed	Strategy Builder	Could not use Set for Period input on indicators

8.0.24.1 Release Date

March 12, 2021

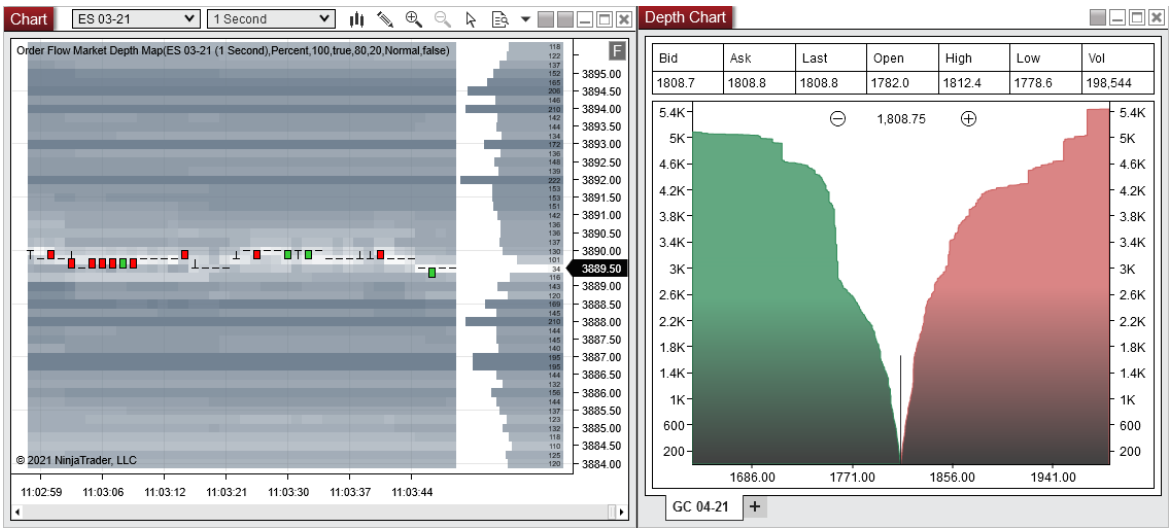
Issue #	Status	Category	Comments
14854	Fixed	Strategy	Intermittently CloseStrategy() did not close positions
14861	Fixed	Strategy Builder	Reopening a strategy and editing it's actions resulted in an error

8.0.24.0 Release Date

March 10, 2021

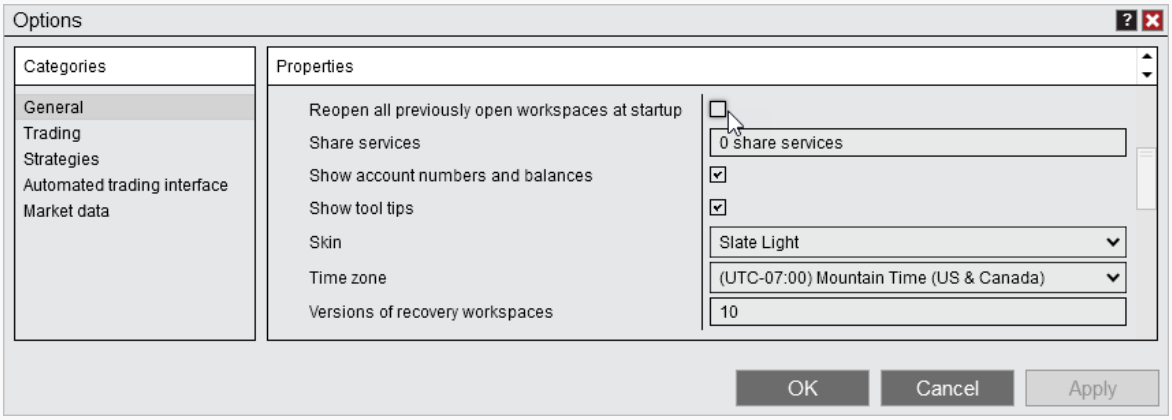
Features

Enhancements to take advantage of additional levels of depth
SuperDOM, Depth Chart, Order Flow +
Feature #14781



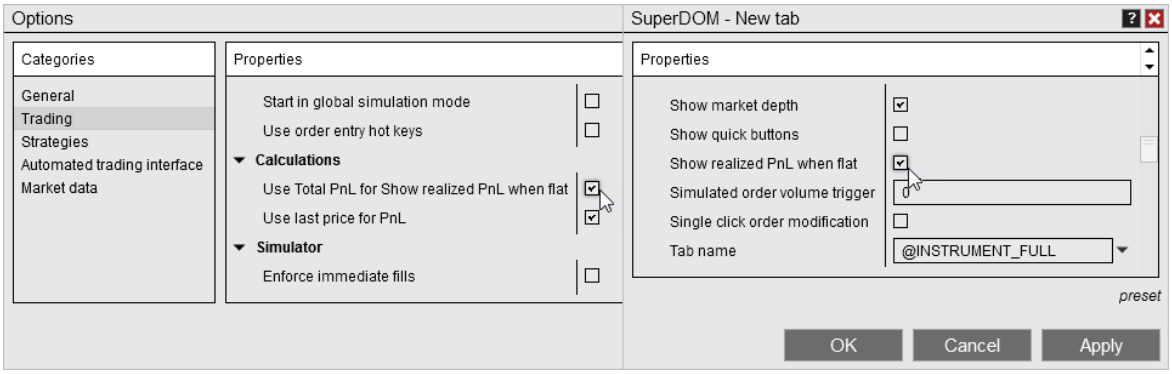
Order Flow Market Depth Map had a setting added called **Number of levels to track**, which can be configured to track more than the default 10 levels of depth.
Depth Chart now is enabled for futures and cryptocurrency instruments.
SuperDOM columns performance has been enhanced to handle more levels of depth. Multiple data providers offer more than 10 levels of depth. NinjaTrader Continuum & CQG have begun adding more levels of depth on popular instruments.

Multiple workspaces will open at startup only when opted in to do so
Workspaces
Feature #14761



One of the main causes of increased resources being used is multiple workspaces running at the same time and most people don't even realize that is occurring. To help boost performance, NinjaTrader now will only open multiple workspaces at start up for users who opt into doing so. No action is needed to only open 1 workspace at start up. To opt in for multiple workspaces, go to the **Control Center**> **General** and enable **Reopen all previously open workspaces at startup**.

Commissions template values can now be applied to Realized PnL
Commissions
Feature #14726



Viewing realized PnL when flat from your order entry windows can now be shown in Gross realized PnL or Total PnL. Within order entry windows, the property previously named **Show gross realized PnL when flat** has been renamed to **Show realized PnL when flat**. By default it will continue to show Gross realized PnL. To show Total PnL, within the **Control Center** go to **Tools**> **Trading** and enable **Use total PnL for Show realized PnL when flat**.

Issue #	Status	Category	Comments
14785	Fixed	Alerts	Alerts Log filter was not filtering
14765	Fixed	ATM Strategies	Close operation failed with MIT targets
14777	Fixed	ATM Strategies	ATM stop strategy template could disappear and ATM reverted to Custom after an hour of inactivity
14675	Fixed	Chart	Global Cross-hair did not update with small movements
14688	Fixed	Chart	Duplicating Chart Window then quickly closing via task bar resulted in an error
14697	Fixed	Chart	Renko bars failed to draw on Volumetric charts
14705	Changed	Chart	Charts could not be panned when Center price on scale was used, now it will disable so that it can be panned
14706	Fixed	Chart	Opening a new chart with a template applied prevented the chart from showing it was loading
14762	Fixed	Chart	Indicator Displacement caused chart to scale incorrectly
14662	Fixed	Chart Trader	Pressing escape did not cancel an order move operation
14692	Fixed	Chart	Using Display selected ATM strategy only did not show all orders when ATM selection was

		Trader	set to None
14693	Fixed	Chart Trader	Positions markers did not display in fractions for instruments quoted in fractions
14751	Fixed	Chart, DrawingTools	Placing a draw object then changing instrument then changing instrument back to original did not show that draw object.
14747	Fixed	Chart, DrawingTools	Drawing objects placed on a holiday in the future could display on the wrong bar
14749	Fixed	Chart, NinjaScript	RemoveDrawObjects did not remove global draw objects if chart using indicator was on background tab
14811	Fixed	Connections	Migrating NinjaTrader 7 connections could result in duplicate Kinetick EOD connections
14770	Fixed	Control Center	Connection status stayed green in Accounts tab when disconnected
14774	Fixed	Control Center	Restoring English Workspace in German resulted in an error
14739	Fixed	Database	Some instruments had incorrect information after resetting instruments

14769	Fixed	Database	Realized PnL could show previous value if new value was not received
14789	Fixed	FXCM	Realized PnL in accounts tab could report incorrectly
14808	Fixed	Indicator	Woodies Pivots line widths settings did not work
14795	Fixed	Instruments	Futures instruments could be created without any contract months defined
14736	Fixed	Interactive Brokers	Disconnects from overnight prevented real-time data from coming in
14763	Fixed	Interactive Brokers	Incorrect settings in Traders Workstation could cause space-bar to be entered repeatedly
14779	Fixed	Interactive Brokers	Resolved some instruments that showed with the wrong contract month
14750	Fixed	IQFeed, Data	Volume for indexes did not update in real-time
14752	Fixed	Kinetick	Instruments on NASDAQCM caused a disconnect/reconnect loop
14754	Fixed	Kinetick, Chart	Bid/ask/last tick data on the same panel could plot incorrectly

14696	Fixed	Kinetick, Chart	Reloading data could result in missing bid/ask tick data
14440	Fixed	Licensing	Timezone affected free trial start/end date for 3rd party add-ons
14689	Fixed	Localization	Resolved an error that occurred when adjusting some settings after setting the language to Spanish
14690	Fixed	Localization, Alerts	Alerts with @Nombre in the Text field would not save when language was set to Spanish
14687	Fixed	NinjaScript, DrawingTool	Draw.TextFixed opacity could no longer change if set to 90% or greater
14707	Fixed	NinjaScript, Rithmic	Account.PositionUpdate did not show flat positions for Rithmic
14606	Changed	Options	Simulation accounts now always calculate as FIFO and the FIFO setting was removed since it only affected simulation accounts
14621	Fixed	Order Flow+	Order Flow Volume Profile week/month started on wrong day for some timezones
14701	Fixed	Order	Order Flow Volume Profile using a week profile period did not display correctly with additional data series

		Flow +	
14766	Fixed	Order Flow +	Order Flow Market Depth property, Number of levels to track, did not persist with workspace change
14745	Fixed	Order Flow +, Indicators	Various indicators with weekly or monthly reset now match logic for weekly/monthly bars
14744	Fixed	Playback, Strategy	Rewinding Playback caused strategy indicators to be removed
14778	Fixed	Playback, Strategy	Adding a 1 tick series to a strategy would only pull 5 days of data for the historical tick series when applied in Playback
14814	Fixed	Regionalization	Italian language resources updated
14682	Fixed	Strategy, Chart	Repeatedly enabling then disabling strategy could cause chart to lockup
14188	Fixed	Strategy	Disconnected accounts were listed in the Account drop-down of the Strategy window
14708	Fixed	Strategy	Resolved a scenario where a strategy could show multiple times on the Strategies tab

14710	Fixed	Strategy	Resolved a case where an error in OnExecutionUpdate was not handled properly
14742	Fixed	Strategy	Connection loss could result in a disabled strategy showing as enabled
14771	Fixed	Strategy	Resolved a scenario where a rejected modified order did not stop the strategy since the order was rolled back
14799	Fixed	Strategy	ExitOnSessionClose did not execute on last session of chart if market was open
14801	Fixed	Strategy Analyzer	Attempting to view Walk Forward Optimization results in a new instance of the Strategy Analyzer caused an error on next run
14632	Fixed	Strategy Builder	Indicators without plots incorrectly showed as available
14746	Done	Strategy Builder	User inputs could prevent entering value in some situations
14773	Fixed	Strategy Builder	Numeric value was missing for order submission quantity
14775	Fixed	Strategy Builder	Offset field could become disabled
14796	Fixed	Strategy Builder	Resolved some scenarios of user inputs not being able to be entered/modified as expected

14792	Fixed	Strategy, Chart	Strategy using AddChartIndicator left indicators on chart when calling CloseStrategy
14802	Fixed	Strategy, Trade Performance	Strategy Performance window did not close when strategy was disabled in chart
14643	Fixed	TD Ameritrade	Resolved a scenario where an error resulted in real-time data not coming in
14664	Fixed	TD Ameritrade	Buying power was not displaying
14673	Fixed	Workspaces	Workspace changes were not saved when exiting if a workspace was restored and then it's name was too long
14696	Fixed	Chart, Workspaces	Workspace couldn't load some instruments on charts if database was deleted

3.3.6 8.0.23.2

8.0.23.2 Release Date
December 14, 2020

Issue #	Status	Category	Comments

14721	Fixed	Backup & Restore	Importing backups from 8.0.22.2 that included historical data resulted in an error
14729	Fixed	DrawingTool, Chart	Rolling over a futures that had drawing objects on the chart resulted in the objects not being visible
14562	Fixed	DrawingTool, Templates	Drawing objects with the setting Attach to - Local in the template were removed when changing instruments
14735	Fixed	DrawingTool, Templates	Attach to - All Charts setting was not saved when using a template
14733	Fixed	NinjaScript	With a managed approach, close position occurred after entry order when reversing
14723	Fixed	Playback, Strategy	Strategies in Playback could show order of executions incorrectly
14724	Fixed	SuperDOM	Horizontal grid lines were wider than expected

8.0.23.1 Release Date

November 23, 2020

Issue #	Status	Category	Comments
14717	Fixed	Rithmic	Resolved a scenario that prevented connecting and resulted in an 'account

			already exists' error
147 15	Fixed	Interactiv e Brokers	Positions did not display when submitted to a master FA account
147 14	Fixed	Strategy	Strategies did not disable upon manual position close
147 12	Fixed	Control Center	Auto Rollover still popped up when unchecking instrument and pressing done

8.0.23.0 Release Date

November 16, 2020

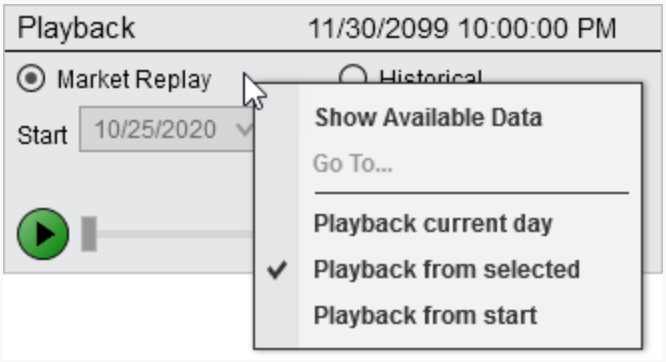
Features

.NET 4.8 Upgrade
General
Feature #14630

NinjaTrader 8 is built utilizing Microsoft's .NET framework. Microsoft recommends upgrading to .NET Framework 4.8 to receive the highest level of performance, reliability, security, and long-term support from Microsoft. This update is expected to be entirely transparent as most users already have .NET framework 4.8 automatically installed via windows update. When installing NinjaTrader, it will prompt you if you need to install .NET 4.8, and if required, you can download it here. Additionally, as a result of this update, Windows 8.1 becomes the minimum supported operating system.

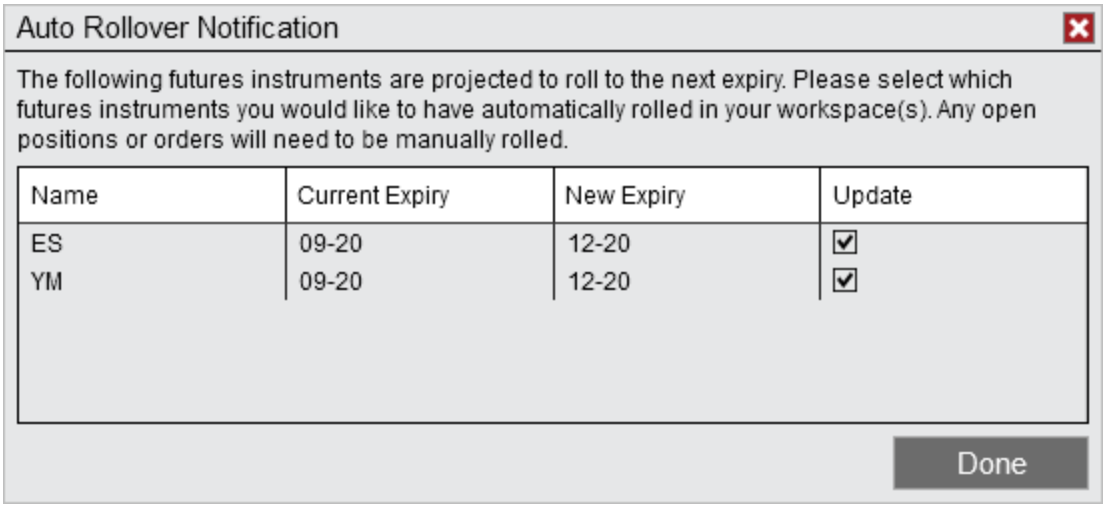
New default playback mode 'Playback from selected'
Playback
Feature #14287

This new mode will load historical data up to the selected time frame and start playback from there. This will increase the speed of changing times for users who don't have scripts or strategies that need to process each data event as it comes in.



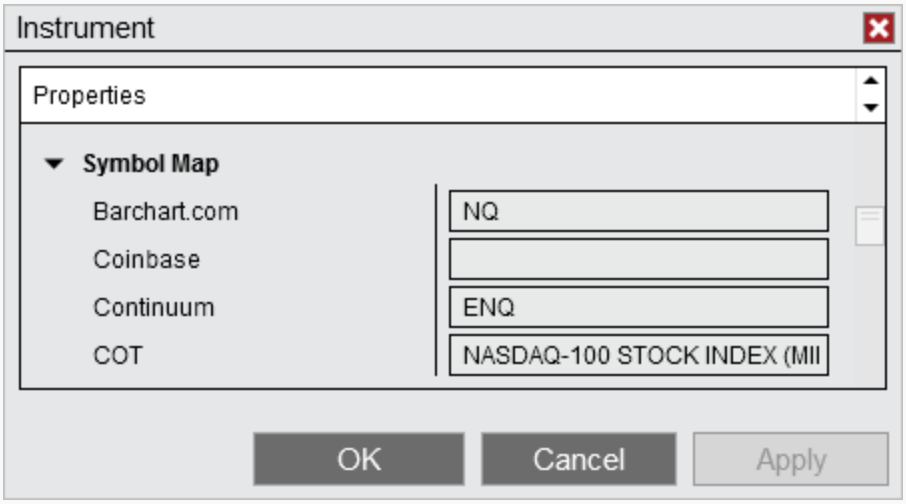
Auto rollover notification
Instruments, DataBase
Feature #14599

A pop up will now appear when starting NinjaTrader or opening a workspace while there is an instrument in the instrument list or workspace that has rolled over. This will help ensure that the latest contracts are loaded with ease.



COT instrument mapping added to Instrument editor
Indicator, Instruments
Feature #14552

Mapping for COT data has been added to the Instrument editor to allow adding COT mapping for newly created instruments or adding to existing ones. For some instruments that do not have COT data you could map to related instruments if desired. For example, you could map COT data from a mini contract to a micro.



Added extended backwards option for Andrew's Pitchfork
DrawingTool
Feature #14553

The extend lines back property adds additional functionality to Andrew's Pitchfork for your technical analysis needs.

Interactive Broker auto reconnects after daily restart
Interactive Brokers
Feature #14510

Both Traders Workstation and Gateway connections will now auto reconnect after their daily restart.

Issue #	Status	Category	Comments
14383	Fixed	Alerts	Time based conditions reverted to Data Series when condition was re-opened
14470	Fixed	Alerts	Changing an indicators label removed it from an alert condition
14473	Fixed	ATM Strategies	The 2nd modification to TIF GTD within ATM parameters was not reflected in the entry window's TIF
14476	Fixed	ATM Strategies	If an order partially filled at the target then filled at the stop, the target could not be canceled
14581	Fixed	ATM Strategies	Stop strategies could not be re-enabled after using the right click menu to make changes to breakeven properties
14453	Fixed	Back up & Restore	There was an error exporting scripts when a PC was set to use Belgian Dutch as the language
14653	Fixed	Bars	Line break bars could have an unexpected spike
14568	Fixed	Chart	Global crosshairs applied time scroll into margins
14578	Fixed	Chart	Indicator plot did not correctly scale when using overlay scale justification and auto scale

14647	Fixed	Chart	The text for z-levels did not match the text color property
14467	Fixed	Chart, NinjaScript	Duplicating a chart with an indicator which uses public variables doubled the value
13995	Fixed	Chart, Strategy	Strategy added indicators could only plot on primary panel
14471	Fixed	Control Center	Using a non-Gregorian calendar resulted in a crash
14544	Fixed	Control Center	Opening with no internet connection resulted in a crash
14638	Fixed	Control Center	Selecting Repair DB then Reset DB caused an error
14641	Fixed	Control Center	Exporting a strategy caused the cursor to change unexpectedly
14669	Fixed	Control Center	Resolved a scenario where a file could still be in use after exiting and trying to start NinjaTrader again
14637	Fixed	CQG	GTD orders restored incorrect date with WebApi
14644	Fixed	CQG	The message indicating an account is not enabled for data was not displaying

14651	Fixed	CQG	OnPositionUpdate could report incorrect position when account was flat
14674	Fixed	CQG, Strategy	Chart could lock up if a multi data series strategy that didn't use the second series moved a stop
14449	Fixed	CQG, Orders	Leaving WebAPI connected over night prevented an OCO order from being modified the next day
14515	Fixed	CQG, Orders	WebAPI ATMs using MIT for profit would not cancel the stop when target was filled
14504	Fixed	DrawingTool	When using 'stay in draw mode' and double clicking on placed text to modify it there could be an error
14521	Fixed	DrawingTool	Trend Channel snapped inconsistently to OHLC price when parallel Y value was modified
14535	Fixed	DrawingTool	Drawing objects with visible set to disabled reappeared when an interval was changed
14562	Fixed	DrawingTool	Drawing objects could be removed when switching instruments when using some settings and default template settings together
14642	Fixed	DrawingTool	Risk Reward drawing tool did not set ratio when drawn from code
14631	Fixed	DrawingTool, Templates	Draw objects were not updated when using 2nd template on same object

14518	Fixed	Indicator	Candlestick pattern Inverted Hammer / Shooting star were incorrect
14601	Fixed	Indicator, DrawingTool	Risk Reward drawing object did not properly update price levels when called from an indicator using Calculate.PriceChange
14496	Fixed	Instruments	Changing instrument type on instrument creation caused an error
14497	Fixed	Instruments	Pasting symbol mapping for an instrument did not work on first attempt
14549	Fixed	Instruments	Select an instrument that was already selected did not update selected instrument text properly
14660	Fixed	Instruments	When a stock chart is open, removing that instrument via the Instruments window caused a crash
14628	Fixed	Instruments, Chart	Added instruments did not appear in instrument list of existing charts
14490	Fixed	Interactive Brokers	Connection interruption while requesting historical data resulted in no data
14522	Fixed	Interactive Brokers, Strategy	Strategy Position did not update in Strategies tab of the Control Center

14533	Fixed	Interactive Brokers, Time and Sales	No shapshot data came through for illiquid contracts
14503	Fixed	Kinetick, IQFeed, Options	Futures options did not have the point value applied
14611	Fixed	Kinetick, Option Chain	AAPL options failed to load
14495	Fixed	Kinetick, Options	Ticksize for treasury options were incorrect
14432	Fixed	Kinetick, SuperDOM	Using multiple connections and experiencing a disconnect could result in a bad depth data
14486	Fixed	Kinetick, SuperDOM	Depth volume would not update from an order until the scroll wheel was used
14530	Fixed	Market Analyzer	Column sorting by description did not work
14505	Fixed	News,	Instrument link did not work with the News window

		Window Linking	
14306	Fixed	Ninja Script	ExitOnSessionClose did not close historical positions when using Wait Until Flat
14462	Fixed	Ninja Script	Was unable to retrieve type info for 'NinjaTrader.NinjaScript.AddOnBase' from assembly 'NinjaTrader.Custom'
14539	Fixed	Ninja Script	Input indicator with secondary series did not trigger on bar update properly when days to load changed
14616	Fixed	Ninja Script	Profit target was not submitted every other time when strategy was disabled/re-enabled if a simulated stop was used
14657	Fixed	Ninja Script Editor	Resolved a scenario where compile could become disabled
14448	Fixed	Ninja Script, Chart Trader	Chart Trader would immediately re-render chart for order modification events even when Chart Trader was disabled
14550	Fixed	Order Flow +	Order Flow VWAP had an error when bars had no volume
14554	Fixed	Order Flow +	Order Flow Volume Profile setting Show POC didn't work with Price profile type
14593	Fixed	Order Flow +	Order Flow Volume Profile would only render a single line if multiple lines were at the same level

14613	Fixed	Order Flow +	Order Flow VWAP could have wrong weekly start time
14634	Fixed	Order Flow +	Order Flow Volume Profile failed to plot when splits and dividends were updated
14633	Fixed	Playback, Strategy, Chart	Start/end dates listed in Strategy Performance reflected start/end date of original chart, not playback start/end dates
14603	Fixed	Rithmic	Connection could get stuck if there was a large amount of events being downloaded
14671	Fixed	Rithmic	Instruments MGC, HO, and RB could have position return with a different exchange
14678	Changed	Rithmic	Add UProfitTrader connection points
14580	Changed	Share Adapter	StockTwits character limit was increased to 1,000
14571	Fixed	Share Adapter	StockTwits share service caused an error when creating for the second time
14480	Fixed	Simulator, Options	Realized PnL and cash value did not match when trading options in simulation
14577	Fixed	Strategy	SetTrailStop incorrectly amended stop price, if an added data series was for a contract older than 1 expiration ago
14620	Fixed	Strategy	A multi-series script that submitted one order, canceled it, then submitted a second

			order ignored the second order when run on historical data
14654	Fixed	Strategy	Mismatched calculation mode between SetStopLoss/SetProfitTarget caused incorrect exit order quantity
14659	Fixed	Strategy	Resolved a scenario where instrument property was not disabled correctly when strategy was enabled
14684	Fixed	Strategy	Quickly disabling/enabling a strategy could result in it's state being our of sync between the Control Center and chart
14385	Fixed	Strategy Analyzer	Moving a column then adding another column caused the first column to revert
14464	Fixed	Strategy Analyzer	Analysis graph could draw incorrectly when plotting scaled in/out position by entry time
14469	Fixed	Strategy Analyzer	Duplicating window with multiple tabs with charts resulted in errors
14474	Fixed	Strategy Analyzer	If Region Format was set to English (Germany) the leading zero and comma was stripped from input display
14485	Fixed	Strategy Analyzer	Duplicating window with a new tab that was not ran resulted in an error
14528	Fixed	Strategy	ExitOnSessionClose orders did not have slippage applied

		Analyzer	
14551	Fixed	Strategy Analyzer	Optimization failed on strategies that enter with no exits when using certain Optimize settings
14575	Fixed	Strategy Analyzer	Bars.Count for a secondary series of a strategy changed between backtest runs
14542	Fixed	Strategy Builder	An indicator offset was not applied with crossAbove/crossBelow operator
14540	Fixed	Strategy Builder	The bars ago for current bar check defaulted to 1 rather than the passed bars ago value when a variable was used for an offset
14541	Fixed	Strategy Builder	Using a numeric value could result in an error
14614	Fixed	Strategy Builder	Assigning value of an added data series to a custom series would not compile
14393	Fixed	Strategy, Chart	Settings IsOverlay to false did not plot indicators to sub panel
14508	Fixed	Strategy, Control Center	Strategies added to Strategies tab of the Control Center didn't appear when opening workspace

14666	Fixed	Strategy, Orders	A strategy which submits a stop loss and exits on session close would submit a stop loss if strategy was enabled while market was closed
14481	Fixed	SuperDOM	ZT was missing depth volume for every other tick
14445	Fixed	SuperDOM, Indicator	Indicators would report historical data as real-time data
14582	Fixed	TD Ameritrade	Disabling then re-enabling an internet connection prevented a proper reconnect
14556	Fixed	TD Ameritrade	Attempting to connect to accounts from countries unsupported by the API caused a crash
14598	Fixed	TD Ameritrade	Resolved a scenario where live order events came in on a simulation license causing errors
14560	Fixed	TD Ameritrade, Orders	An order was rejected if it was being authenticated while the access token was being refreshed
14538	Fixed	TD Ameritrade, Strategy	Profit Target was not submitted with SetProfitTarget if no SetStopLoss was used
13943	Fixed	Tick Replay, Chart	Indicator using Correlation as input series could return incorrect values in TickReplay

14395	Fixed	Tick Replay, Ninja Script	There could be missing ticks when transitioning from historical tick replay data to real-time data
14409	Fixed	Tick Replay, Ninja Script	TickReplay indicators diverged from Calculate.OnBarClose instance on transition to real-time
14501	Fixed	Workspaces	Closing a workspace that had an options instrument that wasn't in the db caused an error
14511	Fixed	Workspaces	Resolved a scenario where using 'Save workspace as' did not properly save it
14573	Fixed	Workspaces	Databox disappeared when saving over the original workspace it was opened in

3.3.7 8.0.22.2

8.0.22.2 Release Date
June 15, 2020

Issue #	Status	Category	Comments
14506	Fixed	DrawingTool	Global drawing tools would not save when closing and saving from another workspace

8.0.22.1 Release Date
June 3, 2020

Issue #	Status	Category	Comments

14492	Fixed	DrawingTool	Global drawing tools duplicated when saving the workspace

8.0.22.0 Release Date
June 2, 2020

Features

Added new TD Ameritrade API
TD Ameritrade
Feature #14132

TD Ameritrade is requiring that we migrate to a new API, which has been implemented and is released in beta. The API has a lot of similarities between the old, with overall functionality expected to be the same. The new API does add option support for equities and utilizes a new connection type which is only available with supported operating systems, Windows 8 and newer. Please report any issues seen via platformsupport@ninjatrader.com.
Note: At this time you may see duplicate executions if you disconnect and reconnect in the same session.

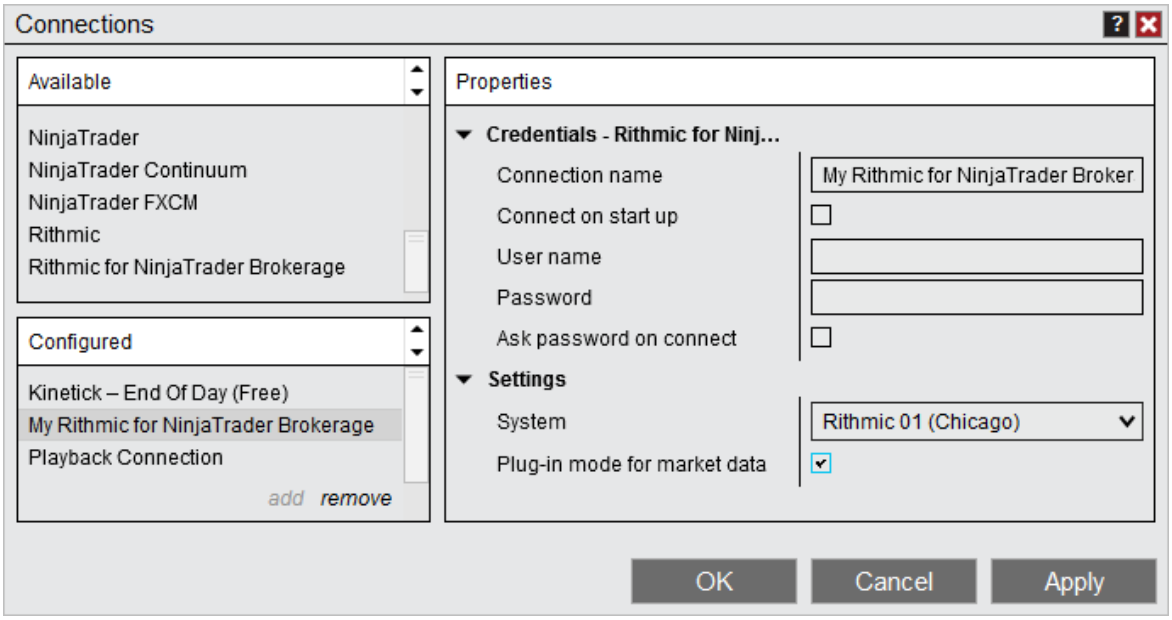
Added COT Indicator

Indicator
Feature #13802

The COT indicator plots data from the Commitment of Traders reports. These are weekly reports (best viewed on a weekly chart) that come out each Friday showing Tuesday's open interest from market participants. These reports are available for many futures instruments. For more information, please see the COT section of the help guide.
Note: You must enable "Download COT data at startup" within Tools> Options> Market data for this indicator to plot.

The Rithmic adapter now support plug-in connection for shared market data
Rithmic
Feature #14439

This mode should only be enabled if instructed by your data provider, it allows re-use of a single market data login between multiple platforms on the same computer and requires some setup prior to use.



Add support for additional Coinbase cryptocurrencies
Coinbase, Instruments
Feature #14437

Cryptocurrency pairs can only be properly added if a 3 digit cryptocurrecny code is supported for it. The following cryptocurrecny codes are now available, adding additional support for Coinbase instrument pairs.

ALG (ALGO), ATM (ATOM) DSH (DASH), EOS, KNC, LNK (LINK), OXT, REP, XLM, XRP, XTZ

Translations improvements
Localization
Feature #14460

NinjaTrader can be localized to many languages including Spanish, German, French, Russian, Portuguese, Russian, Chinese, and Italian. Language selection can be changed via Tools > Options > General > “Language” option in NinjaTrader. These translations have been improved as part of continual refinement.

Issue #	Status	Category	Comments

14402	Fixed	Account Data	Net Liquidation value on stocks was incorrect
14441	Fixed	Alerts, Workspaces	When using 'Save As' on a workspace with active alerts, the alerts would not trigger until the workspace was closed and reopened
14456	Fixed	ATM Strategies	When creating a custom ATM, but then selecting cancel, the custom ATM could become active
14483	Fixed	BarChart, Bars	Non-last qualifying trades showed in historical data after a recent change from BarChart
14411	Fixed	Chart	SPI 06-20 had an incorrect session break on a day of early close
14405	Fixed	Chart, bars	Data was removed from chart upon reconnecting to data feed for ASX index futures instruments
14428	Fixed	Chart, DrawingTool, NinjaScript	Global draw objects created from NinjaScript could duplicate when instrument was switched by linked Market Analyzer
14355	Fixed	Chart, Template, Indicator	When opening a new chart with a template applied that had an indicator, an error occurred if the instrument was switched while still loading
14357	Fixed	Connections, Strategy	An enabled strategy became disabled when using multiple connections and the second connection disconnected

14351	Changed	Control Center	Removed tool tip timeout from Strategies tab
14259	Fixed	Control Center	Updated an error dialog that instructed to reinstall when a reinstall is not needed
14382	Fixed	Control Center	Changing risk template on a simulation account did not show a message that a restart would be needed
14461	Fixed	Control Center	Changing account denomination resulted in multiple pop ups
14400	Fixed	Control Center, Database	Find feature did not work in the "Rollover futures instruments" grid
14407	Fixed	CQG, Continuum	Some WebAPI errors caused a disconnect when not necessary
14088	Fixed	CQG, Continuum	Resolved incorrectly mapper settlement and last close prices with Web API
14396	Fixed	DrawingTool	Switching between series with different "Show global draw objects" settings then setting a drawing to global could show an invalid error
14365	Fixed	DrawingTool, Chart	Drawing tools could be removed when rolling over, switching instruments, then going back to the original unrolled instrument
14362	Fixed	DrawingTool	Drawing tools could move to other instruments when changing instruments

		, Chart	then rolling over
14457	Fixed	DrawingTool, Chart	Copy and pasting a drawing tool with a template set to left scale justification resulted in an error
14379	Fixed	eSignal	Pakistan stocks displayed incorrect date stamp on Monday daily bars
14414	Fixed	FXCM, Account Data	Realized PnL did not show upon connecting or reconnecting
14398	Fixed	Historical Data Window	Downloading data while a chart was loading could result in request getting stuck
14345	Fixed	Hot Key	CTRL+SHIFT+TAB hot key did not work if 'Show tabs' was disabled
14359	Fixed	Indicator, Chart Trader	Applying FX Tile caused Chart Trader to switch to Sim101
14346	Fixed	Indicator, NinjaScript Editor	Indicator variable settings could reset to default when adding additional variables to indicator
14342	Fixed	Instruments	Deleting instruments from recently used instrument list resulted in instrument list closing
14394	Fixed	Instruments	Forex and cryptocurrency instruments could have master symbols that were invalid

13813	Changed	Interactive Brokers	The required version of Traders Workstation has been updated to 978.2c
14314	Fixed	Interactive Brokers, Connections	Auto connect did not work with a simulation license if a market data window was open
14458	Fixed	Interactive Brokers, Option Chain	Strike prices would not populate for custom NSE instruments
14312	Fixed	Market Analyzer	Auto sort could resort instruments that have the same value when refreshing
14416	Fixed	Market Analyzer	The "Profit loss" and "Traded Contracts" columns did not reset when Sim101 was reset
14318	Fixed	Market Analyzer, Indicator	VolUpDown indicator column did not populate on Mondays
14371	Fixed	NinjaScript	Exporting a strategy with dependent indicators would error if PC's OS language was set to Hungarian
14291	Fixed	NinjaScript Editor	Using 'Save as' on a strategy was not creating a new class name
14390	Fixed	NinjaScript	Renaming a sub folder would lose scripts excluded from compilation

		Editor	
14430	Fixed	NinjaScript, DrawingTool	Draw.Arc with template overload did not use template settings
14401	Fixed	NinjaScript, Property Grid	Using a GroupName with a colon resulted in an error
14391	Fixed	Optimization Fitness	MinAvgMae optimization fitness metric was using Max MAE value
14250	Fixed	Order Flow +	Order Flow Volume Profile would not plot with some interval and session template settings when started
14463	Fixed	Output Window	Minimizing the output window then attempting to open a new one did not bring the existing output window forward
14339	Fixed	Strategy	Multi-instrument strategies did not take some trades when a more granular series was added of each instrument
14348	Fixed	Strategy	Closing chart with a strategy calculating could keep strategy in Control Center
14422	Fixed	Strategy	A sell stop order could get an error that it was above the close price when it was not
14431	Fixed	Strategy	An unmanaged strategy that had it's position reversed when transitioning from historical to real-time had mismatched trades
14321	Fixed	Strategy	Backtests had no results when first running with no data then running again while connected for data

		Analyzer	
14323	Fixed	Strategy Analyzer	A resource heavy strategy could get an error when opening an optimization's result in a new window
14337	Fixed	Strategy Analyzer	Closing a Strategy Analyzer window during start-up could result in an error
14377	Fixed	Strategy Analyzer	Trade and Execution prices could show 2 minus signs if price was negative
14380	Fixed	Strategy Analyzer	Strategy performance did not sort by entry or exit price consistently
14381	Fixed	Strategy Analyzer	Launching a chart from Strategy Performance could result in an error
14388	Fixed	Strategy Analyzer	Backtesting a strategy that uses a DayOfWeek parameter could result in a crash
14408	Fixed	Strategy Analyzer	Sorting by quantity did not work as expected
14412	Fixed	Strategy Analyzer	AI Generate could have an error when clicking on indicators

14455	Fixed	Strategy Analyzer	Strategy Analyzer chart froze when minimizing window after a backtest was run
14465	Fixed	Strategy Analyzer	Resolved a scenario where a chart tab would have contents transparent when switching between other windows
14364	Fixed	Strategy Builder	Compiling a strategy that called an index instrument not mapped yet caused an error
14404	Fixed	Strategy Builder	A couple options had some invalid drop down settings
14341	Fixed	Strategy, Chart	Opening strategies settings quickly after applying could result in a lock up and error when trying to close
14397	Fixed	Strategy, Control Center	Removing a strategy from the Control Center then restarting without saving the workspace resulted in the strategy returning
14352	Fixed	SuperDOM	Order entry window did not regain focus after dismissing global sim mode message box
14447	Fixed	SuperDOM	Adjusting active ATM strategy orders while a new ATM is selected generated inaccurate order quantities
14442	Fixed	TD Ameritrade, Strategy	Syncing a short position while flat could send a sell order rather than a sell short order

14399	Fixed	Tick Replay, Bars	Chart could be missing most recent bar when using Tick Replay and changing session templates
14425	Fixed	Trade Performance	Trade Performance window date was based on PC time zone, not NinjaTrader time zone

3.3.8 8.0.21.1

8.0.21.1 Release Date

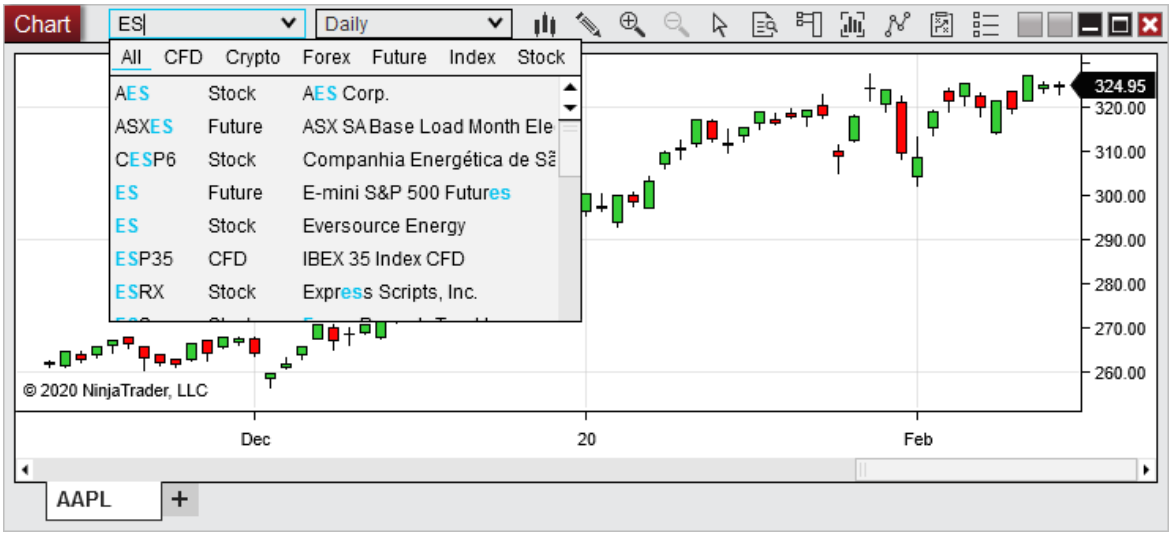
March 23, 2020

Issue#	Status	Category	Comments
14376	Fixed	Chart	Some combinations of tick chart values and data series resulted in a crash
14088	Fixed	Market Analyzer	Resolved some scenarios where Net Change columns showed incorrect values
14356	Fixed	Rithmic	Could not connect to Rithmic US West System

8.0.21.0 Release Date

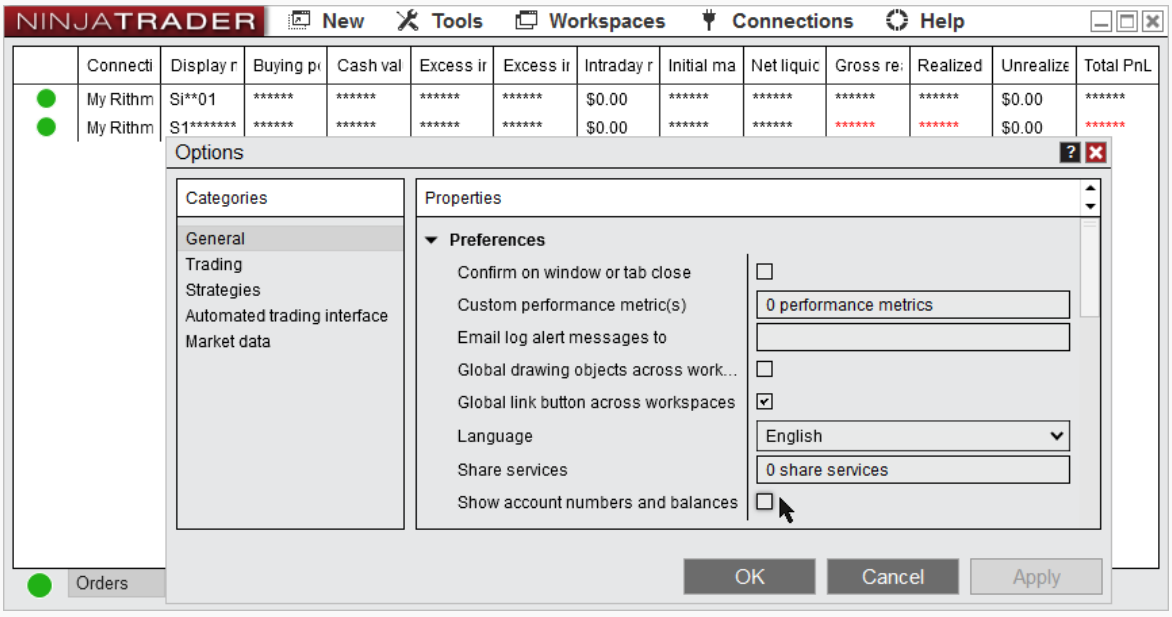
February 25, 2020

Features
Added a quick search enhancement to the instrument selector Instruments Feature #13754 The new quick search shows instant results to quickly look for and change instruments without having to open a search.



Added 'Show account numbers and balances' setting which can be disabled to hide these values
Account Data, Options
Feature #13492

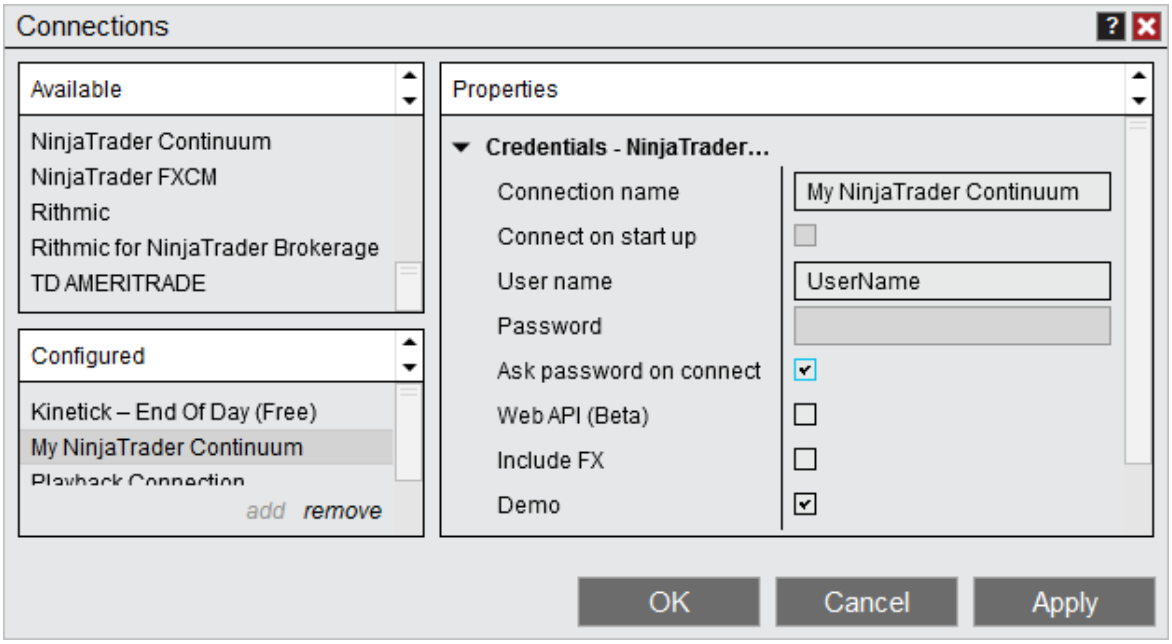
Private account values can now be hidden from view by unchecking the 'Show account number and balances' property. This gives the ability to have additional privacy when sharing your screen or using NinjaTrader around others.



Added setting to always 'Ask password on connect'
Connections

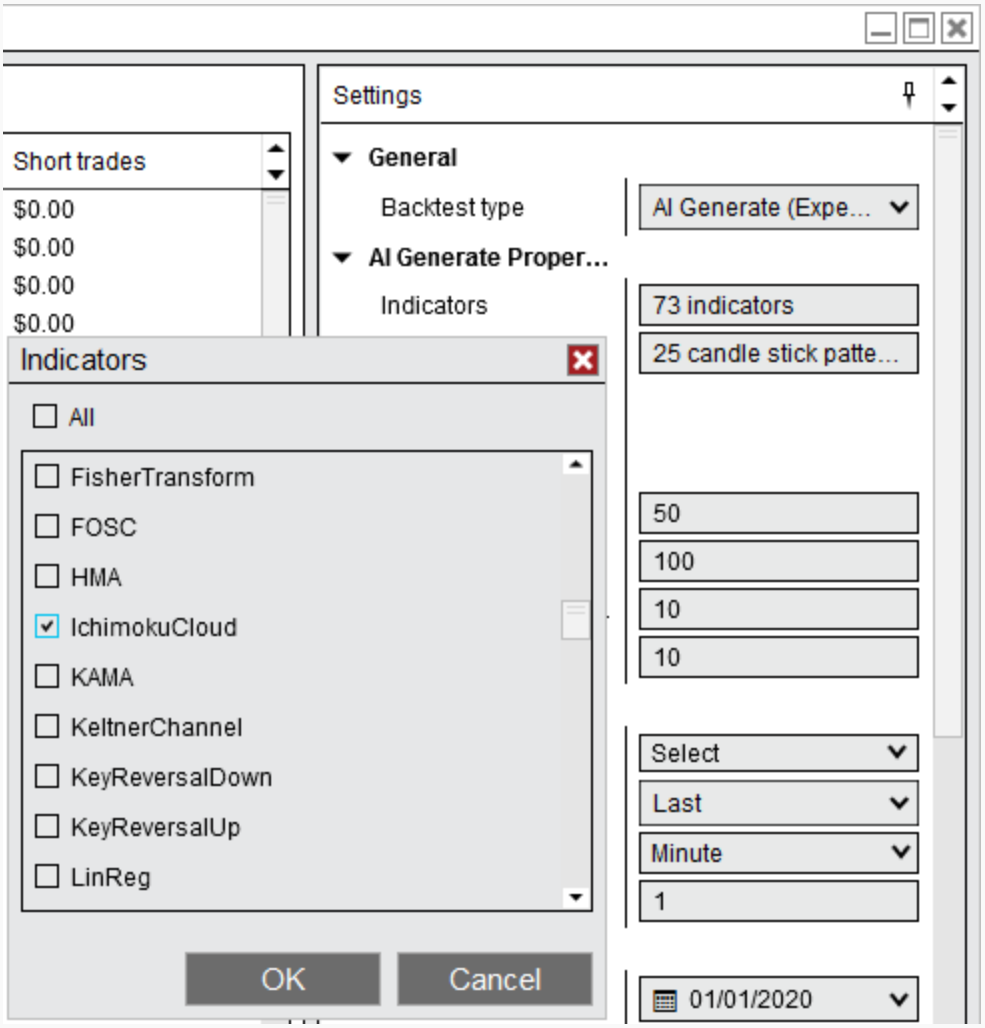
Feature #13057

Enabling 'Ask password on connect' will make it so your account password is not saved and the password will need to be manually entered each time you connect.



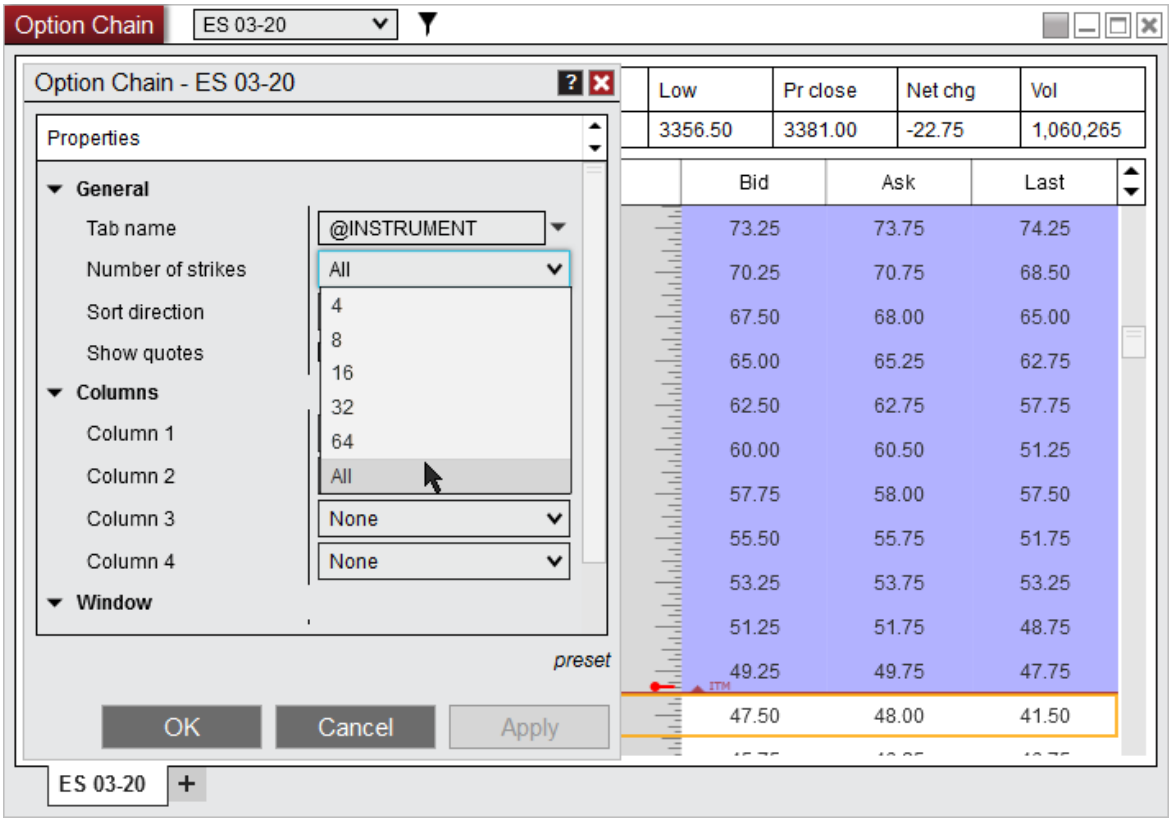
Enhanced AI Generate for use with third party indicators
Strategy Analyzer
Feature #13928

The AI Generate feature has now been expanded to allow the use of 3rd party single series indicators



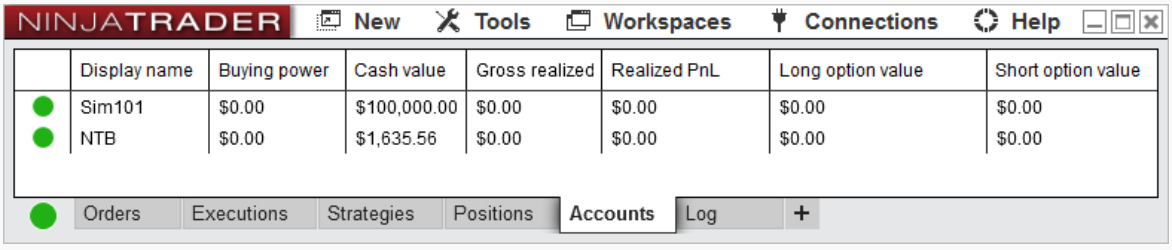
Expanded 'Number of strikes' settings to include '64' and 'All'
Option Chain
Feature #14235

Additional settings have been added to allow additional levels of strikes to view.



Added Short Option Value and Long Option Value columns to Accounts tab
Account Data
Feature #14231

Long/Short option values can now be displayed within the Accounts tab by right clicking on the Accounts grid, selecting Properties, and checking Long option value and/or Short option value.



NinjaTrader Brokerage customers now can view account Initial, Intraday and Net Liquidation account values
Continuum, Account Data
Feature #13657

Multiple account data values (Unrealized PnL, Initial Margin, Excess Initial Margin, Intraday Margin, Excess Intraday Margin, Net Liquidation) will now be calculated and displayed for NinjaTrader brokerage customers using the NinjaTrader Continuum connection.

Note: If upgrading NinjaTrader the database must be reset to access these account values.

Added support for PnL currency conversion of the Hong Kong Dollar
Account Data
Feature #13799

When trading instruments that have the Hong Kong Dollar currency (such as the HSI & MHI), the PnL will now convert to the accounts denomination

Major improvements on Spanish translations
Localization
Feature #14189

Added improvements to Spanish translations to provide a more complete experience for Spanish speaking clients.

Issue#	Status	Category	Comments
14035	Fixed	Account Data	Resolved a scenario where partially filled orders were not properly reflected
14266	Fixed	Alerts	Active alerts pop up warning could show after the workspace was already closed
14303	Fixed	Alerts, Orders	When enabling an alert which submitted a market order it could send an order both when selecting Apply and Ok
14304	Fixed	Alerts, Orders	Order Submitted as part of an alert could get stuck in a submitted state
14240	Fixed	ATM Strategies	ATM Strategy which used a Simulated Stop failed to close

14243	Fixed	ATM Strategies, Orders	Native OCO orders could get multiple cancel requests, resulting in an error
14262	Fixed	Bars	Reloading bars could intermittently result in a crash
14207	Fixed	Chart	Go To function on live charts did not function correctly with secondary data series applied
14260	Fixed	Chart	Indicator plot justification set to Overlay could be moved out of visual range
14325	Fixed	Chart	Scrolling to the first bar on the chart then compressing the time axis caused the chart to lose the ability to click and drag
14209	Fixed	Chart, Indicator	Copied indicators shared the same Z-order level
14222	Fixed	Code Wizard, NinjaScript	New scripts could be named with existing NinjaScript names, resulting in an error
14088	Fixed	Continuum, CQG, Market Analyzer	Net change columns displayed incorrectly with WebAPI
14297	Fixed	Continuum, CQG, Orders	WebAPI OCO orders when modified could incorrectly show an unknown order message

14261	Fixed	CQG, Strategy, Orders	High frequency order changes could result in an error
14288	Fixed	cTrader	cTrader instruments in database could have wrong trading hours
14285	Fixed	Data	Preferred connection setting for options was not working
14216	Fixed	DrawingTool	Trend Channel snapping after modification resulted in Y values rounded to an unexpected price
14310	Fixed	DrawingTool	Applying a template to a drawing object placed by a script set DrawnBy property to null
14302	Fixed	DrawingTool, Alerts	Arc drawing tool was incorrectly set to not support alerts
14244	Fixed	DrawingTool, Chart	Drawing objects did not appear in drawing object window if panel was scaled opposite
14245	Fixed	DrawingTool, Chart	Resolved an error that could occur when left, right, then left clicking once again on a drawing object
14220	Fixed	DrawingTool, Workspaces	Existing Region Highlight Y Drawing Objects disappeared when set as global
14290	Fixed	FXCM, Orders	A rejected order change that was then changed again before the error came in would get stuck in pending
14269	Fixed	Indicator,	Multi series indicator which draws global drawing objects could have duplicate

		DrawingTool	objects when applied to another chart with a different instrument
14099	Changed	Installer	Updated the error received when a database is for a newer version than what is installed
14229	Fixed	Interactive Brokers	Orders submitted from Traders Workstation would not cancel in NinjaTrader
14233	Fixed	Interactive Brokers	Could not connect if there was a working order submitted from Traders Workstation
14249	Fixed	Interactive Brokers	A position on N225 displayed with the wrong expiry
14293	Fixed	Interactive Brokers	A forced disconnect could still show as connected
14201	Fixed	Interactive Brokers, Data	Stocks traded on the Bolsa de Madrid exchange would not get data
14212	Fixed	Interactive Brokers, Orders	Flatten everything when used with option positions was unable to close the option position
14253	Fixed	Interactive Brokers, Orders	Option orders could get a rejection

142 04	Fixed	IQFeed	Resolved a scenario where a lost connection resulted in a crash
142 05	Fixed	IQFeed	Last Price did not update after disconnecting then reconnecting
142 41	Fixed	Market Analyz er	An error occurred when selecting an instrument after adding it to a blank row
142 02	Fixed	NinjaS cript	RemoveDrawObject did not work historically
142 18	Fixed	NinjaS cript	Split Entries did not have fully protected position when strategy was re-enabled with ImmediatelySubmit
142 56	Fixed	NinjaS cript	Resolved a scenario where removing a drawing tool placed by an indicator removed the indicator when it shouldn't
142 65	Fixed	NinjaS cript	Removing plots from an indicator applied to an unopened workspace locked up the chart when opening the workspace
142 84	Fixed	NinjaS cript	Using the TrendLine indicator in a strategy could result in an error
142 38	Fixed	Option Chain	Option Chain did not rollover
142 19	Fixed	Option Chain, Kinetic k	Option Chain could get an error when disconnecting from Kinetick End Of Day (Free)
142 26	Fixed	Order Flow +	Volumetric bars delta percent statistic with positive values could have no color gradient
142 50	Fixed	Order Flow +	Order Flow Volume Profile would not plot the profile until time since the session

			started was greater than data series interval
14267	Fixed	Order Flow +	Resolved a scenario where Order Flow Volume Profile got an OnBarUpdate error
14317	Fixed	Playback, Alerts	Alert rearm seconds was not respected when connected to Playback
13788	Fixed	Playback, NinjaScript	Resolved a scenario where resource heavy scripts could cause Playback to stop updating
14177	Fixed	Playback, NinjaScript	Historical data for added series was not loading when connected to Playback
14274	Fixed	Playback, Strategy	Enabling a strategy which uses ExitOnSessionClose and ImmediatleySubmit did no close historical position when expected
14300	Fixed	Rithmic	Could not connect to Rithmic 01 (Singapore) system
14254	Fixed	Strategy	When loading a template, then loading another template, the original template value stayed applied
13582	Fixed	Strategy Analyzer	Back tests using High Order Fill Resolution of 1 tick could see positive slippage
14307	Changed	Strategy Analyzer	Removed tool tip timeout

14197	Fixed	Strategy Analyzer	An error occurred when restoring Strategy Analyzer with AI Generate selected
14221	Fixed	Strategy Analyzer	AddChartIndicator() could fail for AI generated script containing CandleStickPattern
14248	Fixed	Strategy Analyzer	When optimizing a strategy that uses enums, if you compile before a test with the strategy selected in the optimizer, it no longer iterates the enums
14268	Fixed	Strategy Analyzer	Incorrect error showed when there were too many parameters to optimize
14270	Fixed	Strategy Analyzer	AI generated strategy generated MovingAverageRibbon code incorrectly, causing compile errors
14282	Fixed	Strategy Analyzer	An out of focus Strategy Analyzer chart could go transparent
14320	Fixed	Strategy Analyzer, Chart	Trading template applied to a backtest was not applied to a chart spawned from Executions/Trades send to Chart
14182	Fixed	Strategy Builder	Custom series named after an indicator caused a crash
14186	Fixed	Strategy Builder	Custom Series name used in Action could be changed, causing compile errors

14279	Fixed	Strategy Builder	ExitOnSessionCloseSeconds could be left blank, resulting in compile errors
14315	Fixed	Strategy Builder	Variables could incorrectly be named slippage, resulting in an error
14309	Fixed	SuperDOM	The X to cancel all bid or ask orders showed when no account was selected
14273	Changed	TD Ameritrade	Order submit wait time was increased to reduce timeouts
14252	Fixed	TD Ameritrade	Ticks came in with incorrect time stamps when time zone was set to UTC
14203	Fixed	TD Ameritrade, Data	Some custom instruments would not load historical data
14305	Fixed	TD Ameritrade, Hot Key, Orders	Placing orders with Hot Keys could unexpectedly get overbought/oversold errors
14247	Fixed	TD Ameritrade, Orders	An error could occur when canceling orders with a TIF or DAY
14257	Fixed	TD Ameritrade, Strategy	Resolved a scenario where a market order would get an error

14228	Fixed	Workspaces	Resolved a scenario where loading a workspace and closing a chart before it loaded resulted in an error

3.3.9 8.0.20.1

8.0.20.1 Release Date

December 23, 2019

Attention TD Ameritrade Users:

- For TD Ameritrade uses there is a new connection process to authorize the username and password. To continue to connect you must update to NinjaTrader 8.0.20.0 or newer by December 31, 2019. See the link here for more information

Issue#	Status	Category	Comments
14246	Fixed	TD Ameritrade, Orders	Session is now renewed every hour to ensure order submission works as expected

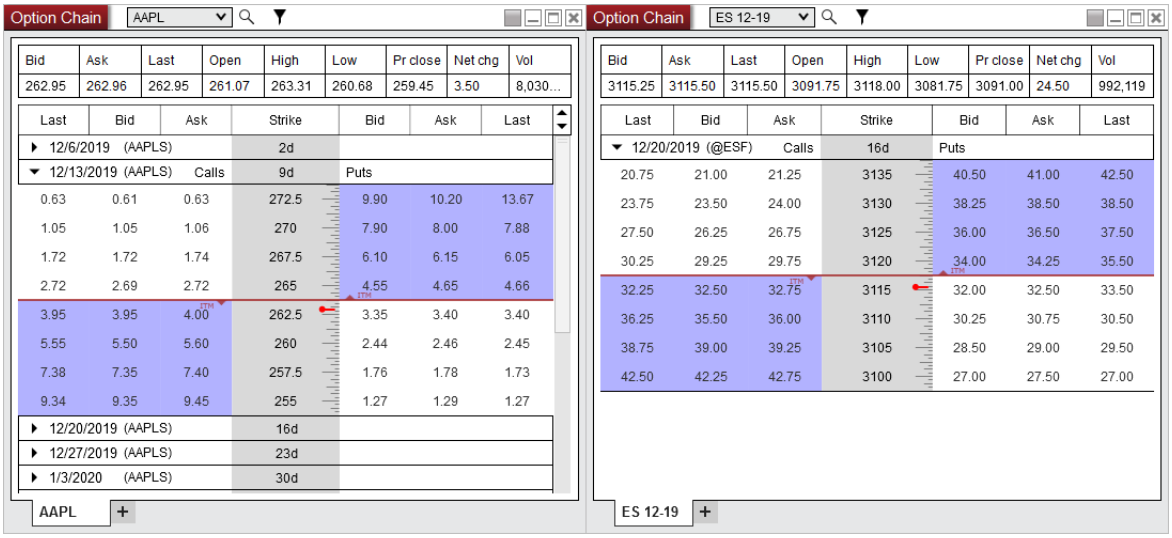
8.0.20.0 Release Date

December 5, 2019

Features

Added options support for Kinetick, IQFeed, and Interactive Brokers (beta)
Kinetick, IQFeed, Interactive Brokers, Options
Feature #13676

With Kinetick, IQFeed, and Interactive Brokers you can now access options on futures and equities. At this time these features are in beta.



Major improvements on Portuguese translations and sound files
Localization
Feature #14185

Added improvements to Portuguese translations provide a more complete experience for Portuguese clients.

Issue#	Status	Category	Comments
14083	Fixed	Alerts	Clicking an Alert Dialog box caused the box to move off screen
14113	Fixed	Alerts	Alerts Log did not save column settings
14014	Fixed	Alerts, Chart	Corrupt alert could prevent other alerts on the same chart from working
14090	Fixed	ATI	CLOSESTRATEGY command did not close orders or position from Atm Strategy
13981	Fixed	ATM Strategies	Target Chase activated without the target being touched

14002	Done	ATM Strategies	Reverse at stop/target orders would get canceled on disconnect and get stuck on cancel submitted
14129	Fixed	ATM Strategies	ATM strategy template was not saving when Chase if Touched was applied/taken off
14164	Fixed	ATM Strategies	An ATM with 'MIT enabled for Profit' would submit the MIT target order with a stop price of zero if position was added to an active ATM strategy
14135	Changed	Backup & Restore	Improved handling of references for backup/restore process
14009	Fixed	Chart	Resolved a scenario where cross hair cursor icon was incorrect after restarting
14045	Fixed	Chart	Dragging a data series from one chart to another did not keep bar width from originating chart
14055	Fixed	Chart	X axis label was missing on minute chart with some scaling settings
14073	Fixed	Chart	Resolved an error that could occur when the data box was saved off screen in a workspace
14077	Fixed	Chart	'No time scroll' cross hair setting did not display associated hot key
14133	Fixed	Chart	Chart templates would grow in size from saving information unrelated to the template
14038	Fixed	Chart, Window	Duplicate in new window while using instrument link could lose track of panel index

		Linking, NinjaScript	
14062	Fixed	Control Center	Removing three DLLs which were applied to a chart would cause a crash
14152	Fixed	Control Center, Commissions	Executions tab commission column did not honor account denomination property
14027	Fixed	Core	Failed to process install file error now auto repairs so it should only occur once
14123	Added	CQG	Added Open Interest with WebAPI
14142	Fixed	CQG	WebAPI would not display real time index data
14145	Fixed	CQG	Resolved a scenario where some WebAPI orders would be accepted at a different price
14146	Fixed	CQG	WebAPI would not attempt to reconnect after a lost connection
14018	Fixed	Depth Chart	Property presets were not applied at start-up
14101	Fixed	DrawingTool	Resetting a drawing object template caused drawing object to disappear
14125	Fixed	DrawingTool	Moving the ruler tool outside of chart boundaries resulted in a loop of errors
14138	Fixed	FXCM	Resolved a scenario where connection loss could get stuck
14081	Fixed	Historical Data	Some import errors did not display in the format expected

		Windo w	
1403 6	Fixed	Hot Key, Market Watch	Market Watch hot key was not listed under Global Hot Keys
1410 2	Fixed	Interacti ve Brokers	Bars with a volume of zero would show with a volume of one
1416 0	Fixed	Interacti ve Brokers	Resolved a scenario where orders would return to an incorrect contract month that does not exist
1414 9	Fixed	Interacti ve Brokers	Orders submitted at an invalid forex prices did not throw order rejection/error
1396 6	Fixed	Interacti ve Brokers , NinjaSc ript	Stop order was unexpectedly canceled when disabling a strategy
1405 0	Fixed	Interacti ve Brokers , Positio n Display	VX futures positions did not show up on order entry position display
1411 9	Fixed	Licensi ng	3rd Party Licensing incorrectly accepted user defined IDs with spaces
1418 7	Fixed	Licensi ng	Vendor licensing filter by Name returned no results

14115	Fixed	Localization, Commissions	Commissions Dialog Instrument Type was not localized
13967	Fixed	Market Analyzer	Resolved a scenario where configured indicator name didn't update as expected
14065	Fixed	Market Analyzer	Cross above/below cell conditions did not work as expected
14127	Fixed	Market Analyzer	Bar graph percent did not properly span full column width
14139	Fixed	Market Analyzer	Bar graph did not show negative values
14140	Fixed	Market Analyzer	Bar graph tooltip value was not rounded
14060	Fixed	Market Analyzer, Indicator	Correlation indicator produced error when applied
13965	Fixed	NinjaScript	Resolved a scenario where canceling a stop order from a strategy resulted in an error
14067	Fixed	NinjaScript	Switching tabs could lose cursor position
14163	Fixed	NinjaScript	Resolved a scenario where a DLL included in a backup file did not get added to Bin/Custom on restore

13969	Fixed	NinjaScript	Updated SampleOnOrderUpdate to track all execution and ensure the fills match before submitting stop market and limit orders
14112	Fixed	NinjaScript Editor	Importing NinjaScript file will caused focus of NinjaScript tab to change
14170	Fixed	NinjaScript Editor	Find window remained visible when changing workspaces
13996	Fixed	NinjaScript, ATM Strategies	GetAtmStrategyRealizedProfitLoss sometimes reported 0 after AtmStrategyClose was used
14053	Fixed	NinjaScript, Chart	Global draw objects placed by script plotted incorrectly on lower time frame charts
13848	Fixed	Option Chain	'Loading...' could stay stuck after disconnect
14097	Fixed	Options Chain	Middle ruler jumped around as new market data came in
14084	Fixed	Order Flow +	Order Flow Volume Profile POC and Value Area lines extended outside of trading hours
14104	Fixed	Order Flow +	Order Flow Volume Profile price profile could plot letters from outside of trading hours
14105	Fixed	Order Flow +	Order Flow Volume Profile price boxes overlapped when profile alignment was set to right

14108	Fixed	Order Flow +	Order Flow Volume Profile could cause chart to lag when 'Display mode' was set to 'Outline'
14167	Fixed	Order Flow +	Order Flow Volume Profile composite profile trading hours property did not affect the profile
14171	Fixed	Order Flow +	Order Flow Volume Profile Initial Balance accounted for 1 extra bar of data
14192	Fixed	Order Flow +	Order Flow Volumetric bars did not apply gradient to statistic values in some scenarios
14173	Fixed	Performance	Recent instruments could retain unnecessary information
14012	Changed	Performance, Chart	Improved performance for 1440 minute charts
14041	Fixed	Performance, Chart	Unchecking 'Equidistant bar spacing' could result in lag
14054	Fixed	Playback	Start and End Dates did not save when disconnecting and reconnecting
14068	Fixed	Playback	A multi-series strategy behaved differently if applied on the strategies tab vs a chart
14103	Fixed	Playback, NinjaScript	ImmediatelySubmit was submitting duplicate orders in Playback
14154	Fixed	Playback, NinjaScript	Strategy using CloseStrategy() caused a freeze

14087	Fixed	Playback, Strategy, Trade Performance	Error occurred when viewing real-time trade performance in Playback
14150	Fixed	Position Display	Submitting an order to an invalid instrument resulted in a crash
14162	Fixed	Stock Import	Could not import stocks to create instruments with numbers
13972	Fixed	Strategy Analyzer	Opening AI Generate Strategy Analyzer result in new Strategy Analyzer resulted in an error
13976	Fixed	Strategy Analyzer	Parameters tool-tip in the Log showed the parameters for the wrong test
13990	Fixed	Strategy Analyzer	Double clicking a Walk Forward result in the Log resulted in the Order, Execution, and Trades displays do not populating
13998	Fixed	Strategy Analyzer	Performance value in Results grid did not match value in summary during optimization
14032	Fixed	Strategy Analyzer	Backtests with a high order fill resolution of 1 tick resulted in an error
14051	Fixed	Strategy	Monte Carlo report for MaxConsecutiveWinners and MaxConsecutiveLosers were reversed

		Analyzer	
14120	Fixed	Strategy Analyzer	Optimizer could get an error when running off historical data with some sets of data and settings
14124	Fixed	Strategy Analyzer	Compiling reset some parameter settings
14148	Fixed	Strategy Analyzer	Backtest with a commission template applied and 'Display' set to 'Percent' incorrectly formatted commission value after a restart
14151	Fixed	Strategy Analyzer	Resetting strategy template then running a backtest changed parameter values after backtest
14046	Fixed	Strategy Builder	Using an indicator as an input resulted in a compile error
14143	Fixed	Strategy Builder	Order Flow + indicators were incorrectly available in the Strategy Builder
14179	Fixed	Strategy Builder	User defined inputs would allow quotation marks when it should not
14181	Fixed	Strategy Builder	Creating an action which sets custom series to its value 1 bar ago would change after a restart and reopening of the strategy in Strategy Builder
14166	Fixed	Strategy	Using system indicator names for user inputs was incorrectly allowed

		Builder	
14147	Fixed	Strategy, Chart	Unchecking enabled on a strategy applied to chart, then removing the strategy without OK/Apply caused chart to retain strategy
14080	Fixed	SuperDOM	Dynamic SuperDOM could not select ATM drop down reliably
14121	Fixed	SuperDOM	Resolved a scenario where there could be a bars ago error when loading an indicator
14033	Fixed	SuperDOM, NinjaScript	Indicator that adds a daily series would not load
14111	Fixed	SuperDOM, Window Linking	Window linking passed through to other workspaces when 'Global link button across workspaces' was disabled
14064	Fixed	Tick Replay, Strategy	Enabling a strategy with Tick Replay after a restart resulted in a crash
14093	Fixed	Tick Replay, Bars	Resolved a scenario where some scripts saved in a workspace with Tick Replay resulted in an error
14074	Fixed	Workspaces	DataBox was left off screen when opening workspace and clicking yes to move windows to primary monitor

3.3.10 8.0.19.1

8.0.19.0 Release Date
September 23, 2019

Attention IQFeed Users:
- For NinjaTrader 8.0.19.0, it is mandatory that you update to the latest IQFeed 6.1 client.

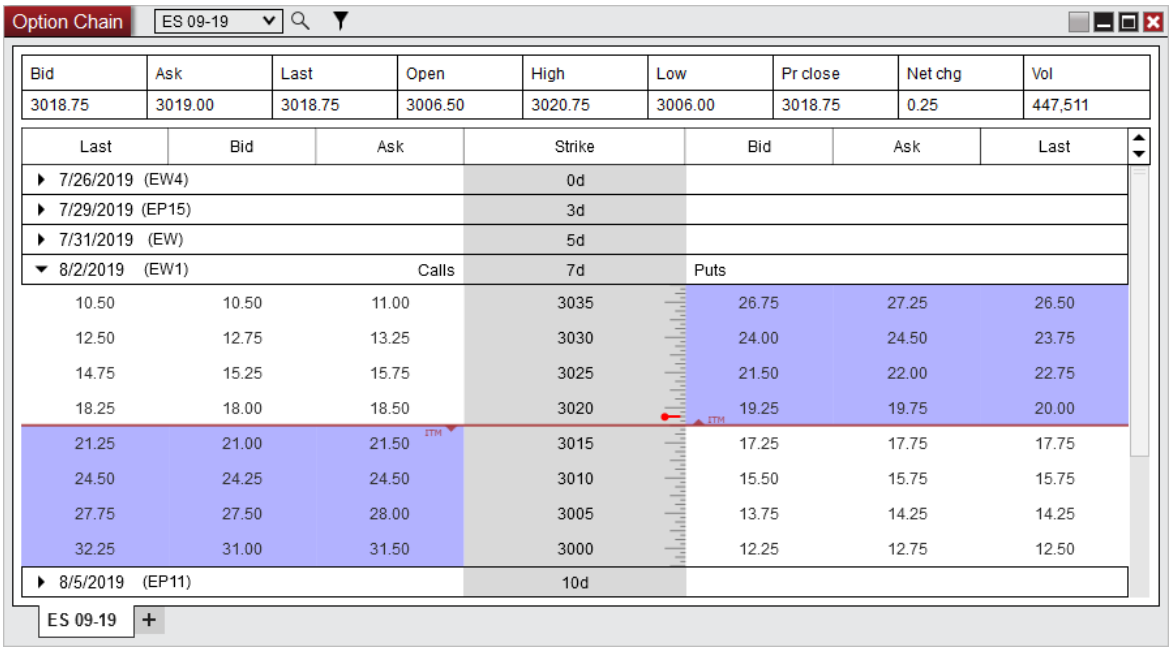
Attention developers:
- When working with D2DFactory to create Direct X resources, this must be done from the charts UI thread otherwise there will be a performance impact. This is now enforced in 8.0.19.0. If your script does not follow this policy it will not be compatible with 8.0.19.0 and a log error will be thrown. Please contact platformsupport@ninjatrader.com if any questions.

Features

Added Option Chain and support for Options

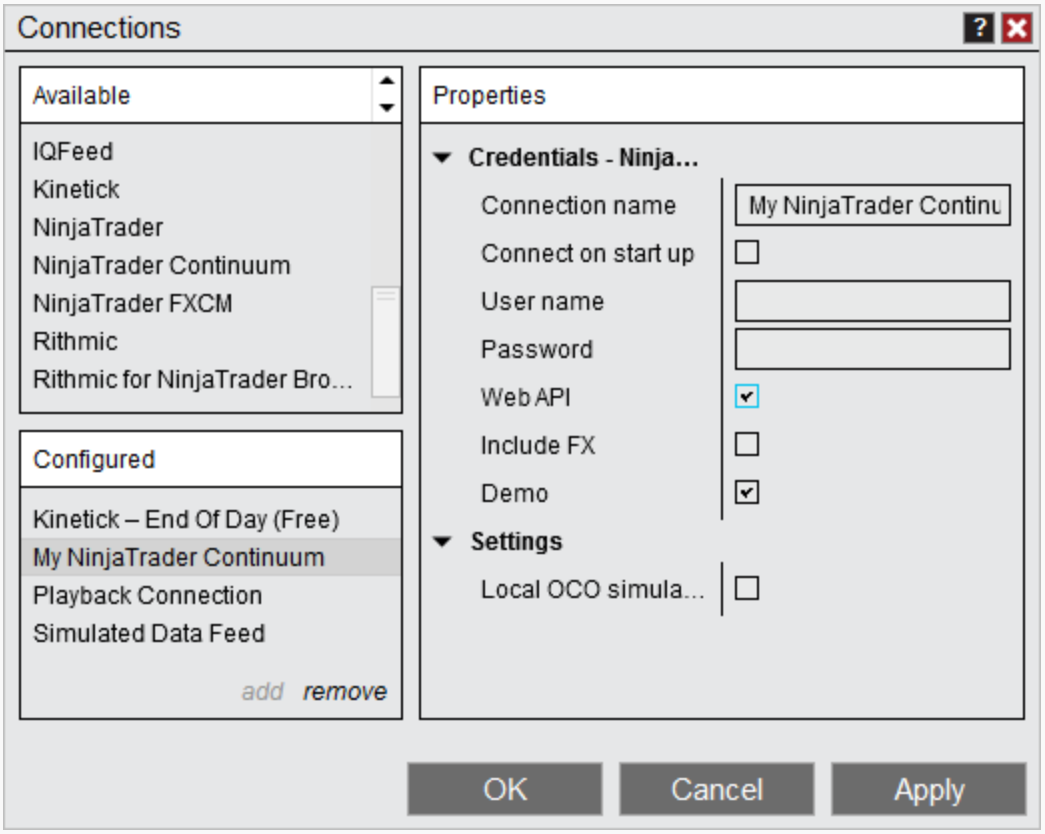
Option Chain
Feature #13844

Options support has been added and can be accessed with the new Option Chain window for users utilizing NinjaTrader Continuum connection and have enabled 'Web API' (Beta). Linking with order entry windows and selecting a bid or ask quote within the option window will load the option into the order entry window for order placement. See the Option Chain section for more information.



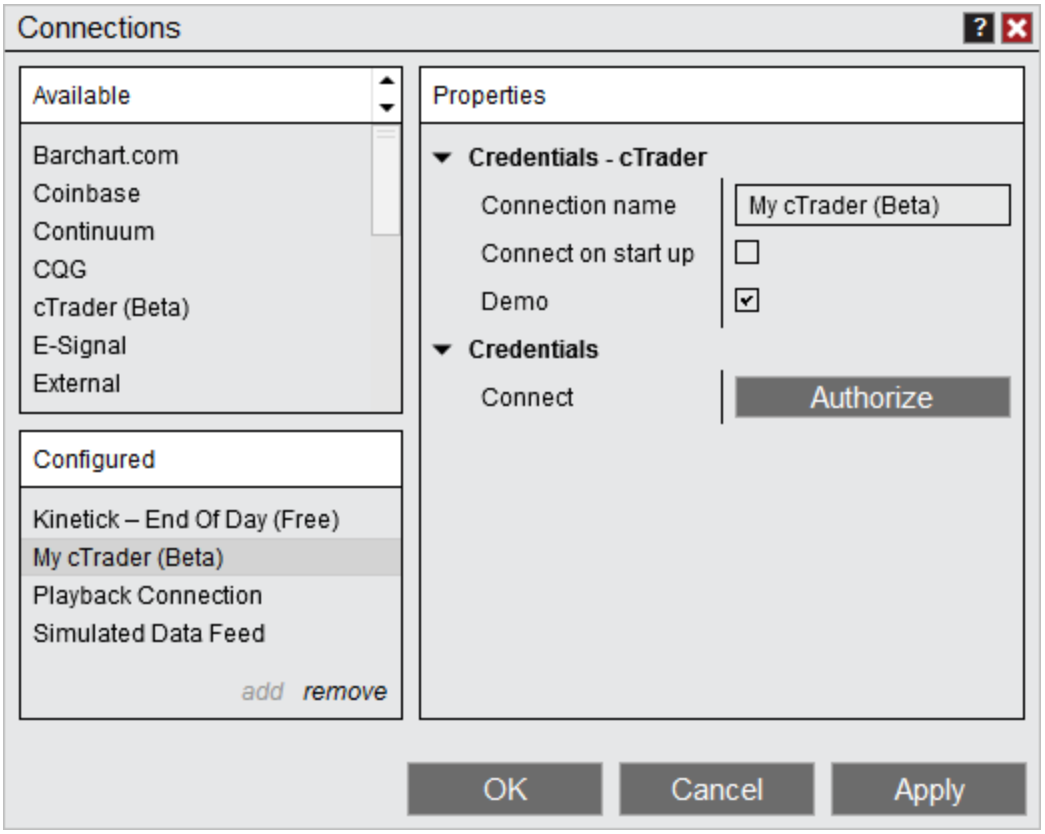
Added Web API support for NinjaTrader Continuum in Beta
NinjaTrader Continuum
Feature #13421

The new Web API adds support for options to NinjaTrader Continuum



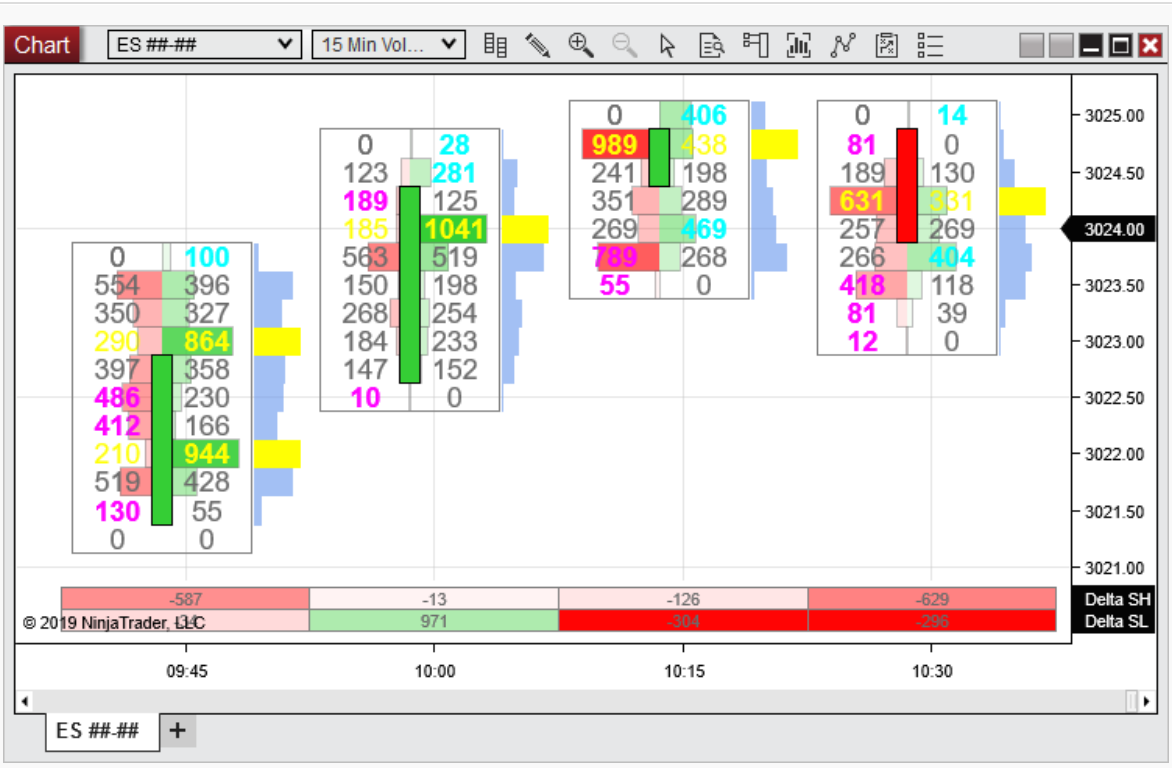
cTrader available in beta
cTrader
Feature #13895

With an enabled license key you can now access your cTrader 'netting' account(s) in NinjaTrader. This is currently in beta.



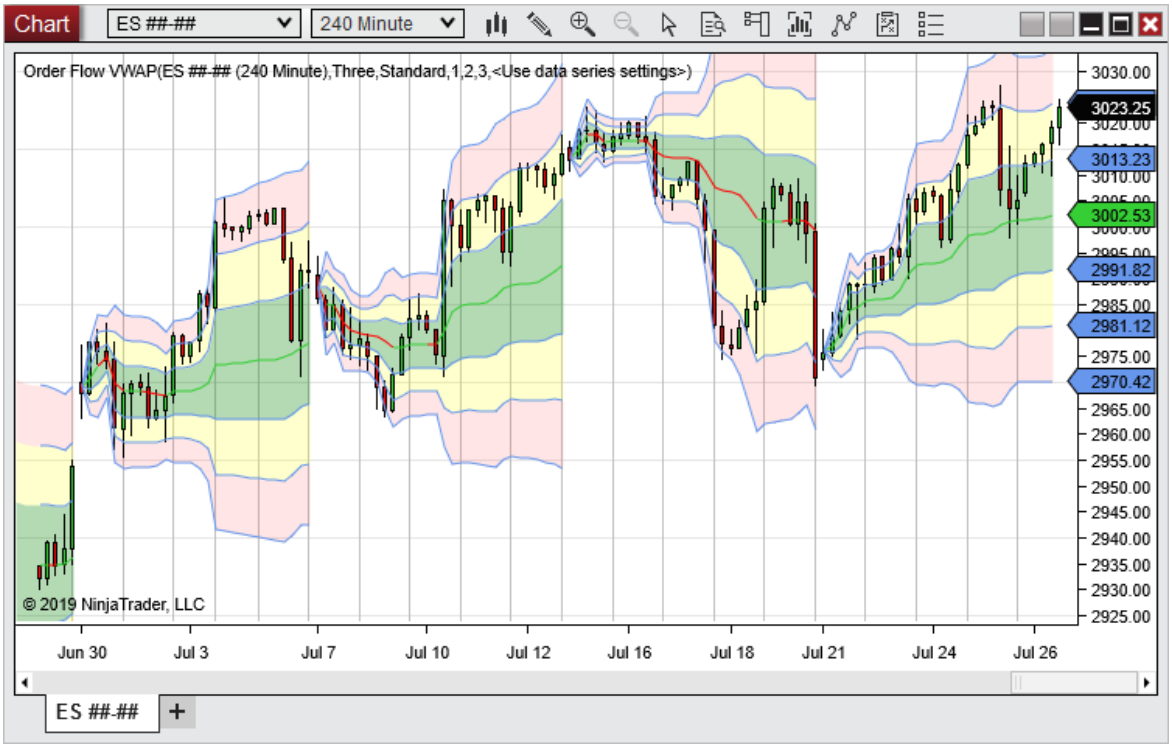
Order Flow Volumetric enhancements
Order Flow +
Feature # 13614

Multiple features have been added to further increase customization and functionality.
The candlestick can now be centered on plot.
A Size filter property has been added to only include values greater than the threshold.
Delta SL & Delta SH have been added to the bar statistics to measure how long since the delta touched the high/low of the bar.
Imbalance can now be calculated horizontally.
Bar volume distribution can now be set to the right of the bar.



Order Flow VWAP enhancements
Order Flow +
Feature # 13611

Multiple features have been added to further increase customization on functionality.
A Reset interval can now be set to Sessions, Weeks, or Months.
Each standard deviation can now have color and opacity individually set.



Added FX Correlation Window
FX Correlation
Feature # 13661

The FX Correlation window is used to display a correlation between multiple forex instruments. Values close to 1 indicate movement in the same direction. Values close to -1 indicate movement in opposite directions. Values near 0 indicate no correlation. See the FX Correlation section for more information.

FX Correlation | Year

	AUDUSD	EURCHF	EURGBP	EURJPY	EURUSD	GBPUSD	USDCAD	USDCHF	USDJPY
AUDUSD		0.58	0.06	0.75	0.67	0.42	-0.41	-0.21	0.59
EURCHF	0.58		-0.39	0.66	0.42	0.67	-0.11	0.36	0.67
EURGBP	0.06	-0.39		0.00	0.30	-0.74	-0.35	-0.62	-0.25
EURJPY	0.75	0.66	0.00		0.78	0.55	-0.46	-0.27	0.86
EURUSD	0.67	0.42	0.30	0.78		0.41	-0.61	-0.67	0.38
GBPUSD	0.42	0.67	-0.74	0.55	0.41		-0.11	0.12	0.50
USDCAD	-0.41	-0.11	-0.35	-0.46	-0.61	-0.11		0.56	-0.20
USDCHF	-0.21	0.36	-0.62	-0.27	-0.67	0.12	0.56		0.13
USDJPY	0.59	0.67	-0.25	0.86	0.38	0.50	-0.20	0.13	

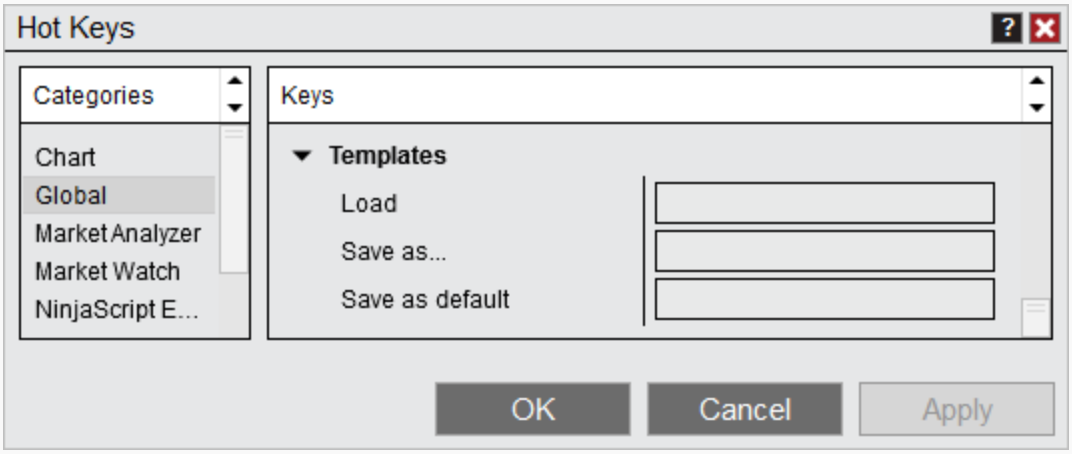
FX Correlation

Added Correlation indicator
Indicator
Feature # 13755

The correlation indicator will plot the correlation of the data series to a desired instrument. Values close to 1 indicate movement in the same direction. Values close to -1 indicate movement in opposite directions. Values near 0 indicate no correlation.

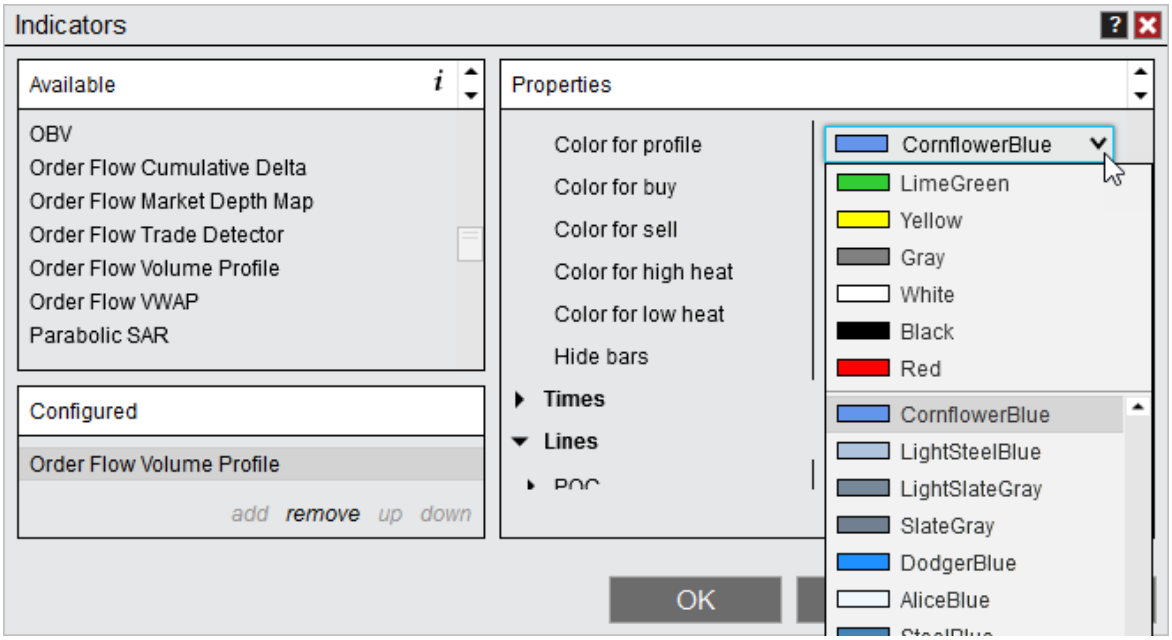
Added additional hot keys
Hot Key
Feature # 13534

The new hot keys make working with NinjaTrader faster and easier.
Charts - Auto scale and return for charts
Workspaces - Save all Workspaces
Templates - Load/ Save as../ Save as default



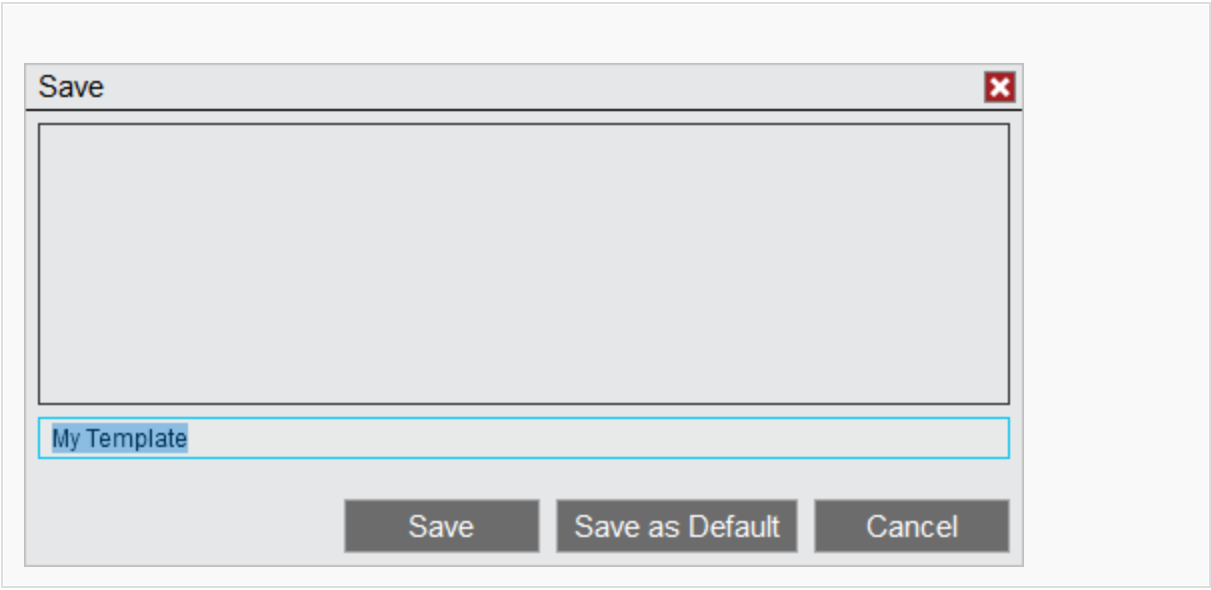
Added recently used colors to the top of the color picker
UI
Feature # 13700

Now the most recently used colors used are placed at the top of the color picker, making it easier to quickly select frequently used colors.



Added save as default button
Templates
Feature # 13672

Save as default has been added to the Indicator, Strategy, Strategy Analzyer, and Drawing tool template dialogs for a more intuitive experience.



Issue#	Status	Category	Comments
13781	Fixed	Alerts, Market Analyzer	Resolved a scenario where CrossAbove/Below alert did not trigger when expected
13980	Fixed	ATM Strategies	Stop strategy logic did not resume when manually disconnecting/reconnecting
13679	Fixed	Barchart, Charts	Historical data was not loading for tick based charts from midnight to the current time
13851	Fixed	Bars	Kagi chart style rendered lines when brush was set to transparent
13989	Fixed	Bars, Order Flow +	A renko chart with Order Flow Cumulative Delta applied could get an error
13840	Fixed	BarsType	Point and figure chart with Price set to Highs and Lows could display large erroneous bars

13941	Changed	Chart	Improved performance for charts which have lots of rendering
13963	Changed	Chart	Optimized crosshair performance
13723	Fixed	Chart	Renaming a chart tab that isn't the active tab resulted in an error
13738	Fixed	Chart	Zoom in selection box could draw into x and y axes
13768	Fixed	Chart	Compressed chart with Nymex Energy RTH template caused unreadable time axis
13798	Fixed	Chart	Charts were missing data around rollover date with UTC+8
13821	Fixed	Chart	PnF charts could time-stamp a session close bar with the next days time
13833	Fixed	Chart	Resolved a scenario where removing a second data series from a chart resulted in an error
13945	Fixed	Chart	Expanding the price axis too far resulted in the time scales disappearing
13947	Fixed	Chart	Typing numbers in a chart less than 10 would change the minute interval rather than return invalid instrument
13959	Fixed	Chart	Resolved a scenario where removing a second data series, that was using imported data, resulted in an error
13926	Fixed	Chart, Drawin	Show global drawing objects set to false could result in duplicate drawing tool names

		gTools	resulting in conflictions
13784	Fixed	Chart Trader, Hot Key	Modifying orders with Hot Keys were slow to display change on Chart Trader
13805	Fixed	Chart, Strategy	Loading a strategy template allowed the strategy to be enabled when disconnected resulting in an error
13596	Fixed	City Index	Resolved a scenario where attempting to trade an unmapped CFD had unexpected results
13734	Fixed	City Index, Orders	Positions were not volume weighted
13960	Fixed	CoinBase	Resolved a scenario where converting time zones for L2 timestamps resulted in an error
13902	Fixed	Connections	An error occurred when trying to edit an account while connecting
13827	Changed	Control Center	Changed display of strategy parameter tooltips on Strategies tab to match Strategy Analzyer
13901	Fixed	Control Center	Getting Connected window could be on top of error when incorrect credentials were entered
13931	Fixed	Control Center	Removed erroneous Add Account right click menu item from Accounts tab
13792	Fixed	Control Center	When connected to Playback Filter by list showed Sim101 rather than Playback

		, Playback	
13747	Fixed	Data Grids	Multi-Line tool-tips did not display as expected
13743	Fixed	Data Grids, Orders	Orders grid multi-line rejected messages did not display as expected
13681	Fixed	Data, Playback	When downloading current day data at times it would not download to the current time
13724	Fixed	DrawingTool	Anchor values changed when attaching to all charts
13910	Fixed	DrawingTool	Andrew's Pitchfork did not display when chart anchors were far out of range
13838	Fixed	DrawingTool, Chart	Global drawing objects were merged into old contracts on reload of historical data
13954	Fixed	DrawingTool, Workspaces	Resolved some scenarios where global drawing objects would duplicate or no longer be global when working with multiple workspaces
13787	Fixed	eSignal, Data	Historical data was not downloaded for yesterday when equidistant bar spacing was enabled
13746	Changed	FXCM	Changed addresses to https
13847	Fixed	Historical Data	Excluded bars were not excluded on reload of historical data

		Window	
13875	Fixed	Historical Data Window	Cryptocurrency historical data would not import with decimals for volume
13732	Fixed	Historical Data Window, Playback	Unexpected contracts showed in Historical Data 'Edit' window when using the Playback connection
13898	Fixed	Hotlist Analyzer	Workspace with more than 1 Hotlist window would not retain the name of the Hotlist selected on restart
13690	Fixed	Indicator	Running ROC on Cumulative Delta resulted in an error
13753	Fixed	Indicator	Wiseman indicators did not display an indicator name in the "Configured" list after their "Label" setting text was removed
13763	Fixed	Indicator	Block Volume's real-time value did not match it's historical value
13884	Changed	Interactive Brokers	New adapter is out of beta
13916	Fixed	Interactive Brokers	Resolved a scenario where an error from modifying the order quantity did not show a pop up error

13899	Fixed	Interactive Brokers, Data	Real-time data could be a tick off what was expected with beta
13818	Fixed	Interactive Brokers, Strategies	Unmanaged multi instrument strategies submitting orders to 10 instruments caused strategy position to become out of sync
13789	Fixed	Instruments	Tick size addition 0.00000001 was not working as expected
13797	Fixed	Instruments	Changes made to copied instrument lists could be reflected in the original list
13687	Fiked	Kinetick, Data	Last Close was not loading for some instruments after workspace was closed and reopened
13702	Fixed	Kinetick, News	Resolved a scenario where a crash could occur while News window was open
13817	Changed	Kinetick, IQFeed	Improved daily bar handling for after hours charting
13722	Fixed	Market Analyzer	After setting the background for the grid once it could not be reset to another color and retained it's value
13748	Fixed	Market Analyzer	Removing cell while scroll-bar was visible could result in an error
13793	Fixed	Market Analyz	Enable color distribution setting could change foreground color unexpectedly

		er	
13794	Fixed	Market Analyzer	T & S trend column did not update when changing the rows instrument
13814	Fixed	Market Analyzer	Updating a row from a valid instrument to an invalid one continued to show values from previous instrument
13937	Fixed	Market Analyzer	Resolved a scenario where adjusting indicator column settings was not reflected in the label
13780	Fixed	Market Analyzer, Workspaces	Column widths did not save in workspace
13866	Fixed	Market Analyzer, Workspaces	Saving workspace with an invalid instrument in the Market Analyzer resulted in an error
13689	Fixed	Market Watch, Market Analyzer	Market Watch and Market Analyzer ChartNetChange column did not use max available scale range at times
13940	Fixed	NinjaScript	Importing a script that has a warning when importing, the warning could not be selected since Import Successful window was on top
13979	Fixed	NinjaScript, Strategy Builder	A strategy built in the Builder with an action to print Volume with a multiplier applied would not compile

13951	Fixed	NinjaScript, Chart	Strategies with ImmediatelySubmit resubmitted orders after disconnecting then reconnecting when strategy was applied to a chart
13993	Fixed	NinjaScript, Tick Replay	Resolved a scenario where GetTime() resulted in an error when connecting on startup
13704	Fixed	NinjaScript Editor	Lock up could occur when compiling then opening a new NinjaScript Editor, Chart, or Strategies dialog window
13942	Fixed	NinjaScript Editor	Error could occur if compile was pressed repeatedly
13973	Fixed	NinjaScript Editor	Qty selector pad could be available for the NinjaScript editor indicator wizard
13890	Fixed	NinjaScript Editor, NinjaScript	Opening NinjaScript Editor with bad 3rd party dll loaded resulted in a crash
13717	Fixed	NinjaScript, Chart	IsOverlay could not be set in State.Configure
13733	Fixed	NinjaScript, Chart	Using the global drawing object overload caused duplicated objects when switching instruments if the window was linked
13882	Fixed	NinjaScript, Indicator	ZigZag indicator was returning incorrect bar values for the HighBar and LowBar method

14000	Fixed	NinjaScript, DrawingTool	A NinjaScript that utilizes DrawingTool templates could have an error when reloading NinjaScripts or opening the workspace
13949	Changed	Orders	Conversion Rate logic for CFD executions will prioritize the CFD currency rate over the FX currency rate
14026	Fixed	Orders	Restored execution rate could be erroneously updated after a disconnect and reconnect
13957	Changed	Order Flow +	Improved Order Flow Market Depth Map performance
13678	Fixed	Order Flow +	Order Flow Volume Profile indicator with a template which sets a color to default resulted in an error
13804	Fixed	Order Flow +	Volume Profile could cause region to go out of sync with plots at times
13820	Fixed	Order Flow +	Order Flow Cumulative Delta values were off relative to Order Flow Volumetric at beginning of session
13904	Fixed	Order Flow +	When Volumetric bar's text was hidden some rows could be highlighted together in one color
13929	Fixed	Order Flow +	Resolved a scenario where the first real-time drawn Order Flow Volume Profile would disappear once the bar closed
13946	Fixed	Order Flow +	Order Flow Volume Profile with Price profile did not plot the z
13953	Fixed	Order Flow +	Volumetric bar used with a secondary non-volumetric bar would result in the secondary

			series width getting stuck on 1 when compressing and decompressing
13796	Fixed	Order Flow +, Drawing	Drawing an Order Flow Volume Profile which has a default template already set rendered the tool on top of bars rather than behind
13930	Fixed	Market Analyzer	Fixed 'Calculating...' text alignment
13782	Fixed	Playback	Resolved some scenarios where Playback controller did not display
13920	Fixed	Playback	Playback would play beyond the end data when PC had German date/region settings
13948	Fixed	Playback, Historical Data	Playback data did not show in the Historical Data window on first download
13810	Fixed	Playback, Strategy	Resolved a scenario where enabling a strategy while connected to playback resulted in a crash
13897	Fixed	Risk	On a new installation orders submitted to sim account with NinjaTrader Brokerage Default Risk template applied threw an error on order submission
13824	Fixed	Regionalization, Historical Data Window	Importing data with language set to Spanish resulted in an error

13767	Fixed	Regionalization, Strategy Analyzer	Optimization results right click context menu text could be cutoff in German
13735	Fixed	Regionalization, Tool Tips	Some tool-tips were not localized properly
13684	Fixed	Regionaliztion, Control Center	Log and trace files failed to send when platform language was set to German
13867	Fixed	Rithmic, Instruments	Opening chart for ZQ 08-19 resulted in an error
13850	Changed	Share Adapter	Changed Sender Display Name to From Name
13854	Changed	Share Adapter	Updated default AOL Mail settings
13574	Fixed	Strategy	Strategy on renko bar would cancel SetProfitTarget & SetTrailingStop orders on 1st trailing stop modification
13594	Fixed	Strategy	An error could occur after enabling then disabling a strategy and attempting to change it's properties

13713	Fixed	Strategy	Resolved some scenarios where no strategy is selected when opening the strategies window
13731	Fixed	Strategy	Resetting Sim101 account then re-enabling a strategy resulted in an error
13595	Fixed	Strategy Analyzer	Stop limit orders could fill outside bar on gap scenarios in backtesting
13695	Fixed	Strategy Analyzer	Custom performance metrics did not work for multi-instrument optimization combined row
13803	Fixed	Strategy Analyzer	Viewing optimization result on second open returned 0's
13807	Fixed	Strategy Analyzer	When changing from Backtest to Optimize the strategy that was selected was lost and defaults to the first in the list
13819	Fixed	Strategy Analyzer	Optimizer results were not copying over optimized data series value correctly
13829	Fixed	Strategy Analyzer	Could attempt to open combined results which resulted in an error
13831	Fixed	Strategy Analyzer	Resolved a scenario where running an Optimization on a strategy with drawings could result in an error

13837	Fixed	Strategy Analyzer	Resolved a scenario where if NinjaScript Editor was open it would be brought in front of other windows by Strategy Analyzer actions
13842	Fixed	Strategy Analyzer	Error occurred when clicking View Strategy in Optimization results
13874	Fixed	Strategy Analyzer	A genetic optimization that optimized doubles could give parameters that are outside of the step range
13888	Fixed	Strategy Analyzer	Multile screens with different DPI settings caused irregular behavior
13887	Fixed	Strategy Analyzer	Error could occur when hovering mouse over NinjaTrader windows in the Taskbar while in Strategy Analyzer
13894	Fixed	Strategy Analyzer	Opening a walk forward optimization in a new tab resulted in an error if it was from the view results window
13905	Fixed	Strategy Analyzer	Chart display had an unexpected right click context menu on the toolbar
13961	Fixed	Strategy Analyzer	Right-clicking in the Strategy Analyzer Log without selecting a row resulted in an error
13975	Fixed	Strategy	Clicking a result of AI Generate then selecting Display Chart had Sample ATM Strategy listed in the top left of the chart

		Analyzer	
13982	Fixed	Strategy Analyzer	AI Generated strategy which used CrossBelow and Close Series would not compile
13984	Fixed	Strategy Analyzer	AI Generate Trade Result listed Strategy as Sample ATM
13986	Fixed	Strategy Analyzer	Strategy Analyzer Column Result listed SampleATMStrategy on AI Generate Run
13987	Fixed	Strategy Analyzer	Log filters were not working
13696	Fixed	Strategy Analyzer, NinjaScript	Calling RemoveDrawObject did not not remove objects from chart in backtest or optimization
13783	Fixed	Strategy Builder	DEL key did not work for manually entered strings
13776	Fixed	Strategy Builder	When using a Custom Series CrossAbove Numeric value an error occurred
13777	Fixed	Strategy Builder	When adding a data series a current bar check was not added resulting in an error

13785	Fixed	Strategy Builder	Quantity input had to be deleted before a new value could be entered
13716	Fixed	Strategy, Control Center	DaysToLoad set in SetDefaults was not applied when adding a strategy to the Strategies tab
13881	Fixed	Strategy, Control Center	Resolved a scenario where a position update on the Strategies grid resulted in an error
13816	Fixed	Strategy, Order Flow +	Order Flow Cumulative Delta values incorrectly reported in strategy script when session line crossed
13572	Fixed	SuperDOM	Changing properties of a column and applying it affected the width of all columns
13703	Fixed	SuperDOM	Reloading historical data with volume column applied failed to load
13706	Fixed	SuperDOM	Volume column width modified on reload of historical data
13709	Fixed	SuperDOM	Modifying the size of the price ladder when a column is applied then adding another column reset the price ladder size
13822	Fixed	SuperDOM	A column set to invisible then back to visible did not display
13860	Fixed	SuperDOM	Position entry marker was not updated on account change
13872	Fixed	SuperDOM	Resetting sim account caused bid and ask values to not display until scrolled

13908	Fixed	SuperDOM	Increased rendering performance when using indicators and resolved a deadlock scenario
13911	Fixed	SuperDOM	Resolved a scenario where a configured indicator display name did not update
13933	Fixed	SuperDOM	Orders tab filter was not maintained when duplicating window
13913	Fixed	SuperDOM, Templates	Resolved a scenario where a template resulted in an error
13725	Fixed	SuperDOM, ATM Strategies	ATM Target +- buttons were not selectable if ATM strategy was submitted with "Select active ATM strategy template on order submission"
13891	Fixed	Templates, Orders Flow +	Using a template could result in Volumetric bars loading with wrong chart style
13944	Fixed	Tick Replay, Order Flow +	Reloading NinjaScript while using Tick Replay and Order Flow Cumulative Delta could result in an error
13826	Fixed	Trade Performance, NinjaScript	The label for EntryTime was labeled as ExitTime in ToString
13923	Fixed	UI	Resolved an error that could be caused by a corrupt UI file
14005	Fixed	Workspaces,	Closing background workspace caused Databox to disappear

		Chart	

8.0.19.1 Release Date

October 18, 2019

Issue #	Status	Category	Comments
13857	Fixed	Translations	Updated various Portuguese translations
14040	Fixed	CQG/ Continuum Web API	Fixed various issues account and order issues
14042	Fixed	IQ Feed	Fixed incorrect handling for tick-based charts
14043	Fixed	Order Flow + - VWAP	Fixed 'session' reset of VWAP in certain scenarios
14052	Fixed	Translations	Fixed F1 links to go to appropriate translated help resources
14056	Fixed	Barchart	Resolved a real-time data issue
14058	Fixed	Chart	Global drawing objects could no longer change visibility after switching instruments
14069	Fixed	IQ Feed	Historical bars were timestamped incorrectly for minute bars
14070	Fixed	Order Flow + VWAP	Standard deviation lines would appear in the Drawing Objects dialog even when disabled

14078	Fixed	IQ Feed	Resolved 'Illegal characters in path' error when attempting to connect
14098	Fixed	Kinetick	Resolved an issue where the current daily bar would not be refreshed properly on open of a new chart
14100	Fixed	UI	Configuring a connection with the same name could cause conflicts
14106	Fixed	UI	Resolved error "The given key was not present in the dictionary" error on connecting to a provider under a certain scenario

3.3.11 8.0.18.1

8.0.18.0 Release Date

May 6, 2019

Features

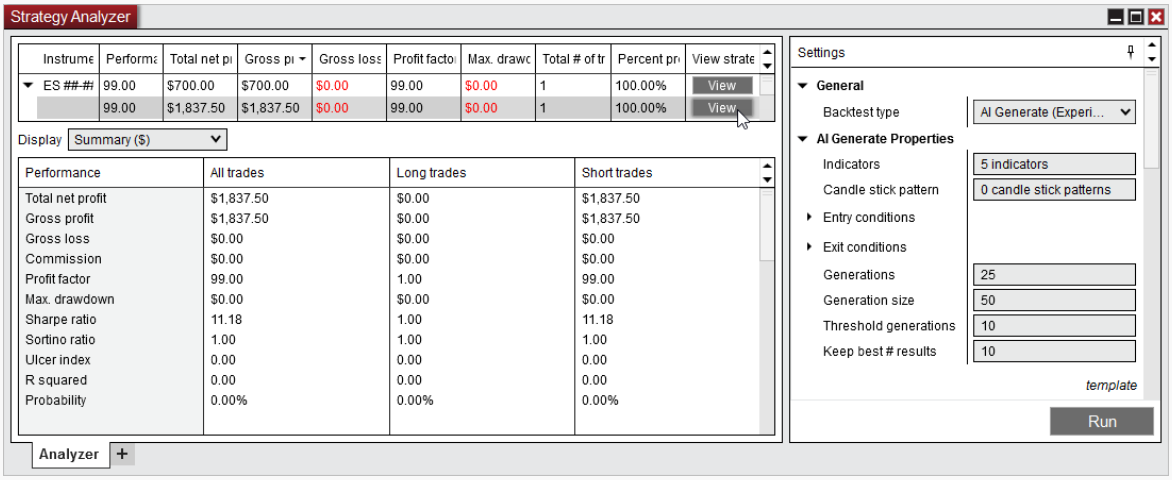
Added Market Watch
Market Watch
Feature #13434

With the new Market Watch window you can quickly see the net change throughout the session, since last close. At the top right of each tile you will see the price with the net change below it in points and percent.

Added an Automatic Strategy Generator (Experimental)
Strategy Analyzer
Feature #13039

Automatically generate a NinjaScript strategy with the new experimental* “AI Generate” backtest type. Configure any number of indicators and/or candle stick patterns to be considered and sit back while NinjaTrader does all the work. After running the AI Generate it will list for you the best strategies from the results. Selecting View will then pull up the strategy code in the NinjaScript Editor so you can view it and save it if desired.

**This is a NinjaTrader experimental feature that may or may not evolve over time.*



Added Equivolume charts

Charts

Feature #13444

With equivolume bars you you can easily see which bars received the largest volume (indicated by being the widest) and which bars received the least volume (indicated by being the thinnest) in comparison to the other bars in view.

Added automatic "Trend Lines" indicator

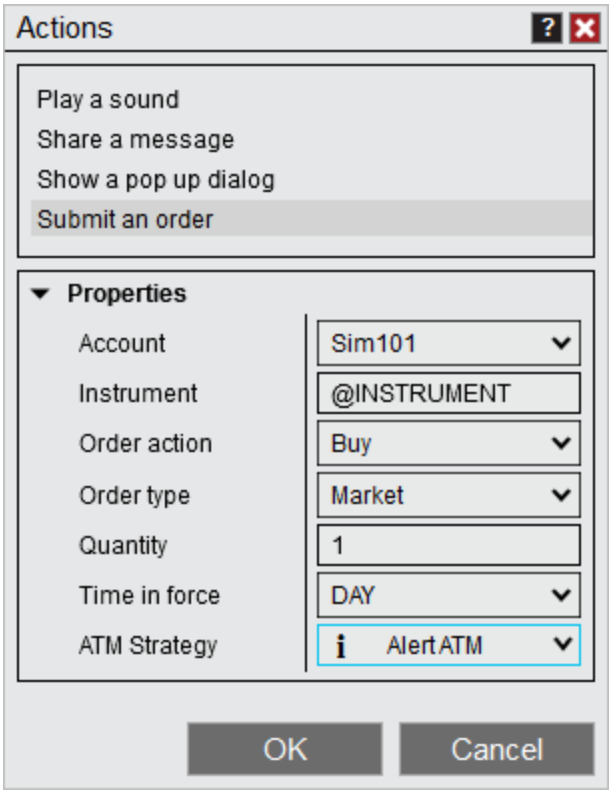
Indicators

Feature #13315

When a high swing is followed by a lower high swing, a trend line high is automatically plotted. When a low swing is followed by a higher low swing, a trend line low is automatically plotted.

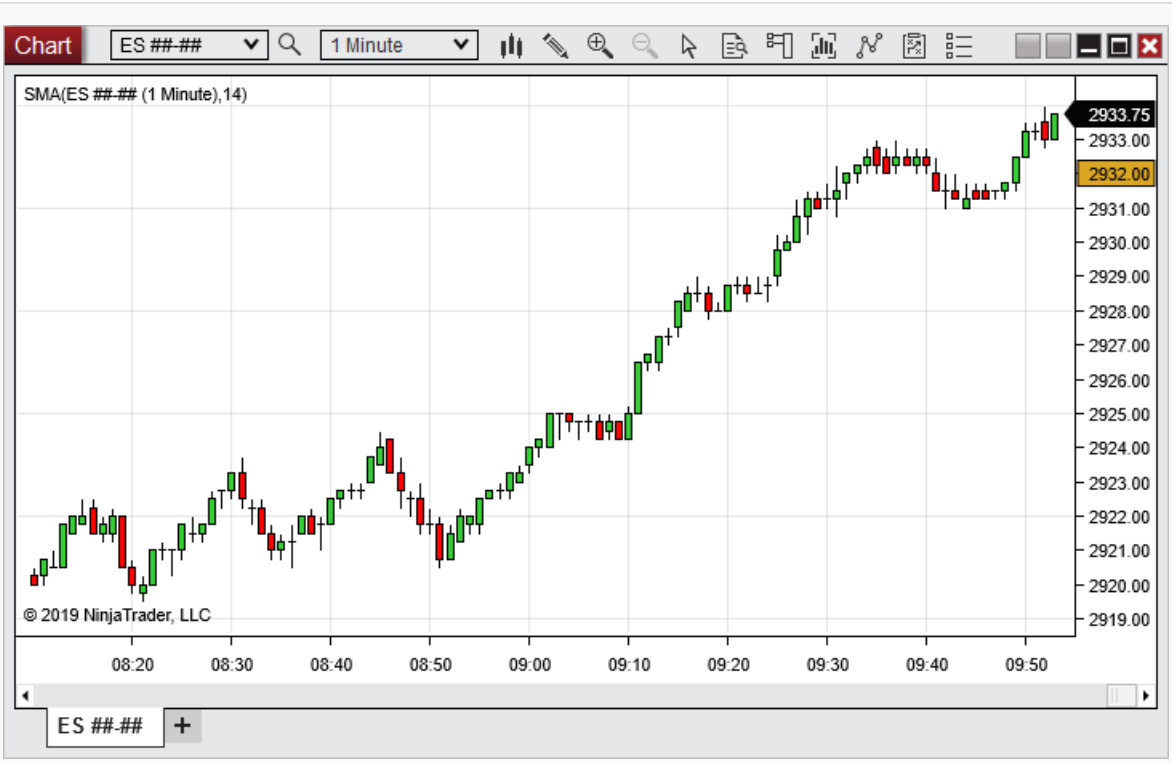
Added the ability to submit an ATM order on Alert's Actions
Alerts, ATM Strategies
Feature #13398

With Submit An Order you can now select a saved ATM strategy to apply to the order.



Added Price Box plot style
Chart, Indicators
Feature #13561

The Price Box plot style will display just a price box in the y-axis, without plots on the bars.



Added NinjaScript Editor support for Visual Studio 2019
NinjaScript Editor
Feature #13666

NinjaScript Editor now supports Visual Studio 2019 and will default to that if multiple versions are installed

General updates to sound files and new regionalized sounds
Localization
Feature #13469

Some new modern voices have been added and are available in multiple languages

Issue#	Status	Category	Comments
13321	Fixed	Alerts	Alert objects could change unexpectedly in multi-series scenario

13505	Fixed	Alerts	Volume alerts using arithmetic offsets fired without applying the offset
13597	Fixed	Alerts	Resolved a scenario where on condition reversed alert did not rearm properly after resume/restart
13565	Fixed	ATM Strategies	Resolved a scenario where using currency mode did not submit stop as expected
13622	Fixed	ATM Strategies, Chart Trader	Resolved a scenario where modifying a stop that was submitted in another window resulted in an error
13624	Fixed	Backup & Restore	An error occurred when restoring from a foreign language backup file
13530	Fixed	Chart	Background image could not be changed if corrupt image file was selected
13568	Fixed	Chart	Charting a stock instrument with no default exchange selected and displaying the mini databox resulted in an error
13625	Fixed	Chart	Resolved a scenario where chart would freeze when removing a drawing object
13635	Fixed	Chart	Transparent plots with both negative and positive values caused lagging in the chart
13669	Fixed	Chart	Resolved a scenario where loading historical executions resulted in an error
13510	Fixed	Chart Trader	Chart Trader could be too small in width on duplicated chart window
13613	Fixed	Chart Trader	Chart became unresponsive when an ATM stop with reverse was modified more than once

13509	Fixed	Chart, Strategy	VWAP could be missing regions when hosted in strategy enabled from the Control Center
13621	Fixed	Chart, Templates	Setting Show Tabs to False in a chart template could cause tabs to go missing unexpectedly
13617	Fixed	Coinbase	Disconnect could result in a crash
13626	Fixed	Coinbase	Resolved an error that occurred from real-time data
13662	Fixed	Control Center	Email successfully sent message was marked as an error rather than an alert
13671	Fixed	CQG, SuperDOM	Instruments with no volume could not scroll down in the SuperDOM
13610	Fixed	DrawingTool	Calling draw tools which use PriceLevels with templates from compiled assembly could fail to get price levels from template
13529	Fixed	FXCM, Orders	Resolved a scenario where positions did not update
13552	Fixed	G2, Orders	Resolved a scenario where past orders would be logged again unexpectedly
13600	Fixed	Indicator	Drawing Tool Tile did not display the Path icon correctly
13612	Fixed	Indicator	Swing Input unexpectedly could be used when BarsArray was passed as an input
13558	Added	Instruments	Added tick size of 0.00000001
13520	Fixed	Instruments	Instrument selector could show CFD and index prefix unexpectedly

13555	Fixed	Instruments	Improved instrument type search filter to better give the results expected by default
13665	Fixed	Interactive Brokers, Market Analyzer	Some instruments would not receive fundamental data that was expected
13586	Fixed	Kinetick	Historical data requests for the ZT could result in an error
13516	Fixed	Market Analyzer	Blank row dividers for chart columns unexpectedly had less opacity when selected
13615	Changed	Market Analyzer	Net Change did not display fractional prices for points display mode
13618	Fixed	Market Analyzer	Adding an indicator column then changing the indicators property caused the name to unexpectedly change
13653	Fixed	Market Analyzer	Date columns had an error when value did not exist
13660	Fixed	Market Analyzer	Resolved a scenario where columns could show '…' unexpectedly
13587	Fixed	Market Analyzer, Alerts	Alerts that are checked against an indicator value and a numeric value did not rearm on reversal of the condition
13511	Fixed	NinjaScript	Strategies with a secondary series prevented Strategies tab Filter Only Active Strategies option from being disabled
13605	Fixed	NinjaScript	Resolved a scenario where a valid NinjaScript import file failed to import
13633	Fixed	NinjaScript	Strategy indicator could get stuck in calculating after dragging and dropping an

			indicator
13549	Fixed	NinjaScript Editor	Renaming file did not rename the name property
13638	Fixed	NinjaScript, ATM Strategies	AtmStrategyCreate could submit orders to non-primary series despite BarsInProgress check
13551	Changed	Order Flow +	Changed Order Flow Volume Profile tick resolution error text to be more clear
13603	Fixed	Order Flow +	Order Flow Trade Detector could throw an error when hovering over a marker
13616	Fixed	Order Flow +	Selecting Order Flow Volume Profile resulted in an error on DST switch
13507	Fixed	Playback	Resolved a scenario where historical data could be removed that was older than the Playbacks start date
13581	Fixed	Playback, Chart	Switching trading hours could result in opening bar to display incorrectly
13601	Fixed	Playback, DrawingTool	Resolved a scenario where drawing objects could be removed when opening/closing workspaces
13641	Fixed	Playback, DrawingTool	Global drawing objects drawn on historical bars while connected to live data could be removed from closed workspace when you connect/play playback data
13569	Fixed	Playback, Strategy	Strategy could not enable on playback after terminated
13571	Fixed	Playback, Strategy	Rewinding playback could duplicate strategy added chart indicators

13623	Changed	ShareAdapter	Default SMTP config settings for Yahoo were updated
13562	Fixed	Skins, Strategy Builder	When using the dark skin, strategies listed in Strategy Builder were unreadable
13532	Fixed	Strategy	Changing the time-frame for multiple active strategy instances could leave parts of the instances disabled
13547	Fixed	Strategy	Date field remained active after strategy was enabled
13567	Fixed	Strategy	SetParabolicStop could move the order to the wrong side of the market resulting in an error
13585	Fixed	Strategy	Resolved a scenario where reloading NinjaScripts resulted in duplicates of strategy indicators
13588	Fixed	Strategy	Resolved a scenario where enabled strategies showed as not enabled
13564	Fixed	Strategy Analyzer	False date picker error could occur after clicking reset through the templates.
13647	Fixed	Strategy Analyzer	Strategy Analyzer optimization results tool tip could display in an unexpected format
13650	Fixed	Strategy Analyzer	When using the strategy optimizer if you closed a secondary window while optimizing it would result in a crash
13506	Changed	Strategy Builder	Improved performance for compiling or opening scripts
13513	Fixed	Strategy Builder	Cancel button could trigger program unresponsive pop-up for large strategy

13576	Fixed	Strategy Builder	Condition builder embedded indicator parameters would not visually update
13628	Fixed	Strategy Builder	An error could occurred when adding an indicator with simple font
13664	Fixed	Strategy Builder	Checking for first bar of the session resulted in an unneeded check
13554	Fixed	Strategy, Chart	Strategy hosted indicator regions could disappear on reload
13548	Fixed	Strategy, Workspaces	Strategies still showed on chart of workspace when the strategy was saved to a new workspace
13542	Fixed	SuperDOM	Could show more indicator lines than expected
13544	Fixed	SuperDOM	Adding two instances of Bollinger resulted in an error
13570	Fixed	SuperDOM	Column configuration lost selection upon applying changes
13579	Fixed	SuperDOM	Indicator Instances duplicated when changing calculate settings
13606	Fixed	SuperDOM	Resolved a scenario where the previous ATM template was retained unexpectedly
13608	Fixed	SuperDOM	An unexpected error could occur when applying an indicator the license was not enabled for
13667	Fixed	SuperDOM, Licensing	An ATM template could stick if the workspace was closed then the license key was switch from live to free edition
13578	Fixed	SuperDOM, Playback	Resolved some scenarios where the SuperDOM did not display as expected while Playback was paused

13489	Fixed	TD Ameritrade, Orders	In some scenarios the position size was reported incorrectly
13580	Fixed	Time and Sales	A Time and Sales which was open prior to a disconnect/reconnect, upon reconnect only printed as above ask
13640	Fixed	Trade Performance	Avg MAE on Summary tab was missing trade size adjustment when summary displayed in pips
13651	Fixed	Trade Performance	Trade Performance presets did not load with new window
13599	Fixed	UI	Fixed message box text for instrument roll over
13630	Fixed	UI, Controle Center	Preferred connection real-time default selection was missing for Cryptocurrency
13636	Fixed	UI, Indicators	Resolved a scenario where the indicator properties window could open partially out of view
13550	Fixed	UI, ATM Strategies	Custom stop strategy tool tip text was missing
13656	Fixed	Workspaces	Switching workspaces that had a databox could result in a freeze
13557	Fixed	Workspaces	Resolved a scenario where 'Save Workspace' confirmation dialog did not work as expected
13658	Fixed	Workspaces	Resolved a scenario where saving and closing workspaces with defaults names could open an unexpected workspace

8.0.18.1 Release Date

June 4, 2019

Issue#	Status	Category	Comments
13741	Fixed	Commissions, Risk	Risk and Commissions templates did not update on start up
13742	Fixed	FXCM, Position Display	Resolved a scenario where account could fall out of sync from live account
13739	Fixed	Interactive Brokers, Data	Real time ^TICK data only updated in positive values
13745	Fixed	Reginalization, TopStep Trader	Connection name translated poorly when platform language was set to Spanish
13744	Fixed	Strategy Analyzer	Could not open optimization result in new tab/window

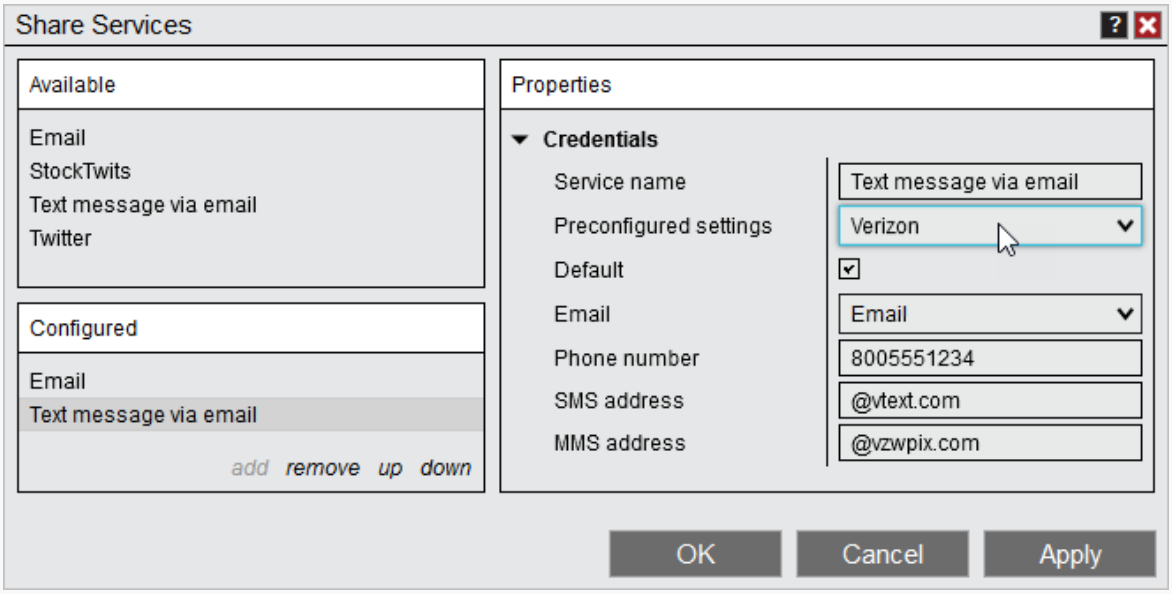
3.3.12 8.0.17.2

8.0.17.0 Release Date
January 28, 2019

Features
Added Drawing tool tile indicator Indicators Feature # 13389 The Drawing tool tile indicator adds the ability to have a floating tile in the chart that can be customized to quickly access the most commonly used drawing tools.

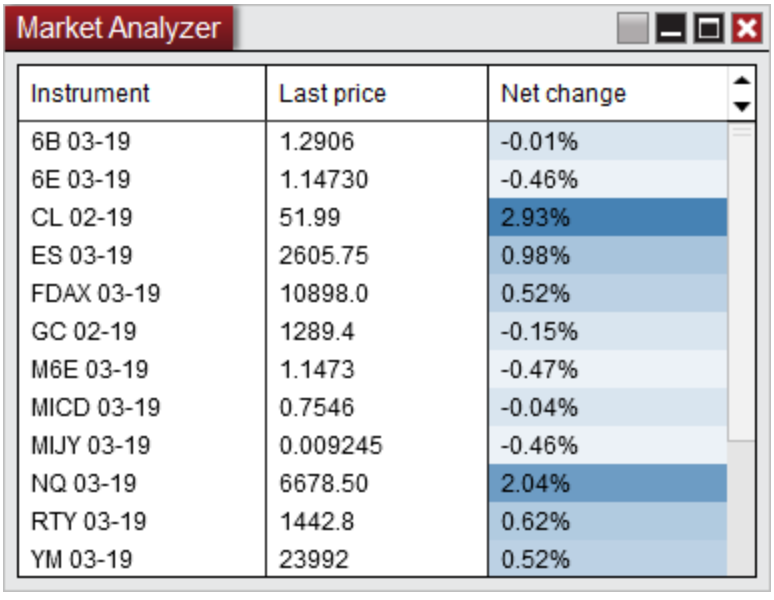
Added Text message via email Share Adapter
Options
Feature # 13131

The Text message via email feature come preconfigured with typical setting for most major mobile phone providers so you can share item to your mobile phone with SMS or MMS messages.



Added Enable color distribution column property
Market Analyzer
Feature # 13193

This column property will function like a heat map to allow easy identification of where values fall on a scale or compared to one another by distributing a range of colors.

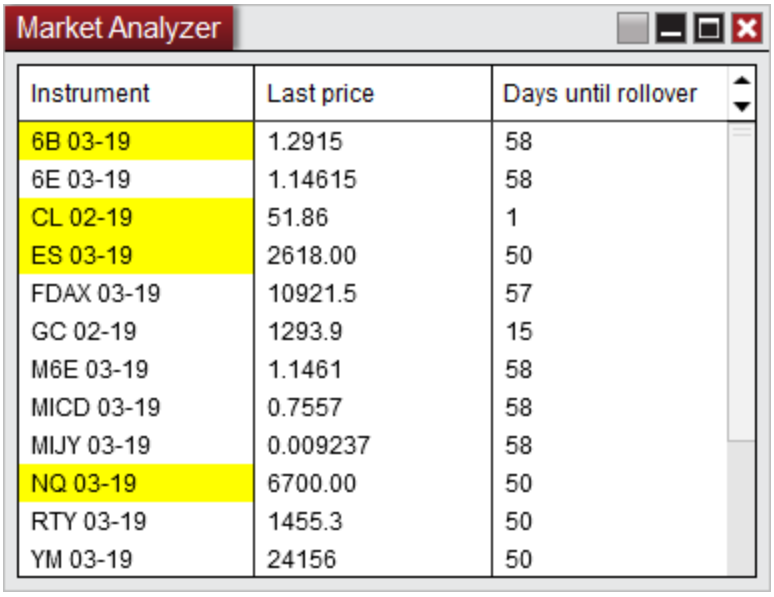


Added Go to... ability to chart scroll-bar right click menu
Chart
Feature # 13367

The Go to... feature will enable you to quickly move to the specified time on a chart.

Added a Days until rollover column
Market Analyzer
Feature # 13293

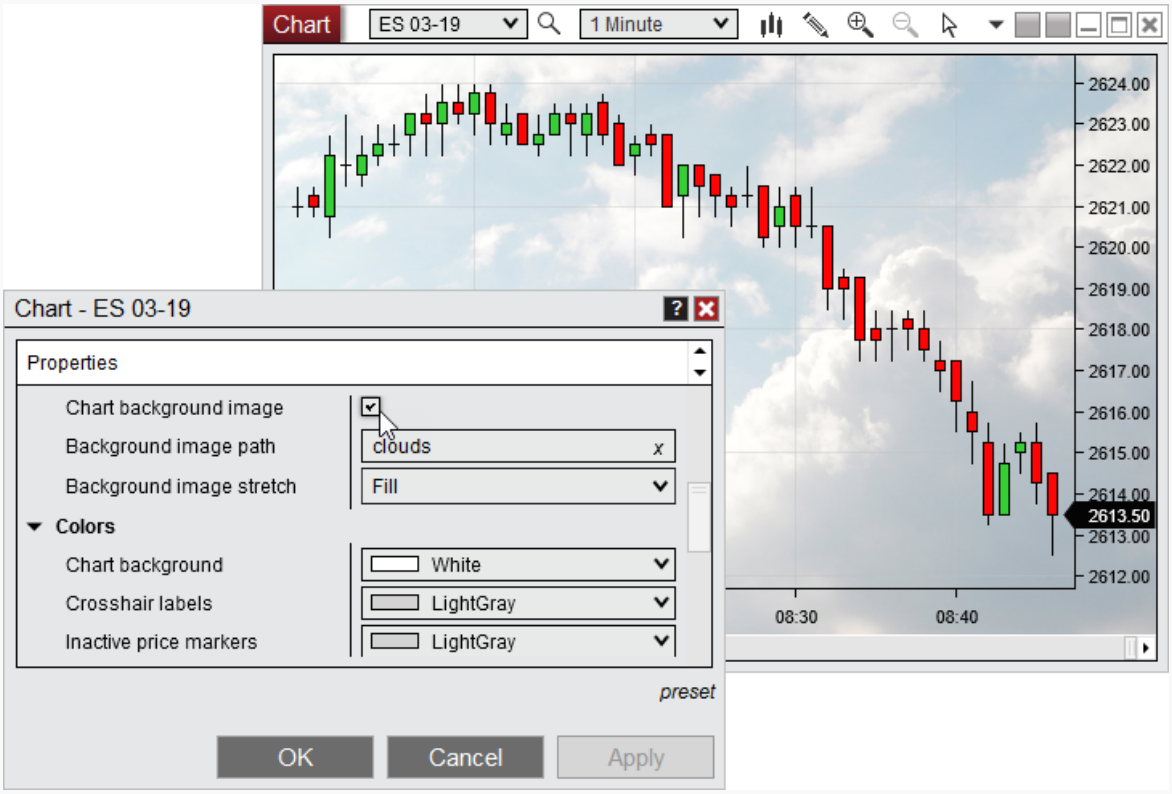
This column creates an easy visualization of when futures are set to rollover and can be used with an alert to notify when it's time to rollover.



Instrument	Last price	Days until rollover
6B 03-19	1.2915	58
6E 03-19	1.14615	58
CL 02-19	51.86	1
ES 03-19	2618.00	50
FDAX 03-19	10921.5	57
GC 02-19	1293.9	15
M6E 03-19	1.1461	58
MICD 03-19	0.7557	58
MIJY 03-19	0.009237	58
NQ 03-19	6700.00	50
RTY 03-19	1455.3	50
YM 03-19	24156	50

Added a property to set an image as a background for the chart
Chart
Feature # 13240

A custom background image can now be applied to charts to help easily differentiate charts or enhance the visual appeal.

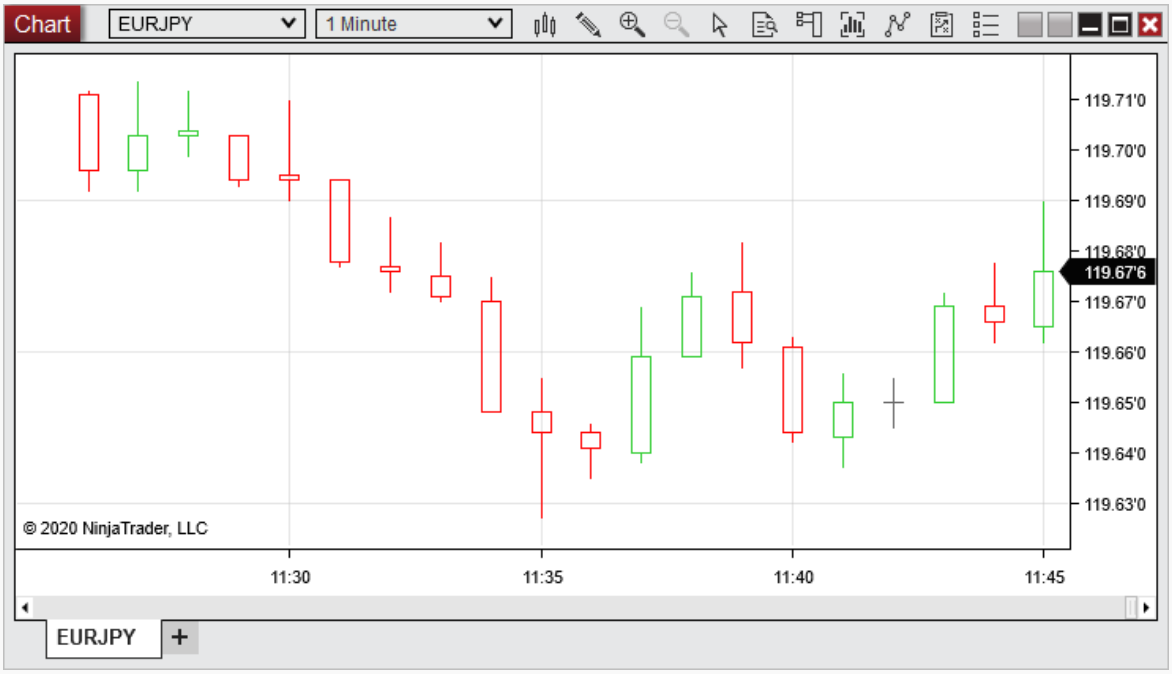


Added ability to duplicate a window, including all it's tabs
Core
Feature # 13399

The ability to duplicate an entire window and all it's tabs greatly reduces the time to recreate a window's set up.

Added Hollow Candlestick Chart style
Chart
Feature # 13435

Hollow candlesticks can color the outline/wicks the up/down bar colors and has a separate color for doji bars, all with a transparent center.

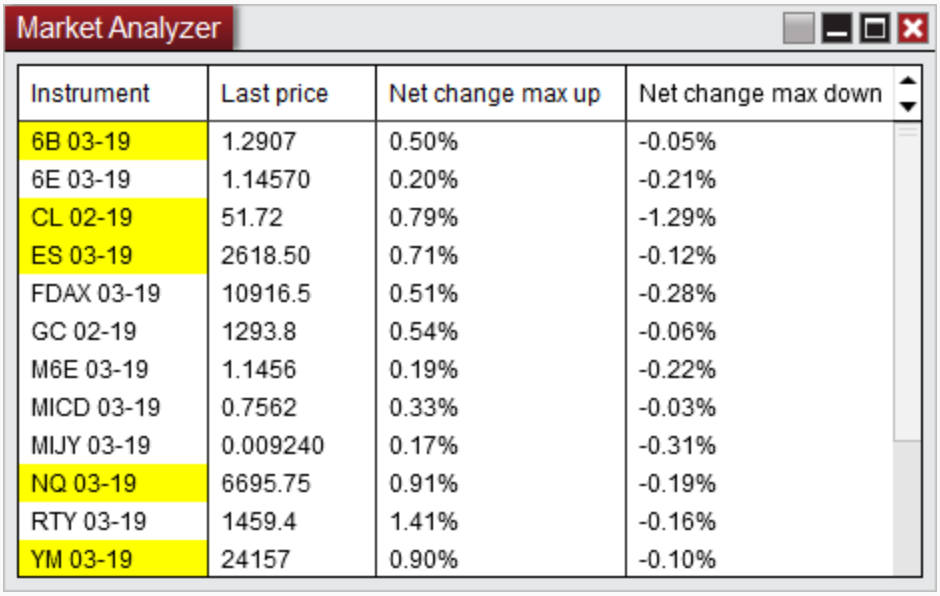


Added 'Path' Drawing Tool
DrawingTool
Feature # 13270

The path drawing tool adds the ability to plot multiple connecting lines without it needing to be a shape.

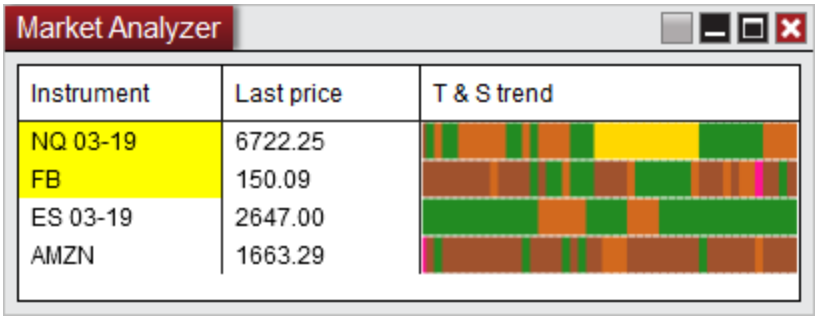
Added Net change max up and down columns
Market Analyzer
Feature # 13094

These columns will display the max and min the net change has been for the session.



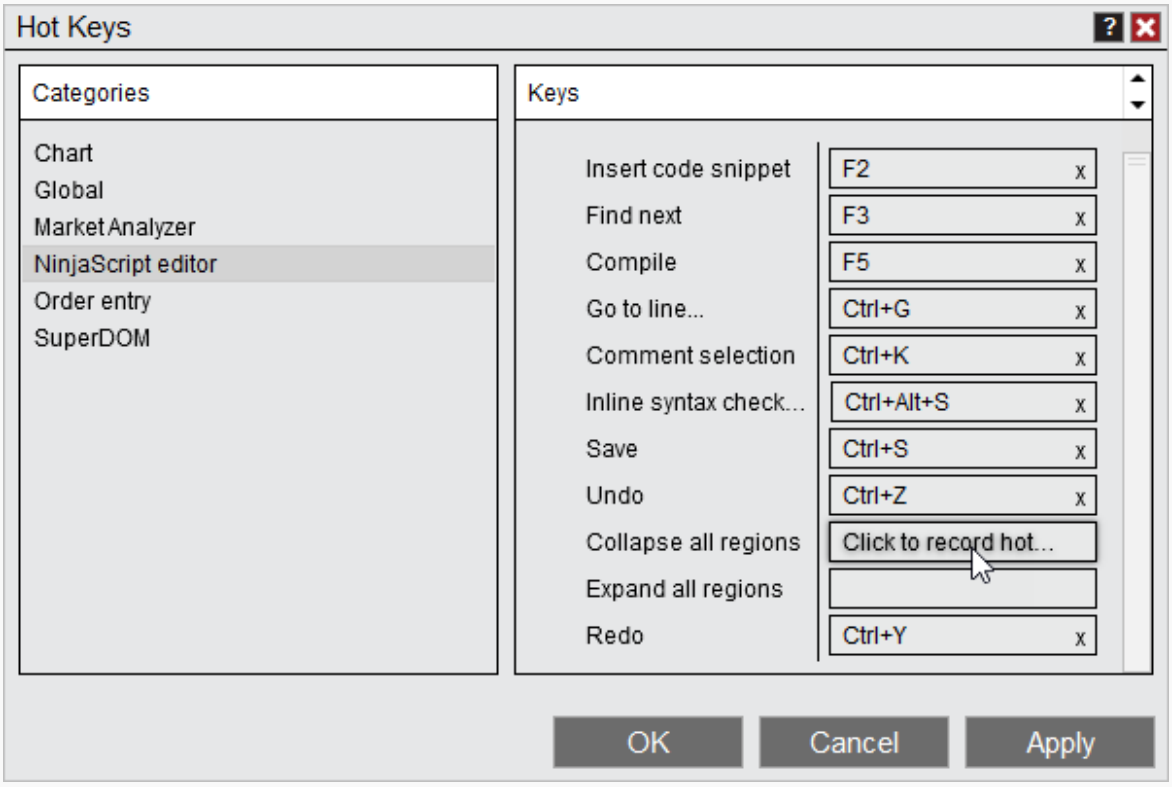
Added T & S trend column
Market Analyzer
Feature # 13062

This column will show a colored display indicating a history of where orders filled in comparison to the bid and ask price



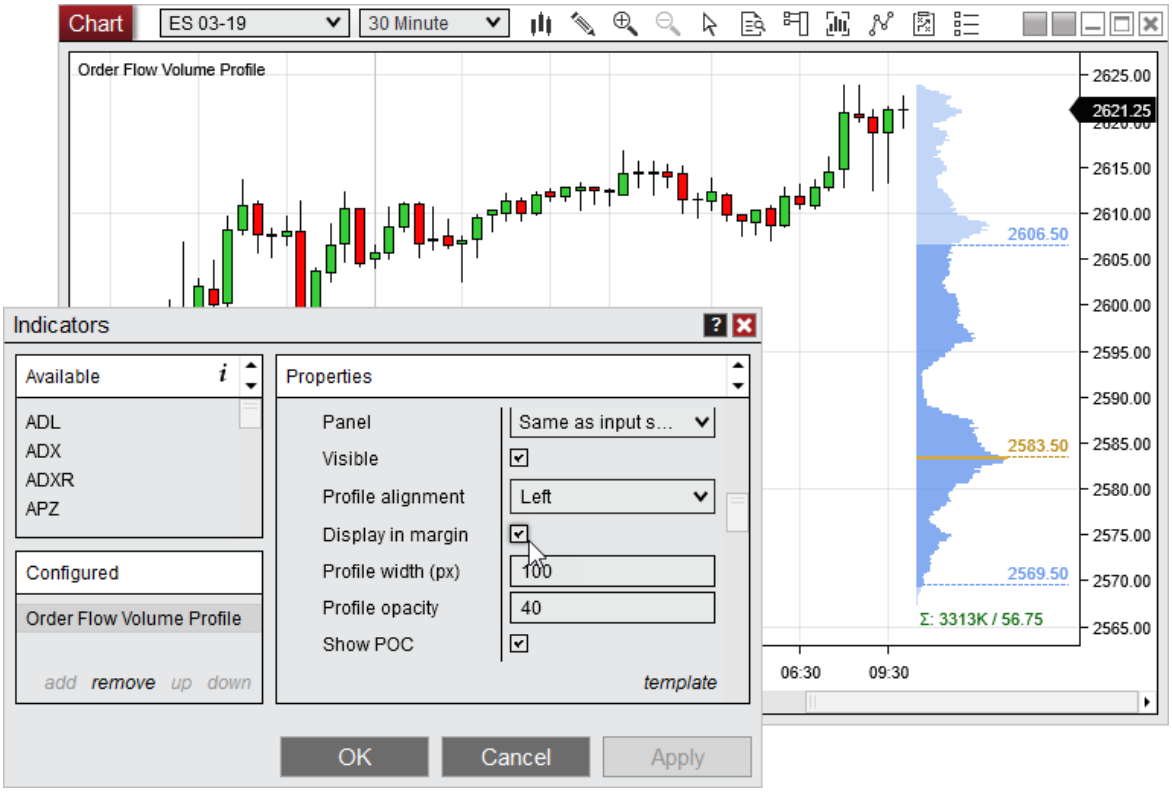
Added Collapse all regions & Expand all regions hot keys for the NinjaScript Editor
Hot Key, NinjaScript Editor
Feature # 13255

With these hot keys you can quickly expand and collapse all regions to make it easier getting to what you need.



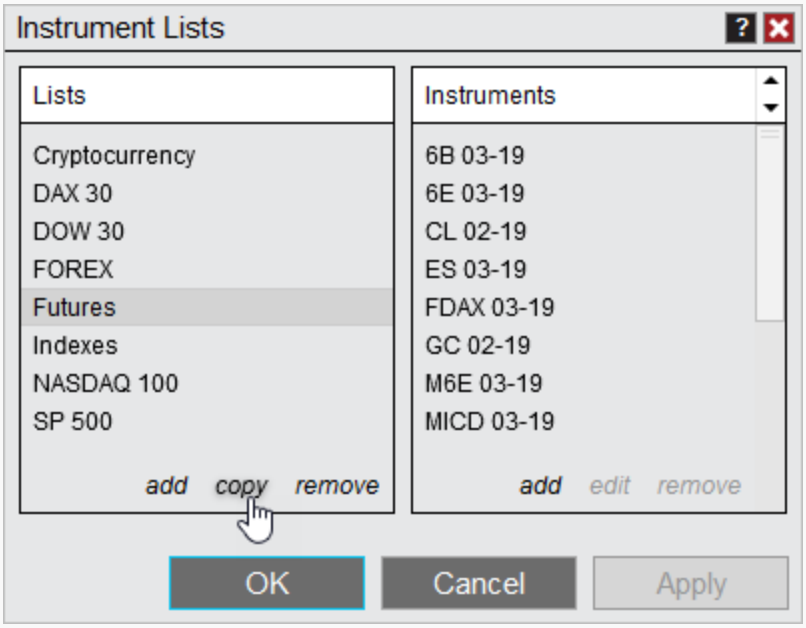
Order Flow Volume Profile in composite mode now has Display in margin option
Order Flow +
Feature # 13316

With the Display in margin option you will be able to plot the profile without overlapping the bars when using composite mode.



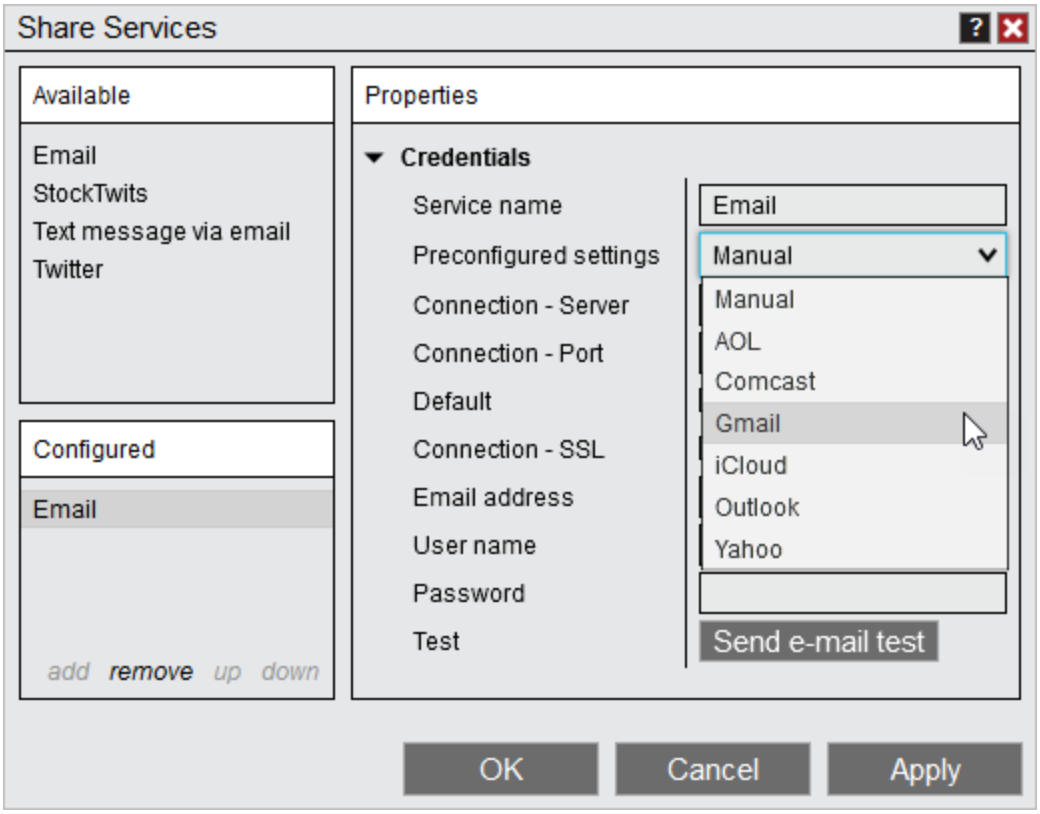
Added the ability to copy an instrument list
Instruments
Feature # 13199

Copying an instrument list can be used to quickly duplicate an existing list that you may want to make some adjustments to for a new list.



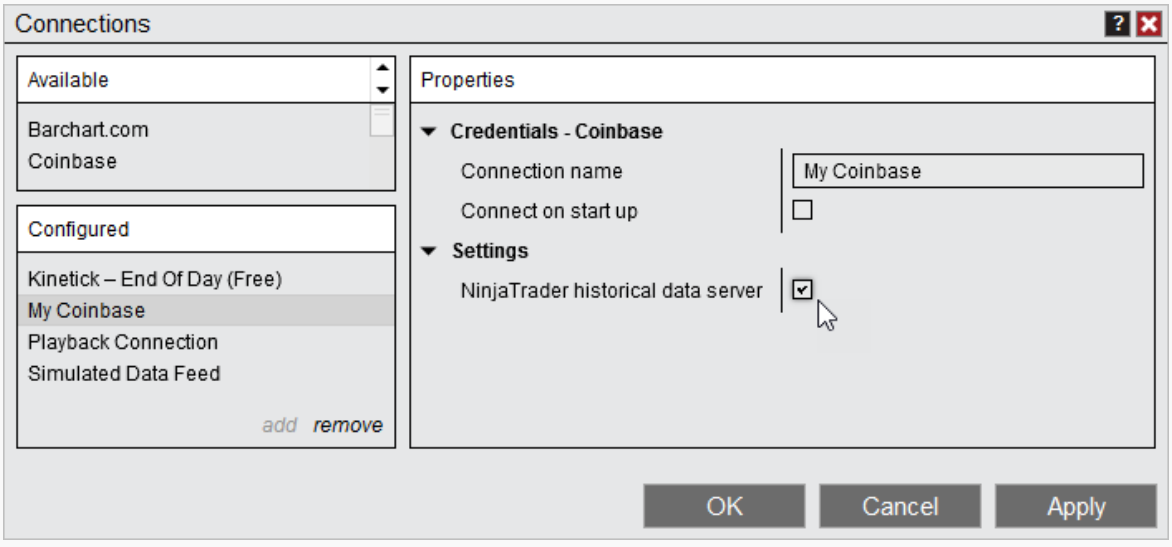
Share email adapter is preconfigured with SMTP settings for common providers
Options
Feature # 13128

The email share adapted now includes common settings for the most popular providers, making adding a provider even easier.



Added historical data support from NinjaTrader servers for Coinbase
Coinbase
Feature # 13410

With NinjaTrader servers you will be able to access historical data faster and get tick data.



Added support for XAGUSD/XAUUSD commodity CFD's
Interactive Brokers, Instruments
Feature # 13329

Although commodities are not supported, these popular instruments can now be accessed as CFDs and will properly route to the commodity CFD in Traders Workstation.

Added support for distributing NinjaScripts that include workspace files
NinjaScript
13299

Similar to how templates can be included with NinjaScript packages, now workspaces could be included.

Added SizeFilter overload for Volumetric bars
Order Flow +, NinjaScript
Feature # 13274

Now the SizeFilter for Volumetric bars will be accessible with NinjaScript.

Added Italian language support
Regionalization

Feature # 13391

You can now configure NinjaTrader for Italian

Issue #	Status	Category	Comments
13204	Fixed	Alerts	Percent offsets did not work as expected
13327	Fixed	Alerts	Arithmetic decimal offsets were not allowed
13442	Fixed	Alerts	Rearm of On condition reversed was not working as expected when there were multiple conditions
13404	Fixed	Attach Order To Indicator	Increasing the quantity of an order attached to an indicator did not inherent attached function
13217	Fixed	Bars	Resolved some scenarios where loading bars could result in an error
13371	Fixed	Basic Entry	Price selector lost fractional value when deselected
13209	Changed	Chart	Send to > Chart feature now sends to the Data Series so the desired chart type/template can be selected
13233	Fixed	Chart	Chart templates/duplicate chart tab did not retain fixed left scale of indicator
13394	Fixed	Chart	In some scenarios the chart style icon could be incorrect

13397	Fixed	Chart	There was no visual cue on the Global Cross Hair which would indicate which chart the mouse cursor was hovering over
13491	Fixed	Chart, NinjaScript	Resolved a scenario where custom bar types wouldn't properly scale
13323	Fixed	Chart Trader	Resolved some scenarios where Chart Trader could be re-sized incorrectly
13260	Fixed	Chart Trader, Indicator	FXTile panel could be overlapped by Chart Trader panel
13280	Fixed	Chart, DrawingTool	Switching data series and then removing all drawing objects from a chart could affect the previous data series
13387	Fixed	Chart, Indicator	Adding an indicator while NinjaScript was reloading could result in an error
13286	Fixed	Chart, Strategy	Strategy could plot on incorrect input series on multi-series chart
13269	Fixed	Coinbase	Resolved a scenario that resulted in repeated disconnects and reconnects
13275	Fixed	Coinbase	Resolved a scenario that prevented download of more than a week of minute data
13336	Fixed	Coinbase	Daylight savings time could prevented data from downloading
13460	Fixed	Coinbase	Historical daily bars were time stamped 1 day off

13334	Fixed	Continuum, CQG	Resolved a scenario where a time zone setting prevented connecting during daylight savings time change
13259	Changed	Control Center	Made Send to support window not modal and added CC to self option
13284	Fixed	Control Center	Email support could say the message sent when it did not
13314	Changed	Control Center	Updated format of the Help menu
13258	Fixed	Control Center, NinjaScript	When more scripts were exported than the size of the window there was no scroll-bar to see the additional scripts
13313	Fixed	Core	When having a modal window open then switching programs then going back to NinjaTrader, the modal window would not pull into the front
13431	Fixed	Core	Made tool tips more consistent
13283	Fixed	DrawingTool	Resolved a scenario where Region highlight could result in an error
13326	Fixed	DrawingTool, Workspace	Drawing Object Attached to All charts with Visible unchecked did not retain unchecked setting on relaunch

13290	Changed	Forex.com	Updated connection options to always use G2 as that is now the only option from Forex.com
13297	Fixed	Forex.com	Resolved a scenario that where a connection loss occurred resulting in a crash
13297	Fixed	Forex.com, Data	When connecting a historical gap could appear on chart if using a chart series preset
13306	Fixed	Forex.com, Chart	After loading a tick chart then switching to another chart type the Loading text would not remove once complete
13186	Fixed	Forex.com, NinjaScript	Resolved a scenario where canceled simulation orders could result in an error
13243	Fixed	FX Board	Futures instruments added to the Forex list would get added to the FX Board tiles collection
13408	Fixed	FX Pro	If no instrument was selected, open orders would show on the orders grid
13224	Fixed	FXCM	Resolved a case where some accounts could not connect
13298	Fixed	FXCM, Orders	Orders could be slower than with previous releases
13252	Fixed	Indicator	ZigZag could not run on an input series that produced negative values
13388	Fixed	Indicator	Candlestick indicator had an error if strength was set to 0

13403	Fixed	Indicator	Typing to select indicator in available list didn't always work
13423	Fixed	Indicator	FX Tile error text flashed on while changing to valid instrument
13433	Fixed	Indicator	Added text indicating Woodie's CCI panel requires minimum 3 daily bars to calculate if less than 3 are available
13462	Fixed	Indicator, Chart	Indicator with IsOverlay=False and DrawOnPricePanel=True would draw objects in indicator panel rather than price panel
13493	Fixed	Installer	Reinstall did not recompile as expected
13487	Fixed	Installer	A large cache could cause upgrade to take long
13402	Changed	Instruments	Addition of 0.000005 as tick size
13440	Fixed	Instruments, NinjaScript	Resolved a scenario where updating instruments quickly after rolling over futures resulted in a error
13386	Changed	Interactive Brokers	Notification from Traders Workstation no longer send as alerts or errors, which resulted in pop ups
13422	Fixed	Interactive Brokers	Resolved a scenario that prevented connecting
13484	Change	Interactive	For equities, if a market maker ID is not received the selected exchange will display as

	d	Brokers, Level II Window	the market maker
13468	Fixed	Interactive Brokers, Instruments	YM 03-19 showed as YM 12-99
13502	Fixed	Interactive Brokers, Instruments	VX 03-19 failed to load
13425	Fixed	Interactive Brokers, Orders	Resolved a scenario where part filled logs were duplicated
13056	Changed	IQFeed	Supported Client Software version updated to 6.0.1.1
13483	Fixed	IQFeed	A disconnect then reconnect could result in data not resubscribing when using multiple connections
13300	Fixed	Kinetick	Resolved a scenario were a reconnect was not attempted after a lost connection
13481	Fixed	Kinetick	Disconnecting during a connection loss could result in an error

13236	Fixed	Kinetick, Workspaces	Connecting to Kinetick EOD from the 'Get connected' screen would load the default workspace after each open
13227	Fixed	Licensing	Resolved a scenario where settings didn't fully update when changing licenses
13333	Fixed	Licensing	Vendor licensing allowed blank name which couldn't be managed
13369	Changed	Localization	Refined Simplified Chinese Mandarin localization
13405	Fixed	Localization, Instruments	Instrument type in instrument manager could display incorrectly
13449	Fixed	Localization, Log	English logs could show localized resources
13234	Fixed	Market Analyzer	Label cell did not work properly after drag and drop when non-text columns were applied
13291	Fixed	Market Analyzer	Resolved some rendering errors for Chart - Net change
13467	Fixed	Market Analyzer, NinjaScript	Resolved a scenario where a custom column was not terminated after the window was closed

13475	Fixed	Market Analyzer, Playback	Chart - Net Change did not plot when connected to Playback
13365	Fixed	NinjaScript	Resolved a scenario where accessing the Strategies window repeatedly resulted in an error
13418	Fixed	NinjaScript	BarsRequest could return incorrect bars if Global Merge Policy was different from BarsRequest Merge Policy
13428	Changed	NinjaScript	Monte Carlo simulation is now multi-threaded
13231	Fixed	NinjaScript Editor	Resolved links for compile errors that went to pages with no documentation
13235	Fixed	NinjaScript, Chart	Resolved a scenario where auto scaling a chart could result in an error
13488	Fixed	NinjaScript, Chart	Draw.TextFixed in sub panel moved on time axis scroll
13096	Fixed	NinjaScript, DrawingTool	Updating the end anchor from NinjaScript could fail on multi-series chart
13432	Fixed	NinjaScript, DrawingTool	Resolved a scenario where rendering a rectangle resulted in an error

13497	Fixed	NinjaScript, Workspaces	When restoring a workspace, OnWindowRestored() was called before OnWindowCreated
13441	Fixed	Options, Data	Tick filtering did not work as expected
13245	Changed	Order Flow +	Updated Order Flow Market Depth Map default settings to improve visualization
13261	Fixed	Order Flow +	Order Flow Volume Profile on a tick chart could cause a rendering error
13263	Fixed	Order Flow +	In some scenarios the Order Flow Volume Profile's Extended naked POC extended when it shouldn't have
13276	Fixed	Order Flow +	Order Flow Volume Profile type price plotted some letters too early on minute resolution
13277	Fixed	Order Flow +	Order Flow+ Volume Profile Composite did not plot with certain settings
13289	Fixed	Order Flow +	Order Flow Volume Profile added addition line labels
13308	Fixed	Order Flow +	Order Flow Volume Profile could plot monthly profiles a day early
13381	Fixed	Order Flow +	In some scenarios Order Flow Volume Profile price charts could show the C for close out of line

13400	Fixed	Order Flow +	Order Flow Market Depth Map extend last known value printed incorrectly in fast moving markets
13470	Fixed	Order Flow +	Volumetric bars could return different values in code vs chart if a gap in price occurred
13406	Fixed	Order Flow +, Chart	Resolved a scenario where chart became unresponsive after switching the instrument
13430	Fixed	Order Flow +, Playback	Order Flow Cumulative Delta could see discrepancies when reloaded in Playback
13377	Fixed	Order Flow +, Tick Replay	Order Flow + Volume Profile indicator plot was displaced when used with Tick Replay
13364	Fixed	Orders	Orders tab would not update Quantity when a strategy order was submitted to secondary series
13420	Fixed	Orders	GTD orders were not handled as expected
13376	Changed	Output Window	Added Copy to right click menu of the NinjaScript Output window
13287	Fixed	Playback	Rewinding Playback with an alert attached to a multi-series indicator resulted in an error

13455	Fixed	Playback	Playback could get stuck when daylight savings ends with some time zones
13498	Fixed	Playback	Controller end date could be before start date in some scenarios
13215	Fixed	Simulator, Trade Performance	Simulator MAE/MFE/ETD could change after end of day
13393	Fixed	Strategy	In some scenarios draw methods would not plot if the strategy was applied in a chart then enabled in the Control Center
13417	Fixed	Strategy	Resolved a scenario that resulted in a double canceling of an order which then caused a crash
13451	Fixed	Strategy	When running 2 strategies on the same instrument, if 1 was disabled the 2nd stopped updating it's PnL values on the Strategies tab
13463	Fixed	Strategy	Calling EnterLong() and then ExitLongLimit() could result in an error
13192	Fixed	Strategy Analyzer	When loading an Optimization from a log, Summary results did not match the Results grid, when using High Order Fill Resolution
13218	Fixed	Strategy Analyzer	Sending a log from a backtest to an already open NinjaScript Editor resulted in an error
13324	Fixed	Strategy Analyzer	Adding Results Columns to an Optimization caused column misalignment

13379	Fixed	Strategy Analyzer	Walk forward optimization Start/End column disappeared when setting display to points
13409	Fixed	Strategy Analyzer	When switching between multiple instruments results of a strategy that draws text an error could occur
13411	Fixed	Strategy Builder	Setting VolumeUpDown as Input in Momentum caused a duplicate instantiation
13494	Fixed	Strategy Builder	Condition Builder showed some indicators that are not available in the Condition Builder
13303	Fixed	Strategy, Orders	Resolved a scenario where specific settings resulted a simulated stop became a real stop when the strategy transitioned to real time
13295	Fixed	Strategy, Templates	Running an Optimization then setting a template was not working as expected
13448	Fixed	Strategy, Templates	Strategies applied on the Strategies tab of the Control Center could change renko brick size when enabled, if a strategy template exists
13282	Fixed	SuperDOM	Column width was not maintained after a restart
13474	Fixed	SuperDOM, ATM Strategies	Resolved a scenario where the APQ column could get an error if using an ATM in a fast moving market

13250	Fixed	SuperDOM, NinjaScript	Resolved a scenario where reloading NinjaScripts could get stuck
13288	Fixed	TD Ameritrade	Resolved a scenario where an additional day of data would download unexpectedly
13472	Fixed	TD Ameritrade, Chart	Some time zones could prevent charts from updating
13419	Fixed	Trade Performance	Performance analysis grid on weekly period during holiday could display incorrectly
13256	Fixed	Workspaces	Switching between workspaces that had a data box caused the data box to become blank
13438	Fixed	Workspaces	Resolved a scenario where saving a workspace with same name as a previously saved workspace, they were combined

8.0.17.1 Release Date
February 6, 2019

Issue #	Status	Category	Comments
13533	Fixed	Order Flow +	Order Flow Market Depth Map did not plot above last price
13522	Fixed	NinjaScript, Drawing Tool	Renamed Path NinjaScript access to PathTool to prevent potential compile conflicts

13537	Fixed	Localization	When NinjaTrader was set to 'Spanish', mail to support did not work as intended
13546	Fixed	NinjaScript	References to an older version of Newtonsoft would result in an error

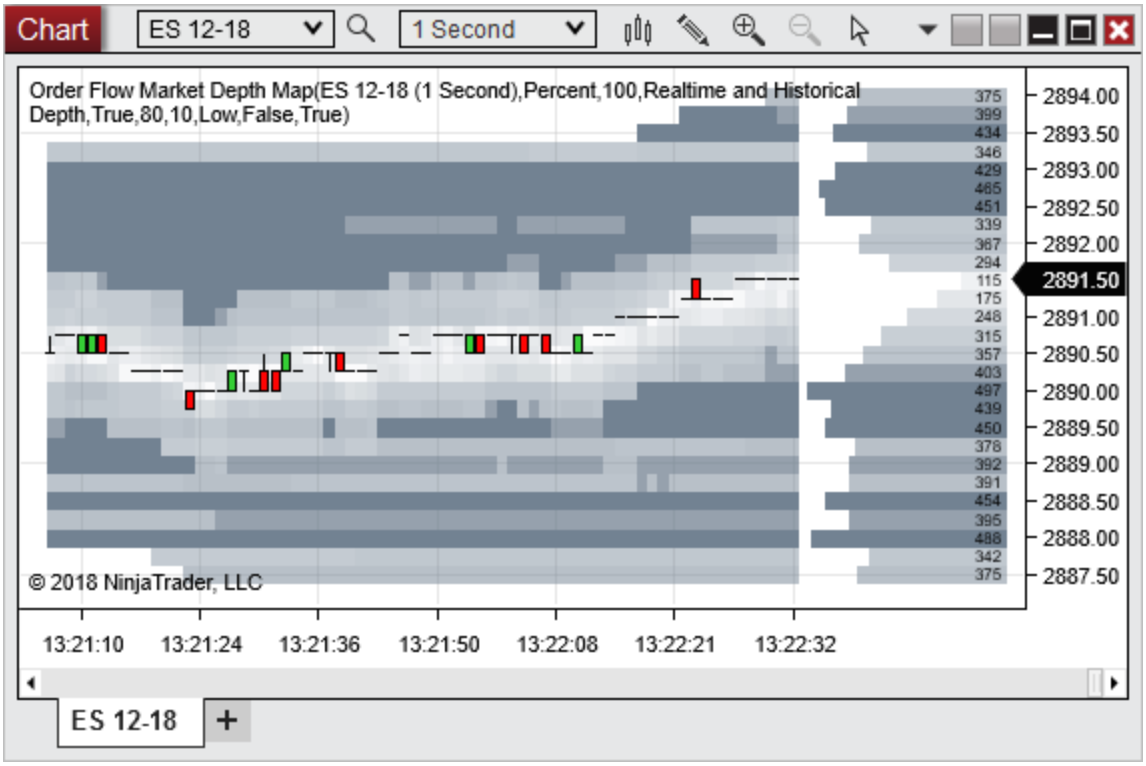
8.0.17.2 Release Date
February 14, 2019

Issue #	Status	Category	Comments
13560	Fixed	Control Center, Playback	Resolved account handling for a few scenarios
13595	Changed	Drawing Tool	Improved Path tool double click to end function

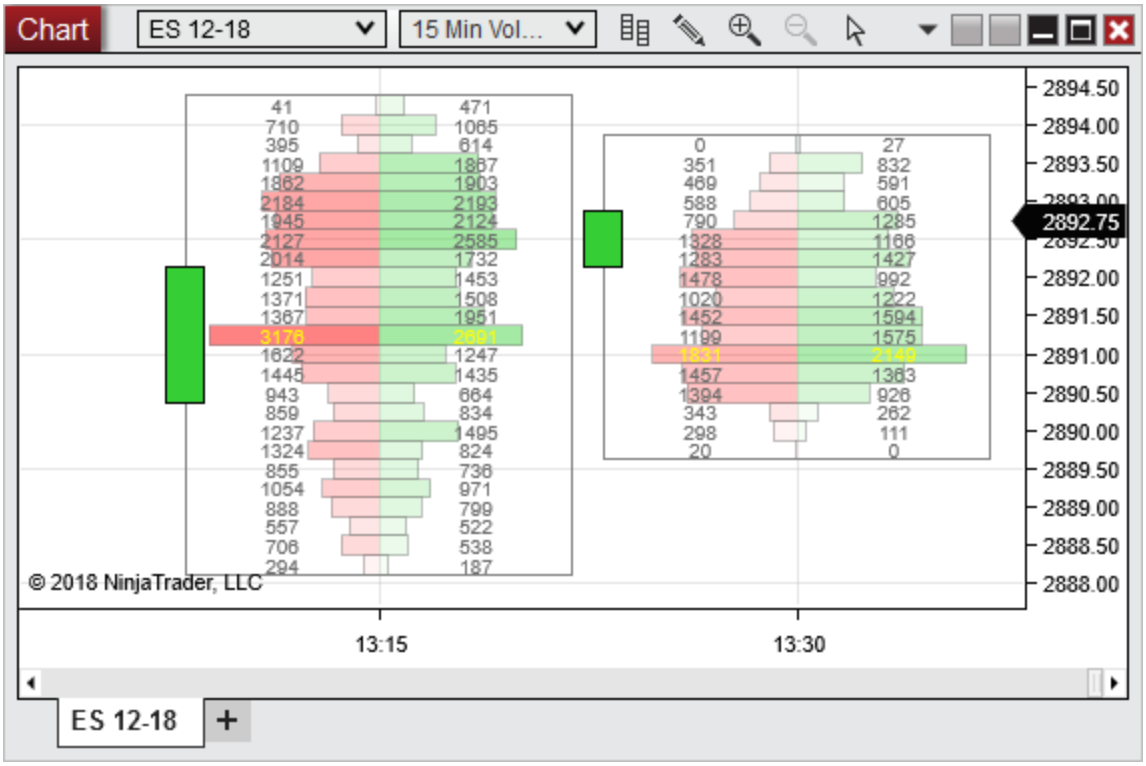
3.3.13 8.0.16.3

8.0.16.0 Release Date
October 16, 2018

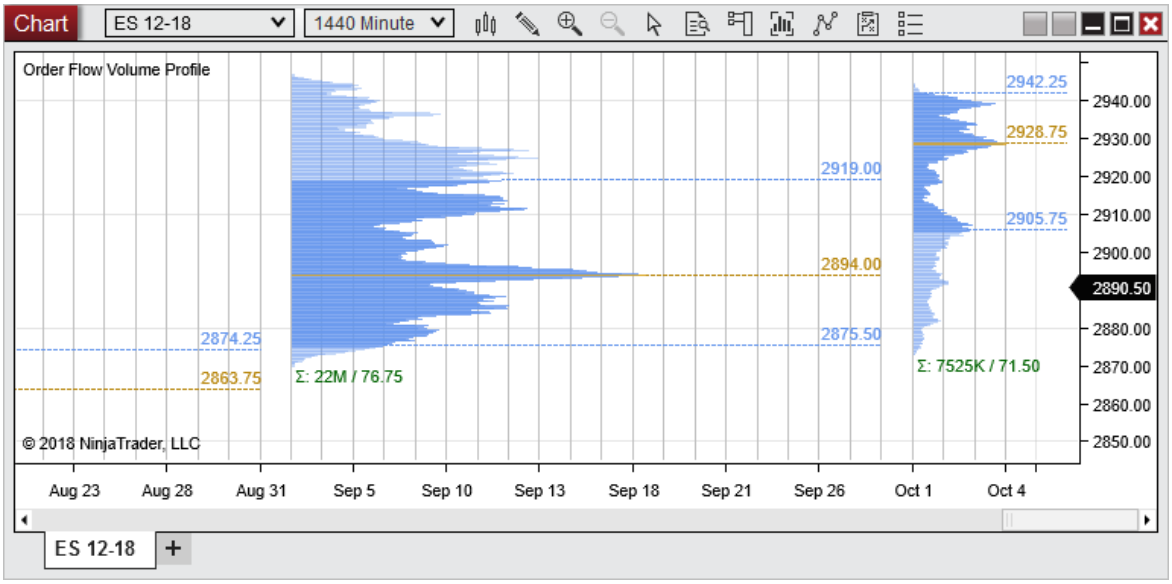
Features
Added additional features to Order Flow + items Order Flow + Feature # 13084 The Order Flow Depth Map now has a real-time display that graphs the highest depth volume seen at each price level for the bar type it is applied to (1 second in the example below). This creates a simple visualization of where the largest depth is.



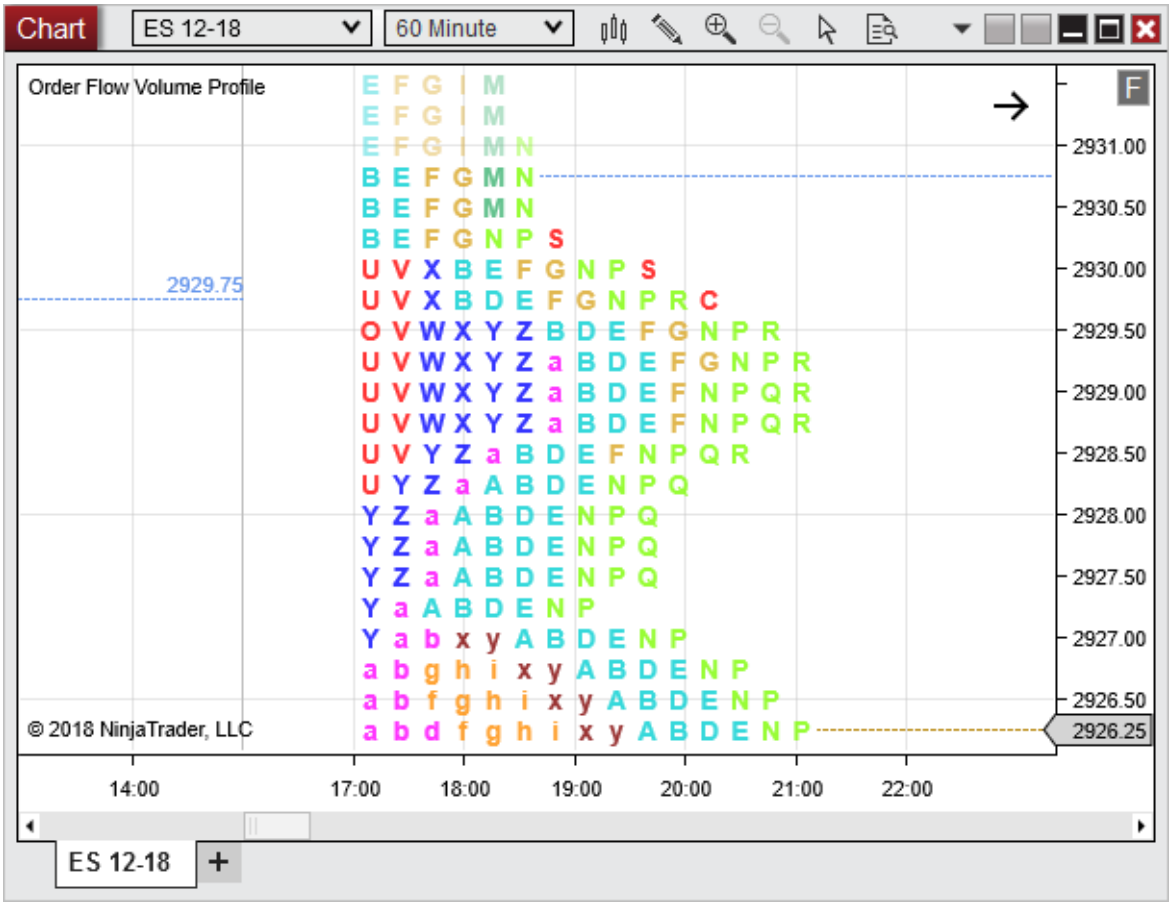
Volumetric bars now have a profile view to easily identify where the largest volumes occurred.



Order Flow Volume Profile using large composite modes such as month will build from the start of the month rather than a month back from the current date.



Order Flow Volume Profile with a Price Profile Type will now show an O and a C to represent the open and close.

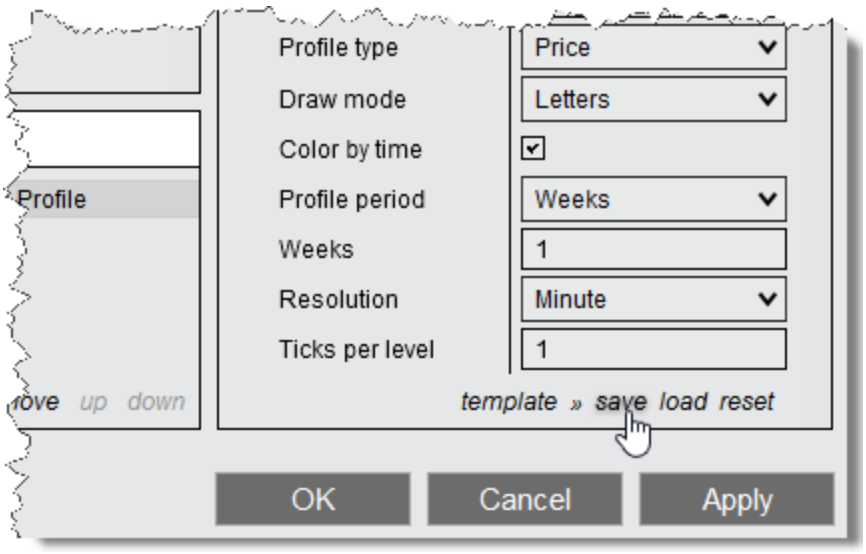


Added the ability to resize the Chart Trader
Chart Trader
Feature #13132

Now the Chart Trader size can be adjusted to your preference.

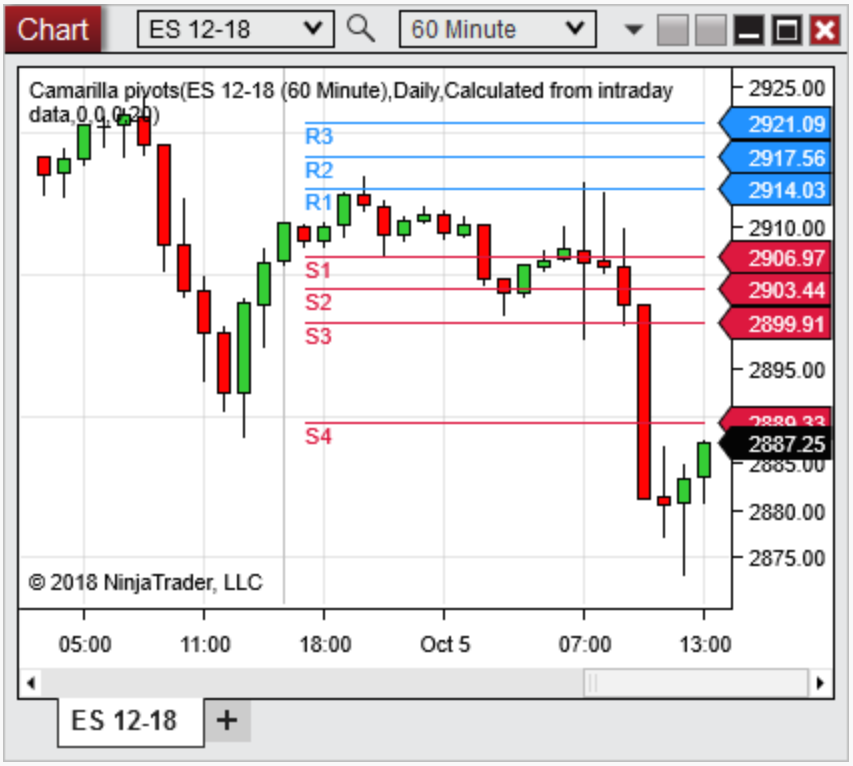
Added templates for Indicators
Indicator
Feature #13129

Added the ability to save templates for indicators to more easily manage different configurations of indicators for your needs.



Added Fibonacci and Camarilla Pivots
Indicator
Feature #12834

Pivot indicators calculate areas of potential support and resistance based on values from the prior trading session.



Added Choppiness Index
Indicator
Feature #12779

The Choppiness Index is an indicator that determined if the market is choppy (trading sideways) or not choppy (trading within a trend in either direction). A high value is more choppy. A lower value indicates the market is trending.

Added Vortex
Indicator
Feature #12780

The Vortex is an oscillator used to identify trends. A bullish signal triggers when the VIPlus line crosses above the VIMinus line. A bearish signal triggers when the VIMinus line crosses above the VIPlus line.

Added Block volume
Indicator
Feature #13069

Block Volume can track how many block trades occurred within a bar. It can display the number of block trades or the total volume of the block trades. This could even be used to count how many ticks occurred within a bar. Historical tick data is required to historically calculate this indicator.

Added FX Tile
Indicator
Feature #13071

The FX Tile display a tile similar to the tiles in the FX Board, directly on the chart. It will display the spread, can be moved around, and can even be used to place order when Chart Trader is enabled.

Added Psychological Line
Indicator
Feature #12778

The Psychological Line displays a ratio of what percent of bars were up bars over a specified number of bars. A higher ratio may indicate the price is more likely to drop. A lower ratio may indicate the price is more likely to rise.

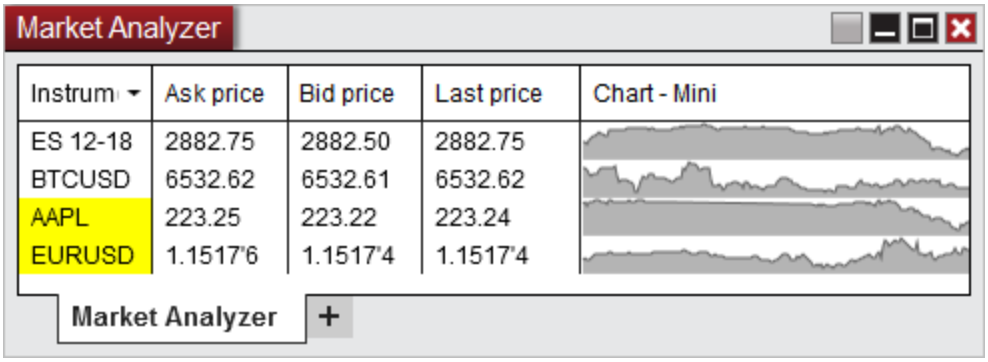
Improved icons
UI
Feature #13130

Implemented new icons to further increase the clean and sleek feel of the platform.

Added Chart - Mini column
Market Analyzer

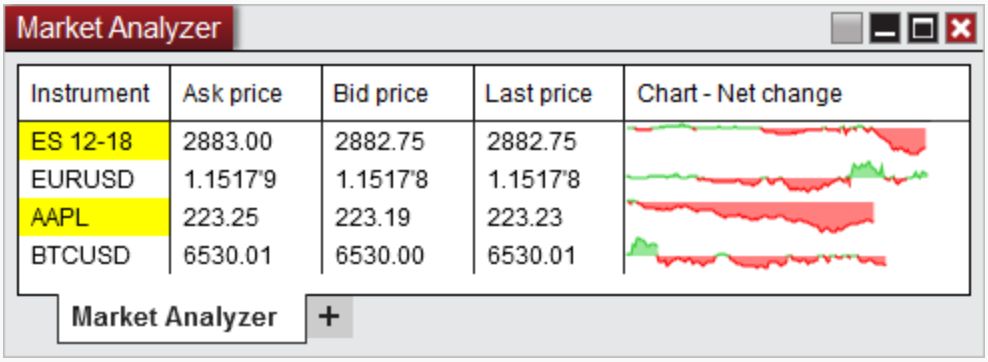
Feature #12893

The Chart - Mini displays a user specified span of time for a quick view of the market. The time does relate across rows. So, if it is 1:05pm and it has a 5 minute span, it will display the last 5 minutes from 1:00pm to 1:05pm



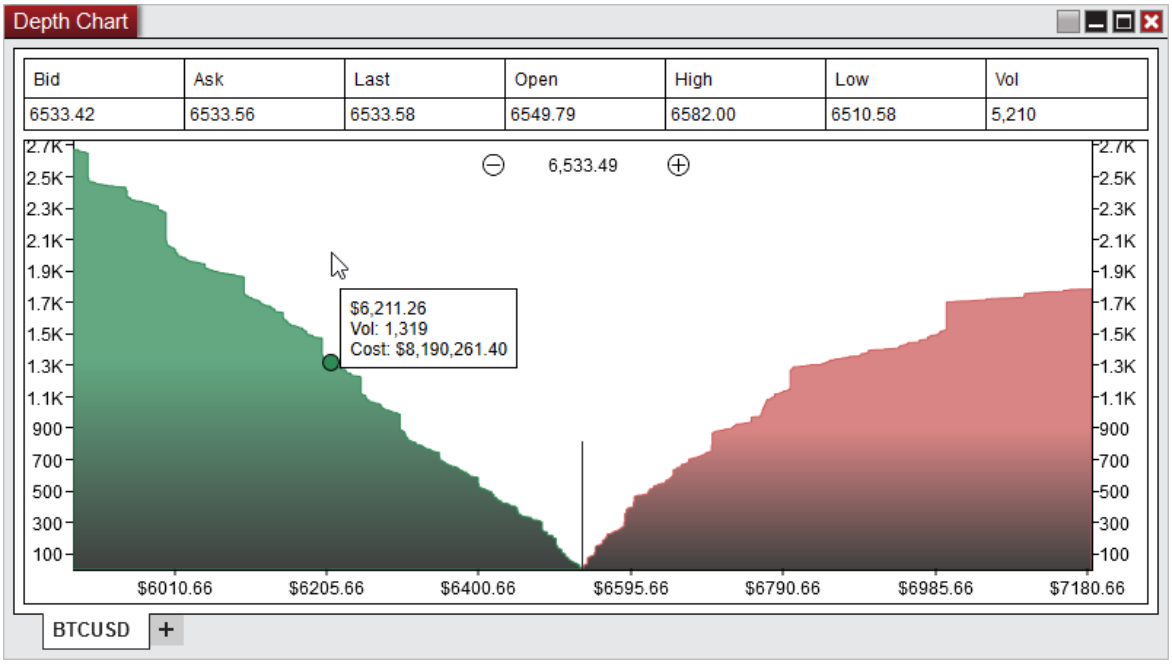
Added Chart - Net change column
Market Analyzer
Feature #13101

The Chart - Net change column is a quick visualization to see the movement of the net change for the session. The start of the column is the session open and the end is the session close. The time does not relate across rows.



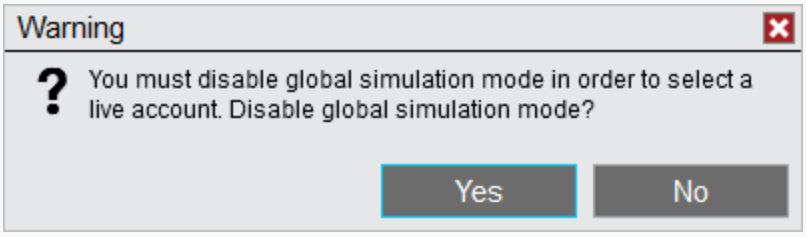
Added a Depth Chart for cryptocurrencies
Depth Chart
Feature #12326

The Depth Chart is a utility for cryptocurrencies that shows an aggregated view of the current order box. This creates an easy display to measure the resistance per side for the price to move.



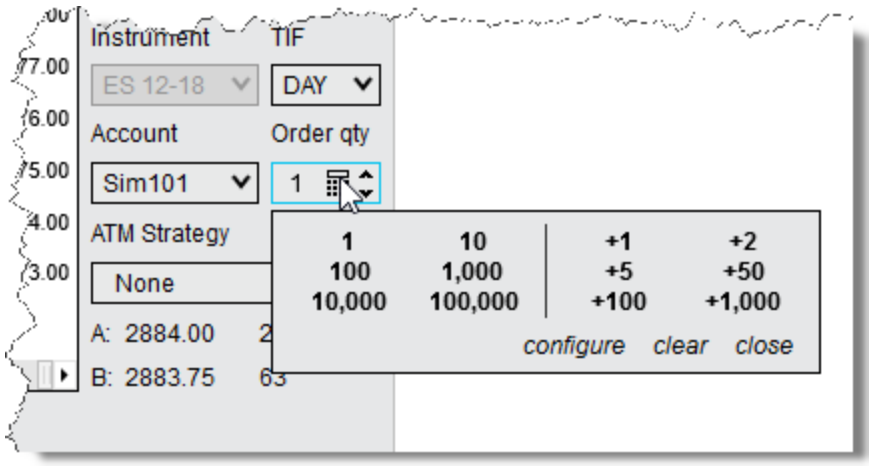
Added the ability to disable global simulation mode directly from the warning message
Orders
Feature #13067

Previously when global simulation mode was enabled and you would select a live account you would have to click through the Control Center to disable global simulation mode. Now it can be done quickly and easily, right from the warning dialog.



Added icon to access the QTY Pad
Order
Feature #13048

The QTY Pad was a feature often missed since it was accessed by middle mouse clicking. Now we have added a button to access the QTY Pad, which can quickly adjust your order quantity and has the ability to configure the available options.

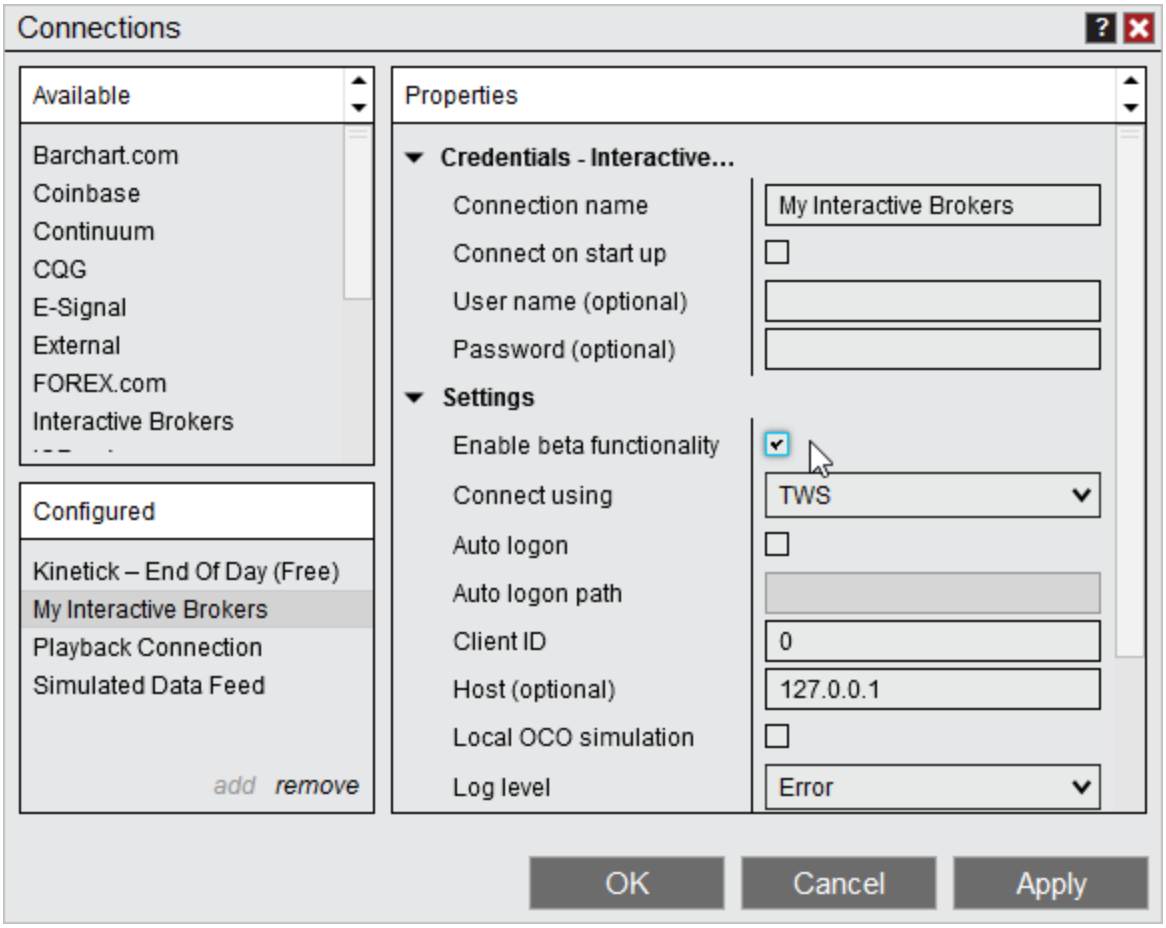


Added option to match data series width for indicators using a bar plot style
Chart, Indicator
Feature #12303

Often times when using an indicator that uses a Bar Plot style, you want the width to match the data series. Setting it and then adjusting the charts compression would then result in them not matching again. Now there is an option to have it dynamically match.

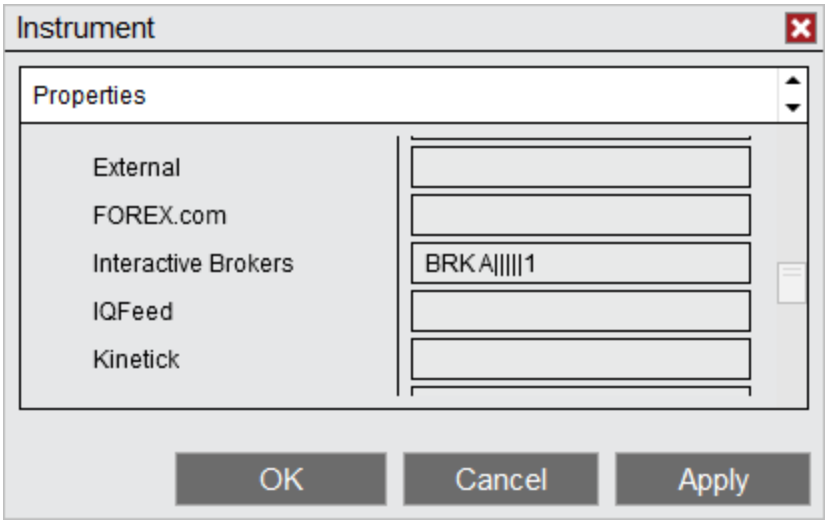
Updated required version of Traders Workstation for Interactive Brokers to 973 and added the ability to enable additional functionality that is in beta
Interactive Brokers
Feature #13053

Beta function can be enabled within the connections properties.



The following functionality has been implemented for beta.

- Unfiltered real-time data
- Support for linked accounts
- Improved level II data handling
- The equities volume multiplier is now applied to all equities
- Added the ability to remove equities volume multiplier by adding ||||1 to the instruments symbol mapping



Added the ability to use the greatest right side margin when it is set in both a script and the chart
Chart, NinjaScript
Feature #13070

Prior if a script set a right side margin and you had a custom right side margin they could conflict with each other. Now we have added the ability to use the greatest margin.

Issue #	Status	Category	Comments
12951	Fixed	Alerts	Alerts could not be set to occur at the same time each day
13147	Fixed	Bars	Requesting bars with GetBars could yield wrong data
13042	Fixed	Chart	Cross hair cursor on X-Axis on future date did not show the date
13068	Fixed	Chart	Multi-data series charts sometimes would not plot the session break line
13138	Fixed	Chart	Ray line using 1st series price copied to 2nd series price if chart had multiple data series

			in the same pane
13178	Fixed	Chart	Chart interval selector cut off portion of window when too many intervals were used
13181	Fixed	Chart	In some scenarios using Default 24x5 Trading hours on the CL resulted in missing daily bars for Mondays
13207	Fixed	Chart	Panels heights on charts could be minimized all the way down, preventing ability to properly adjust the panel
13038	Fixed	Chart, Data	In some scenarios historical data was not being requested up to the current date
13077	Fixed	Chart, Indicator	Woodies CCI could have an error when switching data series
13090	Fixed	Chart, NinjaScript	Additional spacing for text at the bottom left of the chart applied to panels that did not have the NinjaTrader copy write text
13167	Fixed	CoinBase, Chart	Volume chart type did not display as expected
13023	Fixed	Coinbase, Data	Real time volume was not including full volume
13194	Fixed	Commissions	Resolved an error that could occur when changing commissions templates
13116	Fixed	Control Center, Orders	In some scenarios simulated orders to a live account did not show the stop price In the Orders tab
13025	Fixed	Database	Resolved a scenario where after removing a custom bar type a multi-data series chart may not roll over

13047	Fixed	DrawingTool	Drawing object properties could not be guaranteed accessible when an indicator accessed the collection in State.Terminated on chart close
13121	Fixed	DrawingTool	Order Flow Volume Profile Drawing tool did not work correctly when in UTC+02 time zone
13185	Fixed	DrawingTool	Ruler drawing tool rounded incorrectly for instruments whose ticksize has more than 6 fractional digits
13201	Fixed	DrawingTool	Applying a template to a Polygon resulted in the object disappearing
13142	Changed	eSignal	Updated supported eSignal client to 12.9.4919.1048
13078	Fixed	eSignal, Data	Real-time and historical forex volume did not match
13105	Fixed	Forex.com	Resolved a scenario where an ATM's stop could successfully change but showed as Change Submitted then couldn't be canceled
13155	Done	Forex.com	Forex.com connection was removed from beta
13020	Fixed	FXCM	Update FXCM API to 1.6
13064	Fixed	FXCM, Orders	Orders and executions could show more than 1 event update
13060	Fixed	Historical Data Window	Was unable to import MetaStock data

13206	Fixed	Hot Key	Hot keys to switch snap mode did not function while drawing an object
13087	Fixed	Hot Key, Chart Trader	Some order entry Hot Keys stopped working when Chart Trader was hidden
13033	Fixed	Indicator	Resolved some scenarios that resulted in the Chalkin money flow displaying incorrect values and incorrect x-axis scaling
13100	Fixed	Indicator	In the indicator properties window, extended the ability to click the 'i' to show the description for a selected indicator to configured indicators
13119	Fixed	Indicator	Pivots indicators could not be run on top of bars with secondary daily type
13169	Changed	Indicator	Rearranged the properties to some of the newer indicators for consistency
13044	Fixed	Indicator, Tick Replay	When using Volume profile with TickReplay and split session trading hours an error could occur
13000	Fixed	Interactive Brokers	US equities did not have a multiplier applied for volume when only the default exchange was selected
13188	Fixed	Level II Window	Tracking market makers did not work if the connection displayed them with lower case letters
13054	Changed	Localization	Translated additional areas of the platforms for the available languages
13212	Fixed	Localization,	Pivot indicators could select incorrect PivotRange depending on localization

		Indicator	
13074	Fixed	Market Analyzer	Blank name property for indicator column resulted in an error
13089	Fixed	Market Analyzer	Indicator column title displayed inconsistently on workspace/template restore
13133	Fixed	Market Analyzer	VWAP column did not use decimals
13180	Fixed	Market Analyzer	Bar graph colors for indicator columns got set to blank when selecting two bar graph indicator columns in a row
13222	Fixed	Market Analyzer	Historical multi-series NinjaScript bar processing was not working as expected
12941	Fixed	NinjaScript	Resolved a scenario where a NinjaScript error could not be accessed until the indicator properties window was closed
13110	Fixed	NinjaScript	Foreign language characters in class names generated incorrect code
13141	Fixed	NinjaScript	SetStopLoss / SetProfitTarget orders could get stuck in Initialized when a strategy attempted to resume with ImmediatelySubmit
13191	Fixed	NinjaScript	Resolved a scenario where exporting NinjaScript resulted in an error
13079	Fixed	NinjaScript Editor	Save as had stopped updating NinjaScript name property

13104	Fixed	NinjaScript Editor	When a Find dialog was open in the NinjaScript editor, it was on top of all non modal windows
13177	Fixed	NinjaScript Editor	After closing a tab the Ctrl + Tab keyboard combination resulted in an error
13034	Fixed	NinjaScript, Chart	Reload all historical data stoped secondary series in a script from updating on another chart
13075	Fixed	NinjaScript, DrawingTool	Default templates for text drawing object could shift text placement when used in a script
13183	Fixed	NinjaScript, DrawingTool	Resolved some scenarios where regions were misaligned or did not plot
13190	Fixed	NinjaScript, Order Flow +	Bars.BarsSinceNewTradingDay was not resetting as expected on Volumetric bars
13198	Fixed	Order	Simulated GTD orders did not work correctly
13092	Fixed	Order Flow +	Order Flow Volume Profile's Extended Naked Point of Control feature did not work in some cases
13113	Fixed	Order Flow +	Cumulative Delta returned a 0 on first live tick
13134	Fixed	Order Flow +	Order Flow Depth Map historical could have some blank areas
13145	Fixed	Order Flow +	Order Flow Volume Profile with limited tick data caused session rendering to be off

13196	Fixed	Order Flow +	Volumetric bars didn't properly work with cryptocurrencies
13203	Fixed	Order Flow +	OrderFlow VolumeProfile would not display initial balance lines in some scenarios
13157	Fixed	Playback, Adapter	Resovled a scenario where recording data for Playback stopped real-time data from coming in.
12862	Fixed	Properties Grids	When typing to select an item, was not able to append what is typed to more precisely select an item
13172	Fixed	Strategies, Orders	Disabling and enabling strategies could result in orders stuck in state Initialized or CancelPending
13024	Fixed	Strategy	In some scenarios simulation orders did not cancel when a strategy was disabled
13173	Fixed	Strategy	Updating strategy properties from strategy grid to chart could fail
13210	Fixed	Strategy	Changing hosted indicator properties while strategy was running resulted in an error
12986	Fixed	Strategy Analyzer	Optimizer enums got set to their coded values and ignored user values on the details run
13080	Fixed	Strategy Analyzer	During the optimizer details run, invalid information could be sent to the output window and logs
13111	Fixed	Strategy Analyzer	Changing user input type from double to int caused error between optimizations

13114	Fixed	Strategy Analyzer	Walk Forward Optimization did not show an error if there were no properties to optimize
13124	Fixed	Strategy Analyzer	Could not us Volumetric bars with a base period type of Range
13137	Fixed	Strategy Analyzer	Saving a backtest template after an optimization caused standard backtests to change and use optimization parameters
13027	Changed	Strategy Builder	Renamed some stop / target names to be more clear
13081	Fixed	Strategy Builder	Adding an additional data series and setting an internal variable resulted in an error
13102	Fixed	Strategy Builder	Setting a variable action with an offset referencing a 2nd data series resulted in an error
13003	Fixed	Trade Performance	When Windows was using UK date formatting, Trade Performance's start and end dates became invalid after 24 hour
13162	Fixed	Trade Performance	Filters would reset when selecting Generate
13205	Fixed	Trade Performance	Display mode was not persisted and did not save to the workspace
13164	Fixed	Trading Hours	In some scenarios using the Add Monday through Friday feature for sessions would not save to the template

13106	Fixed	Window Linking, Chart	Interval linking to color Link All did not work with charts
13097	Fixed	Workspaces	Resolved a scenario that could result in a saved workspace being blank

8.0.16.1 Release Date
October 17, 2018

Issue #	Status	Category	Comments
13249	Fixed	NinjaScript	Resolved an unintentional code breaking change

8.0.16.2 Release Date
October 29, 2018

Issue #	Status	Category	Comments
13294	Changed	Chart	Reduced chart panel minimum height
13257	Fixed	Chart Trader	Width did not persist when hiding/unhiding
13251	Fixed	Interactive Brokers	With beta functions enabled position updates could show on the wrong expiry
13268	Fixed	Market Analyzer	Chart - Net change did not properly function when opened before the session start

13262	Fixed	Trade Performance	In some scenarios filters could become unchecked unexpectedly

8.0.16.3 Release Date
November 2, 2018

Issue #	Status	Category	Comments
13328	Fixed	Chart	Resolved a scenario where chart borders would not display as expected

3.3.14 8.0.15.1

8.0.15.0 Release Date
July 30, 2018

Feature #	Status	Category	Comments
12351	Added	Order Flow +	Added Order Flow Volume Profile indicator
12381	Added	Order Flow +	Added Order Flow Volume Profile drawing tool
12916	Added	Coinbase	Added Cryptocurrency support via Coinbase (See the Coinbase connection guide here)
12798	Added	Drawing	Added Polygon drawing tool
12873	Added	Drawing	Added Time Cycles drawing tool

12777	Added	Indicator	Added McClellan Oscillator indicator
12818	Added	Indicator	Added Relative Vigor Index indicator
12909	Added	Indicator	Added Wiseman Awesome Oscillator indicator
12910	Added	Indicator	Added Wiseman Alligator indicator
12911	Added	Indicator	Added Wiseman Fractal indicator
12871	Added	NinjaScript Editor	Added an animated icon to show when compiling in the NinjaScript editor
12882	Added	NinjaScript	Added overload for NinjaScript: AddDataSeries(string instrumentName), allowing same Bars Period with a different instrument
13014	Added	Strategy	Added Parabolic Stop support for NinjaScript
12937	Added	Strategy Analyzer	Added Max Strength optimization metric
12967	Added	Strategy Analyzer	Added Probability statistic to the summary tab
12305	Added	Workspaces	Added automatic backup of saved workspaces which can be restored under Tools > Database Management

12938	Added	Chart, Indicator	Added "Calculating..." label for when indicators are still processing historical data
12942	Added	Historical Data	Added data importer for tickdata.com

Issue #	Status	Category	Comments
12722	Changed	Alerts	Improved Alert Pop Up message formatting
12855	Fixed	Alerts	Resolved scenario where alerts using global drawing objects no longer functioned after a restart
12884	Fixed	Alerts	Removed invalid drawing objects pop up that could occur when rolling over
12921	Fixed	Alerts	Reloading NinjaScript in a chart resulted in alerts tied to indicators or drawing objects to stop working
12977	Fixed	Alerts	Exporting from Alerts log to Excel caused the time to be exported incorrectly
12854	Fixed	ATM Strategies	Resolved some scenarios where adjusting settings could result in a TIF error
12970	Fixed	ATM Strategies	Modified stop value was not respected when scaling in with an ATM that uses simulated stops
12952	Fixed	BarType	Heiken Ashi based on tick bars created different, rather than modified, bars
12666	Fixed	Chart	Increasing a chart's days to load did not function if it was increased by less than 4 days

12794	Fixed	Chart	In some scenarios moving an indicator to a different panel could leave an empty panel
12861	Fixed	Chart	Price marker disappeared on non-equidistant chart when more granular series reached right edge of screen
12869	Fixed	Chart	When equidistant was set to false multi-series charts with time-based and daily series used wrong series for latest bar
12878	Fixed	Chart	Chart bars refresh could result in incorrect panel values
12975	Fixed	Chart	Incorrect dates were shown with global cross-hairs when the cursor was along the x-axis
12860	Fixed	Chart	Orders on instruments that display values as fractions did not display order as a fraction when modifying
12979	Fixed	CQG, Continuum, Connections	Resolved a case where some order states prevented a connection
12968	Fixed	Data	Metastock MASTER file was not recognized by File Explorer
12974	Fixed	Data	There was no pop up error when attempting to import Metastock data using the 64-bit version
12972	Fixed	Drawing	When snapping Trend Channel to price it would move in tick increments away from copied price instead of snapping to tick boundaries
12863	Fixed	DrawingTool	Moving draw objects could change anchor points on multi-series chart

12876	Fixed	DrawingTool	Resolved an error that occurred when drawing a vertical line to an empty panel
13011	Fixed	DrawingTool, Templates	Saving a brush to a Text drawing tool's default template overrode brush values passed into NinjaScript
12947	Fixed	eSignal, Data	For some instruments level II data would not update
12932	Fixed	FXCM, Orders	Resolved a scenario where a filled order showed as active/working
12890	Fixed	Indicator	Auto Scale option within the Price Line indicator was showing last price as "0" on a reconnect
12894	Fixed	Indicator	Volume profile indicator did not draw for non-intraday periods
12903	Fixed	Indicator	Displacement did not work properly in Regression Channel indicator
12907	Fixed	Indicator	Indicators with multiple DataSeries could intermittently see 'null' reference during 'SetState'
12925	Fixed	Indicator	Woodies CCI had incorrect panel values
12980	Fixed	Indicator	The Darvas indicator was not creating a well formed Darvas box
13004	Changed	Indicator	Updated CandleStickPattern indicator to be consistent and optimized
12946	Fixed	Indicator, Chart	When Indicators were active and multiple linked charts switch instruments at the same time an error occurred

12866	Fixed	Instruments	Splits were not sorted when changes were applied in instruments dialog
12881	Fixed	Instruments	Pop up dialog for if instrument had server changes had incorrect icon + label text
12915	Fixed	Instruments	Resolved a scenario where option to add an instrument to a list displayed the wrong instrument if using multiple tabs
12962	Fixed	Instruments	Resolved a scenario where if an instrument was incorrectly defined it caused a crash rather than just an error
12940	Fixed	Interactive Brokers	Currently building daily bar was missing for forex instruments
12865	Fixed	Interactive Brokers, Data	Level II data for live accounts was different than paper accounts and Trader Workstation
12889	Fixed	Interactive Brokers, Data	Real-time forex bid tick data differed from real-time last tick data
12987	Fixed	Interactive Brokers, Orders	Resolved an error that could occur if an order was terminal and then was also attempted to be canceled
12848	Fixed	IQFeed	Optimized loading logic so that historical data which is not available is not requested, resulting in a longer load time
12841	Fixed	Market Analyzer	Indicator column logic did not support DisplayName override

12845	Fixed	Market Analyzer	Columns would format to the instrument currency instead of the account currency
12853	Changed	Market Analyzer	Added up and down action buttons to configured section in the object dialog
12990	Fixed	Market Analyzer	Total Row was not formatted the same way as column data
12630	Fixed	NinjaScript	Charts could stop loading and display return to present icon in some scenarios in which a script quickly updates a draw object
12917	Fixed	NinjaScript	Resolved a null check for areaBrushDevice that could result in an error
12948	Fixed	NinjaScript	Indicators window could list duplicate instances for strategy added indicators
12964	Fixed	NinjaScript	Block comments on non-NinjaScript properties caused them to be treated as NinjaScriptProperty in generated code
12966	Fixed	NinjaScript	If the Dot Dash Style was selected for a NinjaTrader.Gui.Stroke the stroke would not render
12976	Fixed	NinjaScript	The ReadOnly attribute did not work for strategies on a chart
12280	Fixed	NinjaScript Editor	Collapsed regions could expand unexpectedly when editing document
12875	Changed	NinjaScript Editor	Updated Actipro (Third Party Component used for NinjaScript Editor and Output Window)

13013	Fixed	NinjaScript, Market Analyzer	Load data based on bars did not work for a multi-time frame series indicator
12844	Fixed	Order Flow +	Depth Map and Trade Detector were incorrectly available in the Market Analyzer and Strategy Builder
12892	Fixed	Order Flow +	Resolved support for some bar types on Volumetric and made code more consistent with other scripts
12920	Fixed	Order Flow +	Order Flow Cumulative Delta did not work on non-intraday charts
12944	Fixed	Order Flow +	Order Flow indicators no longer suspend when inactive as it could affect the values
13008	Fixed	Order Flow +	Order Flow Cumulative Delta on Line Break charts would not form new bars
13010	Fixed	Output Window	Resolved an error that occurred when a window was closed that was printing many lines
12656	Fixed	Performance	Resolved a rare scenario in which quickly working with indicators could increase memory that would not clear
12850	Fixed	Playback	When no Playback data is available to download for the selected instrument/date, there was no message saying no data
12843	Fixed	Playback, NinjaScript	Replaying historical data and using Position.GetUnrealizedProfitLoss could cause Playback to lockup
12836	Fixed	ShareAdapter	Facebook share service discontinued

12919	Fixed	Simulator	Newly created local simulation accounts now have a prefix of 'Sim'
12918	Fixed	Strategy	Resolved a scenario where a high fill resolution series that is more granular than primary bars prevented limit orders from filling
12830	Fixed	Strategy Analyzer	Walk forward optimization displayed chart with incorrect series
12905	Fixed	Strategy Analyzer	Small values were not populating properly in the results grid when displaying percent data
12927	Fixed	Strategy Analyzer	Profit values did not display consistently when display type was set to percent
12928	Fixed	Strategy Analyzer	When running a strategy with slippage set to 0, it will show in the optimizer row but not in the summary for that row
12989	Fixed	Strategy Analyzer	Performance metric value R2 was not functioning as expected
12901	Fixed	Strategy Builder	Configurations using input variables in the condition builder could result in an error
12902	Fixed	Strategy Builder	Setting stop or target values to an indicator resulted in an error
12984	Fixed	Strategy	When making an indicator created in the Condition Builder plot to a chart, the reference

		Builder	to the plotting indicator was lost in NinjaScript
12997	Fixed	Strategy Builder	Combined Results were lost when duplicated to new tab
13001	Fixed	Strategy Builder	Resolved an error that could occur when viewing code that used counter conditions
12887	Fixed	SuperDOM	SuperDOM Quantity Pad 'Apply' button did not update pop-up list when pressed
12988	Fixed	SuperDOM	Columns could reduce width when connecting and selecting accounts
13002	Fixed	SuperDOM, Playback	Volume Column did not reload data in Playback
12885	Fixed	TD Ameritrade, Historical Data Window	Daily historical data downloaded today's data with tomorrow's timestamps
12991	Fixed	Trade Performance, Playback	Was unable to filter out Playback execution once disconnected
12922	Fixed	Window Linking	When using the arrow keys in the Alerts dialog to switch instruments, linked windows would not switch instruments
11176	Fixed	Workspaces	A crash could occur when switching workspaces while printing to output and connecting

8.0.15.1 Release Date
August 1, 2018

Issue #	Status	Category	Comments
13041	Fixed	Order Flow +	Order Flow Volume Profile reverted custom right side margin when applied
13049	Fixed	Order Flow +	Restoring Order Flow Volume Profile drawing tool instance from workspace would sometimes fail
13050	Fixed	Order Flow +	Order Flow Volume Profile Indicator could include more underlying minute data then expected
13051	Fixed	Order Flow +	Order Flow Volume Profile could throw an exception when multiple load requests were pending
13052	Fixed	Interactive Brokers, Data	Reverted fix for 12865 due to unexpected side effects

3.3.15 8.0.14.2

8.0.14.0 Release Date
May 24, 2018

Feature #	Status	Category	Comments
12763	Added	Indicator	Added Moving Average Ribbon
12781	Adde	Indicator	Added Disparity Index

	d		
12782	Added	Indicator	Added Money Flow Oscillator
12750	Added	Indicator	Added Price Line
12756	Added	Indicator	Added Net Change Display
12713	Added	Instruments	Added Tadawul exchange support

Issue #	Status	Category	Comments
12643	Fixed	Alerts	Indicator alerts under certain conditions were not rearming on condition reverse
12696	Fixed	Alerts	Alerts set to rearm 'on bar close' for multi-series indicators would rearm on each tick
12772	Fixed	Alerts	Alerts based on Trend Channel drawing tool did not trigger on extended portion
12821	Fixed	Alerts	Resolved a scenario where Action template did not save screen-shot action
12662	Fixed	ATM Strategies	Resolved a scenario where an error resulted in a crash
12829	Fixed	ATM Strategies	ATMs with a GTD TIF was not applying the desired date

12689	Fixed	Backup & Restore	Restoring a workspace with a NinjaScript Editor window could result in a hang
12694	Fixed	BarsType	Was unable to create new bars types when custom type with BarsPeriodType 32768 or greater existed
12578	Fixed	Chart	Charts could become unresponsive when three 1 tick data series were requested
12631	Fixed	Chart	Horizontal grid lines set to fixed was unable to set a value lower then 1
12642	Fixed	Chart	Setting a chart panel's scaling to consider the entire date range did not work for indicator panels
12729	Fixed	Chart	Resolved a scenario where Access Violation exceptions could occur as Chart properties were modified
12731	Fixed	Chart	Heiken Ashi bars produced a bar every tick when Break at EOD was unchecked
12758	Fixed	Chart	Data box window ignored unchecked 'Always on top' property setting when closed then reopened
12767	Fixed	Chart	Data series dialog will now will load all public defined brushes defined in custom ChartStyle's
12768	Fixed	Chart	X-axis and scroll bar could render incorrectly on weekly charts
12808	Fixed	Chart	Chart was stuck with "Loading..." text when provider does not support instrument type
12732	Fixed	Chart, Indicator	Resolved a scenario where removing then re-adding an indicator while connecting and loading resulted in an error

12835	Fixed	Control Center, Regionalization	Lock up could occur on language selection change
12626	Fixed	Control Center, Strategy	Within the Strategies tab with Filter only active strategies enabled, having strategies selected then disabling them could result in a disabled strategy still showing
12796	Fixed	Control Center, Strategy	Strategy grid could miss flagging scenarios where children instruments were out of sync
12832	Fixed	Control Center, Strategy	Time in force field would not populate when using GTD
12654	Fixed	CQG, Continuum, Orders	Resolved a scenario in which an order error would repeat if a change request was made for the same price and quantity
12811	Fixed	DrawingTool	Gann Fan drawing tool did not use snap modes
12740	Fixed	DrawingTool, NinjaScript	Compiling then reloading NinjaScripts on a chart could remove some drawing objects custom levels
12806	Fixed	eSignal, Data	Some instruments were unable to receive data although symbol mapping was correct
12710	Fixed	eSignal, Instruments	Custom mapped exchanges were ignored
12723	Fixed	Forex.com, Data	Resolved a scenario in which an error would occur when requesting data
12646	Fixed	FX Board, FX Pro	CFD spread calculation was off

12804	Fixed	FXCM	When running overnight, the next daily bar would not plot with NinjaTrader historical data servers
12709	Fixed	G2	An error that was unknown could result in a crash
12736	Fixed	G2	Historical data could duplicate bar volume on session break
12838	Fixed	G2	It was possible to make two G2 connections, which is not supported
12675	Fixed	G2	Real-time data was not received while using a simulation license
12699	Changed	G2	Native historical data volume and real-time volume changed to 10K
12717	Changed	G2	Instrument look up changed to find entry with exact string
12725	Fixed	G2	In some scenarios unaccounted for order updates resulted in an order with a price of zero
12641	Fixed	Google	Google discontinued
12805	Fixed	Hot Key	Custom Orders Hot Keys could be entered with Hot Keys already in use
12755	Fixed	Indicator	NaN value was produced for @stdError for the upper and lower plots on the first bar of a chart
12650	Fixed	Indicator, Chart	Indicators with indicator-as-input could have bad historical values when loading and

			switching tabs
12687	Fixed	Indicator, Drawing Tools, Chart	In some scenarios an indicator drawn Regression Channel would change direction when scrolling a multi-series chart left and right
12661	Fixed	Instruments, Orders	Changing rollover date removed trade history
12416	Fixed	Interactive Brokers, Orders	Externally submitted orders outside of NinjaTrader were unable to be canceled or modified
12697	Fixed	IQFeed, Data	In some scenarios bid and ask data would not complete downloading
12482	Fixed	Market Analyzer	Quickly adding labels and blank rows result in an error
12585	Fixed	Market Analyzer	Resolved a scenario where adjusting a columns settings reduced the columns width
12599	Fixed	Market Analyzer	An invalid selection of an input series was available for non-indicator columns
12576	Fixed	Market Analyzer, NinjaScript	Bar graph columns did not display the correct color after reloading NinjaScript if the value of CurrentBar did not change between MarketData events
12644	Fixed	NinjaScript	Was unable to cleanup objects terminated when a tab was removed
12669	Fixed	NinjaScript	Tick Replay could doubling Input[0] values in certain hosted scenarios
12797	Fixed	NinjaScript	Resolved a scenario where an indicator would get an error as additional data series were added to the chart and objects were moved to different panels

12815	Fixed	NinjaScript	Creating preset then changing Indicator plots caused error on selecting indicator in indicator dialog
12741	Fixed	NinjaScript	Dash style and opacity can now be set with PriceLevel constructor
12737	Fixed	NinjaScript Editor	Adjusted transparency of collapsible region background highlighting
12827	Fixed	NinjaScript Editor	Resolved a scenario where an error could occur when a strategy was removed from the NinjaScript Editor
12826	Fixed	Oanda, FXCM	The next days daily bar did not plot as expected when running overnight
12667	Fixed	Options	Auto close position did not include the ability to define seconds
12450	Fixed	Order Flow +	Resolved some scenarios that would result in missing data or incorrect values
12640	Fixed	Order Flow +	Resolved a scenario where Order Flow VWAP did not reset on session breaks
12679	Fixed	Order Flow +	Multi-Series Order Flow Plus indicators were not suspending when inactive
12711	Fixed	Order Flow +	Resolved a scenario where Order Flow Cumulative Delta could cause an exception
12747	Fixed	Order Flow +	Resolved some scenarios where Trade Detector would throw an error
12774	Fixed	Order Flow +	Resolved a scenario where Volumetric bar stats could be incorrectly colored on current bar
12757	Fixed	Order Flow +,	Using Order Flow VWAP in the Strategy Builder resulted in an error

		Strategy Builder	
12706	Fixed	Playback, Chart	In some scenarios zooming in and out the chart prevented execution markers from displaying
12681	Fixed	Playback, Indicator	Indicator loading another indicator or series could see an error when rewinding playback
12674	Fixed	Regionalization, NinjaScript	Set() method could throw exception in OnStateChange() if Language was set to a non-English language
12620	Fixed	Simulator, ATM Strategies, Orders	Resolved a scenario where snapshot data could unexpectedly move a stop strategy
12670	Fixed	Strategy	Darvas box indicator when used in a strategy showed 0 for lower level
12795	Fixed	Strategy	Strategies did not adopt account position if they canceled orders on start up
12651	Fixed	Strategy Analyzer	Slippage was not rounding to tick size
12817	Fixed	Strategy Analyzer	Log incorrectly displayed Data series column for previous tests
12624	Fixed	Strategy Analyzer	Switching to the chart display when no back-test was ran resulted in a transparent chart
12611	Fixed	Strategy Builder	When certain indicators set as an input series the code generated would result in a compile error

12628	Fixed	Strategy Builder	An indicator with an enum defined in a class inside a namespace resulted in a compile error
12743	Fixed	Strategy Builder	Condition Builder did not perform type checking when adding grouped conditions
12754	Fixed	Strategy Builder	Was not able to select an added data series as an input
12761	Fixed	Strategy Builder	Copying then unlocking a Strategy Builder script prevented the strategy from being loaded by double-clicking the error output
12771	Fixed	Hot Key	Recently modified quantity value did not update until the control was unfocused, potentially impacting Hot Key order entry
12538	Fixed	Strategy, Tick Replay	Strategies that added a chart indicator and were using Tick Replay resulted in an error
12671	Fixed	TD Ameritrade	Updated connectivity handling for more resiliency
12600	Fixed	TD Ameritrade	Resolved a scenario where historical data requests could result in an error
12636	Fixed	TD Ameritrade, Data	Friday and weekends daily data was not displaying
12765	Fixed	TD Ameritrade, Data	Added data throttling of 2 historical data requests per second
12793	Fixed	TD Ameritrade, Data	No longer throws an error as no data is available when returned from an historical data request

12819	Fixed	TD Ameritrade, Data	Historical data downloaded today's data with tomorrow's timestamps
12716	Fixed	Tick Replay	Closing NinjaTrader while Tick Replay was loading resulting in freezing
12719	Fixed	Tick Replay, Strategy	In some strategy configurations, Tick Replay could be defaulted back to disabled unexpectedly
12760	Fixed	Time and Sales	Changes made to the properties grid were lost when changing context menu properties
12714	Fixed	Trade Performance	Strategies set to immediately submit had incorrect trade matching if strategy was already in a position when started
12769	Fixed	Trade Performance	Journal did not sort by date as expected
12825	Fixed	Trading Hours	Templates could add rows with same start and end time
12584	Fixed	Workspaces, Chart	Resolved a case where closing NinjaTrader while the workspace was still loading resulted in an error
12822	Fixed	Workspaces, Data	Closing a workspace while data is loading could result in errors

8.0.14.1 Release Date

May 30, 2018

Issue #	Status	Category	Comments

12857	Fixed	Playback, ATM Strategies	ATM stop market orders being simulated on playback did not execute
12802	Fixed	Chart	Corrected default screenshot file name generation

8.0.14.2 Release Date
June 14, 2018

Issue #	Status	Category	Comments
12867	Fixed	Forex.com	Forex.com adapter could restore an order incorrectly on reconnect
12912	Fixed	NinjaScript	Resolved an error on scripts that required 'OnRenderTargetChanged' to process after state 'Configure', which is now guaranteed
12923	Fixed	Indicator	Invalid operation exception could be thrown be thrown during Chart Panel rendering for indicators

3.3.16 8.0.13.1

8.0.13.0 Release Date
March 26, 2018

Summary
In addition to standard bug fixes in this release, we have implemented various performance optimizations related to chart rendering.We have also been iterating on the 'Order Flow +' tools launched in preview last release. Additionally, we identified a help guide documentation inconsistency for NinjaScript developers resulting in the following advisory being posted: Notice for NinjaScript developers overriding OnRender

Issu	Status	Category	Comments

e #			
12468	Added	Indicator	Added total volume and price to hover values in Order Flow Trade Detector
12594	Changed	Interactive Brokers	Removed Forex 'Lot Size' setting from connection configuration as it can be configured manually per account
12462	Changed	Indicator	Removed 'Plot current value only' from CurrentDayOHL as it is a duplicate function of changing the plots to Horizontal lines
12586	Changed	Indicator, NinjaScript	Order Flow Cumulative Delta previously shared its DeltaType enum with Volumetric Bars which was not ideal for exporting NinjaScript. NinjaScript developers whom utilized the preview release of 'Order Flow +' will require a small signature update: Order Flow Cumulative Delta
12506	Changed	Indicator, NinjaScript	Order Flow VWAP now uses Trading Hours template to set custom start/end time. NinjaScript developers whom utilized the preview release of 'Order Flow +' will require a small signature update: Order Flow VWAP
12497	Changed	Chart	Implemented various chart rendering performance optimizations
12474	Changed	Indicator	Implemented Order Flow Trade Detector performance optimizations and resolved alignment on second charts
12469	Fixed	Indicator	Order Flow Cumulative Delta could have plotted unexpectedly after reloading a chart

12472	Fixed	Indicator	Order Flow Cumulative Delta could not be used as the input to another indicator
12430	Fixed	Indicator	Resolved various Order Flow Market Depth Map display issues
12446	Fixed	CQG, NinjaScript	Fast order changes could be rejected from a strategy using SetTrailStop in fast moving market
12464	Fixed	Chart, Bars Type	Polished various Order Flow Volumetric parameters and resolved various loading issues
12476	Fixed	Indicator	Resolved some scenarios where Order Flow VWAP could plot unexpectedly
12453	Fixed	Chart	Global Cross Hair on two or more charts with an indicator's scale justification set to left could cause the 2 charts to report different times
12480	Fixed	Chart	Changing series via toolbar quick selector could fail in some multi-series chart scenarios
12490	Fixed	Chart	Disconnecting then reconnecting with a multi-series equidistant chart could sometimes result in 'MultiEqSaveTimes' error reported in the log
12541	Fixed	Chart	In some scenario the chart could get stuck loading after disconnect/reconnect cycle
12489	Fixed	Chart Trader	Using the keyboard 'Enter' button on an order modification on Chart Trader would use a prior QTY value
1242	Fixed	Chart, BarsTy	Line break charts with second base periods did not wait the specified time to receive the close

7		pe	value
12554	Fixed	Chart, DrawingTool	Drawing tools did not appear in list when copied from chart with different data series
12539	Fixed	DrawingTool	Draw objects copied between charts will no longer have offsets applied as it does for repeat copies to the same chart
12540	Fixed	Esignal	Equity volume had incorrect multiplier applied
12503	Fixed	Indicator	Changing the font of the chart had no effect on BarTimer, VolumeCounter, and TickCounter
12367	Fixed	NinjaScript	Implemented convenience method DrawObjects.ToList() which should be used when iterating over Drawing Objects in the chart
12530	Fixed	Indicator	NinjaScript draw objects could disappear when deleting a separate indicator from a chart with an overlay scale justification
12501	Fixed	Indicator	Some Indicators wouldn't draw if bar time was the same for multiple bars
12502	Fixed	Indicator	Volume Counter indicator did not work with Volumetric and Heiken Ashi Bars
12577	Fixed	Indicator	When multiple overlay indicators shared a secondary panel and one of them was made invisible the others could be moved to the primary panel
1254	Fixed	Interactive	Forex position in sub accounts were not shown when connected to adviser account

7		Brokers	
12553	Fixed	Kinetick	Resolved a case were a connection loss would reconnect but not stream market data
12394	Fixed	Market Analyzer	Alerts could trigger when cross above/below conditions were not true
12426	Fixed	Market Analyzer	Input series price type was not saved in workspace or presets
12487	Fixed	News	If an instrument list filter was applied, it would not restore properly on loading the workspace
12513	Fixed	NinjaScript	Region and Plots could be misaligned for bars with same time-stamp
12516	Fixed	NinjaScript	Regression channel could draw improperly if a multi-series indicator was used as the input
12408	Fixed	NinjaScript	Saving an indicator in Visual Studio removed the indicator from the editor and created a temp folder
12496	Fixed	Playback	Historical Playback's Go To feature would not adjust chart time for proper playback
12349	Fixed	Playback	In some scenarios if a script got an error then play was pushed on Playback a crash could occur
12424	Fixed	Share Adapter	Emailing to support could result in an error if files were too large of character length was too long

12559	Fixed	Strategies	MIT profit order could not be adjusted using AtmStrategyChangeStopTarget
12219	Fixed	Strategies	In some scenarios on historical fill backtesting, 2 orders filling on the same bar could yield an unexpected price for one of the two orders
12525	Fixed	Strategies	Uniquely named order could be ignored per EntriesPerDirection despite flat position on multi-series strategy
12212	Fixed	Strategies	In a back-test stop limit orders had slippage applied that should only be for market orders
12551	Fixed	Strategies	Properties with the same name but different group names resulted in only one appearing in the Optimizer
12564	Fixed	Strategies	Slippage on market orders was stricter than expected and didn't account for bars full low/high
12425	Fixed	Strategies	Strategy templates would load correctly but display incorrectly in Strategies tab
12435	Fixed	Strategy Builder	Selecting next and back repeatedly could result in an error
12403	Fixed	SuperDOM	Resolved a scenario in which canceling orders from a strategy could result in a null reference error
12415	Fixed	SuperDOM	When using a specific license configuration, it was not possible to select non-futures instruments in the Dynamic SuperDOM
1235	Fixed	TD Ameritr	Resolved some cases in which data would not stream as expected

4		ade	
12413	Fixed	TD Ameritrade	External GTD orders did not properly receive the associated date
12545	Fixed	TD Ameritrade	News headlined were being requested too frequently
12428	Fixed	Time and Sales	Below bid/Above ask highlights were not being detected
12458	Fixed	Workspaces	Default End Of Day workspace would save although a custom workspace was saved

8.0.13.1 Release Date
March 29, 2018

Issue #	Status	Category	Comments
12367	Fixed	NinjaScript	DrawObject.ToList() method returned a parameter type that clashed with the LINQ extension method for .ToList(), which is already in use by NinjaScript could result in a compile error
12368	Fixed	Chart	Trying to select a custom ChartStyle in the DataSeries dialog when you had custom ChartStyles installed would result in errors

3.3.17 8.0.12.0

Release Date
January 30, 2018

Feature #	Status	Category	Comments
11943	Added	Bars Type	Added Order Flow - Volumetric Bars. This feature requires a lifetime license. This feature is in preview/beta.
12264	Added	Indicator	Added Order Flow - Depth Map. This feature requires a lifetime license. This feature is in preview/beta.
12275	Added	Indicator	Added Order Flow - VWAP. This feature requires a lifetime license. This feature is in preview/beta.
12200	Added	Indicator	Added Order Flow - Cumulative Delta. This feature requires a lifetime license. This feature is in preview/beta.
12274	Added	Indicator	Added Order Flow - Trade Detector. This feature requires a lifetime license. This feature is in preview/beta.
12400	Added	Instruments	Added support for exchanges: SHFE (ShangHai Futures Exchange), DCE (DaLian Commodity Exchange), CZCE (ZhengZhou Commodity Exchange), SHSE (ShangHai Security Exchange), SZSE (ShenZhen Security Exchange)
12168	Added	Localization	Added 'Chinese (Simplified)' localization

Issue #	Status	Category	Comments
12317	Fixed	Alerts	Selecting an alert generated from an indicator in the Alerts window could result in an error

12377	Fixed	Alerts	Restored Price Type selection on Alerts, Ask/Bid "price type" were no longer be selectable
12320	Fixed	ATM Strategies	ATM strategy only did not display the correct position color when using Display selected ATM strategy only mode in combination with simultaneous virtual long and short position.
12333	Fixed	ATM Strategies	A stack overflow message could occur when using reverse at stop and reverse at target simultaneously on the same ATM strategy
12344	Fixed	ATM Strategies	ATM stop strategy parameter behaved as tick mode when it should have been pip mode when editing an active ATM strategy via the right-click menu
12246	Fixed	Bars, NinjaScript	Resolved a scenario where multiple inflight requests for historical data on the same instrument with different request periods (e.g. 5 days back vs 10 days back) could result in data corruption.
12316	Fixed	Chart	Data box would freezes when generating new workspace with "Save As"
12324	Fixed	Chart	Interval selector did not show custom bar's names as expected
12357	Fixed	Chart	When using some combinations of time frames the global crosshair did not work as expected
12379	Fixed	Chart	Disabling Break at EOD prevented renko and range bars from loading data before the end of last year
12395	Fixed	Chart	Reordering chart tabs that have strategies with plots running in the background resulted in error messages and lost bars

12368	Fixed	Chart, Indicator	In some dialogs restoring indicators presets continued to use previously saved settings
12378	Fixed	Control Center, Strategy	Generating a Trade Performance then disabling a strategy and disconnecting quickly would still showed the strategy as enabled
12353	Fixed	CQG	Minor CQGT API Update required to fix wrong formatting on ^TRIN
12371	Fixed	Data	Importing data between a source time zone with a daylight saving times and a target timezone without daylight savings could result in a incorrect historical data
12386	Fixed	Drawing	Drawing objects with chart anchors in the future were removed when disconnecting from a live data feed
10013	Fixed	Drawing Tool	Resolved a scenario where drawing tools would fail to draw when called from a script
12321	Fixed	Drawing Tool	Resolved a scenario of a D2DERR_WRONG_STATE error
12372	Fixed	Drawing Tool	Moving a horizontal line was not updating the end anchor price in the properties
12387	Fixed	Drawing Tool	Trend Channel displays the Trend and Parallel in both the Properties and Levels
12328	Fixed	eSignal, Simulator	An error could occur when modifying a profit target or stop loss while the chart was loading
12352	Fixed	FX Pro	CFD prices displayed with incorrect rounding
12329	Fixed	Google, Market Analyzer	Loading fundamental data in some cases resulted in an error

12331	Fixed	In Product Announcement	Resolved a scenario where in product announcements could be repeated
12396	Fixed	Interactive Brokers	Modifying externally placed orders would get stuck in pending change
12402	Fixed	Level II Window	Enabling Size divided by 100 (stocks only) did not persist after a restart
12315	Fixed	Localization, Alerts	Alerts log did not show indicator alerts in several languages
12364	Fixed	NinjaScript	Resolved a case where removing a blank line of code resulted in errors
12399	Fixed	NinjaScript	Control Center and add-ons were not being closed as expected when shutting down NinjaTrader
12297	Fixed	Orders	High frequency order changes triggered from SetTrailStop could cause order change rejections on the provider
12343	Fixed	Orders	Max Position Size would at times incorrectly restrict ATM orders that were valid
12348	Fixed	Playback	Unchecking Playback current day only could prevent Playback controller from functioning as expected
12355	Fixed	Strategies	Argument Exception errors occurred when many strategies were enabled all at once
12342	Fixed	Strategy	In some scenario's High Order Fill Resolution would cancel a profit target or stop loss before intended

12361	Fixed	Strategy	In some scenarios modifying the quantity of stop market orders would fill the original size
12339	Fixed	Strategy Analyzer	Optimization fitness options were not saved with Multi-Objective Optimization template
12350	Fixed	Strategy Builder	Using variables that held the name 'True' or 'False' at the beginning of the variable name were incorrectly handled
12354	Fixed	TD AMERITRADE	Resolved some scenarios of data streaming/connection losses and improved reconnection handling
12347	Fixed	TD AMERITRADE	IndexOutOfRangeExceptions could result in data no longer streaming
12375	Fixed	Tick Replay	Multi-series Tick Replay with Calculate.OnBarClose could give wrong values for bid/ask historically
12365	Fixed	Trade Performance	When generating a Trade Performance report NinjaTrader could become unresponsive
12366	Fixed	Workspaces	Strategies were removed from strategies tab after multiple restarts

3.3.18 8.0.11.1

8.0.11.0 Release
December 6, 2017

Feature #	Status	Category	Comments
10314	Chan	Chart	Cross-hair performance improvements

	ged		
12224	Added	Chart Trader, Hot Key	Added a Chart Trader (Hidden) Hot Key
12221	Changed	CQG, Continuum	Updated CQG and Continuum adapter to version 7.00.704. **CQG/Continuum adapter is in beta**
12098	Added	Rithmic	Added Rithmic "MDP 3.0 Summary" connection point option
12262	Added	ShareAdapter	Added support for 280 characters to Twitter share adapter

Issue #	**Status**	**Category**	**Comments**
12302	Fixed	eSignal	Futures contracts in 2018 did not provide real-time data. **Required update for eSignal users**
12245	Fixed	TD AMERITRADE	Resolved some scenarios where orders would get stuck in Submitted or ChangeSubmitted
12312	Fixed	Account Data	There was no message that a reconnect is required when changing account denomination
12186	Fixed	Account Performance	Sharpe / Sortino ratio were not calculating as expected
12190	Fixed	Alerts	Display text was inconsistent for numeric value conditions between

			Chart alert and Market Analyzer alert
12215	Fixed	Alerts	Indicators created on close could not trigger alerts using ask/bid price type
12237	Fixed	Alerts	In cases of having lots of alerts that were enabled at start up the dialog window could be too large to select desired options
12294	Fixed	ATM Strategies	Could not fully disconnect after entering ATM
12279	Fixed	ATM Strategies	Auto trail did not work as expected when the stop was manually modified
12086	Fixed	ATM Strategies, Attach Order To Indicator, SuperDOM	Indicator tracking disable warning didn't trigger as expected
12180	Fixed	ATM Strategies, Orders	Exiting ATMs which were scaled in with exit orders which had part fills could get an error
12164	Fixed	Bars	Tick based charts could change with some combinations of 'Merge policy' and 'Break at EOD'
12122	Fixed	Chart	Multi series chart could exhibit incorrect plot behavior
12133	Fixed	Chart	Multi-series equidistant chart had unexpected plot behavior in future
12134	Fixed	Chart	Bar time series repositioned when switching tabs
12141	Fixed	Chart	Auto scale was not working properly on displaced plots

12167	Fixed	Chart	Data Series Label changed when switching instruments
12174	Fixed	Chart	Heikin-Ashi preset days to load as 3 rather than 5 on first load
12175	Fixed	Chart	Adjusting the vertical scale at first would move in the wrong direction
12178	Fixed	Chart	Down bar outline did not work for Open/Close chart style
12194	Fixed	Chart	Some Trading hours with similar start/end times going from this year to the next time-stamped yearly bars differently
12205	Fixed	Chart	Chart rendering to a software bitmap resulted in an error
12208	Fixed	Chart	Pivot label box not wide enough to display some larger fonts
12226	Fixed	Chart	Always On Top chart with Indicators dialog open still received text input and opened instrument selector overlay
12229	Fixed	Chart	In some scenarios of drawing into the future on multi-time frame charts performance could degrade
12230	Fixed	Chart	Removed unexpected opacity setting from Chart Properties> Lines
12235	Fixed	Chart	Selecting panel top/bottom borders to re-size could be difficult
12241	Fixed	Chart	Using the interval link and then selecting a weekly chart prevented linking from working

12281	Fixed	Chart	In some scenarios custom chart intervals were not switching the interval
12295	Fixed	Chart	In some scenarios the scroll-bar could be lost on non-equidistant charts
12313	Fixed	Chart	Same time-stamped indicator plots only showed plot value in the Mini Data Box for the first time-stamp
12210	Fixed	Chart Trader	On multi instrument charts, non-market orders could not be placed to non-primary data series
12188	Fixed	Chart, Drawing	Draw objects could be unselectable on high bar spacing non equidistant chart setups
12120	Fixed	Chart, Indicators	Changing indicator input series could result in plots not being drawn in correct panel
12114	Fixed	Chart, NinjaScript	In some scenarios OnRender() index error resulted in a blurry chart
12252	Fixed	Chart, NinjaScript	Resolved some scenarios where resizing or dragging a chart with scripts using OnRender could get an error
12256	Fixed	Chart, Playback	In some scenarios switching between a data provider and playback could result in a gap on the chart
12258	Fixed	Commissions	Commissions were not applying to Chart Strategy Performance reports
12271	Fixed	Control Center	Orders tab did not allow arranging or removing of Instrument column

12140	Fixed	Control Center, Orders	Resolved some scenarios of orders from past days showing in the Control Center
12160	Fixed	Control Center, Strategy	Strategy grid was not updating when editing strategy to an instrument list
12093	Fixed	Data Grids	In some scenarios Filter By Account did not work as expected
12300	Fixed	Database	Resetting the data base showed an error in the trace on simulation accounts
11959	Fixed	DrawingTool	Regression channel anchors were not working as expected on daily charts or higher
12084	Fixed	DrawingTool	Drawing objects did not adopt snap to bar snap mode
12101	Fixed	DrawingTool	Global drawing objects moved over time
12124	Fixed	DrawingTool	Resolved some scenarios of region index out of range errors
12135	Fixed	DrawingTool	Modifying global TrendChannel could freeze the chart
12148	Fixed	DrawingTool	Region could not be aligned with Plot as expected
12232	Fixed	External	Custom instruments had an error upon opening chart when connected to External
12165	Fixed	Forex.com	Could not connect to multiple Forex.com connections when using NinjaTrader's historical data servers

12136	Fixed	Google	LSE instruments were not able to be mapped as expected
12263	Fixed	Google	Switching between daily and monthly charts could result in only 1 bar
12266	Fixed	Google	DAX30 symbols with 'Xetra' exchange enabled did not load
12268	Fixed	Google	Index instruments could not be loaded outside US region
12131	Fixed	Indicator	Custom preset caused missing 'panels' in the 'panel' property drop down list
12151	Fixed	Indicator	ZigZag could not displace
12192	Fixed	Indicator	Regression Channel's Upper and Lower bands became improperly scaled when another indicator with ScaleJustification set to Overlay was applied to the same pane
12251	Fixed	Indicator	Value Plot property was blank for indicators with unnamed plots
12253	Fixed	Indicator	The only selectable plot for Volume Up/Down was the up plot
12283	Fixed	Installer	Resolved a scenario where a migration would not complete
12137	Fixed	Instruments	In some scenarios the Point Value for index and stock instruments could be set to something other than 1
12157	Fixed	Instruments	MICD, MIJY, MISF futures were not rolling over

12088	Fixed	IQFeed, Kinetick	In some scenarios some ticks were not being received
12285	Fixed	Localization	Resolved a scenario where a backup couldn't be completed with some localization settings
12079	Fixed	Market Analyzer, Interactive Brokers	Live accounts were not populating PnL columns
12156	Fixed	News	The news reading pane would not show when Show Reading Pane was checked
12100	Fixed	NinjaScript	In some scenarios exported assemblies could not import
12118	Fixed	NinjaScript	Profit Targets and Stop Losses submitted to the secondary instrument expired and canceled
12176	Fixed	NinjaScript	In some scenarios assemblies with drawing tools could not apply templates
12191	Fixed	NinjaScript	Bars.PercentComplete did not work across session boundaries
12209	Fixed	NinjaScript	Resolved a scenario where strategies could crash on loading
12211	Fixed	NinjaScript	Resolved some scenarios of NinjaScript labels not displaying as expected
12217	Fixed	NinjaScript	Resolved a scenario where adjusting a property grid on a multi instrument strategy could result in an error

12119	Changed	NinjaScript	CreateOrder() method has added TimeInForce and OrderEntry parameters. The older signature is now deprecated but will remain for backwards compatibility
12240	Fixed	NinjaScript	Resolved a scenario where applying a template made from an enabled strategy could result in an error
12248	Fixed	NinjaScript	Resolved some scenarios where indicators adding a data series could result in lock ups
12249	Fixed	NinjaScript	Accessing Instrument outside OnBarUpdate context could trigger Index exception
12273	Fixed	NinjaScript	VolumeZones was not properly set to IsChartOnly=True
12276	Fixed	NinjaScript	Custom namespace folders displayed in incorrect order
12246	Fixed	NinjaScript, Bars	In some scenarios or requesting bars the number of bars returned could change when reloading historical data
12129	Fixed	NinjaScript, Strategy	BarsInProgress value changed inside OnBarUpdate with multi-series strategies
12184	Fixed	Orders	Unexpected order price occurred on error when reusing an old OCO ID
12297	Fixed	Orders	High frequency order changes could cause order change rejections on the provider
12234	Fixed	Playback	Resolved some scenarios where drawing objects would be removed

			when moving the slider forward or disconnecting
12154	Fixed	Strategy	In some scenarios Synchronize All Strategies did not sync as expected
12162	Fixed	Strategy	Resolved a scenario where a strategy would lock up the Control Center if ran there but if ran from a chart it worked
12166	Fixed	Strategy	Strategies grid was not loading default Days to Load
12170	Fixed	Strategy	Disabling/enabling strategies that track and cancel orders could get error and have orders stuck in CancelPending
12236	Fixed	Strategy	Canceling applying a template to an enabled strategy could unexpectedly make properties editable
12307	Fixed	Strategy	Adding indicators from a strategy would always apply to the first panel
12115	Fixed	Strategy Analyzer	Walk Forward Optimization slippage was not populating
12177	Fixed	Strategy Analyzer	Resolved a scenario where double clicking on strategy results resulted in a value cannot be null error
12196	Fixed	Strategy Analyzer	Resolved a scenario where a backtest would hang
12213	Fixed	Strategy Analyzer	Average time in market from Results was not matching the Summary
12238	Fixed	Strategy Analyzer	SetStopLoss and SetProfitTarget orders cancelled when using a high order fill resolution if the secondary series was more granular than primary

12242	Fixed	Strategy Analyzer	Attempting to add an indicator to the Chart after applying a template resulted in an error
12270	Fixed	Strategy Analyzer	Average Time in Market in exported spreadsheet had incorrect values
12272	Fixed	Strategy Analyzer, Chart	Resolved a scenario where the last bar would update unexpectedly
12083	Fixed	Strategy Builder	In some scenarios Inputs and Variables that were in use could be edited which resulted in an error
12096	Fixed	Strategy Builder	Strategy Builder was not filtering OnBarUpdate to process primary only
12153	Fixed	Strategy Builder	Compiler Errors occurred with additional Data Series as input for indicator
12163	Fixed	Strategy Builder	Resolved a scenario where selecting an input resulted in an error
12172	Fixed	Strategy Builder	With language set to German adding Addition Data of Daily loaded Tick instead
12193	Fixed	Strategy Builder	Using Load in condition and actions windows resulted in an error
12243	Fixed	Strategy Builder	Duplicating the Conditions tab resulted in an error
12265	Fixed	Strategy Builder	Localization settings were not working in the Strategy Builder as expected
12110	Fixed	Strategy, SuperDOM	NinjaScript strategy orders modified in the SuperDOM were not synchronizing the position

12287	Fixed	Strategy, UI	Switching input series to a chart applied strategy from the Control Center did not update the display of what input series it was applied to
12138	Fixed	Trade Performance	Orders section was unable to filter items
12139	Fixed	Trade Performance	Instrument Type Filters would get removed after unchecking and generating filters
12152	Fixed	Trade Performance	In some scenarios MAE, MFE, ETD were not reported correctly after re-connecting
12290	Fixed	Workspaces	Control Center's "Always on top" setting was not saved to UI.xml
12291	Fixed	Workspaces	Alerts Log priorities were not saved with workspace on restart for current tab

8.0.11.1 Release
December 13, 2017

Issue #	Status	Category	Comments
12335	Fixed	CQG, Continuum	Required data was missing for Order Modification and Order Cancellation messages.
12327	Fixed	Market Analyzer	Saved templates would not maintain custom column labels.

3.3.19 8.0.10.0

Release Date
November 7, 2017

Issue #	Status	Category	Comments
12239	Fixed	Rithmic and MB Trading	Logic to parse the decade on a futures instrument will expire in 2018 for Rithmic and MB Trading connection technologies. Resulting in the wrong decade being resolved for any 2018 futures expiry. **Required update for all Rithmic and MB Trading users, please see Advisory #32 for more info.**

3.3.20 8.0.9.0

Release Date
September 12, 2017

Attention existing NinjaTrader 8 Users: As a consequence of a bug fix in 8.0.9.0, any indicators in a Market Analyzer that have non-primary plots selected, saved in a workspace, will be set back to the primary plot. Users will need to re-select the desired plot and re-save the workspace

Feature #	Status	Category	Comments
12090	Added	Localization	French language support introduced

Issue #	Status	Category	Comments

11958	Fixed	Market Analyzer, Indicator	Some indicators did not have the option of what plot to select
11931	Fixed	ATI	Was unable to cancel order with OIF file
11933	Fixed	ATI	Reversing a position in an OIF file resulted in a crash
11975	Fixed	ATM Strategies	Canceling creation of a custom ATM then placing an order could attempt to apply an ATM
12040	Fixed	ATM Strategies	In some scenarios the ATM Strategy Selection Mode did not function as expected when switching instruments
11634	Fixed	Bars	Resolved some scenarios that resulted in cache data errors
11911	Fixed	Chart	Creating a monthly chart with insufficient days back resulted in an error
12071	Fixed	Chart Trader	Reverse was not using the selected TIF
12032	Fixed	Chart, Indicators	Indicator with Draw.Ray or Draw.Text was resulting in bars margin to be indented left
11681	Fixed	Chart, Indicators	Chart with rendering indicator could freeze when switching instruments
12048	Fixed	Chart, Strategies	In some scenarios options were greyed out when attempting to add multiple strategies to a chart
11983	Fixed	Chart, Workspaces	Resolved some scenarios where the bar series wouldn't keep right side margin when workspace was restored

12025	Fixed	CQG	Close price was not getting adjusted by settlement
11919	Fixed	Data	In some scenarios combinations of PC time zone and chart type could result in cached data not matching historical
12064	Fixed	Data, NinjaScript	Loading data based on days, then bars, then days again could result in inconsistent 2nd series prints
11660	Fixed	Drawing	Global draw objects could be lost on panel drag / scale justification change
12009	Fixed	Drawing	In some scenarios charts with large time frames and drawing objects into the future resulted in chart lag
11916	Fixed	DrawingTool	Bars would no longer snap when anchors moved off chart
11952	Fixed	DrawingTool	Ruler would not show count of bars until time-stamp was no longer the same
12008	Fixed	Forex.com	Resolved some scenarios where an order could get an error saying the results are invalid
11815	Fixed	Google, Instruments	Bovespa stocks did not load daily data
11928	Fixed	Google, Market Analyzer	Average Daily Volume column resulted in an error
12004	Fixed	Historical Data Window	Exporting bid/ask tick data showed fake values rather than zeros for data type columns other than what was exported
11834	Fixed	Indicator	Regression Channel did not displace

11964	Fixed	Indicator	Restoring indicator settings to default could remove Panel selection
12005	Fixed	Indicator	Indicators that have public series would not return as an indicator
12052	Fixed	Indicator	Max and min 'indicators' with renko bars returned an invalid value
12067	Fixed	Indicator	In some scenarios large ZOrder values were ignored
11929	Fixed	Instruments	Sorting on instrument type in the Instrument Editor wasn’t alphabetical
11942	Fixed	Instruments	Attempting to edit the Description field of forex instruments resulted in an error
12020	Fixed	Instruments	Instrument selector was not always selecting the default exchange when using the search icon
11895	Fixed	Kinetick	Historical data was receiving some non-last-qualifying data
11978	Fixed	Localization	German localization was resulting in an error with some log messages
12082	Fixed	Market Analyzer, Indicator	Accessing the Indicator column with some custom scripts could result in an error
11894	Fixed	NinjaScript	Referencing BarsArray during OnRender then switching bars type caused a crash
11912	Fixed	NinjaScript	Description for some indicators was missing
11914	Fixed	NinjaScript	Script could be enabled in non connected state in some scenarios

11927	Fixed	NinjaScript	Suspended indicators were waiting for a new tick before resuming
11934	Fixed	NinjaScript	TraceOrder was showing more output prints than expected
11938	Fixed	NinjaScript	OnRenderTargetChanged for ChartStyles was resulting in an error
11966	Fixed	NinjaScript	Multi-series indicators prevented proper syncing with any Series<T> constructor for primary series
11976	Fixed	NinjaScript	Exceptions for DrawingTools OnStateChange() was not being logged
11981	Fixed	NinjaScript	OpenCloseStyle was not able to use GetCandleOutlineOverrideBrush
11982	Fixed	NinjaScript	Resolved a scenario where Order Fill Resolution couldn't be changed
12022	Fixed	NinjaScript	Overloads were missing from <Anchor>.MoveAnchor()
12074	Fixed	NinjaScript	If a sound was not located and threw an error, that error was not automatically handled by NinjaTrader
11856	Fixed	NinjaScript, Chart	In some scenarios a secondary series added to a chart could change behavior of NinjaScript
11924	Fixed	NinjaScript, Chart	Chart rendering would fail with a specific sequence of loading NinjaScript and enabling the Chart Trader
11941	Fixed	NinjaScript, Chart	Panel MIN MAX began calculating incorrectly from plot values with small granularity

11953	Fixed	NinjaScript, Chart	Moving a strategy indicator to a new panel resulted in an error
12026	Fixed	NinjaScript, DrawingTool	Chart drawing objects would disappear at times when the chart time was shifted
12054	Fixed	NinjaScript, Market Analyzer	Referencing Indicator series from Market Analyzer column was resulting in an error
12006	Fixed	NinjaScript, Orders	In some scenarios a High Order Fill Resolution could result in duplicate close orders
12065	Fixed	NinjaScript, Playback	Error could occur when using BarsSeries.GetClose
11915	Fixed	NinjaScript, Tick Replay	Changing set order quantity let you select Order Fill Resolution when Tick Replay was enabled
11898	Fixed	NinjaScript, DrawingTool	Having a chart with a script that places drawing objects and then switching instruments could results in an error
11985	Fixed	Options	Print Hot Keys was not listing all categories
11940	Fixed	Orders	Adjusting target while stop filled could result in order stuck in pending change if trading in simulation
11920	Fixed	Playback	Resolved some scenarios where Playback could stop auto scrolling forward and/or strategy stopped placing trades
11918	Fixed	Playback, NinjaScript	Drawing Lines attached to last price could detach from price and error
12023	Fixed	Playback, NinjaScript	NinjaScritps could get error on rewind of data

12051	Fixed	Property Grids	Removing Label from strategy had unexpected results of how the strategy label should display
11962	Changed	ShareAdapter	Updated Facebook Share Adapter API
11171	Fixed	Simulator	In some scenarios realized PnL reported double or halved values
11956	Fixed	Strategy Analyzer	Drag and drop of an indicator into the y-axis of another panel didn't move the indicator
11989	Fixed	Strategy Analyzer	Columns for the Results were not saving with preset template or workspace
12002	Fixed	Strategy Analyzer	Prints displayed twice in the Output window when a backtest is run with the Display Selector set to anything but Summary or Settings
12003	Fixed	Strategy Analyzer, Chart	In some scenarios strategy applied indicators could reset the panel they are applied to when accessing settings
11892	Fixed	Strategy Analyzer, NinjaScript	Optimizing a script that called PriorDayOHLC caused an error when IsInstantiatedOnEachOptimizationIteration was set to false
11917	Fixed	Strategy Builder	Using the TRIX indicator as an input series generates a message that strategy could not compile when the strategy did properly compile
12007	Fixed	Strategy Builder	The ResourceType property of DisplayAttribute was incorrectly being used for strings not defined in NinjaTrader

12080	Fixed	Strategy Builder	Using Pivots required adding daily bars even when selecting calculate from intraday data
11922	Fixed	Strategy Builder	Optimization Fitness Wizard had an invalid Input Parameter
12045	Fixed	SuperDOM, ATM	Display Selected ATM Strategy Only was not showing PnL for simulation accounts
12049	Fixed	SuperDOM, NinjaScript	Adding multiple volume columns and then switching instruments while reloading data/scripts could result in an error
11926	Fixed	Tick Replay	At times there was an error when using multi-series and switching instruments
11889	Fixed	UI	Resolved some scenarios where interacting with a message box on start up flashed all windows caption bars multiple times
11831	Fixed	Window Linking	Indicators could be plotted incorrectly / not update when chart symbols got quickly changed from Market Analyzer

3.3.21 8.0.8.0

Release Date
July 26, 2017

Feature #	Status	Category	Comments
11864	Done	eSignal	Required version of eSignal has been updated to 12.7.4.5.40
11744	Added	MES Capital	MES Capital Simulation Trading connection added

Issue #	Status	Category	Comments
11552	Fixed	Data	Resolved a scenario where recording live data as historical could corrupt historical data cache
11802	Fixed	ATI	Passing a value of zero was changing the stop price to zero rather than leaving it unmodified for Order Instruction Files
11891	Fixed	ATM Strategies, Chart Trader	In some scenarios shutting down with a workspace that had Chart Trader and an ATM could result in an error
11853	Fixed	BarsType, NinjaScript	Heiken-Ashi bars returned too many bars when added to script
11565	Fixed	Chart	Point and figure charts were not properly displaying on 4k monitors with display zoomed
11767	Fixed	Chart	Using multiple monitors with different DPI settings was resulting in an error when restoring minimized charts
11879	Fixed	Chart	In some scenarios a memory leak was occurring after a reload/close of an existing chart
11816	Fixed	Chart Trader, Drawing Tool	Attempting to select an order that is at the same price as a horizontal line was sometimes selecting the line rather than the order
11808	Fixed	Chart, Databas	Rolling over a chart with multiple tabs of different instruments could select the

		e	wrong instrument in the instrument selector
11759	Fixed	Chart, Strategy	In some scenarios adding a strategy to the chart resulted in an error and removed bars
11826	Fixed	Chart, Tick Replay	Resolved some scenarios where a chart couldn't load from cache after changing the chart Type
11765	Fixed	Control Center	Orders tab showed incorrect date/time format
11854	Fixed	Control Center	Connection status indicator was not always displaying the correct connection color
11756	Fixed	Core, NinjaScript	Sending 2 different emails to the same share service from on bar update resulted in an error
11791	Fixed	Data, Backup & Restore	Historical data for custom instruments was not being restored
11671	Fixed	Drawing	Resolved some scenarios where global drawing objects could result in the error "Collection was modified; enumeration operation may not execute"
11760	Fixed	Drawing Tool	Fibonacci inner lines were not drawn to exact price displayed
11774	Fixed	Drawing Tool	Fibonacci extension lines plotted incorrectly when chart scale was set to logarithmic
11778	Fixed	Drawing Tool	Text drawing object could not be selected when chart's right scale type was logarithmic

11782	Fixed	Drawing Tool	Resolved multiple items related to the anchors and selection of the Regression Channel not functioning as expected
11790	Fixed	Drawing Tool	When setting the chart scale to logarithmic and using multiple panels in some scenarios you could not place a drawing object
11906	Fixed	Drawing Tool	Fibonacci Tools allowed opacity values outside of the expected range
11757	Fixed	Drawing Tool, Chart	In some scenarios global drawing objects were not showing a tag in Drawing Objects window
11781	Fixed	Drawing Tool, Chart	When moving some drawing tools on a logarithmic scaled chart the cursor will move away from the drawing object
11855	Fixed	Drawing Tool, Chart	Setting an anchor point into the future would not plot at the correct time
11833	Fixed	Drawing Tool, NinjaScript	ChartAnchor was incorrectly created in Fibonacci and shapes drawing tools which could result in errors with some scripts
11830	Fixed	Drawing Tool, Strategy, Strategy Analyzer	Running a strategy with a drawing tool from the Strategy Analyzer or Strategies tab was resulting in an error
11668	Fixed	Historical Data Window	Disconnecting/reconnecting while downloading data resulted in loading message not going away
11780	Fixed	Historical Data	In some scenarios editing an instruments settings removed it's saved historical data

		Window, Instruments	
11805	Fixed	Indicator	CandlestickPattern was not using State.Historical for Chart Control
11845	Fixed	Instruments	Instrument Selector would only work for equities that have Default checked for the exchange
11899	Fixed	Licensing, UI	Vendor Licensing allowed invalid characters to be used
11803	Fixed	Market Analyzer	Adding an indicator column was inconsistently selecting different indicators by default
11838	Fixed	Market Analyzer	In some scenarios added columns were wider than expected
11800	Fixed	Market Analyzer, NinjaScript	MTF indicator could provide incorrect results when reloading NinjaScript
11807	Fixed	NinjaScript	Updated Help Guide's Add On Framework example that was resulting in a lock up when clicking Connect
11884	Fixed	NinjaScript	BarsToLoad on secondary bar series was using primary bar series BarsToLoad
11763	Fixed	NinjaScript, Playback	Subscribing to Market Data could prevent playback from progressing
11849	Fixed	Playback	Recording playback data could get added into the previous days playback data in

			some scenarios
11893	Fixed	Playback	Resolved a scenario where the Playback controller could get stuck
11466	Fixed	Playback, NinjaScript	Resolved some errors that could occur when working with multiple custom data series
11650	Fixed	Playback, UI	Resolved some scenarios of adjusting data series not functioning as expected while playback controller is resetting
10257	Done	Regionalization	Updated and extended translations to other languages
11797	Fixed	Regionalization, Data	Resolved an error that could occur when importing data with UTC+1 timezone
11740	Fixed	Strategy	AdoptAccountPosition was not recognizing orders previously submitted from strategy
11828	Fixed	Strategy Analyzer	Setting Input Series of an indicator to Default Input displayed as Close although it was properly Default Input
11839	Fixed	Strategy Analyzer	Strategy statistics were not consistent in same scenarios (MAE, MFE, ETD)
11818	Fixed	Strategy Analyzer, Chart	Going to Logs and viewing a chart from a previous backtest that used a custom session template did not display with the custom session template
11821	Fixed	Strategy Analyzer, Drawing Tool	In some scenarios a strategy using drawing tools were resulting in an error

11738	Fixed	Strategy Builder	Editing Enter Position settings was producing an error in some scenarios
11824	Fixed	Strategy Builder	Group Conditions was not showing secondary series
11851	Fixed	Strategy Builder	Resolved a scenario where compiling and have the Strategy Builder open could result in an error
11905	Fixed	Strategy Builder	Duplicating a Set was adding a space to the draw object's tag ID
11785	Fixed	Strategy, Control Center	In some scenarios using multiple strategies on the same instrument could display an average price to a strategy without a position in the Strategies tab
11858	Fixed	SuperDOM	APQ values did not clear after an order was moved
11723	Fixed	TD AMERITRADE	In some scenarios orders status was not being updated on submitted orders
11829	Fixed	Templates, Chart	Applying a chart template was disabling Tick Replay
11877	Fixed	Trade Performance	Filter items do not display in alphabetical order
11784	Fixed	Visual Studio Integration	Was unable to restore NinjaScript backup after saving solution with Visual Studio
11888	Fixed	Workspaces	In some scenarios a message displayed to reposition windows off the screen when no windows were off the screen

3.3.22 8.0.7.1

8.0.7.0 Release

June 6, 2017

Feature #	Status	Category	Comments
11562	Added	Control Center	When the Control Center's width is reduced menu items now change to icons
11597	Added	Forex.com	Added support for account denomination mapping to CHF, JPY, USD, EUR, GBP
11545	Added	FX Board	Add ability to add instruments using the keyboard
11549	Added	Hot Key	Added additional Hot Key functions for tabs
11672	Added	Ninja Script Editor	Add Support For Visual Studio 2017 (requires 15.2 or later)
11746	Added	Regionalization	Added regionalization support for Portuguese and completed more Spanish regionalization

Issue #	Status	Category	Comments
11476	Fixed	Alerts	In some scenarios Alerts wouldn't activate

11580	Fixed	ATI	Bars building from data received from LastPlayback with an API was building inconsistently
11488	Fixed	ATM Strategies	In some scenarios an 'unable to load ATM strategy template file' error could occur
11745	Fixed	ATM Strategies	Scaling out of an ATM did not properly modify order quantities
11750	Fixed	ATM Strategies	Large quantities could result in OCO not applying to all orders
11717	Fixed	ATM Strategies, CQG	In some scenarios an inflight execution could exit a position and send a close order
11574	Fixed	Bars, Chart	Error occurred when using a Default 24x7 daily chart with Brasilia timezone
11603	Fixed	Bars, Chart	Crosshair label was inconsistent with x-axis label on Brasilia time with Default 24x7 hours
11724	Fixed	Bars, Instruments	Forex data displayed zeros if tick size was adjusted
11704	Fixed	Bars Type, Chart	Removed custom bars type could still be referenced by workspace
11470	Fixed	Chart	Canceling instrument switch with applied alert resulted in wrong instrument in drop down menu

11514	Fixed	Chart	Clicking search button on a chart searched for the data series
11525	Fixed	Chart	Zoom icon and zoom box could display unexpectedly in some scenarios
11605	Fixed	Chart	Copy and Paste menu items didn't display hot key
11653	Fixed	Chart	X-Axis time in the future plotted incorrectly on multi-time-frame chart with different period types
11667	Fixed	Chart	In some scenarios opening a second chart tab resulted in an error
11694	Fixed	Chart	Sub-menu stayed open after item was selected if it was in an overflow panel
11699	Fixed	Chart	Importing minute data and generating day bars resulted in duplicate or incorrectly dated bars
11725	Fixed	Chart	At times clicking on a charts panel while switching tabs locked the chart image
11727	Fixed	Chart	At times selecting the y-axis while switching tabs results in an error
11742	Fixed	Chart	Removing series from multi-series chart could result in an error
11747	Fixe	Chart	Mouse left-click and drag worked unexpectedly when equidistant bar spacing was disabled

	d		
11624	Fixed	Chart, Drawing	Drawing tools did not work on 4th panel of chart if occupied by an indicator alone
11592	Fixed	Chart, DrawingTool	Global arrow line could be incorrectly rendered in non equidistant charts
11526	Fixed	Chart, NinjaScript	Chart Styles were not properly disposing
11736	Fixed	Chart, Playback	Scrolling with fixed scale, center price on scale, and Playback connected caused range and scale to change
11627	Fixed	Chart, Strategy Analyzer	DrawRegion could duplicate on Strategy Analyzer charts
11690	Fixed	Chart	There was an error when changing the data series that had an indicator applied to it that was based off of another indicator
11531	Fixed	Commissions	Adding commissions to ambiguous instrument resulted in an exception
11537	Fixed	Connections	Crash occurred when disconnecting in some scenarios with multiple connections

11633	Fixed	Control Center	In some scenario menu items were skipped over when using arrow keys
11657	Fixed	Control Center	Category and Message columns of Log Grid could be removed and then couldn't be added back
11503	Fixed	Control Center	Filled Buys/Filled Sells column on Account grid was not counting partial fills
11713	Fixed	Control Center	Account Data Window did not save to workspace
11601	Fixed	Connections	Starting NinjaTrader with no internet connection resulted in a crash
11582	Fixed	Data	Downloading historical tick data for current day erased last hour of previous day.
11618	Fixed	Data Grids	Exporting data grids to CSV with values over 1000 was adding an invalid comma
11500	Added	Database	Added equities that were missing from the database
11516	Fixed	Drawing	On multi-series charts drawing object anchor points could adjust unexpectedly
11616	Fixe	Drawing	Merging/separating series on multi-series chart could render duplicate global draw objects

	d		
11706	Fixed	Drawing, NinjaScript	Resolved some errors related to drawing object rendering
11631	Fixed	Drawing, NinjaScript	RemoveDrawObject did not work when object was not on the price panel
11613	Fixed	DrawingObjects, NinjaScript	Resolved some scenarios of error failed to call OnRender() for chart object 'Ray'. "External component has thrown an exception."
11102	Fixed	DrawingTool	TrendChannel could display incorrectly in some scenarios where the anchor was off the chart
11540	Fixed	DrawingTool	Fibonacci Extension could place the price level label on the wrong side
11589	Fixed	DrawingTool	Ruler tick size format was wrong after switching instruments on chart
11594	Fixed	DrawingTool	Removed unused BackgroundOpacity property from RiskReward drawing tool
11654	Fixed	DrawingTool	Drawing object with future-anchors behaved poorly on multi-series chart
11658	Fixed	DrawingTool	VerticalLine selection points could be incorrectly distributed on additional chart panels

11703	Fixed	DrawingTool	Risk Reward tool with Auto scale enabled resulted in Y-axis to greatly extend
11714	Fixed	DrawingTool	When Gann fan's price levels were removed upon adding a new level a percentage field was given
11721	Fixed	DrawingTool	Restoring drawing heavy workspace could result in error
11730	Fixed	DrawingTool	Text objects moved after accessing the Drawing Objects window
11412	Fixed	DrawingTool, Strategy Analyzer	Memory could increase when reusing tag for a drawing object
11669	Fixed	eSignal	BANKNIFTY and NIFTY futures instruments were not able to receive data
11524	Fixed	Hot Key	Sell Stop limit order placed with hot key placed as simulated Sell Stop limit order
11674	Fixed	Hot Key	In some scenarios Hot Key order entry showed order confirmation twice
11508	Fixed	Indicator	There were missing spaces in TickCounter, VolumeCounter, and BarTimer
11534	Fixe	Indicator	Set ParabolicSAR to calculate on price change and resolved a case when its plot could back up

	d		from it's previous value
11628	Fixed	Indicator	Volume Profile with certain window sizes did not display latest profile as expected
11632	Fixed	Indicator	When adding BarTimer to a chart it briefly showed a disconnected message while still connected
11655	Fixed	Indicator	Swing Indicator was not outputting values for certain intervals
11659	Fixed	Indicator	BarTimer wouldn't function with simulated data feed if there were no historical bars
11591	Fixed	Interactive Brokers	In some scenarios switching instruments could remove execution markers
11604	Fixed	Interactive Brokers	Gateway connection stayed yellow when active orders/positions were present
11607	Fixed	Interactive Brokers	In some scenarios a working profit target/stop loss from Traders Workstation were not reported
11619	Fixed	Interactive Brokers	In a strategy shorting an instrument that that wasn't available to do so stopped the strategy and required manual cancellation of the order
11615	Fixed	IQFeed	Connection could stay in Connection Lost on failed log in attempt

11553	Fixed	Market Analyzer	Row highlight could not be disabled
11567	Fixed	Market Analyzer	Realized PnL for a simulation account reset inconsistently in different windows
11590	Fixed	Market Analyzer	Account drop-down was not populating all available accounts if the Market Analyzer was created before a data feed connection was started
11662	Fixed	Market Analyzer	Label row background color would reset when column background color changed
11676	Fixed	Market Analyzer, NinjaScript	Null references could result in a crash
11568	Fixed	Market Analyzer, Playback	In some scenarios adding an indicator column resulted in an error
11732	Fixed	Market Analyzer, SuperDOM	Indicator properties changed orientation in SuperDOM and Market Analyzer

11504	Fixed	NinjaScript	Enum could revert back to default parameter after compile
11518	Fixed	NinjaScript	SetProfitTarget as MIT order had limit price and was not modified correctly
11555	Fixed	NinjaScript	BarsSinceEntryExecution resulted in multi-series error on OrderFillResolution=High with empty overload
11561	Fixed	NinjaScript	Stop and target handling reverted to per entry execution on disable strategy
11670	Fixed	NinjaScript	Free trials produced an error requiring a restart for the trial to work
11702	Fixed	NinjaScript	Multi-series strategies using StartBehavior.AdoptAccountPosition will add the account's position on an instrument once per matching data series to the strategy's position
11663	Fixed	NinjaScript	Resolved some scenarios of error D2DERR_WRONG_FACTORY when rendering
11737	Fixed	NinjaScript	DrawRegion could be missing after hosting script saw OnBarUpdate() calls
11761	Fixed	NinjaScript	Resolved some scenarios of error Write lock may not be acquired with read lock held
11661	Fixed	NinjaScript	Resolved some scenarios of error on calling 'OnRender' method: Attempted to read or write protected memory

11664	Fixed	NinjaScript	Resolved some scenarios of error D2DERR_WRONG_STATE/WrongState, Message: The object was not in the correct state to process the method
10173	Fixed	NinjaScript Editor	XML Comments were not resolving correctly on DisplayAttribute properties
10207	Fixed	NinjaScript Editor	Was unable to properly collapse and expand regions holding only xml comments
11535	Fixed	NinjaScript Editor	Renaming of folders had unexpected results
11626	Added	NinjaScript Editor	Updated NinjaScript code editor
11734	Fixed	NinjaScript Editor	Right clicking NinjaScript Editor resulted in errors in Visual Studio output
11550	Fixed	NinjaScript, SharedAdapter	Twitter Share Service was not working with scripts
11696	Fixed	NinjaScript, Strategy Analyzer	Strategy indicator input series was defaulting to primary series when 'Optimize Data Series' was enabled in an optimization
11630	Fixed	NinjaScript,	Saving a template for an indicator prevented changes to the plot names in code from taking affect

		Templates	
11600	Fixed	Playback	UTC Amsterdam +1 time zone could result in a lockup
11640	Fixed	Playback	Recorded playback data was not time stamped correctly if disconnects occurred
10869	Fixed	Playback, Tick Replay	Reloading historical data with multiple strategies on multiple tick replay charts could result in an error
11611	Fixed	Strategy	In some scenarios the input series of the strategy instance on the tab was blank
11510	Fixed	Strategy	In some scenarios editing a strategies start behavior results in error and prevented connection
11566	Fixed	Strategy	In some scenarios an error occurred when adding a strategy to the strategies tab
11665	Fixed	Strategy	Account name displayed differently in strategies than other areas
11556	Fixed	Strategy	Error occurred when opening strategies dialog on chart with running strategy after removing source
11413	Fixed	Strategy Analyzer	Memory was not releasing when adding an indicator that draws objects

11522	Fixed	Strategy Analyzer	Optimization could hang on invalid high order resolution period value
11523	Fixed	Strategy Analyzer	In some scenarios running a test on an instrument with the option aggregated checked resulted in a crash
11560	Fixed	Strategy Analyzer	In some scenarios incorrect strategy name could show in drop-down menu
11625	Fixed	Strategy Analyzer	In some scenarios running tests with non-equidistant bar spacing resulted in an error
11505	Fixed	Strategy Builder	Removing a condition tab and compiling did not reflect the change in the code until relaunching Strategy Builder
11506	Fixed	Strategy Builder	In some scenarios a script error could make the Strategy Builder and NinjaScript editor inaccessible
11507	Fixed	Strategy Builder	Share to email resulted in an error
11512	Fixed	Strategy Builder	Calling an indicator in a sub-folder resulted in an error
11521	Fixed	Strategy	Spaces and special characters in the strategy name were able to be input

		Builder	
11571	Fixed	Strategy Builder	BarsSinceEntryExecution was not using Multiseries overloads
11610	Fixed	Strategy Builder	Offset series barsAgo was not accounted for in CurrentBars check
11652	Fixed	Strategy Builder	Could not check if a Date series is equal to a date
11570	Fixed	SuperDOM	Could not change bar type for an indicator
11520	Fixed	SuperDOM	Sending Futures instrument from Market Analyzer to Dynamic SuperDOM resulted in an error
11448	Fixed	TD AMERITRADE	Incorrect order type was used to close position
11637	Fixed	TD AMERITRADE	Out of order events could result in canceled orders showing as working
11643	Fixed	TD AMERITRADE	Incorrect error displayed if 'user is not allowed to access Streamer'

11666	Fixed	Templates	Tick Replay was saved in templates which was invalid when Tick Replay was disabled
11595	Fixed	Trade Performance	Individual trades calculated in pips showed in tenth pips
11698	Fixed	Trade Performance	Orders and Executions did not populate orders if dates were set to current day
11543	Fixed	UI	Modal Windows were globally 'Always On Top' instead of Application level 'Always On Top'
11623	Fixed	UI	Mouse pointer did not change to a caret for End Date setting
11697	Fixed	UI, Drawing Obejects	Drawing object's anchor points didn't use PC's region format
11701	Done	Yahoo	Yahoo discontinued

8.0.7.1 Release

June 21, 2017

Issue #	Status	Category	Comments
11793	Fixe	Playback	Having 'Playback current day only' disabled and moving the slider was only playing 1 day

	d		

3.3.23 8.0.6.1

8.0.6.0 Release
April 17, 2017

Feature #	Status	Category	Comments
11311	Added	FOREX.com	FOREX.com adapter added in (Beta)
10617	Changed	Drawing Tool	Updated the visual style of various Drawing Tools and Trade Markers
11427	Added	IQFeed, Kinetick	'VWAP' Fundamental Data support was added for IQFeed and Kinetick
11459	Added	Rithmic	Added support for Account 'Unrealized PnL' reported from Rithmic displayed in the 'Account Data' window.
11444	Changed	Trade Performance	The 'Trade Performance' window now filter in real-time, no longer requiring the report to be regenerated.
10941	Added	General	Added caption bar context menu for window operations (Restore, Minimize, Maximize, Close). Additionally when a window is maximized you can now drag the caption bar to restore the window.
11452	Changed	General	Reduced the height of tabs.

Issue #	Status	Category	Comments

11421	Fixed	Alerts	Multiple share actions on a single alert attempting to take a screen shot would cause an error
11451	Fixed	Alerts	Resolved a few scenarios where Alert condition did not fire as expected
11477	Fixed	ATM Strategies	ATM selection mode settings prevented REV button from working
11430	Fixed	ATM Strategies	Scaling in with an ATM using Pip mode resulted in unexpected rounding
11469	Fixed	ATM Strategies, FX Board	Canceling out of setting up a new 'Custom' ATM strategy didn't revert to previous ATM strategy selection
11332	Fixed	ATM Strategies, Strategy	Having an active ATM and then enabling a NinjaScript strategy could result in an error
11375	Fixed	ATM Strategies	Orders tab ATM name didn't show full active name until reconnect
11355	Fixed	Bars, Tick Replay	GetBar had error with secondary series when Tick Replay was enabled
11369	Fixed	Basic Entry	Action column wasn't showing order state colors
11467	Fixed	Basic Entry, Chart Trader, FX Pro,	Some areas were transparent in dark skins

		SuperDOM	
11229	Fixed	Chart	Couldn't scroll non-equidistant chart when chart bars building too quickly
11313	Fixed	Chart	Strategy could keep plot executions on chart as finalized by auto close
11399	Fixed	Chart	Displaced indicators showed values in Data Box without displacement
11446	Fixed	Chart	Auto Scale on Indicator panel was not working the same as Data Series panel
11475	Fixed	Chart	Some times zone settings could result in errors
11275	Fixed	Chart	Daily bars took longer to display than expected
11339	Fixed	Chart Trader	Resolved some race conditions that could prevent an OCO order from canceling
11324	Fixed	Chart, Drawing Tool	Pasting text from one instrument to a different instrument prevented it's properties from being available
11376	Fixed	Continuum, CQG	In some scenarios race conditions could prevent OCO orders from canceling
11321	Fixed	Continuum, CQG	Daily volume for current day was higher than expected
11335	Fixed	Control Center	In some scenarios multi-instrument entry prices would not showing in strategies tab
11315	Fixed	Core	Resolved some in-flight-executions scenarios that could prevent OCO orders from canceling

11447	Fixed	Data Grids	Live accounts were not showing under 'Filter by accounts'
11359	Fixed	Data Grids	Order Grid text alignment was off after editing
11445	Fixed	Database	In some scenarios continuous contracts would show in instrument rollovers
11278	Fixed	Drawing Tool	Draw objects could shrink in size when moved into the left chart boundary
11364	Fixed	Drawing Tool	Tag names would increase per object rather than per object type
11479	Fixed	Drawing Tool	Drawing objects would not persist through rollover when Attach To All Charts was selected
11286	Fixed	Drawing Tool, NinjaScript	Re-using drawing tool tag with different type did not replace as expected
11358	Fixed	FX Board	Order grid incorrectly updated when adding targets
11397	Fixed	FX Board	Up and down movement arrows were not displaying as expected
11366	Fixed	FX Board, FX Pro	Adjusting the quantity with the arrows was not modifying by the account set FX lot sizes
11485	Fixed	Hot Key, Orders	Enabling OCO orders would not apply OCO to Hot Key orders
11487	Fixed	Hot Key, Orders	Enabling simulated orders would not simulate Hot Key orders
11385	Fixed	Indicator	Woodies CCI panel was different between NinjaTrader 7 & 8

11443	Fixed	Indicator	Pivots Performance Improved
11462	Fixed	Indicator	BarTimer on daily chart wasn't showing message that it needs to be on intraday data
11316	Fixed	Instruments	Resolved some scenarios that could prevent instruments from showing in search windows
11382	Fixed	IQFeed	Bid, Ask, and Last historical data was displaying the same values
11455	Fixed	IQFeed	With some set ups and time zones not all requested data would load
11379	Fixed	Market Analyzer	PositionAvgPrice column wasn't showing tenth pip values
11429	Fixed	Market Analyzer	Changing the font removed Column background colors
11415	Fixed	Market Analyzer	Resizing columns was not working as expected when resizing window
11344	Fixed	NinjaScript	WaitUntilFlat only waited for the first exit of a position before submitting orders
11368	Fixed	NinjaScript	Resolved some scenarios of the error "Upgradeable lock may not be acquired with read lock held"
11395	Fixed	NinjaScript	Renaming script sub folder did not take affect until restart
11424	Fixed	NinjaScript	Resolved some scenarios of error "Failed to call OnRender() for chart object 'Line':

			'External component has thrown an exception.'"
11326	Fixed	NinjaScript Editor	Improper focus could prevent seeing compile option
11348	Fixed	NinjaScript Editor	NinjaScript editor right click compile option was enabled inconsistently
11293	Done	NinjaScript, Tick Replay	Tick Replay and Order Fill Resolution = High was able to be used in combination
11367	Fixed	Orders	Updating order quantity of OCO orders failed to fully persist OCO
11459	Fixed	Orders	In some scenarios properly cancelled orders could show error 'order can not be cancelled'
11391	Fixed	Other	Automatic Windows Update of video card drivers could result in an error
11340	Fixed	Playback	Using playback data from daylight savings time could result in an error
11438	Fixed	Playback	Historical NinjaScript draw objects were not drawn when using the Playback Slider
11203	Fixed	Strategy	BarsTypes default overrides were not applying to strategies added to strategies grid of Control Center
11418	Fixed	Strategy	In some scenarios restoring strategies could add in duplicates / phantom instances
11458	Fixed	Strategy	Multi-series strategy applied wrong trading hours to second data series when enabled from strategies grid

11482	Fixed	Strategy	In some scenarios a strategy could be edited while still running
11289	Fixed	Strategy Analyzer	No count was returned in the Optimizer for 'this.Results[0].AllTrades.Count'
11317	Fixed	Strategy Analyzer	Selecting 'Open NinjaScript Output' was not functioning
11319	Fixed	Strategy Analyzer	Selecting 'Open result in New Strategy Analyzer' opened with the incorrect backtest type
11386	Fixed	Strategy Analyzer	Aggregate optimization caused identical Performance values for all instruments
11387	Fixed	Strategy Analyzer	Strategy box did not display strategy name if amended namespace
11290	Fixed	Strategy Builder	In some scenarios of multiple alerts with different rearm times only the shorter time would rearm
11347	Fixed	Strategy Builder	A strategy could be named with the same name as pre-existing indicator resulting in errors
11361	Fixed	SuperDOM, UI	At times changing the order quantity could display the wrong quantity in the confirmation window
11448	Fixed	TD AMERITRADE	Short positions were reported as a 'negative' value which throws off internal logic
11330	Fixed	Trade Performance	In some scenarios Average Trade could incorrectly show a value of zero

11356	Fixed	UI	Button Backgrounds for saved items where not updating when changing skins
11406	Fixed	UI	Chart's Taskbar button would not move with the chart when moved to different monitor
11428	Fixed	UI	In some scenarios Simulation Color would not update
11349	Fixed	Visual Studio Integration	File changes in Visual Studio was not triggering compile sound
11383	Fixed	Window Linking, Workspaces	In some scenarios Chart indicator panels reverted to default size when linked

8.0.6.1 Release
April 25, 2017

Issue #	Status	Category	Comments
11578	Fixed	Drawing Tools	Removed boxes around text in drawing objects and reverted minimum chart marker size implemented in feature 10617
11587	Fixed	NinjaScript, Drawing Tools	Bars ago was referencing BarsInProgress rather than the primary bars object
11583	Fixed	Continuum, CQG	Resolved some scenarios where an order couldn't be canceled until a reconnect occurred when using local OCO simulation
11586	Fixed	IQFeed	Daily data was displaying incorrect values

3.3.24 8.0.5.2

8.0.5.0 Release
March 6, 2017

Issue #	Status	Category	Comments
11142	Fixed	ATM Strategies	Reversing an active ATM that had a flat position would not cancel the original ATM
11299	Fixed	Chart	In some scenarios reloading scripts rendered global draw objects invisible
11284	Fixed	Chart	Resolved DirectX errors that occurred in some scenarios
11260	Fixed	Chart	Muli-series charts were plotting future time axis values incorrectly
11240	Fixed	Chart	In some scenarios reloading scripts duplicated global drawing objects
11202	Fixed	Chart	In some scenarios indicators would not plot with horizontal line plot style
11201	Fixed	Chart	Chart background color changed when title bar was held
11181	Fixed	Chart	Focus to chart was not restored after closing data series or indicator window
11167	Fixed	Chart	Preset Bars to Load for daily bars was not applying when switching time frames by keyboard
11159	Fixed	Chart	Merge back adjusted was not properly applying
11147	Fixed	Chart	Bars would stop updating if laptop lid was closed despite it not being in sleep/hibernate

			mode
11123	Fixed	Chart	Improved null checks to prevent exceptions
11118	Fixed	Chart	Anchor points were not plotting on end of displaced indicator
11074	Fixed	Chart	Using a large displacement on an indicator resulted in a lag
11066	Fixed	Chart	In some scenarios an indicator could retrace over itself
11105	Fixed	Chart, Drawing Tool	In some scenarios auto scale of drawing objects on secondary panel didn't properly function
10617	Changed	Chart, Drawing Tool	Increased spacing and dash size of lines
11104	Fixed	Chart, Strategy	In some scenarios drag and drop of strategy plot did not properly function
11215	Fixed	Continuum, CQG	Properly canceled orders reported cancel failed in some scenarios
11216	Fixed	Control Center	Orders/Executions tabs filtering by live account included sim101 orders/executions
10973	Added	Core	Modal on modal windows have a dark overlay applied to better identify the modal window
11254	Fixed	Database	Automatic rollover failed if triggered while data was loading on start up
11296	Fixed	Drawing Tool	Risk/Reward drawing tool could result in crash with specific configurations

11255	Fixed	Drawing Tool	Data settings were not saved or applied when pressing Enter key
11214	Fixed	Drawing Tool	In some scenarios adding a line to the Trend Channel resulted in freezing
11163	Fixed	Drawing Tool	Remove all drawing tools was applying to all instruments and time frames
11122	Fixed	Drawing Tool	In some scenarios risk-reward tool calculated ticks incorrectly
11111	Fixed	Drawing Tool	Changing primary period on multi-series chart with region could produce an error
11079	Fixed	eSignal	Index SP500 was not plotting
11206	Fixed	FX Board	Orders were sorted inconsistently from other order grids
11140	Fixed	Google, Yahoo	Connection loss was not being reported
11304	Fixed	Instruments	Importing a stock with a period converted the symbol mapping to use an underscore rather than just the master symbol
11220	Fixed	Instruments	Stocks imported with invalid characters were not converted and could not be edited
11219	Fixed	Instruments	Use Instrument Settings template was not reflecting changes to default template
11195	Fixed	Instruments	Selecting the search button would not search what was typed in
11168	Fixed	Instruments	Edited default instrument values would reset on restart
11282	Fixed	Interactive	In some scenarios targets showed as external orders

		Brokers	
11129	Fixed	Interactive Brokers	Execution markers were not shown on custom instruments
11291	Fixed	Log	Auto close displayed incorrect time in Log tab
11148	Fixed	Market Analyzer	In some scenarios the Realized PnL showed incorrect currency symbol
11072	Fixed	Market Analyzer	Changing the Instrument column to a bar graph resulted in an error
11308	Fixed	NinjaScript	Auto close disabled strategies on instruments not in auto close list
11262	Fixed	NinjaScript	ChartControl was null in State.Historical of hosted indicator
11183	Fixed	NinjaScript	Minimizing and maximizing a chart with text drawn from a script could produce an error
11182	Fixed	NinjaScript	Changing types after applying indicator to chart would cause errors when opening indicator dialog after recompile
11180	Fixed	NinjaScript	Saving a file in Visual studio resulted in file being removed from .csproj
11174	Fixed	NinjaScript	A crash could occur while printing when an output window is open in another workspace
11115	Fixed	NinjaScript	Drawing objects configured to draw on price panel would move with plot when drag and dropped
11089	Fixed	NinjaScript	Region could draw past first bar of chart

11062	Fixed	NinjaScript	In some scenarios RemoveDrawObjects did not properly function
11189	Fixed	NinjaScript Editor	Renaming an indicator folder resulted in an error
11134	Fixed	NinjaScript Editor	Right clicking and pasting removed focus from the editor
11223	Fixed	NinjaScript, Chart	Negative ZOrder resulted in unnecessary OnRender updates
11135	Fixed	NinjaScript, Chart	On a multi-data series chart with strategies applied to both data series, moving a plot moved both strategy's plots
11297	Fixed	NinjaScript, Drawing Tool	Andrews Pitchfork was not setting extension anchor correctly in NinjaScript
11233	Fixed	NinjaScript, Drawing Tool	Risk-reward was not applying globally when set to do so in a script
11261	Fixed	Playback	When using Go To the minute and second could not be adjusted using arrows
11244	Fixed	Playback	ATM targets would stack in playback if 'wrong-side' limit entry orders were placed
11141	Fixed	Playback	Multi-data series charts would not load renko bars
11166	Fixed	Playback, Alerts	Alerts would trigger while Playback was paused
11165	Fixed	Playback, Alerts	Alerts time stamps were reporting as the current time of the PC clock

11248	Fixed	Playback, NinjaScript	Position.GetUnrealizedProfitLoss() prevented Playback from running
11303	Fixed	Playback, SuperDOM	Volume column was not updating each second as designed
11258	Fixed	Strategy	Brush property was not read-only when strategy was enabled
11236	Fixed	Strategy	Enabling a terminated strategy could result in multiple instances of strategy
10997	Fixed	Strategy	Account position sync took multiple enables to properly display
11310	Fixed	Strategy Analyzer	Switching display mode while backtest ran resulted in no chart displayed
11266	Fixed	Strategy Analyzer	Using both interger and bool properties resulted in an error
11234	Fixed	Strategy Analyzer	Category order of properties was not properly sorting
11209	Fixed	Strategy Analyzer	Scrolling results of optimization on a list of instruments could result in an error
11204	Fixed	Strategy Analyzer	In some scenarios Logs displayed wrong strategy when multiple strategies were in different namespaces
11157	Fixed	Strategy Analyzer	Sending optimization results to a new Strategy Analyzer was not keeping settings
11132	Fixed	Strategy Analyzer	Chart Plot lines disappeared when moving multi-series Strategy Analyzer

11131	Fixed	Strategy Analyzer	Setting optimization graph to performance did not properly list the axis as performance
11120	Fixed	Strategy Analyzer	In some scenarios indicators region removed when adding other indicators
11272	Fixed	Strategy Builder	String input did not escape characters
11269	Changed	Strategy Builder	Unrealized PNL condition was not functioning as expected
11267	Fixed	Strategy Builder	String inputs with a blank default value were not added into set defaults resulting in an error
11232	Fixed	Strategy Builder	Setting IsRising regression channel to true resulted in an error
11217	Fixed	Strategy Builder	In some scenarios Conditions and Actions tabs regenerated after being closed
11169	Fixed	Strategy Builder	Printing an indicator with bars ago set up would return an unexpected value
11103	Fixed	Strategy Builder	In some scenarios condition templates with additional data could result in errors
11087	Fixed	Strategy Builder	In some scenarios text could be selected unexpectedly
11156	Fixed	Strategy, Chart	Moving a strategy's indicator above panel 1 from another panel would also move the strategy plot to the data series panel
11096	Fixed	Strategy, Chart	Removing secondary series with strategy applied could leave a blank panel
11192	Fixed	TD AMERIT	Clicking Close was unable to close out a stop loss order

		RADE, NinjaScript	
11150	Fixed	Tick Replay	Adding an indicator while Tick Replay was loading could result in an error
11125	Fixed	Tick Replay	Applying an incompatible script wasn't showing an exception in some scenarios
11281	Fixed	Trade Performance	Journal was not displaying added notes when start and end dates were set to current date
11241	Fixed	Trade Performance, Workspaces	Display setting was not saving in workspace
11277	Fixed	Trading Hours	Removing all but one EOD option on the Default 24/7 template resulted in a crash
11249	Fixed	UI	In some scenarios context menus would get removed before the mouse could reach it
11212	Fixed	UI	Applying strategy in Control Center was not applying data series presets
11245	Fixed	Window Linking, NinjaScript	In some scenarios switching instruments on linked tabs resulted in flat indicator plots
11130	Fixed	Workspaces	Switching workspaces was not placing windows on top of other windows

8.0.5.1 Release

March 8, 2017

Iss	St	Categor	Comments

ue #	atus	y	
11346	Fixed	Strategy	Editing the properties of any existing NinjaScript Strategy on the Control Center would incorrectly reset any DataSeries value back to defaults. This was a result of a fix for issue: 11212

8.0.5.2 Release

March 15, 2017

Issue #	Status	Category	Comments
11384	Fixed	General	Fixed a bug where fresh new installations of NinjaTrader did not get the latest version of the default instrument lists from the server as expected.

3.3.25 8.0.4.0

Release Date

January 31, 2017

Issue #	Status	Category	Comments
11056	Fixed	Adapter	In some cases quickly disconnecting and reconnecting resulted in an error
10989	Fixed	Alerts	In some scenarios disabled alerts would re-enable unexpectedly when re-opening the workspace
11027	Fixed	Alerts, Drawing Tool	Alerts were not detecting trend line cross overs properly

11091	Fixed	ATM Strategies	Submitting an order to an active ATM strategy for an opposite position left an active strategy
11028	Fixed	Backup & Restore	Drawing object templates were not transferred when restoring from a backup
10855	Fixed	Chart	Using the 'Apply' button on the strategies dialog did not always perform the requested action.
10964	Fixed	Chart	Y-axis crosshair label was not equal to value in the y-axis scale when two data series where on a chart
10971	Fixed	Chart	In specific data load requests for tick data the full range requested did not display in the chart
11002	Fixed	Chart	Non-equidistant bar spacing caused invalid x-axis labels on interval change
11011	Fixed	Chart	Center on price scale with fixed scale incorrectly adjusted on each tick instead of on each new bar
11035	Fixed	Chart	Second data series was included in auto scale calculations despite auto scale being disabled
11060	Fixed	Chart	Region did not render when equidistant bar spacing was set to false.
11061	Fixed	Chart	Region and plot could get out of sync when using a displacement
11064	Fixed	Chart	Region and plot display did not sync with right side margin
11071	Fixed	Chart	When making setting changes the chart could lose your scroll location and reset back to displaying the last bar on chart.
11078	Fixed	Chart	Changing instrument via right click menu resulted in a blank chart

11098	Fixed	Chart	Adding Line On Close to a regular trading hours data series caused an error
11076	Fixed	Chart, DrawingTool	Region with displacement and non-equidistant bar spacing caused incorrect display
11073	Fixed	Chart, Hot Key	Databox was not closing with hot key
11054	Fixed	Chart, NinjaScript	In some scenarios moving an indicator that places drawing objects resulted in an error
10995	Fixed	Chart, SuperDOM	Editing an instrument applied to a chart and SuperDOM could make NinjaTrader unresponsive
11070	Fixed	Chart, Workspaces	Left fixed scale setting was not saving
11047	Fixed	Code Wizard	Resolved an error when trying to edit an added Custom Data Series
11100	Fixed	Commissions	Entering Commission Per Instrument Type in template caused an error
11032	Fixed	CQG, Backup & Restore	Performing a Backup/Restore after you had been connected to CQG to not finish correctly
10789	Fixed	Drawing Tool	In some scenarios the Trend Channels area wouldn't fully fill
10865	Fixed	Drawing Tool	Moving drawing objects on a logarithmic chart scale could incorrectly place anchor points
11031	Fixed	Drawing Tool	Trend Channel line color would not change
11038	Fixed	Drawing Tool	Regression channel disappeared when Price Type was set to 'Median'

11059	Fixed	Drawing Tool	Region displacement was not properly displacing
10968	Fixed	DrawingTool, Indicator	Drawing Tool anchors would break when switching intervals with indicator on chart
10895	Fixed	eSignal	Was not able to reconnecting after manual disconnect
10993	Fixed	FXCM	In some scenarios stacked orders could produce an error
11053	Fixed	FXCM	Daily Charts wouldn't receive real-time data with FXCM historical data server
11081	Fixed	FXCM	Historical data from FXCM's servers showed volume as 100k
11052	Fixed	Hot Key	Chart Trader Hot keys required Chart Trader to be selected
10958	Changed	Indicator	Updated existing indicators and help guide to consistently advise and perform instantiation of Indicators and custom DataSeries in State.DataLoaded
11016	Fixed	Instruments	Instrument Lists were not updating on first time start up
10947	Fixed	Interactive Brokers	A bad bar was generated when left connected past regular market hours
11068	Fixed	Interactive Brokers	In some scenarios connection hung in yellow state when there were active orders
11039	Fixed	Interactive Brokers, Kinetick	In some scenarios current daily bar was removed when using Preferred Connections
11093	Fixed	IQFeed	In some scenarios IQFeed could request too many simultaneous historical data requests
11009	Fixed	Market Analyzer	Zeros were reporting in indicator column for PriorDayOHLC

10917	Fixed	NinjaScript	Strategy stopped trading after historical data was reloaded
11017	Changed	NinjaScript	AddChartIndicator was not working in State.DataLoaded
10922	Fixed	NinjaScript, ATM Strategies	Closing position via AtmStrategyClose method prevented RealizedPnL from updating correctly
11051	Fixed	Orders, Rithmic, FXCM	Long account information could prevent orders from updating
10977	Fixed	Performance	Using Window's Taskbar preview used large amount of resources
10950	Fixed	Playback	Controller became disabled/frozen when using Tick Replay
11024	Fixed	Playback, SuperDOM	Market Depth disappeared/reappeared when scrolling price ladder
10961	Fixed	Strategy	Moving chart tab while strategy was loading caused an exception
11092	Fixed	Strategy	High order fill resolution with multi-series script would disable rather than enable with popup message
10767	Fixed	Strategy Analyzer	In some scenarios the chart was missing plot lines and the chart became transparent
10966	Fixed	Strategy Analyzer	In some scenarios the walk forward optimization results were not matching the summary page
10978	Fixed	Strategy Analyzer	Time frame date was wrong when opening at the start of a new year
10992	Fixed	Strategy Analyzer	Could not change grid row with arrow up and down keys

11004	Fixed	Strategy Analyzer	Show Tabs value was not consistently checked/unchecked
11021	Fixed	Strategy Analyzer	Changing from Summary to Trades display caused strategy to re-run
11055	Fixed	Strategy Analyzer	Changing optimization parameters could leads to an error
11005	Fixed	Strategy Builder	Selecting Close Price could display back as Default Input
11006	Fixed	Strategy Builder	Condition set tab order does not persist in Strategy Builder
11007	Fixed	Strategy Builder	Offset could not use User Defined Input
11048	Fixed	Strategy Builder	Defining a condition in Strategy Builder which was supposed to 'divide' a numerical value didn't actually perform the action
11077	Fixed	Strategy Builder	Default plot was not available for MACD
10970	Changed	Strategy Builder, Code Wizard	Added Use Primary Instrument option for adding data series
11030	Fixed	Strategy, Chart	In some scenarios strategy was not removed as underlying series was removed
10983	Fixed	SuperDOM	Center button was not working when last price was near top/bottom of ladder
10951	Fixed	TD Ameritrade, Indicator	In some scenarios the Bar Timer didn't properly function
11044	Fixed	Trading Hours	Eurex trading hours templates included Martin Luther King holiday

11033	Fixed	UI	Error could occur when using the Vendor Licensing Add On
10991	Fixed	Window Linking	In some scenarios changing time frames on a linked chart caused the current bar to report in different parameters
10915	Fixed	Workspaces	Added Skin Selection as part of installer

3.3.26 8.0.3.0

Release Date
January 9, 2017

Attention existing NinjaTrader 8 Users: As a consequence of bug fix in 8.0.3.0, all DataSeries Trading Hours Templates contained in saved workspace(s) will be reset to factory default settings "<Use Instrument Settings>".

Issue #	Status	Category	Comments
10922	Fixed	ATM Strategies, NinjaScript	Closing position via AtmStrategyClose method prevented RealizedPnL from updating correctly
10876	Added	Backup & Restore	Added restart message when restoring
10907	Fixed	Barchart	Updated DLL version to 1.1.0.9
10814	Fixed	Bars	UTC +2 time zone usage created erroneous bars on a chart
10239	Fixed	Bars	After editing workspace and restarting at times there was an unable to clear cache

			message
10864	Fixed	Chart	A crash occurred when creating daily charts in the evening
10832	Fixed	Chart	Removing indicator from multi-series chart removed unrelated data series as well
10959	Fixed	Chart	Center price on scale setting could cause crash on empty charts
10957	Fixed	Chart	AddOns and Caption Bar alignment was inconsistent
10945	Fixed	Chart	Changing axis line width caused incorrect margins
10935	Fixed	Chart	Chart was enlarged when restore from preview to secondary monitor
10916	Fixed	Chart	Bar spacing was not restoring correctly on multi-series chart
10906	Fixed	Chart	Indicator displacement did not auto scale correctly
10893	Fixed	Chart	Chart could scroll to the right unexpectedly
10881	Fixed	Chart	Chart rendering was failing after rollover of contracts
10877	Fixed	Chart	Trend Channel parallel line did not snap
10858	Fixed	Chart	Loading text did not stay in place for secondary series
10815	Fixed	Chart	Tab name variable tool tips were not defined

10912	Fixed	Chart Trader	Arrow keys moved chart instead of changing order quantity
10969	Fixed	Chart Trader	Right click order menu showed invalid order types for some connections
10780	Fixed	Chart Trader	Resolved a scenario where you could remove a DataSeries which has Scaled Justification Right and Chart Trader orders would no longer display
10925	Fixed	Chart, Bars	In some scenarios parts of Monday's sessions were not plotting
10851	Fixed	Chart, Drawing Tool	Drawing objects 'attached to' was lost when moving to overlay scale
10720	Fixed	Chart, Indicator	Editing indicators while chart is still loading at times cause a lock up
10842	Fixed	Chart, Window Linking	Incorrect chart tab was changing instruments with linked Market Analyzer
10868	Fixed	Connections, NinjaScript	Disconnect Delay Seconds was not recognizing reconnect
10926	Fixed	Control Center	At times NinjaTrader could lock up on start up
10857	Fixed	Control Center	Tab properties font was not saving checked boxes
10900	Added	Data Grids	Drag column away to delete was implemented on all grids
10812	Fixed	Database	Resolved a scenario where changing the NinjaTrader language caused a trading hours error

10847	Fixed	Drawing Tool	Draw.Region was shading incorrectly when using displacement
10845	Fixed	Drawing Tool	There was an inconsistent Z-Order between charts for global draw object
10833	Fixed	Drawing Tool	TrendChannel was not properly drawing when attached to all charts by default
10831	Fixed	Drawing Tool	Occasionally there were exception on removing all drawing objects
10984	Fixed	FX Board	Unhandled exception occurred when disconnecting Kinetick with open FXBoard open
10963	Fixed	FX Board	Instrument drop-down were not changing color for selected tile
10953	Fixed	FXCM	XAUUSD and XAGUSD CFD had incorrect symbol mapping
10809	Fixed	FXCM	In somecases orders could become stuck
10727	Fixed	FXCM	After a lost connection a crash could occur
10601	Changed	FXCM	Updated API version to ForexConnect 1.4.1
8490	Added	FXCM	Added GTD Order Support
10946	Fixed	FXCM, Historical Data Window	Control Center locked up when closing Historical Data window while downloading data
10830	Fixed	Historical Data	Historical data manager text was not readable when selected

		Window	
10885	Fixed	Hot List Analyzer	@HOTLIST was not present in resources
10834	Fixed	Indicator	LinReg generated error with period of 1
10883	Fixed	Indicator, SuperDOM	Indicator's Input series Price type was not saving
10939	Fixed	Instruments	Stock list with unsupported characters could be imported but not edited
10936	Fixed	Instruments	EUREX quarterly rollover dates do not match other EUREX instruments
10928	Fixed	Instruments	Contract Month had invalid date format
10920	Fixed	Interactive Brokers	FA accounts could receive position for unknown symbol
10897	Fixed	Interactive Brokers	Requesting 1 day of historical data downloaded 2 days
10853	Changed	Interactive Brokers	Symbol mapping updated for ICE TF Point Value change
10824	Changed	Interactive Brokers	Updated Traders Workstation to 960.2g
10792	Fixed	Interactive Brokers	Bad order state occurred when order blocked due to TWS precautionary settings

10846	Fixed	Kinetick	Constant connect/disconnect could occur when PC clock was out of sync
10604	Fixed	Kinetick	Historical data was unexpectedly throttled
10944	Fixed	Market Analyzer	Template colors were not restored after closing and opening a workspace
10909	Fixed	Market Analyzer	Removed invalid conditions when using Alerts
10849	Fixed	Market Analyzer	Suspended indicators did not catch up until bar closes after restored
10554	Changed	Market Analyzer	Added log error when using indicators not compatible with end of day data
10335	Fixed	Market Analyzer	Typing label text which exceeded window length triggered instrument search box
10918	Fixed	NinjaScript	LockRecursionException occurred on reloading NinjaScript after changing added series
10929	Fixed	NinjaScript	IndexOutOfRange exception occurred when there was an added series in hosted indicator
10902	Fixed	NinjaScript	AddDataSeries with specified template was not working as expecting in all scenarios
10870	Fixed	NinjaScript	Creating a new strategy while in a bad compile state halted NinjaTrader
10861	Fixed	NinjaScript	Removing an indicator then reloading a chart resulted in an exception
10841	Fixed	NinjaScript	High order fill resolution did not load expected bars when 'bars back' is used

10840	Changed	NinjaScript	Methods .PlaySound/.SendMail/.Share now can be triggered in State==.Active
10758	Fixed	NinjaScript	Exceptions could be generated by Finalized NinjaScript
10839	Fixed	NinjaScript, Strategy Analyzer	MAE was incorrect for multi-series strategies
10884	Fixed	Other	First In Product Announcement was modal only to Control center, other ones are modal across the app as expected
10948	Fixed	Playback	Controller's time-stamp lagged behind bars when paused
10911	Fixed	Playback	Slider became stuck if only two days were downloaded
10888	Fixed	Playback	At times Go To could not select time
10866	Fixed	Playback	Exception was occurring when disconnecting if End Date calendar was open
10291	Fixed	Playback	Migration from NinjaTrader 7 playback data had incorrect time offset
10938	Fixed	Property Grids	Orders Grid Properties allowed removing of Instrument column
10838	Fixed	ShareAdapter	Stocktwits share service log-in was throwing errors and was unresponsive
10982	Fixed	Strategy Analyzer	Assigned account could unexpectedly change on subsequent runs in certain scenarios.

10952	Fixed	Strategy Analyzer	Move to new window was available while running which resulted in an error
10894	Fixed	Strategy Analyzer	Null reference could occur when switching between backtest and optimization
10886	Fixed	Strategy Analyzer	Strategy was running two instances on single run
10850	Fixed	Strategy Analyzer	Repeatedly running a backtest caused memory to increase until there was a crash
10843	Fixed	Strategy Analyzer	There were incorrect tab names
10826	Fixed	Strategy Analyzer	Sorting pinned logs caused pins to be lost
10967	Fixed	Strategy Builder	IsFalling and IsRising was calling the bar index
10949	Fixed	Strategy Builder	MACD's plot name was Default rather than Macd
10932	Fixed	Strategy Builder	Unhandled Exception occurred when opening Strategy Building with compile errors
10836	Fixed	Strategy Builder	Removed invalid settings
10816	Fixed	Strategy Builder	An unhandled exception could occur when adding Actions in a certain scenario
10913	Fixed	SuperDOM	Dynamic SuperDOM was not always shown as a menu item
10821	Fixed	SuperDOM	Removed invalid order references for APQ column
10921	Fixed	TD AMERIT	External orders at times caused exceptions

		RADE	
10672	Fixed	TD AMERITRADE	Linked account hanged on connection attempt
10954	Fixed	Tool Tips	Getting started arrows were hard to see
10903	Fixed	Trade Performance	Journal entries were not hidden/shown based on generated date range
10891	Fixed	Trade Performance	Cut, copy, and paste was not working in all areas
10890	Fixed	Trade Performance	Journal entries were not properly logging notes
10962	Fixed	UI	Help > Email Support not saving email address
10914	Fixed	UI	End keyboard button switched chart tabs after "F" (fixed) button is pressed
10844	Fixed	UI	Resolved scenarios where in product announcement would block migration

3.3.27 8.0.2.0

Release Date
December 5, 2016

Issue #	Status	Category	Comments
10808	Fixed	Alerts	Alerts from removed drawing objects were playing on Playback rewind

10797	Fixed	Alerts	Alerts were activating based on core time rather than Playback time
10762	Fixed	Alerts, Market Analyzer	Alert conditions with Market Analyzer column were triggering an exception
10799	Fixed	ATM Strategies	Info tool tip had a new line at the bottom making the tool tip too long
10533	Fixed	Bars, NinjaScript	Secondary series was using trading hours of primary series unexpectedly
10769	Fixed	Basic Entry	Did not display active live orders
10798	Fixed	Chart	Extra data series was added when opening a Chart
10765	Fixed	Chart	Drawing objects were removed from second data series when changing primary series
10748	Fixed	Chart	PlotBrushes and BarBrushes were failing when indicator was copied to another chart using drag and drop
10732	Fixed	Chart	Could not add multiple indicators to panel 2 in some cases
10710	Fixed	Chart	Boxes printed too long when multiple data series are in panel 1
10709	Fixed	Chart	MultiSeries chart with Global Drawing objects were making charts unresponsive on scrolling
10753	Fixed	Chart Trader	Entry price marker was not matching average price

10751	Fixed	Chart, Chart Trader	Selected account was not duplicating to new chart
10818	Fixed	Chart, Hot Key	Canceling a drawing tool caused drawing tool Hot Keys not to work until chart refocused
10551	Fixed	Chart, NinjaScript	There was an render error on scripts at times when having multiple charts and switching the Period or Instrument
10338	Fixed	Chart, NinjaScript	There was an exception in some cases when unchecking Equidistant Bar Spacing and adjusting the time scale
10282	Fixed	Chart, NinjaScript	Box ChartStyle Bar would sometimes drawn to wrong date
10685	Fixed	Chart, NinjaTrader	Drawing Objects window caused external assembly script to stop updating
10778	Fixed	Chart, Workspaces	Chart scale fixed range restored to automatic range for non primary panels on saved workspaces
10738	Fixed	Control Center	Strategies grid sync red flag was appearing incorrectly
10758	Fixed	Core	Exceptions were being generated by terminated/finalized NinjaScripts
10786	Fixed	Data Grids	Grid highlights were overriding underneath grid color
10766	Fixed	Database	Execution markers would multiply when secondary connection holding traded account was running through a reconnect cycle

10696	Fixed	Drawing	Draw objects were rendering on left side of chart during bar loading
10790	Fixed	Drawing Tool	Saving the Trend Channel's default template caused odd plots
10788	Fixed	Drawing Tool	Trend Channel template prevented manual plotting of the second line
10784	Fixed	Drawing Tool	Trend Channel was drawing the parallel line out of order on indicator panel
10772	Fixed	Drawing Tool	Could not move draw objects in indicator panel
10744	Fixed	Drawing Tool	Trend Channel additional lines option did not exist
10623	Fixed	Drawing Tool	Global drawing objects were disappearing on restart when not applied to the primary data series
10752	Fixed	FX Board	ATM was no longer selected after tile size change
10747	Fixed	FX Board	ATM strategy orders were not being grouped
10743	Fixed	FX Board	Selected ATM strategy was not removing
10735	Fixed	FX Board	Order grid state column displayed incorrect on order rejection
10734	Fixed	FX Board	Display value for TimeLastTick was not resetting when disconnected
10721	Fixed	FX Board	Editing ATM settings from a saved workspace was sometimes resulting in an error

10722	Fixed	FX Board, FX Pro	FXCM default quantities were not matching across order entry windows
10779	Fixed	Hot Key	Global Hot Key for new chart caused unhandled exception with chart focused
10611	Fixed	Indicator	Instrument linked charts caused stochastics indicator to plot 0's after instrument change
10562	Changed	Interactive Brokers	Changed pacing violation logic and real-time data logic to improve load times and prevent bad ticks
10791	Fixed	Licensing	Vendor Licensing was allowing invalid spaces and would not keep some date settings
10810	Fixed	Licensing, SuperDOM	Template settings were not applying with Direct Edition License
10605	Fixed	Market Analyzer	There was an exception on loading indicator data for many instruments
10787	Fixed	NinjaScript	Using SetZOrder() to -1 was resulting in an error
10800	Fixed	NinjaScript Editor	Add-on warning was accessing invalid thread resources
10668	Fixed	NinjaScript, Tick Replay	There were errors at times when toggling Tick Replay while disconnected
10716	Fixed	Orders	Close button was not allowing closing out of in flight position during a close attempt
10705	Fixed	Orders	Stop loss orders attached to indicators were ending up on wrong side of the market

10712	Fixed	Playback	Playback controller button sometimes stopped enabling
10803	Fixed	Strategy Analyzer	Optimization results did not show slippage
10776	Fixed	Strategy Analyzer	Strategy optimizer did not display tool tips for properties
10775	Fixed	Strategy Analyzer	Summary Results % Profitable column was missing
10756	Fixed	Strategy Analyzer	Excessive Strategy Analyzer logs were slowing down NinjaTrader
10660	Fixed	Strategy Analyzer	Opening results to a new Strategy Analyzer was losing bars and getting an error
10804	Fixed	Strategy Builder	Slope did not allow for use of different plots
10703	Fixed	Strategy Builder	Did not allow to check all multi-series price types
10693	Fixed	Strategy Builder	Was not updating real-time changes to script
10837	Fixed	SuperDOM	Columns were not restoring correctly
10773	Fixed	SuperDOM	There was an error on requesting bars series when applying DEMA to SuperDOM
10768	Fixed	SuperDOM	Scroll wheel stopped scrolling price ladder after center button was clicked
10609	Fixed	SuperDOM	SuperDOM indicators not showing from saved workspace
10708	Fixed	TD AMERITRADE	In some cases if a connection could not be made an invalid error appeared

10795	Fixed	Tick Replay	AddDataSeries was not building in sequence with primary data series
10750	Fixed	Tick Replay	At times the SMA would get errors when using Tick Replay and multiple data series
10819	Fixed	Trade Performance	Column properties were missing after window is restored with workspace
10746	Fixed	UI	Enter key was not saving changes when the change was selected via keyboard command
10733	Fixed	UI	Creating instrument lists with the same name but different capitalization was not prevented and resulted in an error
10670	Fixed	UI	Restoring preset was applying before selecting OK
10761	Fixed	UI, Chart	Properties needed multiple clicks on OK to accept
10763	Fixed	Workspaces	Built in workspaces had incorrect default settings

3.3.28 8.0.1.0

Release Date
November 14, 2016

Issue #	Status	Category	Comments
10583	Fixed	Alerts	Drawing Tool Trend Channel did not display plot for selection
10578	Fixed	Alerts	Visibility and indicator suspension logic not working as expected

10572	Added	Alerts, Strategy Builder	Improved user experience on condition builder when setting up comparisons
10576	Fixed	ATM Strategies	Keep selected ATM Strategy template on order submission selected active ATM instead of template
10656	Fixed	Attach Order To Indicator, UI	Indicator name missing for blank labels in Attach Order To Indicator dialog
10694	Fixed	Bars	On Fridays, currently building weekly bar only contained that date
10492	Fixed	Bars	HeikenAshi BarsType Chart label was incorrect
10483	Fixed	Bars	Historical data recording thread locking .ncd file causing various crash reports
10599	Fixed	Basic Entry, FX Board, FX Pro, UI	Some properties were not available when clicking orders grid
10652	Fixed	Chart	Empty panel remained after removing indicator while Chart was loading
10645	Fixed	Chart	Crosshair incorrectly enabled after clicking Chart
10640	Fixed	Chart	Maximize panel button group on Chart moves the historical bar button location on some DPIs
10629	Fixed	Chart	Chart would sometimes freeze while changing properties
10619	Fixed	Chart	Chart right side margin property allowed negative values

10598	Fixed	Chart	Crosshair rendered incorrectly after interval change
10568	Fixed	Chart	Scale justification fixed scale icon remained after scale was no longer used
10566	Fixed	Chart	Non-default bar width applied incorrectly when new tab created via direct type into Chart
10546	Fixed	Chart	Taskbar preview caused extreme memory usage as mouse held on task bar
10540	Fixed	Chart	Jump to execution was not working on time-based Charts
10524	Fixed	Chart	Chart did not auto-scroll on multi-series Chart with global Crosshair
10519	Fixed	Chart	Crosshair disappeared when using a Drawing Tool
10509	Fixed	Chart	Automated strategy indicators could not be dragged and dropped
10507	Fixed	Chart Trader	Chart Trader missing grids when used on another tab
10485	Fixed	Chart	Deleting Chart Panel sometimes caused ChartStyle OnRender error
10392	Fixed	Chart	ChartStyle was changing in advance of new series loading
10487	Fixed	Chart Trader	Simulated Stop-Limit order in Chart trader caused rejected order when triggered
10684	Fixed	Chart Trader, Orders	Chart trader sell stop limit order with positive offset in remained in trigger pending state

10563	Fixed	Chart, DrawingTool	Draw object on non-primary panel was removed on reload NinjaScript
10607	Fixed	Chart, Indicator	Invisible indicators would incorrectly leave label drawn on chart panel
10315	Fixed	Chart, NinjaScript	SetZOrder was not working as designed
10575	Fixed	Chart, Strategy Analyzer	Strategy Analyzer duplicate tab Chart display scale margin was too wide
10484	Fixed	Chart, UI	Higher fill resolution type was blank for HeikenAshi
10481	Fixed	Chart, UI	Date and time was sometimes shown twice in mini data box
10648	Fixed	Connections, Kinetick	Kinetick "globex non-pro fees" option was available with sim key
10571	Fixed	Connections, UI	Pressing enter in username field during account creation generated unexpected message
10600	Fixed	Control Center, Strategy	Strategy grid displayed Sync as false when strategy is in sync with account
10493	Fixed	Control Center, UI	Control Center columns would widen on height changes
10655	Fixed	Data Grids, Orders	Incorrect strategy name displayed in Orders Tab
10678	Fixed	Drawing	Region/RegionHighlight Z-Order was not defaulted beneath other Chart objects as expected

10522	Fixed	Drawing	Gann fan line labels were drawing off-screen
10651	Fixed	DrawingTool	AutoScale was not applied to Ray drawing object
10626	Fixed	DrawingTool	Drawing object templates would not work in exported assemblies
10623	Fixed	DrawingTool	Global Drawing Objects disappeared on restart when not applied to the primary data series
10550	Fixed	DrawingTool	Making Fibonacci levels all invisible would dead lock the Chart
10523	Fixed	DrawingTool	Changing of Reward Anchor Y in Drawing Objects properties did not work
10506	Fixed	DrawingTool	AutoScale Draw.Text was causing incorrect AutoScale and hotkey to not work
10515	Fixed	DrawingTool	DrawingTool Slot Index changed when directly setting anchor to the same value
10521	Fixed	DrawingTool	Compressing the Chart changed angle of Gann fan
10396	Fixed	DrawingTool	Arc Drawing Tool line property affected the straight line instead of the Arc line
10669	Fixed	DrawingTool	Setting Arc drawing object to global resulted in exception
10295	Fixed	DrawingTool	Region Highlight did not recognize Z-Order change
10269	Fixed	DrawingTool	Bar and price did not snap correctly with Trend Channels Drawing Tool

10250	Fixed	DrawingTool	Draw object errors when changing multi series Chart series
10467	Fixed	DrawingTool, Indicator	Indicator using Rectangle type could not be used on two instance/panels of same Chart
10535	Fixed	eSignal	Custom divisors in instrument mapping were ignored on historical data
10721	Fixed	FX Board	Error on getting/setting property 'BarsRequiredToTrade' for NinjaScript 'AtmStrategy'
10634	Fixed	Historical Data Window	Unable to enter OHLC or Price to added rows in historical data
10700	Fixed	Indicator	BuySellVolume and BuySellPressure would not count the last tick of the bar
10662	Fixed	Indicator	Connect on startup caused error on applying NinjaScript
10650	Fixed	Indicator	Heuristics for determining what scale to place an overlay data series/indicator not working as expected
10573	Fixed	Indicator	Clicking apply then clicking ok double applies settings in indicator dialog
10560	Fixed	Indicator	VolumeProfile indicator could cause indicator exception when on empty charts
10559	Fixed	Indicator	Reloading Indicators could cause errors in some situations
10532	Fixed	Indicator	Tick Counter did not work with HeikenAshi bars set to tick
10453	Fixed	Indicator	WoodiesCCI indicator Zone Bars width did not match CCI plot width

10438	Fixed	Indicator	Indicators were not restoring presets when in configured list
10456	Fixed	Indicator, UI	Chart Indicator input series selector was auto-selecting wrong series
10659	Fixed	Instruments	Adding a contract month operation took longer than expected
10638	Fixed	Instruments	Offset values would reset to server offset regardless of rollover date change
10625	Fixed	Interactive Brokers	Real-time equity volume was divided by 100
10577	Fixed	Interactive Brokers	TWS live account was not obeying signal names in managed approach
10513	Fixed	Interactive Brokers	Position was not closed when selecting close under rare circumstances
10479	Fixed	Kinetick	USDCHN instrument was not displaying data
10379	Fixed	Kinetick	Kinetick remain connected when IP changed
10695	Fixed	Log	Missing "reversing..." log message on clicking REV button on order UI
10661	Fixed	Market Analyzer	Default parameters shown in columns window after loading a template
10567	Fixed	Market Analyzer	CurrentText value was not available to NinjaScript after reloading workspace
10527	Fixed	Market Analyzer	Position Avg Price column would display whole numbers only
10644	Fixed	News	Minor news filter UI issues at DPI 125%

10683	Fixed	NinjaScript	RemoveDrawObject(tag) did not remove NinjaScript drawn global objects
10552	Added	NinjaScript	Added ChartBars.GetBarIdxByX to locate center of bar
10543	Fixed	NinjaScript	AddOn stayed in UI after assembly was removed
10541	Fixed	NinjaScript	Multi-series chart with TickReplay and Calculate.OnEachTick was not triggering OnBarUpdate correctly
10548	Fixed	NinjaScript	RenderTarget sometimes had null properties in the middle of OnRenderTargetChanged
10470	Fixed	NinjaScript	Internal exception on closing chart with Multi-Series indicator with heavy load
10462	Fixed	NinjaScript	Provide additional information to client on import of NinjaScript
10528	Fixed	NinjaScript	GetSlotIndexByTime would cause crash on Time-based Bar Spacing
10466	Fixed	NinjaScript Editor	SetState was missing from IntelliPrompt
10536	Fixed	NinjaScript, Workspaces	Workspace persistence was not working for Add-ons immediately after import
10488	Fixed	Orders, Risks	Risk template Max Position Size was not working
10514	Fixed	Other	Updated mail to support to work with Yahoo and AOL domains
10674	Fixed	Playback	Right click in playback controller triggered a data reload

10512	Fixed	Playback	Playback reset caused exceptions in indicators
10382	Fixed	Playback	Playback chart was not always auto scrolling with playback
10504	Fixed	Property Grids	Some property grid combo box's were missing UI hover effect
10718	Fixed	Rithmic	Rithmic adapter for TopStepTrader did not set the correct routing information which resulted in rejected orders
10643	Fixed	Strategy	Live strategy could be visually "enabled" while global simulation mode was on
10579	Added	Strategy Analyzer	Added dialog when opening Strategy Analyzer running 32-bit NinjaTrader on a 64-bit machine
10682	Fixed	Strategy Analyzer	Optimization summary tab was not populating for some results
10666	Fixed	Strategy Analyzer	3D Optimization graph display view would reset when duplicating
10594	Fixed	Strategy Analyzer	Saving Strategy Analyzer preset resets non-default column widths
10556	Fixed	Strategy Analyzer	Resolved issues with drag/dropping tabs
10517	Fixed	Strategy Analyzer	Column presets not working as expected
10590	Fixed	Strategy Analyzer, Templates	Was not loading instrument from template
10591	Fixed	Strategy Analyzer, UI	Preview window with chart display sometimes did not display

10689	Fixed	Strategy Builder	Custom series allowed for values to be set of the wrong type
10676	Fixed	Strategy Builder	Condition Builder incorrectly uses close data series when other Data Series selected
10610	Fixed	Strategy Builder	Resolved exclude from compilation issues
10603	Fixed	Strategy Builder	Remove strategy was not always working correctly
10544	Fixed	Strategy Builder	Strategy causes unhandled exception "item with same key has already been added"
10530	Fixed	Strategy Builder	Condition Builder Indicator-as-input selected plot not in generated code
10472	Fixed	Strategy Builder	Context sensitive help guide not directing to correct location
10658	Fixed	Strategy, Strategy Analyzer	High fill resolution did not use primary series data type
10480	Fixed	Strategy, Tick Replay	Unexpected instance handling of script using bid / ask in tick replay
10616	Fixed	Strategy, UI	Strategy tab template reset data series incorrectly
10663	Fixed	SuperDOM	SuperDOM trade control on left was not saved to/restored from preset
10503	Fixed	SuperDOM	Off by one pixel width on DOM rows
10510	Fixed	TD AMERITRADE	External execution was sometimes not showing on connect

10478	Fixed	Templates	Non-UI Strategy properties were incorrectly saved to strategy template and overwrote developer code
10547	Fixed	Tick Replay	Errors could be generated on Multi-Series TickReplay chart in some situations
10631	Done	Trade Performance	Typo in MultiObjectiveValues
10671	Fixed	Trade Performance	Local PC currency incorrectly shown in Trade Performance display
10549	Fixed	Trade Performance	Executions did not update from grid after removed
10699	Fixed	Trading Hours	Copied trading hours template were unable to rename
10434	Added	UI	Delete key did not remove selected item in all dialog windows
10608	Fixed	UI	Preset >> restore was not working as expected in some areas
10606	Fixed	UI	Indicator name/label could disappear in the indicator properties
10602	Fixed	UI	Load Drawing Tool template dialog was not appearing on same monitor as parent
10542	Fixed	UI	Options text label was using dash instead of hyphen
10516	Fixed	UI	Blank name for risk template could be displayed

10537	Fixed	UI	Pressing Enter after renaming a tab did not rename a tab on all windows
10508	Fixed	UI	NinjaTrader logo was cut off when Save Chart Image was used
10476	Fixed	UI	Inconsistent tab name behavior was identified with no instrument selected
10471	Fixed	UI	Rename tab did not select tab name property correctly

4 Risk Disclosures

Futures, foreign currency and options trading contains substantial risk and is not for every investor. An investor could potentially lose all or more than the initial investment. Risk capital is money that can be lost without jeopardizing ones financial security or lifestyle. Only risk capital should be used for trading and only those with sufficient risk capital should consider trading. Past performance is not necessarily indicative of future results.

CFTC Rules 4.41 - Hypothetical or Simulated performance results have certain limitations, unlike an actual performance record, simulated results do not represent actual trading. Also, since the trades have not been executed, the results may have under-or-over compensated for the impact, if any, of certain market factors, such as lack of liquidity. Simulated trading programs in general are also subject to the fact that they are designed with the benefit of hindsight. No representation is being made that any account will or is likely to achieve profit or losses similar to those shown.

This website is hosted and operated by NinjaTrader, LLC (“NT”), a software development company which owns and supports all proprietary technology relating to and including the NinjaTrader trading platform. NT is an affiliated company to NinjaTrader Brokerage (“NTB”), which is a NFA registered introducing broker (NFA #0339976) providing brokerage services to traders of futures and foreign exchange products. This website is intended for educational and informational purposes only and should not be viewed as a solicitation or recommendation of any product, service or trading strategy. No offer or solicitation to buy or sell securities, securities derivative or futures products of any kind, or any type of trading or investment advice, recommendation or strategy, is made, given, or in any manner endorsed by any NT affiliate and the information made available on this Web site is not an offer or solicitation of any kind. Specific questions related to a brokerage account should be sent to your broker directly. The content and opinions expressed on this website are those of the authors and do not necessarily reflect the official policy or position of NT or any of its affiliates.

5 Risks of Electronic Trading with NinjaTrader

There are risks associated with electronic trading in general. Below are risks that you must be aware of with respect to NinjaTrader.

OCO Handling (One Cancels Other)

NinjaTrader supports multiple different connectivity providers (brokers, exchange gateways, and data feeds) that each have different levels of support for advanced order handling features such as OCO orders. An OCO order is simply a group of linked orders where if one is either filled or cancelled, all other orders that belong to it's OCO group are cancelled. If your connectivity provider does not support OCO orders natively, NinjaTrader will simulate them on your local PC. It is important to understand how these order types behave.

- OCO does not imply that once one order is filled, related orders in the same OCO group are guaranteed to be cancelled. It means that once an order is filled or cancelled, any remaining orders in the same OCO group will try to be cancelled. It is possible (in rare occasions) that order(s) that are part of the OCO group will be filled before the cancellation request has been acknowledged. As an example, let's say you have a stop loss and profit target order as part of an OCO group. The profit target is filled, the market rapidly turns around, the OCO cancellation request is submitted, the stop loss order is filled before the cancellation request is acknowledged. The narrower the spread between your OCO orders the higher the risk of getting filled on an order before it is cancelled in fast moving markets.

- Local PC held simulated OCO orders are dependant on order status events returning from your connectivity provider to trigger the cancellation of OCO orders. If NinjaTrader is offline (internet connection is down or PC crashed) then the simulated OCO functionality will not be operational.

In Flight Executions

There are several functions within NinjaTrader that are based on the current state of your account at the moment the function is invoked. These functions are:

- Close Position
- Flatten Everything

In flight executions are orders that are partially or completely filled between the time that you invoke one of the above functions and the time your connectivity provider acknowledges the order submission/modification/cancellation requests submitted by these functions. Here is an example:

1. You have an open long position for three contracts and several working stop loss and profit target orders for three contracts each
2. You invoke the command "Flatten Everything" which proceeds to cancel all working orders and submit a market order to close the three contract position
3. One of your profit target orders is filled before the cancellation request arrives at the exchange
4. The market order to close the position is also filled for three contracts
5. You now have an open short position for three contracts

This example is generally a rare occurrence. After invoking any of the above commands it is always prudent to check the Control Center's Positions Tab and Orders Tab to ensure that all orders were cancelled and positions flattened. To avoid these situations you should be cautious of using the "Close Position" function when you have orders that are working within a few ticks of the inside market.

NinjaTrader Volume Based Simulated Stop Orders

Please see this section of the Help Guide to understand the risks involved in using volume based simulated stop orders.

6 Terms of Service

NINJATRADER TERMS OF SERVICE AGREEMENT

THIS **TERMS OF SERVICE AGREEMENT** ("Agreement") is made between NinjaTrader, LLC ("Company") and any person ("User") who installs the NinjaTrader Trading Platform ("Platform").Platform

BY CLICKING THE ACCEPTANCE BUTTON OR ACCESSING, USING OR INSTALLING ANY PART OF THE PLATFORM, USER EXPRESSLY AGREES TO AND CONSENTS TO BE BOUND BY ALL OF THE TERMS OF THIS AGREEMENT. IF USER DOES NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, THE BUTTON INDICATING NON-ACCEPTANCE MUST BE SELECTED AND COMPANY SHALL PROMPTLY CANCEL THIS TRANSACTION AND USER MAY NOT ACCESS, USE OR INSTALL ANY PART OF THE PLATFORM. THIS AGREEMENT IS APPLICABLE FOR ALL RELEASED VERSIONS OF THE PLATFORM INCLUDING, BUT NOT LIMITED TO BETA VERSIONS. THIS AGREEMENT MAY BE AMENDED FROM TIME- TO-TIME AT THE SOLE DISCRETION OF COMPANY. COMPANY SHALL PROVIDE NOTICE TO USER OF AMENDMENTS BY POSTING THE UPDATED TERMS OF SERVICE ON COMPANY'S WEBSITE. USER SHALL HAVE THE OPPORTUNITY TO REFUSE SAID AMENDMENTS SOLELY BY REQUESTING TERMINATION OF ACCESS TO THE PLATFORM.

1. Platform Terms

a. ***Description.*** The Platform is proprietary to Company and is protected by intellectual property laws and international intellectual property treaties. User's access to the Platform is licensed and not sold. Platform is a software application that interfaces through various third-party independent software vendor and brokerage API's (collectively "Broker API") for the purpose of analyzing and trading financial markets.

b. ***Use of Third-Party Software Components.*** User is aware that the Platform implements various third-party software, platforms, services, equipment and Broker API's, (collectively "Components"). Company warrants that use of Components is fully licensed for use by Components providers to Company and in-turn to licensed Users of Platform. User shall abide by all Components' individual terms of service agreements, if applicable. **COMPANY MAKES ABSOLUTELY NO REPRESENTATIONS OR WARRANTIES AS TO ANY COMPONENT(S) AND EXPRESSLY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.**

c. ***Third-Party Vendors.*** User is aware that third-party vendors ("Vendors") may a) develop plugins including but not limited to indicators, strategies and other various utilities ("Apps") that interact and/or work within the Platform and b) provide educational services that demonstrate the use of the Platform. Vendors along with their websites, products, services and Apps collectively referred to as "Vendor Content", are independent persons or entities that are in no manner affiliated with Company or any if its affiliates.

1. ***Vendor Content.*** Company and its affiliates are not responsible for, do not approve, recommend or endorse any Vendor Content and it's your sole responsibility to evaluate Vendor Content. Please be aware that any performance information provided by a Vendor should be considered hypothetical and must contain the disclosures required by NFA Rule 2-29(c). If you are interested in learning more about, or investigating the quality of, any such Vendor Content you must contact the Vendor, provider or seller of such Vendor Content. No person employed by, or associated with, Company or its affiliates is authorized to provide any information about any such Vendor Content. Visit the CFTC resources for education regarding the industry and signs of fraud.
2. ***Use of Apps.*** Installation and use of Apps is at User's sole risk. User hereby agrees that Company makes absolutely no guarantees regarding compatibility and is not responsible for the function of Apps individually or with respect to the Platform. **COMPANY MAKES ABSOLUTELY NO REPRESENTATIONS OR WARRANTIES AS TO ANY APP(S) AND EXPRESSLY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS OR IMPLIED,**

INCLUDING WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

d. ***Accessibility and Function.*** User agrees that from time to time, the Platform may be inaccessible or inoperable for any reason, including, without limitation: (i) equipment (hardware) malfunctions, (ii) software malfunctions, (iii) periodic maintenance procedures or repairs which Company may undertake from time to time, or (iv) causes beyond the reasonable control of Company or which causes are not reasonably foreseeable by Company. **Company is not responsible, directly or indirectly, for the performance and/or reliability of Components, system, equipment or otherwise, or User's Internet Service Provider ("ISP").**

e. ***Equipment.*** User shall be solely responsible for providing, maintaining and ensuring compatibility with the Platform, all hardware, software, electrical and other physical requirements for User's use of the Platform including, without limitation, telecommunications and Internet connection(s), ISP, web browsers and/or other equipment, programs and services required to access and use the Platform.

f. ***Grant of License.*** Company grants User, pursuant to the terms and conditions of this Agreement, an exclusive and nontransferable license to use the Platform on a single computer at any one time.

g. ***Remote Access Services*****:** Company may, at its sole option, provide as a courtesy, technical support services, which are subject to the following terms and conditions. User accepts all risks associated with any request or authorization by User permitting Company personnel to remotely access and control User's computer. By requesting and permitting remote access, User acknowledges that User may be providing Company personnel with access to files and data on User's computer. Before permitting remote access, User agrees to close any confidential or personal files and create a backup of any important files. Company personnel are not expected to make copies or download files or to retain any information accessed from User's computer. User's name and contact information provided to facilitate remote access may be logged to process support requests and will be processed in accordance with Company's then-existing privacy policy.

COMPANY MAKES NO WARRANTIES OF ANY KIND WITH REGARD TO TECHNICAL SUPPORT SERVICES AND HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS OF ANY KIND RELATED TO TECHNICAL SUPPORT SERVICE, INCLUDING BUT NOT LIMITED TO ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL COMPANY BE LIABLE FOR ANY DAMAGES, INCLUDING SPECIAL, DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES RESULTING FROM LOSS OF USE, DATA OR PROFITS REGARDLESS OF THE LEGAL THEORY UNDER WHICH SUCH CLAIMS ARE ASSERTED, INCLUDING WITHOUT LIMITATION, ACTIONS BASED ON CONTRACT, NEGLIGENCE OR OTHER TORTIOUS CONDUCT, ARISING OUT OF OR IN CONNECTION WITH THE PROVISION OF TECHNICAL SUPPORT SERVICES. IN NO EVENT SHALL COMPANY'S TOTAL LIABILITY FOR ANY DAMAGES EXCEED THE TOTAL FEES PAID BY USER TO COMPANY HEREUNDER.

2. Security of User's System

User shall be solely responsible for the security, confidentiality and integrity of all messages and the content that User receives, transmits through or stores via the Platform or on any computer or related equipment that is used to access the Platform. User shall be solely responsible for any authorized or unauthorized access to User's account by any person, entity, partnership, organization, association or otherwise.

3. Fees/Licenses

a. ***Collection and Taxes.*** All Fees, taxes and other charges shall be billed to User's credit/charge card or paid by check. In the event that User is provided with use of Platform through a 3rd party reseller ("Reseller), User shall pay the Reseller who in turn shall submit the appropriate subscription fee to Company.

User shall be solely responsible for and shall pay Company or Reseller , if applicable, all sales, use, value-added, personal property or other tax, duty or levy of any kind, including interest and penalties thereon (collectively, "Taxes"), whether imposed now or hereinafter by any governmental authority. User shall promptly pay Company in the event of any refusal by User's credit card issuer to pay any amount to Company for any reason. User agrees to pay interest at the rate of two percent (2.0%) per month on any outstanding balance, together with costs of collection, including attorney's fees and costs, and any applicable bank fees. In the event User fails to pay any amount due as set forth herein, Company may, at its sole discretion, immediately suspend or terminate this Agreement and User's access to the Platform. Company reserves the right to report delinquent accounts to appropriate credit agencies.

b. ***Term/Automatic renewal.*** The term of this agreement shall begin upon User's commencement of the Platform and shall automatically renew on either a monthly or quarterly basis as chosen by the Client at the time of contract initiation. Fixed lease options as chosen at the time of contract initiation do not auto-renew. Termination by User or Company prior to automatic renewal of term must be supplied in written form at least 30 days prior to the expiration of the current term and must comply with the termination procedures set forth in Section 6 of this Agreement. Should the subscription be terminated prior the current subscription period expiration date and pursuant to Section 6 of this Agreement, NO refund shall be issued to User by Company.

c. ***Lifetime Licenses.*** If User purchases a lifetime license to use the Platform, User agrees that without limitation certain features of the software may not be available or supported in perpetuity. User also agrees that Company shall have the right to change features associated with the Platform in Company's sole discretion, and that Company may choose to discontinue support of Platform at any time. User shall not be entitled to a refund of the lifetime license fee under any circumstances. Lifetime licenses are for non-concurrent use, they are non-transferable, and can only be used by the individual that purchased the license. Lifetime licenses cannot be sold or bartered in the future and if such actions are taken the license can be terminated at Company's sole discretion. The lifetime license fee does not include the cost of any TT transaction fees if applicable when a static SuperDOM is requested.

d. **Upgrades.** During the term of the license User shall be entitled to Platform upgrades as provided in the sole discretion of Company. User's entitlement to upgrades shall be limited to the specific edition of the Platform for which the User is licensed. For instance, if User subscribes to Edition A of the Platform, User shall be entitled only to Edition A upgrades and so forth. Platform editions relate to the service level of Platform and shall not be confused with release version number(s).

4. User Representations

User represents and warrants to Company that: (a) User is over the age of eighteen (18) and has the power and authority to enter into and perform User's obligations under this Agreement, (b) all information provided by User to Company is truthful, accurate and complete, (c) User is the authorized signatory of the credit or charge card provided to Company to pay the Fees, (d) User shall comply with all terms and conditions of this Agreement including, without limitation, the provisions set forth in section 5, (e) User, and not the Company, is solely responsible for the security and use of User's password, (f) User has provided and shall provide accurate and complete registration information including, without limitation, User's legal name, address and telephone number, (g) User acknowledges that all right, title, and interest to the Platform belongs to Company. Company reserves all rights not expressly granted to User in this Agreement and that the User may not sublicense, transfer, or assign the Platform, directly or indirectly, to any person, entity, partnership, organization, association or otherwise, for any reason.

5. Prohibited Uses

a. ***Errors, Acts, Omissions and Unacceptable Use.*** User is solely responsible for any and all errors, acts and omissions that occur under User's account or password, and User, directly or indirectly, agrees not to engage in, facilitate, or encourage any unacceptable use of the Platform which unacceptable use includes, without limitation, use of the Platform to: (i) disseminate, store or transmit unsolicited messages, chain letters or unsolicited commercial e-mail, (ii) disseminate or transmit material that, to a reasonable person may be

considered abusive, obscene, pornographic, defamatory, harassing, grossly offensive, vulgar, threatening or malicious, (iii) disseminate, store or transmit files, graphics, software or other material that actually, impliedly, or potentially infringes the copyright, trademark, patent, trade secret, trade name or other intellectual property right of any person, entity, partnership, organization, association or otherwise, (iv) create a false identity or to otherwise attempt to mislead any person, entity, partnership, organization, association or otherwise, as to the identity or origin of any communication, (v) distribute, re-distribute or permit transfer of content in violation of any export or import law and/or regulation or restriction of the United States of America and its agencies or authorities, or without all required approvals, licenses or exemptions, (vi) interfere, disrupt or attempt to gain unauthorized access to other accounts on the Platform or any other computer network, (vii) disseminate, store or transmit viruses or any other malicious code or program, (viii) develop an interface between Platform to Broker APIs without the express written consent from the Company,; or (ix) engage in any other activity deemed by the Company, in its sole discretion, to be in conflict with the spirit or intent of this Agreement.

b. ***Dissemination.*** User may not disseminate software, username(s) and/or password(s) to any other person, entity, partnership, organization, association or otherwise. Internet Protocol ("IP") addresses may be recorded by the Platform to prevent account misuse.

6. Termination

This Agreement is effective upon User's acceptance as set forth herein and shall continue in full force until terminated. User may terminate this Agreement for any reason upon thirty (30) days prior written notice to Company. Company reserves the right, in its sole discretion and without prior notice to User, at any time and for any reason, to: (a) remove or disable access to all or any portion of the Platform, (b) suspend User's access to or use of all or any portion of the Platform, and (c) terminate this Agreement.

7. Disclaimer of Warranties

THE PLATFORM IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USE OF THE PLATFORM IS AT USER'S SOLE RISK. COMPANY DOES NOT WARRANT THAT THE PLATFORM WILL BE UNINTERRUPTED OR ERROR FREE, NOR DOES COMPANY MAKE ANY WARRANTY AS TO ANY RESULTS THAT MAY BE OBTAINED BY USE OF THE PLATFORM. USER REALIZES THAT THERE IS RISK IN TRADING STOCKS AND THAT ASSETS MAY BE LOST AND ARE NOT INSURED. COMPANY IS ABSOLUTELY NOT RESPONSIBLE, DIRECTLY OR INDIRECTLY, FOR USERS' STOCK ORDER, PURCHASE AND SALE ACTIONS. COMPANY MAKES NO OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, IN RELATION TO THE PLATFORM. COMPANY MAKES ABSOLUTELY NO WARRANTIES WITH REFERENCE TO THIRD PARTY VENDOR/BROKER SOFTWARE AND/OR SERVICES.

8. Limitation of Liability

UNDER NO CIRCUMSTANCES SHALL COMPANY, DIRECTLY OR INDIRECTLY, BE LIABLE TO USER OR ANY OTHER PERSON, ENTITY, PARTNERSHIP, ORGANIZATION, ASSOCIATION OR OTHERWISE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL OR PUNITIVE DAMAGES FOR ANY MATTER ARISING FROM OR RELATING TO THIS AGREEMENT, THE PLATFORM OR THE INTERNET IN GENERAL, INCLUDING, WITHOUT LIMITATION, USER'S USE OR INABILITY TO USE THE PLATFORM, ANY CHANGES TO OR INACCESSIBILITY OF THE PLATFORM, DELAY, FAILURE, UNAUTHORIZED ACCESS TO OR ALTERATION OF ANY TRANSMISSION OR DATA, ANY MATERIAL OR DATA SENT OR RECEIVED OR NOT SENT OR RECEIVED, ANY TRANSACTION OR AGREEMENT ENTERED INTO THROUGH THE PLATFORM, ANY DATA LOSS, OR ANY DATA OR MATERIAL FROM A THIRD PARTY ACCESSED ON OR THROUGH THE PLATFORM, WHETHER SUCH LIABILITY IS ASSERTED ON THE BASIS OF CONTRACT, TORT OR OTHERWISE. IN NO EVENT SHALL COMPANY'S TOTAL LIABILITY FOR ANY DAMAGES EXCEED THE TOTAL FEES PAID BY USER TO COMPANY HEREUNDER. SOME STATES PROHIBIT

THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, THUS THIS LIMITATION OF LIABILITY MAY NOT APPLY TO USER. IF USER IS DISSATISFIED WITH THE PLATFORM, USER'S SOLE AND EXCLUSIVE REMEDY SHALL BE FOR USER TO DISCONTINUE USE OF THE PLATFORM AND TERMINATE THIS AGREEMENT IN ACCORDANCE WITH SECTION 6. COMPANY IS NOT LIABLE FOR ANY ITEMS VIEWED OR TRANSMITTED VIA THE PLATFORM. COMPANY IS NOT LIABLE, DIRECTLY OR INDIRECTLY, FOR ANY ACTS TAKING PLACE WHICH ARE NOT VIEWED OR TRANSMITTED VIA THE PLATFORM. COMPANY IS NOT OBLIGATED, DIRECTLY OR INDIRECTLY, TO TAKE ANY STEPS TO PREVENT OR CORRECT ANY ILLEGAL, ABUSIVE OR OTHERWISE INAPPROPRIATE ACTIVITY PERFORMED BY USER, NOR IS COMPANY OBLIGATED, DIRECTLY OR INDIRECTLY, TO ARCHIVE OR OTHERWISE MAINTAIN OTHER REPRODUCTION OF THE CONTENT THAT APPEARS OR IS TRANSMITTED ON THE PLATFORM FOR FUTURE REFERENCE. COMPANY IS NOT LIABLE, DIRECTLY OR INDIRECTLY, FOR ANY ACTION OR INACTION WITH RESPECT TO ANY CONTENT ON THE PLATFORM. COMPANY IS NOT RESPONSIBLE, DIRECTLY OR INDIRECTLY, FOR COMPLIANCE OR LACK THEREOF BY ANY BROKER(S) WITH RESPECT TO ANY APPLICABLE LAWS AND REGULATIONS INCLUDING, BUT NOT LIMITED TO, THOSE LAWS REGARDING OR PERTAINING TO THE TRADING OF SECURITIES. COMPANY MAKES SIGNIFICANT EFFORTS MEETING OR EXCEEDING INDUSTRY STANDARDS TO INSURE THE SECURITY AND/OR FUNCTIONALITY OF PLATFORM RELATED INTERNET TRANSMISSIONS BUT, DUE TO THE INHERENT NATURE OF THE INTERNET, CANNOT GUARANTEE OR WARRANT FUNCTIONALITY AND/OR SECURITY OF INTERNET TRANSMISSIONS.

9. Indemnification

User agrees to indemnify, hold harmless and defend Company, its shareholders, directors, officers, employees and agents from and against any action, cause, claim, damage, debt, demand or liability, including reasonable costs and attorney's fees, asserted by any person, entity, partnership, organization, association or otherwise, arising out of or relating to: (a) this Agreement, (b) User's use of the Platform, including any data or work transmitted or received by User, and (c) any unacceptable use of the Platform, including, without limitation, any statement, data or content made, transmitted or republished by User which is prohibited as unacceptable in section 5.

10. Privacy

a. ***General.*** When reasonably practicable, Company shall attempt to respect and maintain User's privacy. Company shall not monitor, edit, or disclose any personal information about User or User's account, including its contents or User's use of the Platform, without User's prior written consent unless Company has a good faith belief that such action is necessary to: (i) comply with any legal process or other legal requirements of any governmental authority, (ii) protect and defend the rights, interests, or property of Company, (iii) enforce this Agreement, (iv) protect the interests of users of the Platform other than User or any other person, entity, partnership, organization, association or otherwise, or (v) operate or conduct maintenance and repair of Company's services or equipment, including the Platform as authorized by law. User has no expectation of privacy with respect to the Internet in general. User's IP address and Platform generated GUID is transmitted and recorded with each User session.

c. ***Billing/Credit or Charge Card Information.*** Company shall not share billing/credit or charge card information provided by the User with third parties unless written or electronic permission is expressly received from User.

d. ***Use of Aggregate Information.*** Company may, at its sole discretion, share aggregate information (e.g. number of website visits, demographic breakdown, etc.) to third parties by combining aspects of personal information into an anonymous pool.

e. Security of Personal Information. Information security is of the utmost importance to Company, however, no transmission of data over the Internet is guaranteed to be completely secure. Company shall

not guarantee or warrant the security of any personal information transmitted to or from it. Any such transmission is made solely at User's risk.

f. Links. Company's Platform website may contain links to other Internet websites. These websites are not under the control of Company and Company does not control linked websites' privacy and/or user agreements. Company does not grant any warranties (express or implied) nor does Company have any liability for information transferred and conferred to or from linked websites.

g. ***Audits.*** Company may gain access to customers account/trading records for auditing purposes. Such records may be disclosed to an independent audit source. Reasonable and industry appropriate non-disclosure agreement(s) shall pertain to third party auditing sources. Some configurations of Platform may transmit trade execution data over the Internet to a secure database for the purpose of audit tracking.

11. Miscellaneous

a. ***Amendment.*** Company shall have the right, at any time and without prior written notice to or consent from User, to add to or modify the terms of this Agreement, simply by updating the Company's website or by requiring the User to accept an updated Agreement upon installing and using the Platform. User's access to or use of the Platform after the date such amended terms are delivered to User shall be deemed to constitute acceptance of such amended terms.

b. ***Waiver.*** No waiver of any term, provision or condition of this Agreement, whether by conduct or otherwise, in any one or more instances, shall be deemed to be, or shall constitute, a waiver of any other term, provision or condition hereof, whether or not similar, nor shall such waiver constitute a continuing waiver of any such term, provision or condition hereof. No waiver shall be binding unless executed in writing by the party making the waiver.

c. ***Severability.*** If any provision of this Agreement is determined to be illegal or unenforceable, then such provision shall be enforced to the maximum extent possible and the other provisions shall remain fully effective and enforceable.

d. ***Notice.*** All notices shall be in writing and shall be deemed to be delivered when sent by first-class mail or when sent by facsimile or e-mail to either parties' last known post office, facsimile or e-mail address, respectively. User hereby consents to notice by e-mail. All notices shall be directed to the parties at the respective addresses given above or to such other address as either party may, from time to time, provide to the other party.

e. ***Governing Law.*** This Agreement is made in and shall be governed by the laws of the State of Colorado without reference to any conflicts of laws.

f. ***Dispute Resolution.*** Any and all disputes relating to or arising out of this Agreement including, but not limited to, the arbitrability and the validity of this Agreement shall be resolved by binding arbitration in Denver, Colorado.

g. ***Force Majeure.*** If the performance of any part of this Agreement by either party is prevented, hindered, delayed or otherwise made impracticable by causes beyond the reasonable control of either party, that party shall be excused from such performance to the extent that it is prevented, hindered or delayed by such causes.

h. ***Survival.*** The terms and provisions of sections 2, 3, 4, 5, 7, 8, 9, 10 and 11 shall survive any termination or expiration of this Agreement.

i. ***Entire Agreement.*** This Agreement constitutes the complete and exclusive statement of the agreement between the parties with respect to the Platform and supersedes any and all prior or

contemporaneous communications, representations, statements and understandings, whether oral or written, between the parties concerning the Platform.

7 Copyrights

NinjaTrader, LLC acknowledges the following:

Helix 3D Toolkit
Copyright (c) 2012 Oystein Bjorke

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

SharpDX
Copyright (c) 2010-2012 SharpDX - Alexandre Mutel

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Trading Technologies, Inc.
NinjaTrader SuperDOM is licensed under U.S. Patent Nos. 6,766,304 and 6,772,132, U.K. Patent Nos. GB 2,377,527 and GB 2, 390,451 and European Patent No. EP 1 319 211 from Trading Technologies, Inc.

OpenSSL

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
==========
This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).
This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay License

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
All rights reserved.
This package is an SSL implementation written
by Eric Young (eay@cryptsoft.com).
The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed.

If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used.

This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
"This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"
The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).
4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:
"This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE

DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publicly available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

WPF Property Grid
Copyright © 2010, Denys Vuika

Licensed under the Apache License, Version 2.0 (the "License") you may not use this file except in compliance with the License. You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, softwaredistributed under the License is distributed on an "AS IS" BASIS,WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Newtonsoft.Json
Copyright (c) 2007 James Newton-King
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Protocol Buffers
Google's data interchange format

MySql.Data

8 Introduction

Introduction Overview

This section of the Help Guide provides basic information about the NinjaTrader support and education resources available to you as well as helpful information on getting started with NinjaTrader.

8.1 Getting Started

Getting Started with NinjaTrader

This help guide contains a wide range of information on configuring and using all aspects of the NinjaTrader platform, but there a few key pages that can help you to get up and running quickly with the most important concepts for new users: Getting connected to market data, creating charts, saving a **Workspace**, and understanding the NinjaTrader **Control Center**.

- Getting Connected
- Understanding the basics of NinjaTrader Charts
- Understanding Workspaces in NinjaTrader
- Understanding the NinjaTrader Control Center

Once you have covered the basics, the following topics can help you to bridge the gap between basic and advanced understanding of NinjaTrader's features:

- Using the Overlay Instrument Selector to quickly change instruments in a trading window
- Creating and restoring Backup files
- Utilizing Advanced Trade Management (ATM) Strategies
- Analyzing Trade Performance
- Understanding the advanced features of NinjaTrader windows and tabs
- Contacting the NinjaTrader Support team for platform-related help
- Accessing the Support Forum to consult with experts and fellow traders

Getting Started with NinjaScript

This help guide contains educational and reference resources for NinjaScript developers of any experience level. The topics listed below can help you to quickly familiarize yourself with the resources available.

- The Distribution section provides resources for third party vendors distributing their code to end users
- The Editor section provides information on using the built-in **NinjaScript Editor**
- The Educational Resources section includes helpful information on a range of topics related to NinjaScript development.
- The Language Reference includes descriptions and reference information for NinjaScript properties, methods, and classes
  - If you know what you are looking for, the Alphabetical Reference can be used to quickly navigate to a specific Language Reference page
- The Strategy Analyzer is a fully featured module for backtesting and optimizing automated strategies on multiple fitness metrics

If you have questions or require assistance outside of the scope of this help guide, there are several resources available to get help from NinjaScript experts or other developers.

- The NinjaScript Development Support Forums feature discussion on general programming, as well as more specific areas related to different types of NinjaScript objects
- The NinjaScript File Sharing Forums provide an outlet to share NinjaScript objects you create, or to find publicly shared objects created by other developers

If you are looking to develop a specific type of NinjaScript object, the links below will lead you to Language Reference documentation for that Type.

- Add On
- Bars Type
- Chart Style
- Drawing Tool
- Import Type
- Indicator
- Market Analyzer Column
- Optimization Fitness
- Optimizer
- Performance Metrics
- Share Service
- Strategy
- SuperDOM Column

8.2 Getting Help & Support

NinjaTrader Support Policy

It is critical that you can rely on the support and service you receive from your trading platform provider. It is for this reason that NinjaTrader prides itself on its top quality support model that ensures you receive lightning fast and accurate turn around to your support inquiries. We

have found that delivering support electronically allows us to provide high levels of service in a cost efficient manner. Electronic support inquiries can be processed thirty times faster than traditional telephone support models which ensures that you get the necessary information when you need it. No more leaving messages in phantom voice mail boxes and no more waiting for thirty minutes in a telephone queue! Does this mean we do not have telephone support? Absolutely not! If we can't resolve your support inquiry electronically, we will be on the telephone with you right away and if required, login remotely to your PC to expedite a resolution!

So how do I get support?

- This help guide is interlaced with over one hour of instructional video and images
- Pressing F1 key anywhere in the NinjaTrader application will load context sensitive help
- Daily live interactive online training sessions (schedule)
- View online tips and tutorials on our YouTube page.
- Some of our connectivity providers are staffed with NinjaTrader support specialists. Please check with your provider to find out if they have live support for NinjaTrader.
- NinjaTrader Support Forum available 24 hours a day 7 days a week
- Send "**Mail To Support**" from the **Help** menu of the NinjaTrader application
- Send an email to the NinjaTrader support team

Support Priority

It is preferred that you send us a support email from the "**Mail To Support**" sub menu under the **Help** menu of the NinjaTrader application since it provides us with additional trouble shooting information, however, when sending an email to support, please provide the following information:

- Operating system
- Current NinjaTrader version (Can be accessed by selecting the **Help** menu from the Control Center followed by the **About** menu item)
- Who you are connected to for data
- Explanation of your problem

8.3 Learning to Use NinjaTrader

NinjaTrader provides a variety of ways for free and licensed users to learn and master the platform, including this help guide, the Video Library, the Support Forum, and weekly free live training sessions.

▼ Help Guides

Installation Guide

- The Installation Guide outlines the installation steps and provides the minimum PC requirements for NinjaTrader.

Connection Guides

- NinjaTrader is supported by numerous brokers around the globe, as well as a variety of market data providers.
  - You can create a connection to your broker or data feed by following the steps outlined in the specific Connection Guide for your broker or data provider.

User Help Guide

- This user help guide contains sections related to all of the features within NinjaTrader.
- In addition to accessing the help guide online, you can press the F1 key on your keyboard within the NinjaTrader platform to pull up this guide at any time. The guide will automatically open to a page related to the window you are viewing.

Video Library

New User Video Guides

- Reference quick tip videos in the New User Video Guides, designed to help you get up and running with the NinjaTrader platform.

Video Library

- The NinjaTrader Video Library contains a variety of videos on various features within the NinjaTrader platform.

NinjaTrader Support Forum

NinjaTrader Support Forum

- The NinjaTrader Support Forum is a great place to get and offer help, or discuss a range of topics with NinjaTrader experts or other traders
- The Support Forum is also a great resource for NinjaScript developers looking for a community of fellow developers.
- The Support Forum can be searched for specific items using the "search" feature, which quickly resolves the majority of questions.

Free Live Events

Watch Live Futures Trading & Analysis

- Join our Livestreams each weekday as we prepare, analyze and trade the futures markets in real-time.

NinjaTrader Partner Events

- NinjaTrader is pleased to sponsor weekly partner events as a value added service for our clients.
- These events are intended to provide increased exposure to the various trading styles and methods taught by our 3rd Party add on and Educational partners.
- You can email the NinjaTrader sales team today to be added to the partner event email list.

8.4 Using 3rd Party Add-Ons

3rd Party Add-Ons

NinjaTrader's comprehensive and flexible development environment empowers 3rd Party Developers to build rich and integrated apps. These add-ons allow for endless customization & expansion, leveraging 1000s of 3rd party indicators, strategies, and apps to build a custom trading setup to meet your requirements.

▼ Installing Add-Ons

Installing 3rd Party Add-Ons

After you have downloaded 3rd Party Add-On, they can be imported from the NinjaTrader Control Center.

1. From the Control Center window select the menu **Tools** > **Import**> **NinjaScript Add-On...** to open the "Import" dialog window
2. Select the file you want to import
3. Press the "**Import**" button

> **Notes**:
> 1. Your vendor may have different instructions for installing their **3rd Party Add-Ons**. Please check with the vendor for any specific guidelines they may require for installing their products.
> 2. If you receive an error during importing **"You have custom NinjaScript files on your PC that have programming errors..."**, please see the following post for information on how to resolve: How do I resolve NinjaScript Programming Errors?

▼ Understanding the impact of installing Add-Ons

Understanding the impact of installing Add-Ons

NinjaTrader provides a development environment allowing low-level access to 3rd party developers to build integrated indicators, drawing tools, automated strategies and more. An **Add-On** with software bugs can have adverse effects on the entire NinjaTrader application. These add-ons also natively run on your computer, therefore, its important to only install **3rd Party Add-Ons** from sources you trust.

The following symptoms post installation could indicate an Add-On is installed causing negative impact:

- Windows become slow or unresponsive to user interaction
- Market data becomes unusually slow to load or update
- Standard features fail to work as designed
- Other scripts fail to work as designed (custom scripts that work on their own may conflict with each other)
- Lost connections from market data providers
- Error messages are generated at various times
- The entire application shuts down abruptly and without warning

If you run into any of the above symptoms post installation of a **3rd Party Add-Ons** please try uninstalling the **3rd Party Add-Ons** to see if the problem goes away and contact the 3rd party developer for support.

Updating and Removing Add-Ons

Updating Add-Ons

Developers can issue updates to fix issues or add functionality. If you have obtained an updated copy of your **3rd Party Add-On**, you can import the new version using the same steps you originally used to install by going to **Tools** > **Import** > **NinjaScript Add-On...** and selecting the new file. During the import process, you will be given an option to replace the current **Add-Ons** which exists on your PC, which you should accept for each file you wish to update.

> Note: You should always restart NinjaTrader after installing an update to ensure you are running the most recent code.

Removing 3rd Party Add-Ons

Should you identify a problem, or suspect a **3rd Party Add-On** is causing problems, you may wish to remove these files from your system. The exact steps

to remove will depend on how it was distributed. **3rd Party Add-ons** can be installed either as a "**Protected Assembly**" or "**Non-Protected**". Please see the information below on how to proceed.

Removing Protected 3rd Party Add-Ons Assemblies

If you have purchased a **3rd Party Add-On**, it is likely distributed as a **Protected Assembly**. These protected files can be uninstalled by going to **Tools** > **Remove NinjaScript Assembly**. If you cannot find the **3rd Party Add-On** from this dialog, your **Add-On** is most likely a **non-protected** assembly.

Removing Non-Protected 3rd Party Add-Ons

Most free **3rd Party Add Ons** downloaded from online forums and other communities are distributed as **unprotected** c# scripts. These open-source files can be uninstalled using the following steps:

1. From the Control Center window select the menu **New** > **NinjsScript Editor** to open the NinjaScript Editor
2. On the right side, under the "**NinjaScript Explorer**" expand the type of folder of the **3rd Party Add-On** you are trying to uninstall
3. Locate the name of the **3rd Party Add-On (Note:** 3rd Party Add-Ons can be installed in several sub-folders**)**
4. Right click on entry > select "**Remove**"

Temporarily Disabling Add-Ons

Temporarily disabling Add-Ons

Should you start to experience an issue with NinjaTrader, the first step to isolate the problem is to determine if you continue to experience issues without **Add-Ons** enabled referred to as 'Safe Mode'. To enable safe mode, please use the following steps:

1. Exit NinjaTrader
2. Hold the CONTROL key on your keyboard and double-click the NinjaTrader icon.
3. Keep the CONTROL key held down until you see the NinjaTrader Control Center
4. Once you see the Control Center, you can verify you are in safe mode by going to Help > About.

Once in safe mode, you may use NinjaTrader without 3rd party add-ons, allowing you to verify if a problem no longer is present or allowing you to remove a 3rd party add-on.

9 Configuration

Configuration Overview

This section will provide you with guidance regarding various NinjaTrader configuration options and help you setup NinjaTrader for the first time.

9.1 Installation

Installation Overview

Please see the below resources for installing NinjaTrader, if you run into any problem installing the product please contact us at platformsupport@ninjatrader.com and we will quickly assist.

9.1.1 Minimum System Requirements

Minimum PC Requirements

Your PC must meet the minimum requirements listed below to run NinjaTrader
Windows 10, Windows 11, Windows Server 2016 or later 64-bit

- 1 gigahertz (GHz) or faster 64-bit processor
- 2GB RAM
- Microsoft .NET Framework 4.8
- (pre-installed on most PC's and can be downloaded here: Microsoft .NET Framework)
- Screen resolution of 1024 x 768
- DirectX10 compatible graphics card highly recommended

Recommended PC Specifications

NinjaTrader downloadable desktop platform was designed to take full advantage of modern PC architecture. To achieve the highest possible level of performance, NinjaTrader will utilize all available CPU cores and additional memory resources. Depending on your actual usage

with NinjaTrader, you may need more or less resources than the average user. Additional memory will be of direct benefit when running strategy optimizations, and the amount of additional memory needed is proportional to the number of CPU cores available.

- 2 (GHz) or faster quad core 64-bit processor
- 8 GB RAM
- DirectX 10 compatible graphics card
- SSD Hard Drive

9.1.2 Installation Guide

Follow the process outlined below to install NinjaTrader on your PC. To view minimum system requirements or recommended PC specifications, see the Minimum System Requirements page.

Installation Steps

1. If you do not have the Microsoft .NET Framework 4.8 installed on your PC please download and install it from here.
2. Log in to your NinjaTrader account
3. If you have an existing license you can import it by going to Settings, Plans, and then selecting Import a License. If you are upgrading NinjaTrader, you can skip this step and it will automatically be imported once upgraded.
4. Select Download on the left menu then download and install NinjaTrader
5. Firewall Software – NinjaTrader contacts our server on application log in for user validation. If you have a firewall, anti-spyware or other such software running on your PC, please ensure that you grant NinjaTrader permission to access the internet or you may receive a log in error.
6. Once the installation is complete, please review the appropriate Connection Guide to establish a connection to your broker or market data feed service provider

Warning:

- Please also take care that your Documents\NinjaTrader 8\ folder is excluded from any backup operation (cloudbased like Dropbox / OneDrive, or local), as this can create file access conflicts

9.1.3 Clear Browser Cache

How to clear your browser cache

In order to download or upgrade NinjaTrader you may need to clear your browser cache. Common errors that occur when this is the case are Cabinet File Errors and errors involving Temporary Files. If you receive one of these errors when installing or updating NinjaTrader please follow the steps listed below to accomplish a successful download of the NinjaTrader application.

Internet Explorer:

1. Select the **Tools** menu in the top right of the browser
2. Select the menu item **Internet Options**
3. Select the "**Delete your browsing history**" button.
4. Attempt the download again

Chrome:
1. Select the Chrome Menu button which is 3 horizontal lines in the top right of the browser
2. Select Tools
3. Select "**Clear browsing data**"
4. Select all check boxes and confirm the browser clear.
5. Attempt the download again

Firefox:
1. Select the "**Firefox**" menu button in the top left of the browser
2. Select **Options**.
3. In the advanced panel select the Network tab
4. Under the Cached Web Content section click "**Clear Now**"
5. Attempt the download again

9.2 Connecting

The connection to your NinjaTrader account is done within the Log In and Trading Mode windows. Additional connection management is done within the **Connection** menu in the Control Center. To configure connections from additional providers, **Multi-provider** must be enabled.

Connecting Overview

You must establish an account connection to either a NinjaTrader provided connection, your broker or a data feed in order to receive market data and trade either live or in simulation.

› Log In
› Trading Mode
› Playback Connection
› Multi-provider Connections

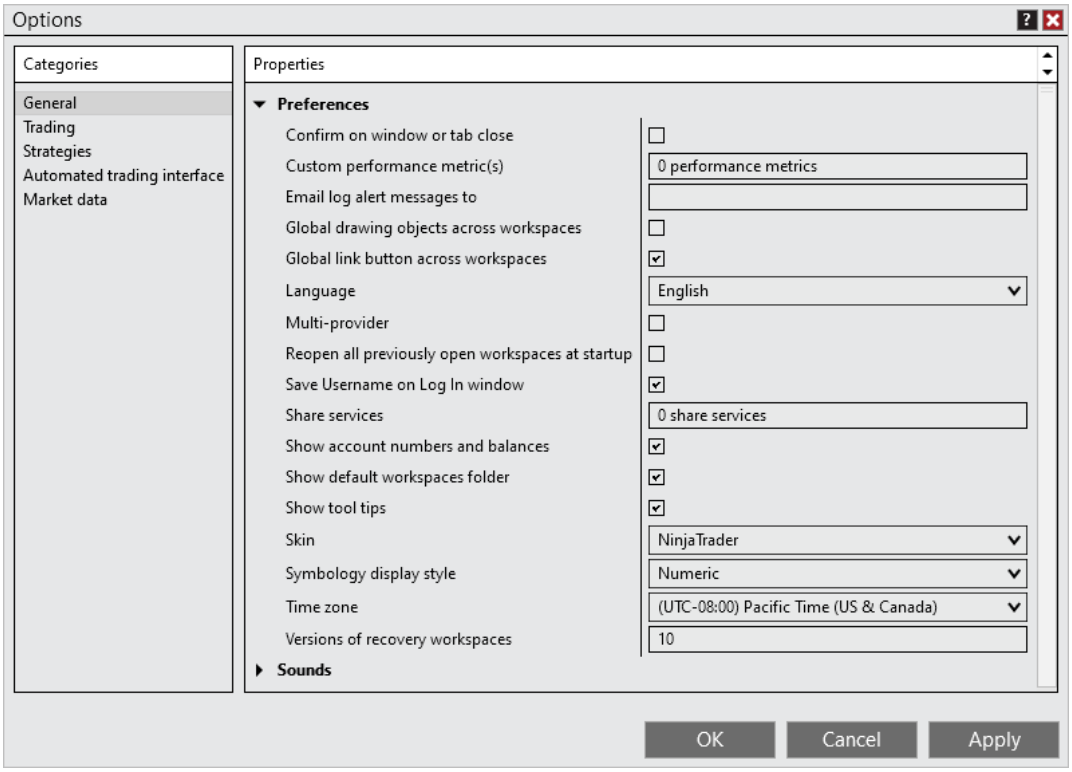
9.2.1 General

The General section sets general application options.

▼ Understanding general properties

General Properties

General properties can be set in the **Options** window with the **General** category selected. Each available property is described below:



Preferences	
Confirm on window or tab close	Enables or disables if the display of a dialogue box to confirm on tab or window close to prevent accidental window closures.
Custom performance metric(s)	Sets which custom performance metrics you would like included in account performance and strategy analyzer results. Performance Metrics are NinjaScript objects which can be created via the **NinjaScript Editor** and installed by third party vendors.
Email log alert	Sets the email address that you would like any alert messages from the log tab in the

messages to	**Control Center** to be sent to automatically. Leaving this field blank disables this feature. Note: For emails to be sent, you must first define a default email account to be used via the **Share Services** property below.
Global Drawing Objects Across Workspaces	Sets whether **Global Drawing Objects** should be applied across all open workspaces.
Global link button across workspaces	Sets whether the global link button will work across all open workspaces or only the current workspace.
Language	Sets the language you would like NinjaTrader to use. Changing this property requires a restart.
Multi-provider	Sets whether the ability to configure and connect to other providers is available. See the Enabling/Disabling Multi-provider Mode section of the Help Guide.
Reopen all previously open workspaces at startup	Enables or disabled the ability to open multiple workspaces at start up.
Save Username on Log In window	Saves the last used Username to the Log In window for next log in.
Share services	Manages your defined social network and email accounts. You must first set up a **Share Service** to enable sharing functionality

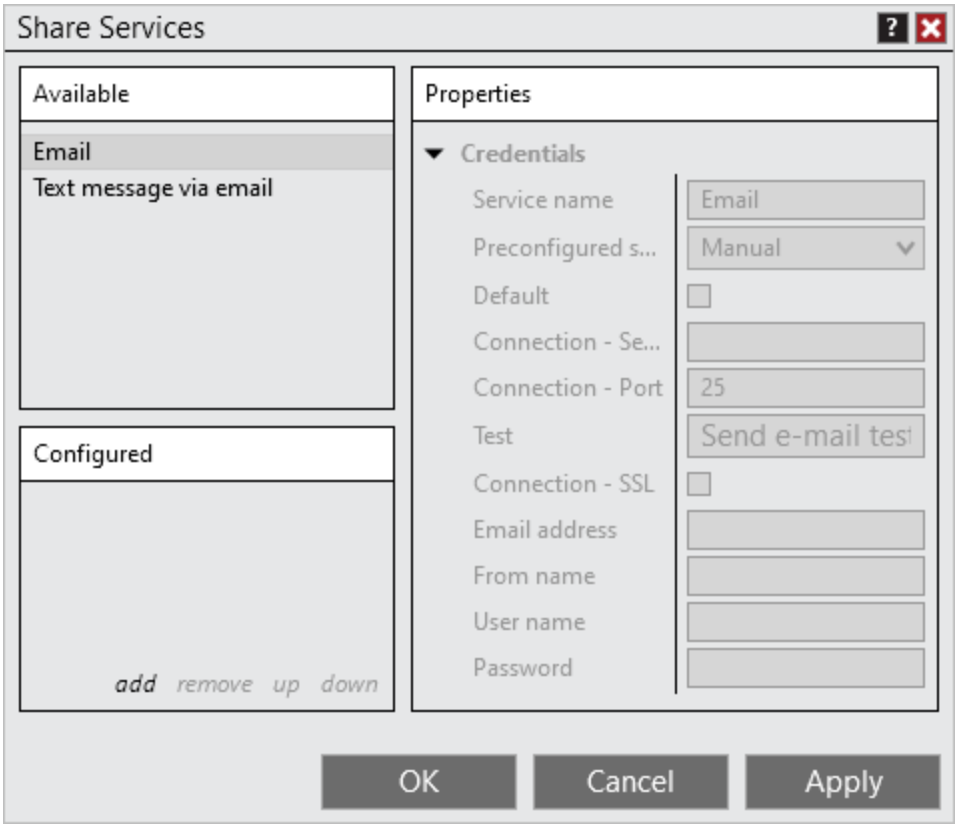
	from NinjaTrader. See the "Managing Sharing Services" section below for more information.
Show account numbers and balances	Sets whether account values that may be considered private are displayed or have asterisks in their place to hide them.
Show tool tips	Sets whether description tool tips will be displayed. Note: Tool tips that show cut-off text will still function.
Skin	Sets the skin you would like to use for NinjaTrader. Changing this property requires a restart. Skins are NinjaScript objects which can be created and modified.
Symbology display style	Selected the style in which symbols are displayed across all your NinjaTrader platforms.
Time zone	Sets the time zone that NinjaTrader will use. All charts and market data will be displayed in this time zone. Time zones are set to your local PC time by default.
Versions of recovery workspaces	Indicates how many version back of saved recovery workspaces to retain
Sounds	
Play consecutively	Sets whether sounds will be queued to play in sequence without overlap, or if simultaneous sounds will play at the same time.
Alert Sounds	All alert sounds are listed in alphabetical order by alert name. Sound files can be replaced by clicking any of these fields, or muted by

	clicking the small X icon to remove the assigned sound file.

Managing Share Services

Share Services

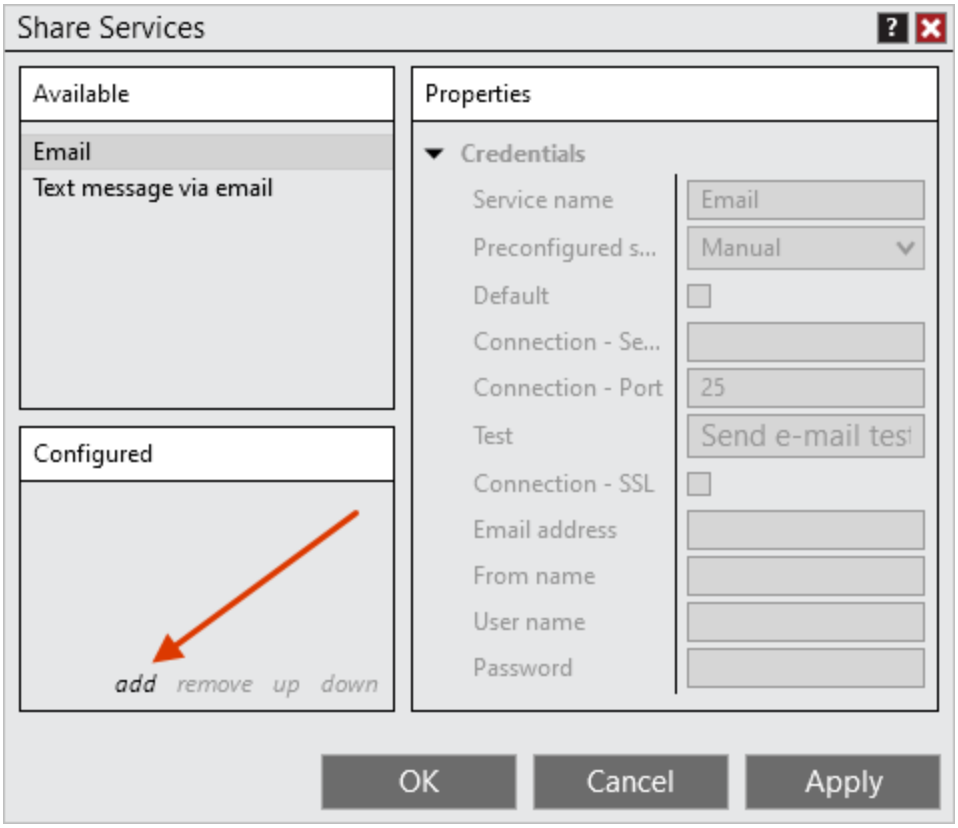
The **Share Services** dialog allows you to set up your various social media accounts. NinjaTrader ships with Sharing Adapters for Email and Text message via email, and it is possible for developers to create their own ShareService in NinjaScript to access other social media outlets.



Depending on the available sharing service you are configuring changes the settings needed to complete the setup. Please see the below guides for setting up each of the sharing services pre loaded with NinjaTrader 8

Email Service Setup

To setup an Email account that can be used to send messages from NinjaTrader select **Email** from the available section and click **add**. The Properties section is now available to enter the needed information to set up your Email **Share Service**. NinjaTrader needs valid SMTP email server that it can use to send outbound emails. Typical settings for some of the most common providers can be quickly entered by selecting your provider from the **Preconfigured settings** menu.



Credentials	
Service name	Sets the name of the **Share Service** that will be used to identify this account when selecting a service to which to share content
Preconfigured settings	Provides typical settings for some of the most common providers

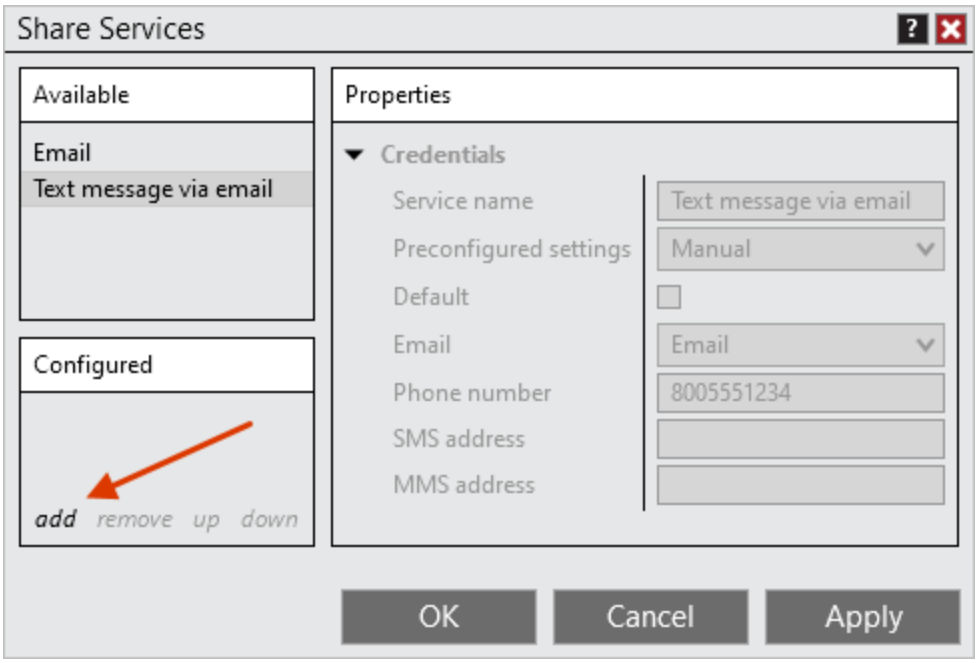
Connection - Server	Sets the server address used for the SMTP connection
Connection - Port	Sets the server port used for the SMTP connection
Default	Sets whether is the default sharing service to be used for automated sharing from NinjaScript. Note: You can only have one default per service type.
Connection - SSL	Sets whether your email server uses Secure Sockets Layer (SSL) security
Email address	The Email Address that will be used for the "From" field when sending outbound emails.
User name	Sets the server user name
Password	Sets the server password
Test	Sends a test email through the server

Notes: Port 465 is not supported for SMTP servers.
AOL uses an App Password for the Password field. This is different than your AOL Password and needs to be configured within AOL's Account Security settings

Text Message Via Email Service Setup

To setup a text messages account that can be used to send messages from NinjaTrader select **Text message via email** from the available section and click **add**. The Properties section is now available to enter the needed information to set up your Text message via email **Share Service**. An email share service must be set up before configuring the Text message via email share service. NinjaTrader needs SMS address and MMS address for the entered phone

number. Typical settings for some of the most common providers can be quickly entered by selecting your provider from the **Preconfigured settings** menu.



Credentials	
Service name	Sets the name of the **Share Service** that will be used to identify this account when selecting a service to which to share content
Preconfigured settings	Provides typical settings for some of the most common providers
Default	Sets whether is the default sharing service to be used for automated sharing from NinjaScript. Note: You can only have one default per service type.
Email	The Email Share Service that will be used to send the text message from

Phone number	The phone number that will receive the text messages
SMS address	The service providers SMS address for sending text messages (do not enter a phone number here)
MMS address	The service providers MMS address for sending multimedia messages (do not enter a phone number here)

9.2.1.1 Creating your own Skin

You can create your own skin by creating a copy of the skin template located in the "My Documents > NinjaTrader > Templates > Skins" directory. Do not modify skin templates directly, as on each new installation they will be overwritten by the NinjaTrader installer. Instead, first make a copy of a skin directory, then rename the folder to the desired skin name. On restart of NinjaTrader, the new skin directory will be detected, allowing you to switch over to the skin to activate it.

Skin File Structure

Skins consist of XAML files corresponding to different windows in the NinjaTrader platform. Each pre-built skin includes a "BluePrint.xaml" file that contains most of the shared application keys that can be used. In addition to this file, you will find individual XAML files for windows such as FXPro, BasicEntry, and Level2.

Creating A New Skin

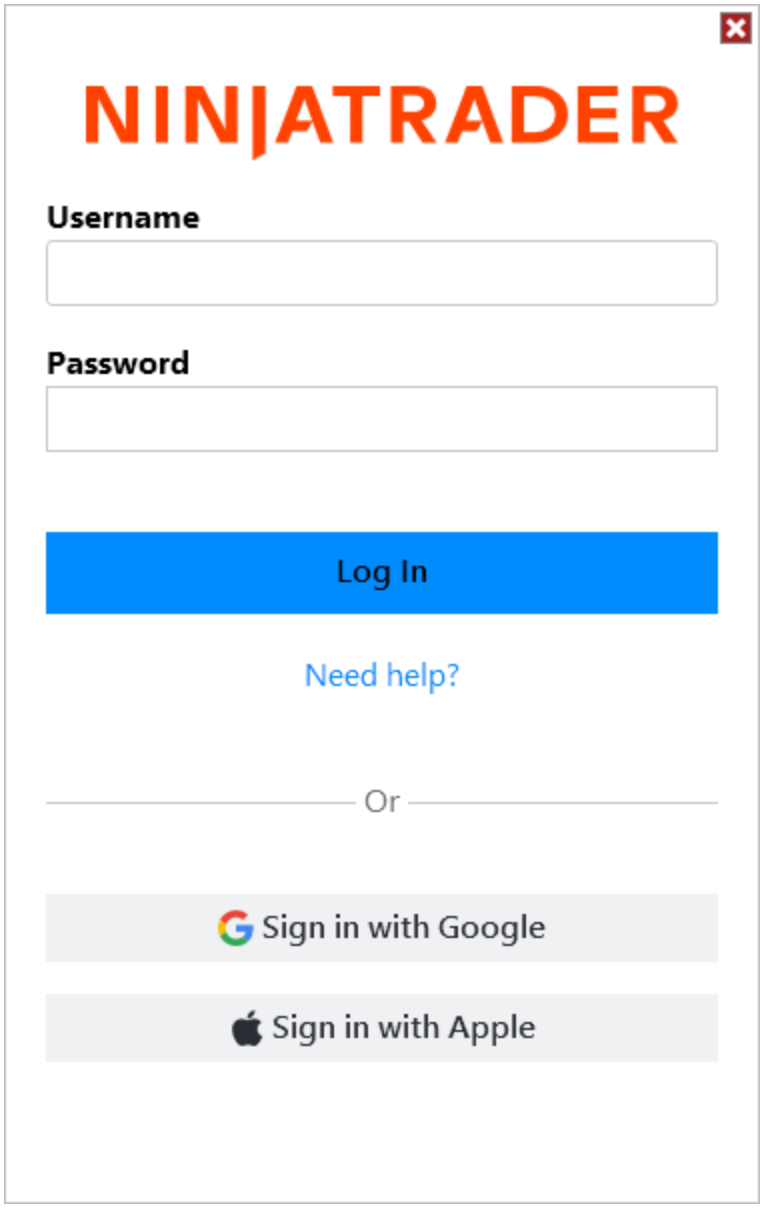
The most efficient way to customize a skin for NinjaTrader is to begin with an existing skin located in the C:\Users\<user>\Documents\NinjaTrader 8\templates\Skins directory on your PC. Each XAML file in the Skins that come pre-loaded with NinjaTrader is fully commented, with notes pointing to the areas of the application affected by each logical grouping of XAML tags. The following general process can be used when creating a new skin.

- Determine which pre-built skin most closely resembles your end goal
- Copy the folder containing all of the skin's files, then paste and rename the folder in the same directory
  - The files in this new folder will comprise your new skin
- Open BluePrint.xaml for your new skin, and begin to edit the XAML tags as desired
- If you wish to test a change at any time, first save the file you are working on, then close and restart NinjaTrader to view the change
- When finished with BluePrint.xaml, repeat the process for each of the other files

- These other files are significantly smaller than BluePrint.xaml

9.2.2 Log In

When starting NinjaTrader you will be presented with a Log In window.



Warning: Your Log In is only for your NinjaTrader account, your entitlements, and Server side ATM Templates. All items within NinjaTrader on your local computer will be available to anyone who logs in: Configured Multi-provider connections, NinjaScripts, Workspaces, etc.

Log In credentials

You will log in with the Username and Password you use on account.ninjatrader.com
If you do not have an account, you can register for an account at account.ninjatrader.com/register
You can also set up a log in using your Google or Apple account. This is only available for a demo.

Log In with Username and Password

Once you have your NinjaTrader credentials, you can enter them under the Username and Password fields, then press **Log In**

Log In with Google or Apple

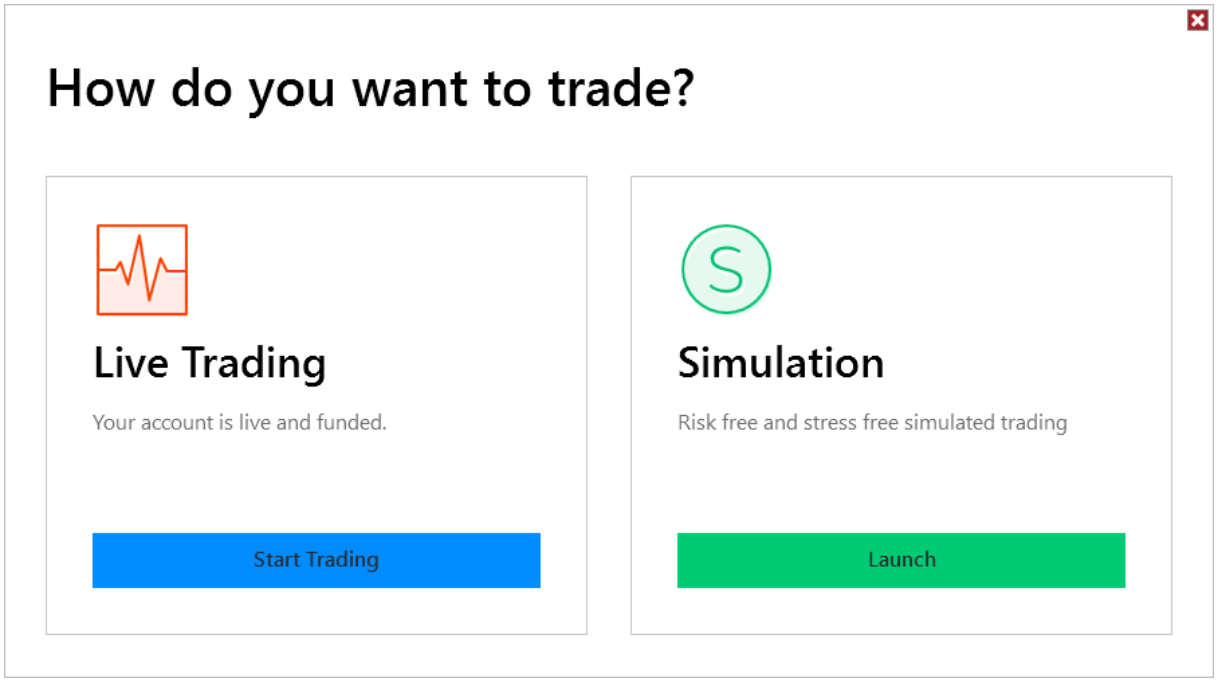
If you registered for a demo using your Google or Apple account, you can select to connect with those options.

Forgot Username or Password

To retrieve a forgotten username or password select **Need help?**.

9.2.3 Trading Mode

After Log In you will be presented with the Trading Mode window.



Live Trading

The Live Trading section will display the current state of your live account. If your account is live and funded, you can select Start Trading to connect.

Simulation

The Simulation section will display the current start of your simulation account. If your account is active your can select Launch to connect.

Note: If Multi-provider is enabled the Trading Mode screen will be skipped. You can then select what to connect to under the **Connections** menu of the Control Center.

9.2.4 Playback Connection

Playback Connection

The Playback connection is a default connection installed with NinjaTrader. Its purpose is for replaying NinjaTrader recorded data files or historical data. See the Playback Connection section of the Help Guide for further details.

9.2.5 Multi-provider Connections

Multi-provider Connection Overview

To enable Multi-provider, within the Control Center go to **Tools**, **Options**, and check **Multi-provider**. See the Enabling/Disabling Muli-provider Mode section of the Help Guide.

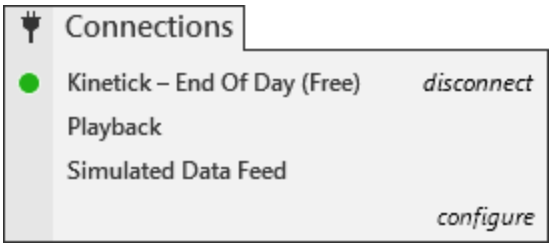
- Connecting to Multi-provider Connections
- Connecting to Kinetick - End Of Day (Free)
- External Data Feed Connection
- Simulated Data Feed Connection

9.2.5.1 Connecting to Multi-provider Connections

▼ Understanding account connections

Account Connections

Account connections can be managed under the Control Center window by selecting the **Connections** menu. Connected connections will have a green circle next to them. You can select what connections you want to connect to or disconnect to. When Multi-provider is enabled, you will be able to configure any additional connection or connect to ones already configured. See the Enabling/Disabling Muli-provider Mode section of the Help Guide. A connection is where you set up your user name, password and any relevant information that allows you to establish a connection to your broker and/or data feed service.



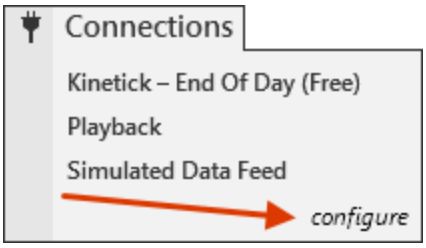
How to create an account connection

Creating an Account Connection

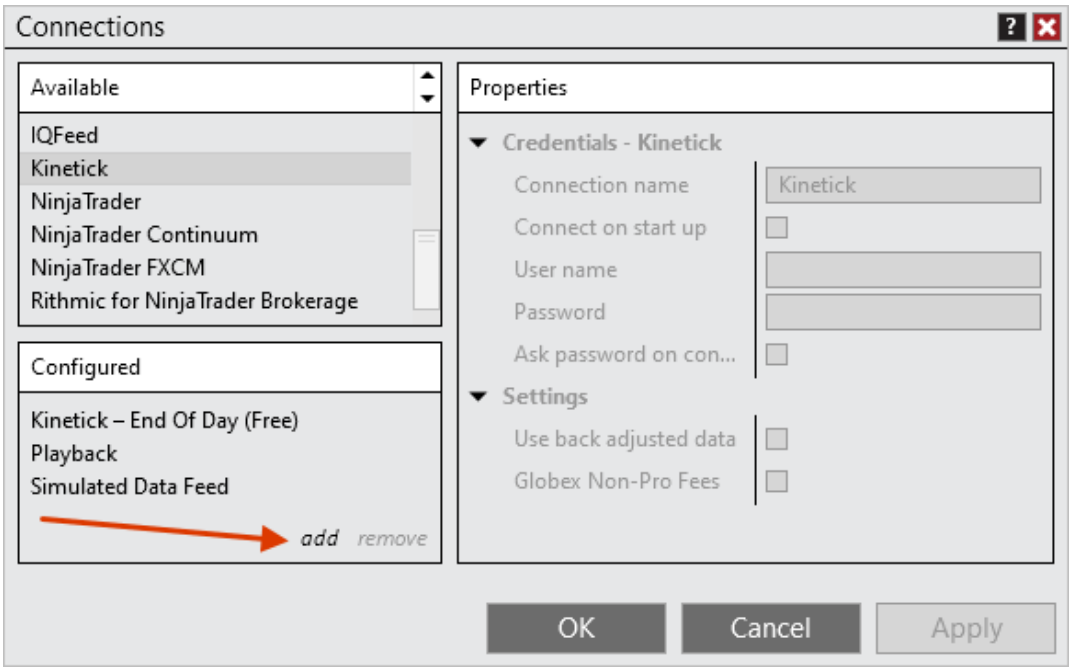
Within the **Connections** menu you can add a connection, change a connection or remove a connection. The following steps use Kinetick as the connectivity provider. This provider is used for demonstration purposes. You can access broker/technology specific connection help information via the NinjaTrader Connection Guide.

To create an account connection:

1. Open the **Connections** window by going to the **Connections** menu within the Control Center and selecting "configure"



2. Select the connection provider you want to create a connection for in the **Available** section and select "add".

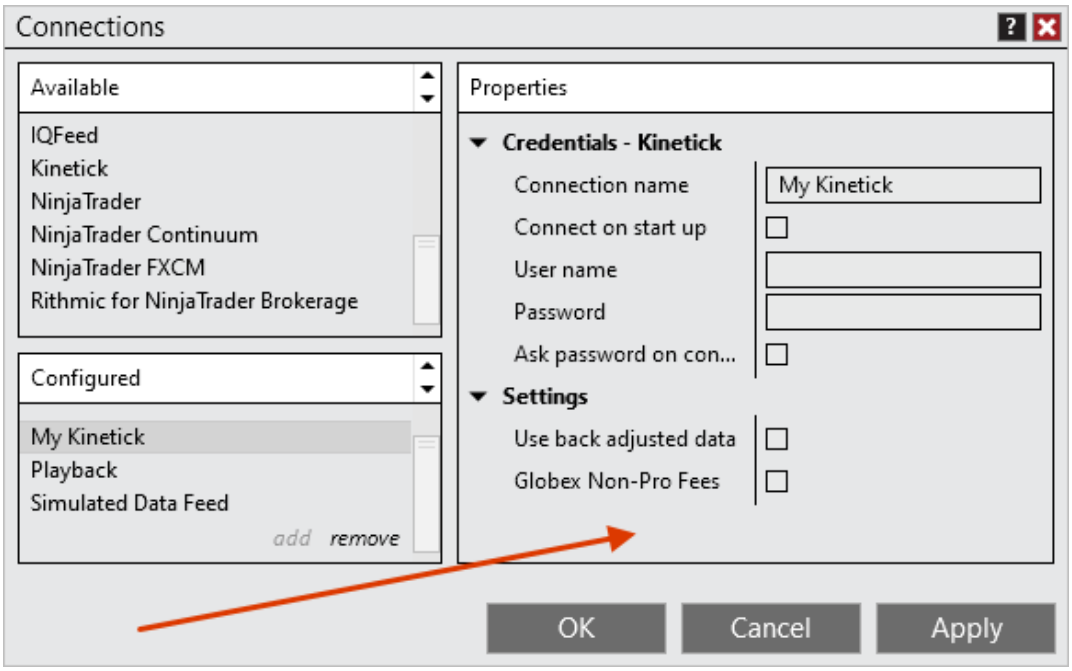


3. After selecting "add" you will be prompted to supply the following information:

- User defined connection name (Only use alphanumeric characters in the connection name)
- Optionally select "Connect on start up" to automatically connect to this connection when NinjaTrader is started.

Note: Please test and ensure your connection is working as expected before using this option as it is possible to input incorrect credentials which could prevent the startup of NinjaTrader.

- Various Settings which are specific to your connection. Please see the NinjaTrader Connection Guide for more information.



4. Press the "OK" button to finish configuring the connection. Now in the NinjaTrader **Control Center** **connections** menu you will be able to select the newly created connection by its connection name to connect.

Warning: All configured Multi-provider connections will be saved to your local system and anyone logging in on your local system will be able to access them.

9.2.5.2 Connecting to Kinetick - End Of Day (Free)

What is Kinetick?

Kinetick provides FREE end of day data for stocks, futures and forex. Real-time data service starts as low as $60 per month and you can qualify to have your CME Globex Futures exchange fees reduced with a qualified brokerage account. Please visit www.kinetick.com for more detailed information.

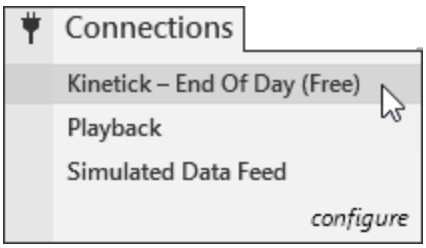
How to connect to Kinetick for FREE end of day data

Connecting to Kinetick

The **Kinetick - End Of Day (Free)** connection provides FREE end of day data for stocks, futures and forex. It is built into NinjaTrader and can be used directly out of the box with no additional steps. (Kinetick is not available for older versions of NinjaTrader 6.5 and earlier)

To connect to Kinetick for FREE end of day data:

1. Left mouse click on the **Connections** menu from the **Control Center**
2. Left mouse click on the **Kinetick - End Of Day (Free)** menu item



Please visit www.kinetick.com for information on signing up for real-time data.

9.2.5.3 External Data Feed Connection

External Data Feed Connection

The **External Data Feed** connection is a default connection installed with NinjaTrader. In combination with the DLL Interface, it provides 3rd party applications the ability to drive NinjaTrader with market data.

This connection targets those traders who have programming experience and wish to create a market data link between their charting or custom application and NinjaTrader which allows them to use the full functionality of NinjaTrader simulator. Please refer to the **Ask** and **Bid** functions of the **DLL** Interface.

9.2.5.4 Simulated Data Feed Connection

Simulated Data Feed Connection

The **Simulated Data Feed** connection is a default connection installed with NinjaTrader. Its purpose is to play internally generated market data for simulation.

This connection can be used for:

- Offline simulated training and practice of NinjaTrader
- Offline testing of strategies

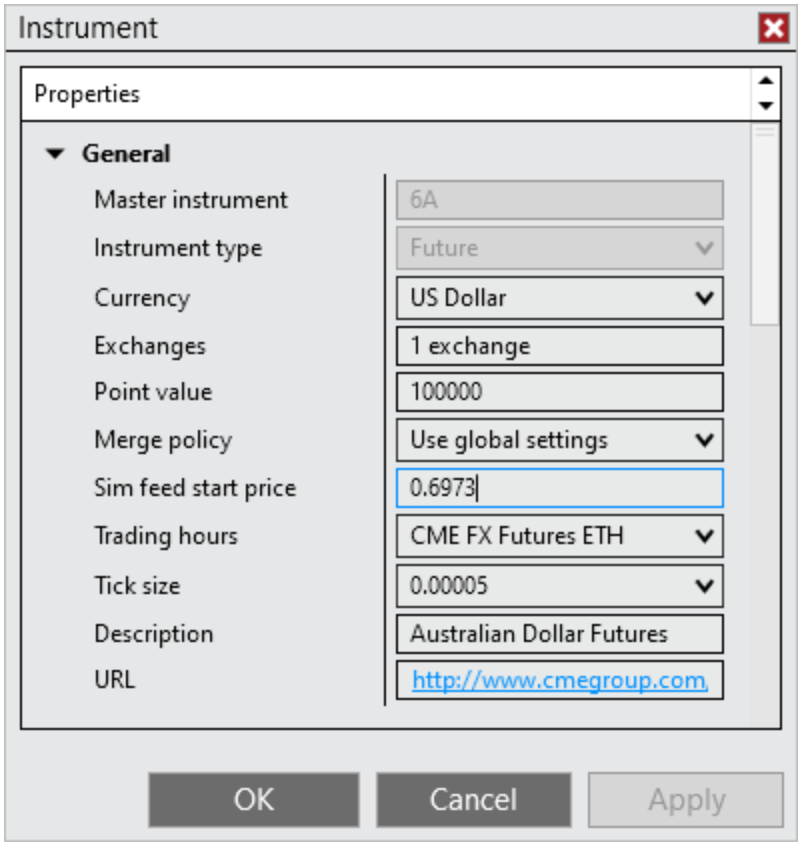
- Offline testing of trade automation using NinjaScript strategies or the NinjaTrader Automated Trading Interface

Notes:
1. This connection is a random internally generated market and has **NO** correlation to real market data
2. This connection does not provide historical data

Tip: The **Simulated Data Feed** will continuously run and generate data once connected and drive all NinjaTrader windows, however please keep in mind the Trading Hours definitions used will still govern for which time periods your window (i.e. Chart, Market Analyzer, SuperDOM Indicators and Columns) can receive the data to display.

Sim Feed Start Price

The **Simulated Data Feed** will automatically use the last price from the last connection as the starting price for the instrument.



Defining the Sim Feed Start Price

To manually set an instrument starting price for use with the **Simulated Data Feed**:

1. Left mouse click on the **Tools** menu in the Control Center and select the **Instrument** menu item
2. Search for the desired instrument and select it
3. Press the **edit** button and set a **Sim feed start price** value

Once you are connected to the **Simulated Data Feed** connection, the instrument will begin simulated trading at the **Sim feed start price** value.

Trend Slider

The **Trend** slider control will appear once connected to the **Simulated Data Feed**. Left mouse click on the slider and drag it up or down to cause the **Simulated Data Feed** to move in that direction.

9.3 Options

Options Overview

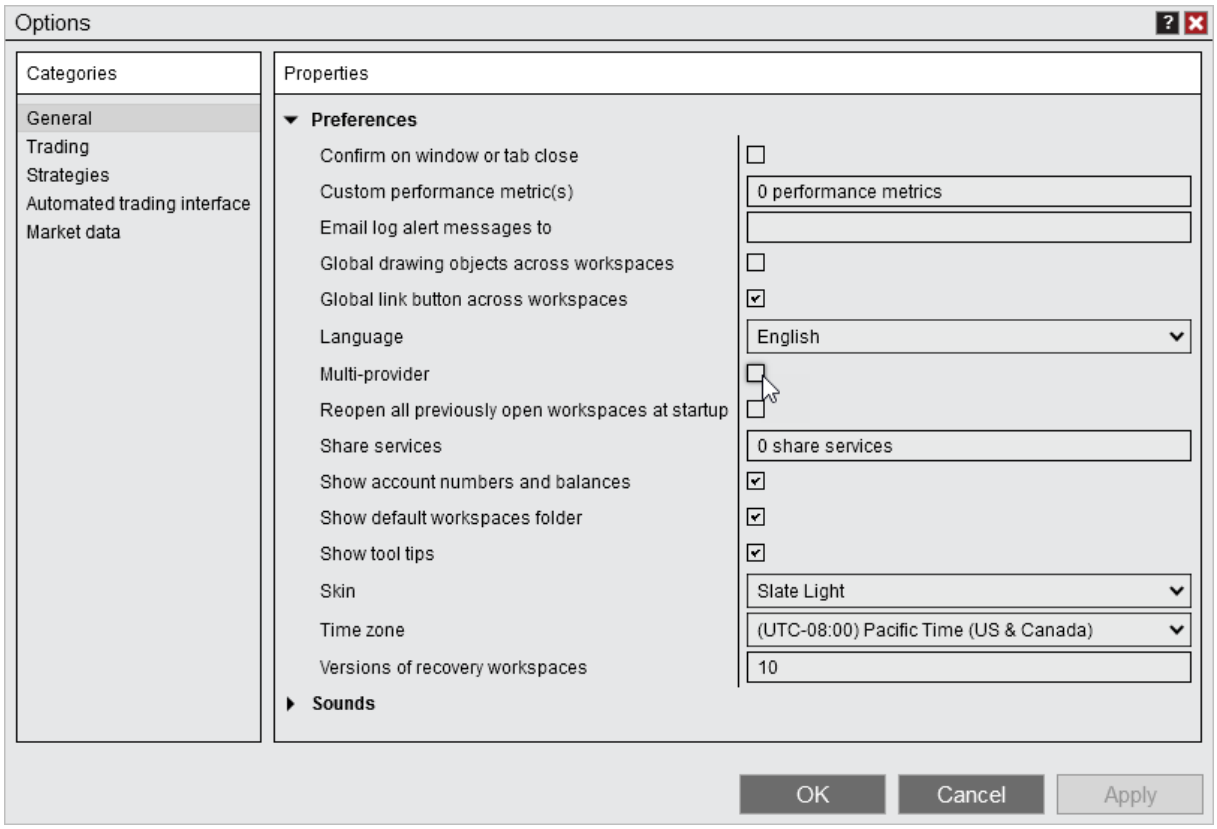
To access the **Options** menu, select the **Tools** menu within the NinjaTrader Control Center and select the menu item **Options**.

Various options can be configured inside the **Options** menu.

- General
- Trading
- Strategies
- Automated trading interface
- Market data

9.3.1 Enabling/Disabling Multi-provider Mode

Multi-provider mode can be enabled or disabled under Tools> Options.



Multi-provider disabled

When logging in you will be presented with the Trading Mode window which will display the states of your Live and Simulation account. You will be able to select what account you want to connect to. Connecting to Live will only connect you to your live account, so that you don't trade to simulation in error. Connection to Simulation will connect you only to your simulation account, so you don't trade live in error. After you make a selection, you will be connected and taken to NinjaTrader. Optionally, you can connect to both Live and Simulation at the same time. To do so, after making your initial connection, you can select either **Live** or **Simulation** under the **Connections** menu.

Shadow strategies for local ATMs will be disabled since there is no local simulation account to send the orders to. Additionally, risk and commissions templates can't be applied to the playback connection, since they need to be applied to the local simulation account.
When Multi-provider is disabled you can opt-in to using Server side ATMs under Tools> Options> Strategies. See the Server Side vs Local ATMs section of the help guide to understand how they function differently. Server side ATMs cannot be used will connected to **Playback**.

Multi-provider enabled

When logging in you will be taken directly to NinjaTrader. You can connect to any configured connections under the Connections menu. Any connections set to **Connect on start up** will

auto connect. Under **Connections** the **configure** button will be available to set up additional providers.
Local simulation accounts will always be available when Multi-provider is enabled. A **Global simulation mode** will be available under **Tools** and is enabled by default to ensure trades are not made to live accounts in error.

Warnings:
Configured Multi-provider connections will be available to any user that connects to NinjaTrader on your local computer.

If you opted into using **Server side ATMs (beta)** and you then enable Multi-provider mode, you will be switched to local ATMs. Server side ATMs are only available for the NinjaTrader connection. See the Server Side vs Local ATMs section of the help guide to understand how they function differently.

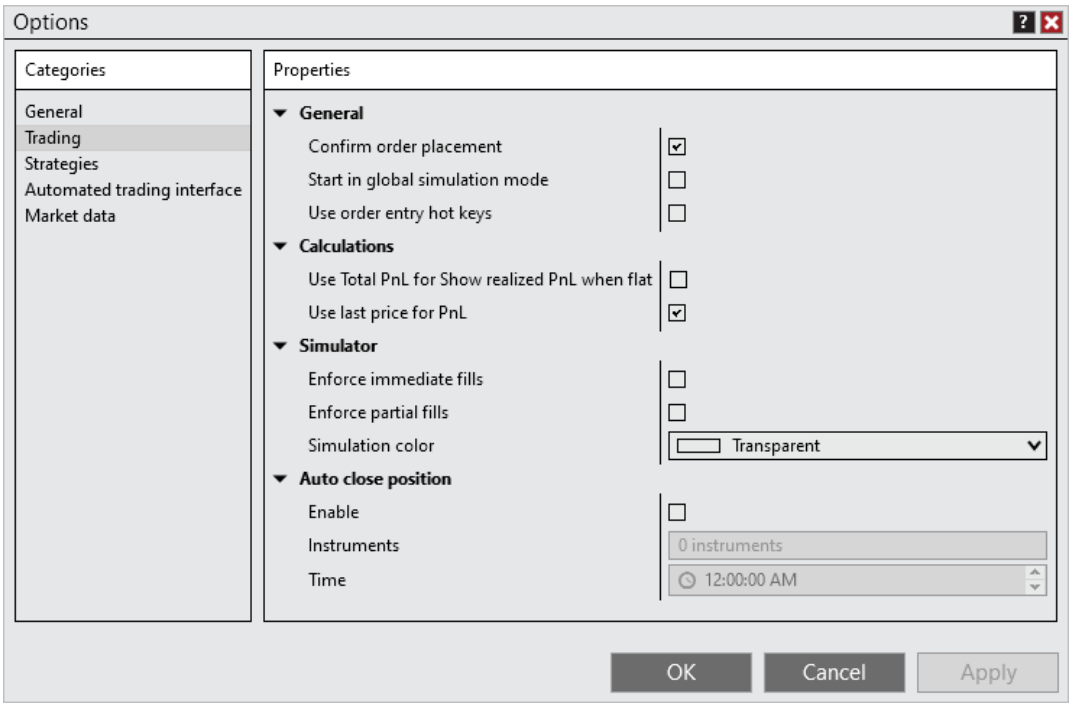
9.3.2 Trading

The **Trading** category sets all trading properties and allows configuration of simulation accounts.

▼ Understanding trading properties

Trading Properties

Trading properties can be set in the **Options** window with the **Trading** category selected. There are several properties each are described below:



General	
Confirm order placement	Sets if NinjaTrader will open a popup to confirm each order placed preventing an accidental order submission.
Start in global simulation mode	Sets if NinjaTrader will start with **Global Simulation Mode** enabled preventing live orders from being submitted until you manually disable **Global Simulation Mode**. This is only available if Multi-provider is enabled.
Use order entry hot keys	If checked NinjaTrader will allow you to submit orders using the order-entry **Hot Keys**, which can be defined in the **Hot Keys** window. See the "Hot Keys" section for more information.
Calculations	

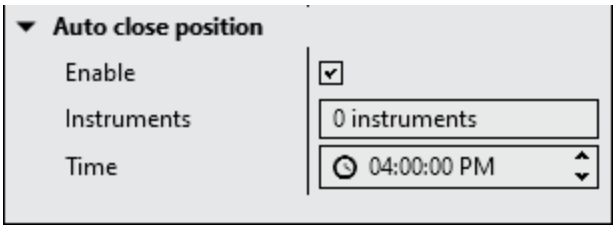
Use Total PnL for Show realized PnL when flat	Enables or disabled Total PnL rather than Gross PnL being displayed in order entry windows when **Show realized PnL when flat** is also enabled.
Use last price for PnL	Sets if the last trade price is used to calculate profit and loss, or if the Bid will be used for long positions and the Ask for short positions.
Simulator	
Enforce immediate fills	Sets if orders on simulation accounts will be filled immediately instead of using the NinjaTrader advanced simulation fill engine.
Enforce partial fills	Sets if partial fills will be forced on simulation orders. When disabled, orders are filled based on the NinjaTrader advanced simulation fill engine.
Simulation color	Sets the background color of any order interface that has a simulation account selected. This feature is disabled if set to "Transparent".
Auto close position	
Enable	Sets if NinjaTrader will close out any positions automatically at the specified time. For see more information, see the *"Understanding the auto close position function"* section below.
Instruments	Sets the instruments for which NinjaTrader will attempt to close positions at the specified time. This is set for each individual order-entry window, and can be set here by selecting the Instruments field and clicking **add** in the window that appears.

	If **Auto close on all instruments** is checked, this will globally work on all instruments in the platform, meaning disabling **Auto close** on an order entry window will have no affect since it is already globally set to close.
Time	Sets the time at which NinjaTrader will attempt to automatically close positions held in the instruments set in the instruments field. **Note**: The time will be based upon the timezone set up in the General section of the **Options** window.

Understanding the auto close position function

Auto Close Position

NinjaTrader can be set to automatically attempt to close a position at a designated time that is configured in the **Tools** > **Options** > **Trading** menu, you can also add instruments through the NinjaTrader trading interfaces via right click and selecting **Auto close position**.



Auto close position	
Enable	Sets if NinjaTrader will close out any positions automatically at the specified time.
Instruments	Sets the instruments for which NinjaTrader will attempt to close positions at the specified time. This is set for each individual order-entry window,

	and can be set here by selecting the Instruments field and clicking **add** in the window that appears If **Auto close on all instruments** is checked, this will globally work on all instruments in the platform, meaning disabling **Auto close** on an order entry window will have no affect since it is already globally set to close.
Time	Sets the time at which NinjaTrader will attempt to automatically close positions held in the instruments set in the instruments field. **Note**: The time will be based upon the timezone set up in the General section of the **Options** window.

Note: This feature is not available to Direct Edition license users. Please contact platformsales@ninjatrader.com for upgrade options.

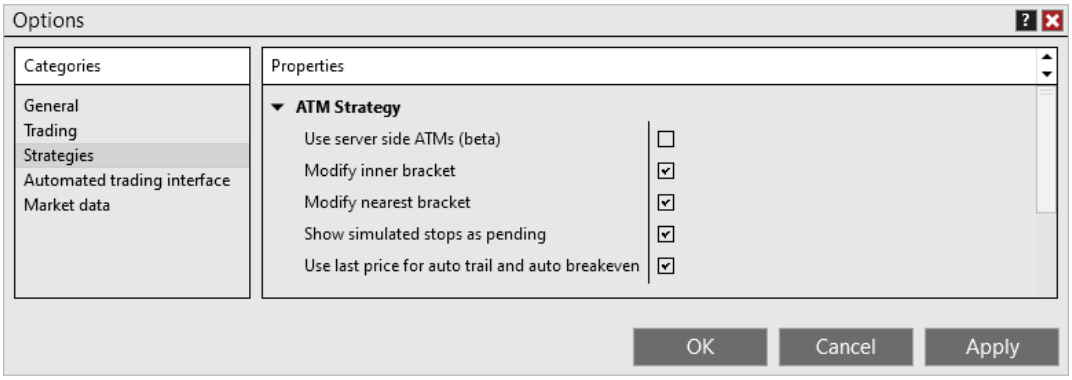
9.3.3 Strategies

The Strategies category sets options regarding handling of ATM Strategies and NinjaScript strategies for automated system trading.

▼ Understanding ATM strategy properties

ATM Strategy Properties

This property group sets ATM strategy handling options.



ATM Strategy	

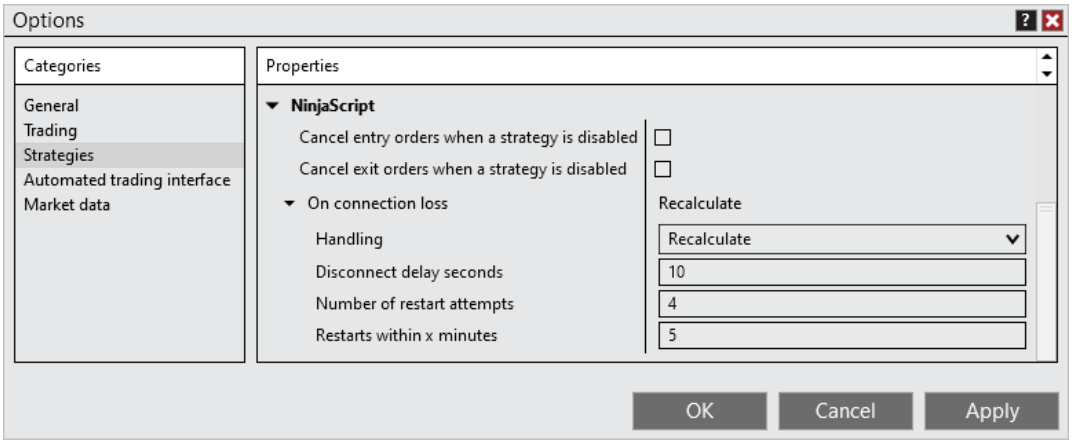
Use server side ATMs (beta)	When enabled the new ATMs that work server side will be available. This is only available when Multi-provider is disabled. See the Sever Side ATMs section of the help guide to understand the differences between them and the legacy ATMs.
Modify inner bracket	When enabled, and when scaling into a position managed by an **ATM Strategy**, the inner bracket of stop loss and profit target order quantities will be modified to reflect the increased position size. When disabled, the outer bracket will be modified. This setting is only available when Use server side ATMs is disabled.
Modify nearest bracket	When enabled, the nearest bracket of stop loss and profit target order quantities are modified when changing the quantity of a stop loss or target order in a multi-target **ATM Strategy**. This property is used in conjunction with "Modify inner bracket". For example, if both "Modify inner..." and "Modify nearest..." are enabled and you modify target 2 from 1 contract to 2 contracts, target 1 order size will be reduced by 1. If you had "Modify inner..." disabled, target 3 order size will be reduced by 1. This setting is only available when Use server side ATMs is disabled.
Show simulated stops as pending	When enabled, simulated stops will show with an order start of Trigger pending. This setting is only available when Use server side ATMs is disabled.
Use last price for auto trail and auto breakeven	When enabled, the last traded price is used to trigger auto trail or auto breakeven functions. When disabled, the Bid is used for long positions, and the Ask is used for

	short positions. This setting is only available when Use server side ATMs is disabled.

Understanding NinjaScript properties

NinjaScript properties

This property group controls how NinjaTrader will run your NinjaScript strategies.



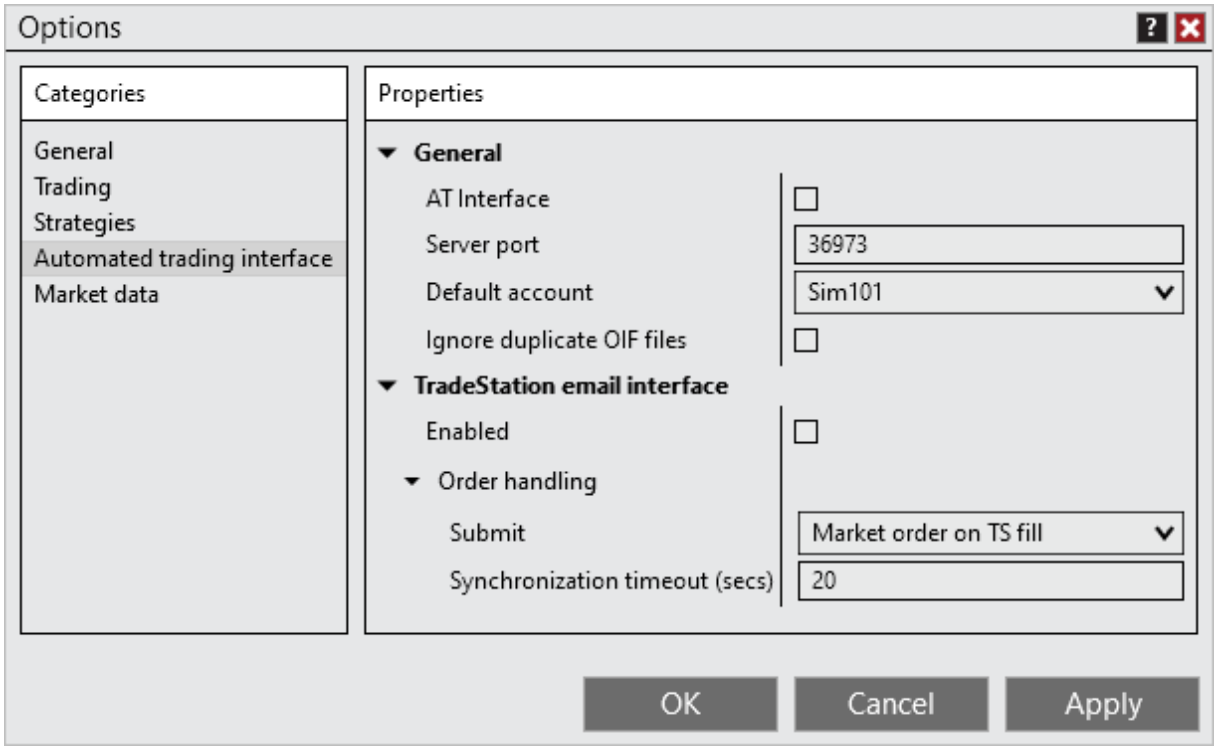
Cancel entry orders when a strategy is disabled	Enables or disables automatic cancellation of a NinjaScript strategy's entry orders when the strategy is disabled.
Cancel exit orders when a strategy is disabled	Enables or disables automatic cancellation of a NinjaScript strategy's exit orders when the strategy is disabled.
On connection loss	Expand this category to set connection loss handling parameters.
Handling	Sets the action a NinjaScript strategy will take after a disconnection occurs:

	Keep Running: Keeps the strategy running and logs the disconnection. When the connection is reestablished, the strategy will resume as if no disconnection occurred. **Recalculate**: The strategy will attempt to recalculate its strategy position when a connection is reestablished and held for longer than 10 seconds. Recalculations will only occur if the strategy was stopped based on one of the conditions below. Should the connection be reestablished before the strategy is stopped, the strategy will continue running without recalculating as if no disconnection occurred. • If data feed disconnects for longer than the time specified in “Disconnect delay seconds”, the strategy is stopped and the disconnection is logged. • If the order feed disconnects and the strategy places an order while disconnected, the strategy is stopped and the disconnection is logged. • If both the data and order feeds disconnect for longer than the time specified in “Disconnect delay seconds”, the strategy is stopped and the disconnection is logged. **Stop Strategy**: Automatically stops the strategy and logs the disconnection when disconnected for more than "Disconnect Delay Seconds". No action will be taken when a connection is reestablished.
Disconnect delay seconds	Sets the number of seconds a disconnection must persist before it is

	recognized by the Disconnect Handling logic
Number of restart attempts	Sets the number of times NinjaTrader will attempt to restart a strategy within the "Restarts within x minutes" time span. The strategy will only restart on a reestablished connection when there have been fewer restart attempts than "Number of restart attempts" within the last "Restarts within x minutes" time span. Otherwise the strategy will simply halt, and no further restart attempts will be made.
Restarts within x minutes	Sets the number of minutes for the "Restarts within x minutes" time span used by "Number of restart attempts".

9.3.4 Automated trading interface

The **Automated trading interface** section sets options for the Automated Trading Interface.



General Properties

This property group sets the general ATI (**Automated trading interface**) properties.

General	
AT Interface	Sets if the **automated trading interface** is enabled. This will not affect the remaining API methods.
Server port	Default port number for communicating with NinjaTrader via the DLL interface.
Default account	Sets the default account for automated trading. If no account is specified the default account is used.
Ignore duplicate OIF files	Enables or disables ignoring duplicate OIF files. If enabled, any OIF files with the same name during the current NinjaTrader session will be ignored.

TradeStation Email Interface Properties

This property group sets the TradeStation email interface properties. Detailed information on the TradeStation email interface can be found here.

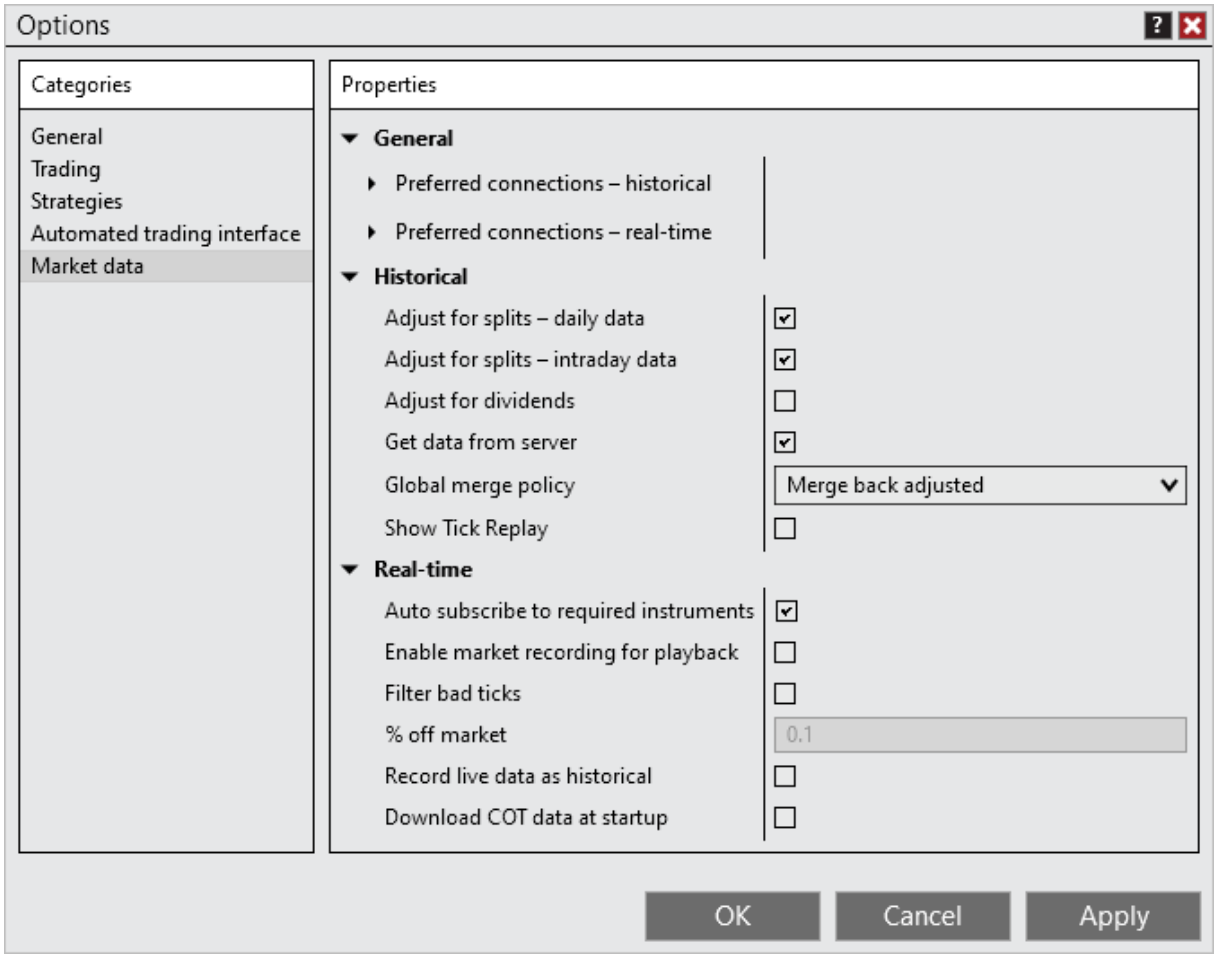
Enabled	Sets if the Tradestation email interface is enabled.
Order handling	
Submit	Sets how NinjaTrader will handle orders submitted from the email interface. **Market orders on TS fill:** NinjaTrader will submit market orders when NinjaTrader receives a strategy order filled email notification from TradeStation. **Submit as is:** NinjaTrader will submit the specified order type (market, limit and stop) when a strategy active order email notification is received from TradeStation. There are additional

	properties that become available when this mode is enabled that allow for some additional protections to attempt to prevent the Tradestation strategy from being out of sync with NinjaTrader, in contrast to the "Submit and forget" method below. **Submit and forget:** NinjaTrader will submit the specified order type (market, limit and stop) when a strategy active order email notification is received from TradeStation.
Delay conversion of unfilled amount to market after TS fill (secs)	Number of seconds NinjaTrader will delay converting any unfilled NinjaTrader orders after Tradestation reports it has filled the orders. This only applies to the order handling "Submit as is" mode.
Synchronization timeout (secs)	Number of seconds NinjaTrader will provide a pop up notification if order are out of synchronization (For example; TS reports a fill but NinjaTrader live order is not filled)
Stop Orders	
Submit	Sets how NinjaTrader will handle stop orders submitted from the email interface. This property only applies to the order handling mode "Submit as is". **Submit as is:** Submits the specified stop order when NinjaTrader receives a strategy active order email notification from TradeStation **Convert to stop limit:** Submits a stop-limit order when NinjaTrader receives a strategy active order email notification from TradeStation for any stop order type. The property "Limit price offset as ticks" will be made available where you set the amount of ticks the limit price is offset from the stop price.

	Submit as simulated stop: Submits a locally simulated stop-market order when NinjaTrader receives a strategy active order email notification from TradeStation for any stop order type. See more information on simulated stops here.
Submit market order if stop order was rejected	If a stop order is rejected for any reason, a market order will be sent. Please see the following section for disclaimer and risks of this feature.

9.3.5 Market data

The Market data section sets options related to market data and database management.



General	

Preferred connections - historical	Sets a connection to be used by NinjaTrader for historical data if it is connected. You can choose a separate preferred connection for each instrument type, expand the triangle to the left of the property name to set a preferred connection.
Preferred connections - real-time	Sets a connection to be used by NinjaTrader for real-time data if it is connected. You can choose a separate preferred connection for each instrument type, expand the triangle to the left of the property name to set a preferred connection.
Historical	
Adjust for splits - daily data	Enables or disables split adjusting historical data for daily data. Some providers already split adjust their daily data and you do not need to adjust it a second time if your provider handles it on their side. Please see this help guide page under the section "Understanding splits and dividends" for more information.
Adjust for splits - intraday data	Enables or disables split adjusting historical data for intraday data. Some providers already split adjust their intraday data and you do not need to adjust it a second time if your provider handles it on their side. Please see this help guide page under the section "Understanding splits and dividends" for more information.
Adjust for dividends	Enables or disables the adjustment of historical data to account dividends, for use with any function that requires historical market data
Get data from server	Enables or disables the retrieving of historical data from the data provider's server. When disabled, only local data stored on your PC will be used.
Global merge policy	Sets the merge policy for Futures contracts: **Do not merge:** historical data is not merged

	Merge back adjusted: NinjaTrader automatically merges and back adjusts historical data **Merge non back adjusted:** NinjaTrader automatically merges, but does not back adjust, historical data For more information on merge policies, see the "Understanding merge policies" section on this page.
Show Tick Replay	When enabled, allows "**Tick Replay**" to be configured from a data series menu. Please see Tick Replay for more information.
Real-time	
Auto subscribe to required instruments	Sets whether NinjaTrader will automatically subscribe to market data for any instruments requiring data throughout the platform to properly calculate PnL when trading non-USD pairs
Enable market recording of playback	Enables or disables market data recording for use with the Playback Connection
Filter bad ticks	Enables or disable filtering of bad ticks. This filtering only works on real-time data and will filter ticks that are a greater then a set percentage away from the last tick. Set the percentage for filtering with the property: "% off market". **Note**: If NinjaTrader receives 2 or more ticks that violate the tick filter we will no longer filter the ticks as the market is assumed to have legitimately gapped up or down.
% off market	Sets the real-time tick filter offset percentage (0.1 equals 1/10 of a percent)
Record live data as historical	Enables or disables the storage of incoming real-time Chart data to your local PC for future historical data requests. If you are connected to a provider that supports historical data, disable this feature.

Download COT data at startup	Allows downloading of Commitment of Traders data for use with the COT indicator

9.3.5.1 Splits and Dividends

Splits and Dividends

NinjaTrader will split and dividend-adjust historical data. This is primarily relevant for backtesting. NinjaTrader uses a fixed level back adjustment of dividends. This means that historical data is adjusted at the fixed amount of the dividend.

For example:

Day 1: Stock trades at $10.00
Day 2: Stock trades at $10.50
Day 3: Stock trades at $11.25
Day 4: Stock goes ex-dividend, the dividend is $0.75, and finishes trading at $10.50

The dividend adjusted historical data now becomes:

Day 1: $9.25
Day 2: $9.75
Day 3: $10.50
Day 4: $10.50 (the ex-day is not adjusted)

Enabling Splits and Dividends

You can enable this data adjusting by selecting the **Tools** menu from the **Control Center** and left mouse clicking on the **Options** menu item. Then select the **Market Data category** and select **Adjust for splits** and/or **Adjust for dividends**.

Warning: Should the historical data you are using come pre-adjusted you should not readjust them a second time.

Connectivity Provider	Split Adjusted		Dividend Adjusted	
	Intraday	Daily	Intraday	Daily
Kinetick www.kinetick.com	NO	YES	NO	NO
BarChart	YES	YES	NO	NO

eSignal	NO	YES	NO	NO
Interactive Brokers	----	----	----	----
IQFeed	NO	YES	NO	NO
TD Ameritrade	----	----	----	----

Adding Splits and Dividends

You must add splits and dividends per instrument in the **Instruments** window. Please see the Adding Splits and Dividends section of the Help Guide for more information.

- NinjaTrader stores historical data in it's local data repository in an unadjusted state
- If the data provider provides adjusted data, NinjaTrader will convert the data into it's unadjusted state prior to local storage

9.3.5.2 Merge Policy

Merge Policy

The **Merge Policy** option can be found in the Market Data category of the **Options** menu and sets how NinjaTrader handles the merging of historical data for futures contracts during a contract rollover. For example: If requesting a chart of the ES 06-15 from March 1st through April 1st, two contract months were the front month during that time span (03-15 and 06-15). The way the chart will display those contracts will depend on the following settings and are illustrated below.

> **Note:** More information on Configuring Rollover Dates and Offsets can be found in the Editing Instruments section of the Help Guide.

MergeBackAdjusted

- Data from each individual expiry month across the time span of the historical data request is loaded
- Offset values will be used to back adjust the historical price data to match the next front month

Selecting this option, the 03-15 data will be merged with the 06-15 data on the date of rollover (March 12th, 2015) and an Offset value will be used to connect the previous 03-15 contract price point with the first 06-15 contract point.

The result is a continuous chart of ES front month data for the dates selected. Price is seamlessly merged between each contract month.

MergeNonBackAdjusted

- Data from each individual expiry month across the time span of the historical data request is loaded
- Offset values are **NOT** used and leaves historical data as raw data

Selecting this option, the 03-15 data will be merged with the 06-15 data on the date of rollover (March 12th, 2015); however, **NO** Offset value will be applied.

The result is a continuous chart of ES front month data for the dates selected. Significant price gaps in the chart may be present due to changes in contract values that were **NOT** Offset.

DoNotMerge

- Data from **ONLY** the selected expiry month across the time span of the historical data requested is loaded
- Offset values are **NOT** used and leaves historical data as raw data.

Selecting this option will only show historical data for the front month selected. The 03-14 data will **NOT** be merged and **ONLY** data for the 06-15 contract will be used.

The result is a chart that goes as far back as there is data for the selected front month, which may be less than the requested date range.

9.3.5.3 Real-time Tick Filter

What is tick filtering?

Tick filtering is a function where each incoming tick is evaluated in relation to the last known price and if it is outside of a user defined percentage value, the tick is thrown away and not distributed to any NinjaTrader object that requires market data such as advanced charts or strategies. This prevents data spikes from showing on your charts and can also prevent unwanted actions taken by automated strategies due to a data spike.

How does it work?

A bad tick is detected if the tick price is less than the last valid traded price - (last traded Price * (1 - bad tick offset as %))
A bad tick detected if the tick price is greater than the last valid traded price + (last traded Price * (1 + bad tick offset as %))
If a bad tick is detected but the prior two ticks were also bad ticks, then the tick being processed is now a valid last traded price and is NOT filtered out

How do I enable tick filtering?

You can enable real-time tick filtering by selecting the **Tools** menu from the Control Center window and selecting the **Options** menu item. The Options dialog window will appear. Within the **Options** dialog window, left mouse click on the **Market data** category. Under the **Real-time data** section you can place a check mark next to **Filter bad ticks** and set the **% off market** value.

When should I used tick filtering?

- If you are using a market data vendor where you often see data spikes come in
- If you trade primarily equities
- If you are running automated strategies where data spikes have implications

9.3.5.4 Multiple Connections

▼ Using multiple connections

Multiple Connections

NinjaTrader supports multiple simultaneous connections to different connectivity providers, and in some cases, to the same connectivity provider allowing you to:

- Connect to and trade through multiple brokers simultaneously
- Connect to your broker and a separate data provider simultaneously

NinjaTrader will use a data feed for real-time or historical data, and by default will subscribe based on the type of instrument supported by the data feed connection and your connection order.

▼ Determining which data source is being used

Determining which data source is being used

When connecting to multiple connections, you must choose which provider will be supplying your real-time and historical data in NinjaTrader.

By default NinjaTrader will attempt to get real-time and historical data from the first connected data provider for the instrument type for which you are attempting to receive data.

The instrument types used for lookup are as follows, for determining which data feed supports which instrument types, please see the Data by Provider page.

- CFD's
- Futures
- Forex
- Indices
- Stocks

Example 1:
1. Connect to a NinjaTrader Continuum broker technology first
2. Connect to a Kinetick data feed second

NinjaTrader Continuum only supports futures, so all futures data would come from that connection, but if you tried to pull stock data, NinjaTrader would pull that data from Kinetick.

Example 2:
1. Connect to a Kinetick data feed first
2. Connect to a NinjaTrader Continuum broker technology second

Since Kinetick supports all instrument types, all data will be pulled from this connection. Any trades or orders submitted always go to the account you select, therefore if using the NinjaTrader Continuum account for order entry, all trades will go through NinjaTrader Continuum even if you are using Kinetick for data.

Connection order is important when determining which provider will be used for real-time and historical data. However, you can choose to set a preferred connection. See the "*Setting Preferred Data Connections*" section below for more details.

> **Note**: In Example 2 above, even if you did not have entitlement on your Kinetick account for certain futures, but you did on NinjaTrader Continuum, it will *not* fall over to NinjaTrader Continuum to pull data for those futures contracts since Kinetick's connection supports the futures instrument type. Data requests will only fall over to the secondary connection when the primary connection does not support the instrument class being requested.

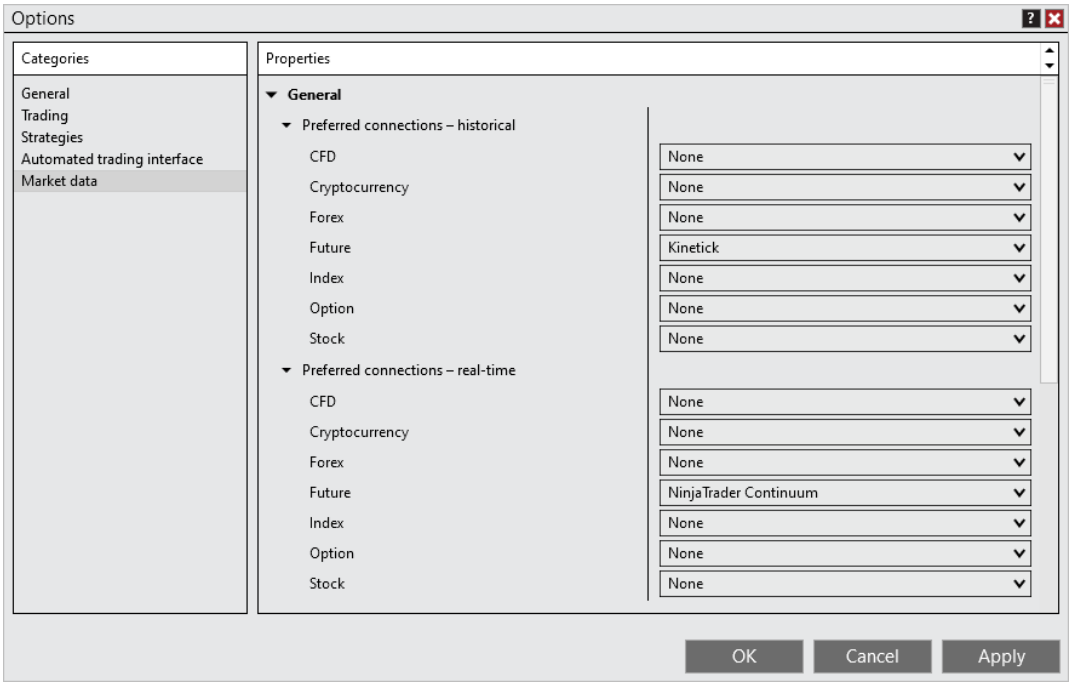
Setting preferred data connections

Preferred data feed connections

Within the NinjaTrader **Control Center** window, select the **Tools** menu, and then select **Options** menu item. In the **Options** window, select the "**Market Data**" category and expand the triangle to the left of **Preferred connections - historical** and **Preferred connections - real-time.** Here you may select a connection technology to use as the preferred connection per instrument type for real-time and historical data, independent of connection order. Setting your preferred connection for both historical and real-time has the advantage of being able to use a different data feed for your live connection and your historical connection.

Example:
1. Set NinjaTrader Continuum as the preferred real-time data feed provider for Futures.
2. Set Kinetick as the preferred historical data feed provider for Futures.
3. Connect to a NinjaTrader Continuum broker technology
4. Connect to a Kinetick data feed

In this example all futures real-time data would come from NinjaTrader Continuum and all futures historical data would come from Kinetick.



9.4 Performance Tips

Performance Factors

There are many variables that contribute to overall performance of the NinjaTrader application.

- Different connectivity providers (market data vendors and broker technologies) that NinjaTrader supports vary in their level of real-time data service. For example, providers who deliver unfiltered tick data (submit all market ticks) will impose heavier processing load than a vendor that provides throttled market data.
- The load you place on the NinjaTrader application (running 200 charts will consume more processing power than running only 20 charts)
- The capability of your PC hardware (are you running a brand new state of the art machine or a 4 year old PC with limited RAM)
- A single bad 'setting' can cause performance issues and a single active Third Party script can cause general performance issue. See the Understanding the impact of installing Add-Ons section.

Note: High memory usage may not be related to poor performance. It takes computer resources to figure out when to free up memory that is no longer being used. So, rather than using resources to constantly determine what memory to free up, the system will determine the best time to free up memory when it is needed.

Optimize Performance

The following are some suggestions that can help you fine tune your NinjaTrader installation to run optimally.

1. Close any unneeded open workspaces. Although a workspace may not be in view, it does still use system resources when open.
2. Exit any unneeded Charts, SuperDOMs, Market Analyzers, etc. in your open workspace(s). Additionally, if you have multiple tabs in any windows that you do not need, exit out of those.
3. Ensure that every indicator applied is using the Calculate setting of On price change or On bar close. There is no benefit to using On each tick unless the indicator deals with volume or tick counting.
4. Remove unneeded indicators from Charts, Market Analyzers, and SuperDOMs.
5. Reduce the Days to Load on Charts and on Market Analyzer indicator columns. This can be especially effective with any tick-based series.
6. Use Tick Replay sparingly and only when needed. For example, a simple Pivot indicator which just uses the current and previous daily price levels would not see any advantage from using tick replay. In contrast, a Volume profile indicator which relies on the exact sequence of trades to calculate various levels would greatly benefit from using tick replay.
7. Remove unneeded drawing objects. Especially over time as you have more drawing objects on your charts the performance can decline since there is more to render.
8. Workspaces which have a single or fewer number of instruments are unable to take advantage of multi-core processors. For example, having 30 charts with the same instrument will not perform as well as 30 charts of different instruments on a multi-core processor. Reducing the number of windows with the same instrument can increase

performance. Alternatively, workspaces that have one or only a few instruments would benefit from a processor with a higher clock speed.

9. Reduce the amount of charts with Global crosshairs enabled.
10. Set crosshair to Draw cursor only. Systems with less powerful GPUs can benefit from the reduced rending which results in this change. This setting is within the chart's properties.
11. Reset and clear history of your simulation account. A large amount of simulated trades can grow the size of your data base. If you do not need this historical information, completing this step will free up more resources.
12. Restart NinjaTrader daily.
13. Disable market recording for Playback and Record live data as historical if it's not needed.
14. Reduce the number of tickers expanded in the Options Chain.
15. The "Show Volume Text" property in the SuperDOM's Volume column can impact PC performance and the speed of rendering objects in the SuperDOM. This property is disabled by default to minimize the performance impact, and when disabled, you can hover your mouse cursor over any Volume row to view the exact volume at that row.
16. Some computer configurations provide two graphics cards (integrated vs. dedicated). To get the most out of chart rendering performance, enable a high-performance dedicated graphic processor.
17. Use a hardwired internet connection. Wireless and satellite connections can drop packets or have a poor latency. If a low latency connection is not available, a VPS can be a great solution.
18. Set your connection to connect on start up. This will make it so your workspace only needs to load once, rather than once when you start NinjaTrader and a second time when new data is download after you connect.
19. Reduce the number of applications running on your system.
20. Windows search indexing can also place an extra burden on your processor when running NinjaTrader, negatively impacting performance. It is recommended to exclude the folders listed below from indexing, as well, which can be done via the Windows Control Panel.

    C:\Users\User\Documents\NinjaTrader 8

    C:\Program Files (x86)\NinjaTrader 8

Playback

1. Remove unused instrument subscriptions in playback. In your playback setup, be mindful for which instruments you have added (for example in a Market Analyzer or via Charts) you would have data to playback actually available, as each instrument subscription here would consume CPU cycles and thus contribute to performance experienced. For example, having the SP500 index added in your Market Analyzer but then only replaying MSFT data is expected to have lower performance in contrast to having only this one MSFT instrument listed in the Market Analyzer as well.

Additional Diagnostics

If you're still having performance issues, follow these diagnostics steps to isolate the problem.

1. Start NinjaTrader in Safe Mode.

    Safe Mode will prevent NinjaTrader from:

Loading workspaces.
Connecting on start-up.
Loading custom assemblies.
Getting instrument updates from the server.

To enable safe mode, please use the following steps:
Exit NinjaTrader.
Hold the CONTROL key and double click the NinjaTrader icon.
Keep the CONTROL key held down until you see the Control Center.
You can verify you are in safe mode by going to Help -> About.

2. Open some windows and test performance without loading any custom indicators.
3. If everything looks good, attempt to open up your workspace. You may need to close your workspace, without saving it, then reopening it. If this works well, installed custom assemblies may be the cause.
4. If performance is poor with the workspace in safe mode it could be that the workspace is too resource intense for your system, which may be from the scripts being used.
5. To determine if there are specific scripts that are resulting in poor performance you will need to recreate your workspace and add back any scripts one at a time to see which one may be resulting in poor performance.
6. For further assistance, or assistance with any of these steps, please write into platformsupport@ninjatrader.com

10 Operations

Operations Overview

10.1 Advanced Trade Management (ATM)

ATM Overview

ATM Strategies can be accessed from the ATM Strategy Selectors located in various Order Entry interfaces

NinjaTrader provides you with the flexibility to trade with or without an Advanced Trade Management (ATM) Strategy. ATM Strategies are designed to provide discretionary traders with semi-automated features to manage their positions. This is NOT to be confused with NinjaScript Strategies for automated trading systems.

ATM Strategy

- Definition and Benefits
- ATM Strategy Parameters
- ATM Strategy Selection Mode
- Stop Strategy
- Auto Breakeven
- Auto Trail
- ATM Strategy Templates
- Example #1
- Example #2

Advanced Options

- Auto Chase
- Auto Reverse
- Shadow Strategy

Misc

- Close at Time
- Indicator Tracking
- FAQ

What is an ATM Strategy?

Before you enter a trade you already know where you are going to place your Profit Target(s), where you will set your Stop Loss, and how many contracts you will trade. You may also have rules and conditions for managing your trade such as; once there is 1 point in profit you will move your Stop Loss to breakeven and once there is 2 points in profit you will move your Stop Loss to protect 1 point in profit. These rules and conditions make up your personal trade methodology, or as we call it, your strategy. In NinjaTrader, an ATM Strategy is a collection of orders that represent your entries, exits, stops and targets along with sub-strategies (Auto Breakeven, Auto Chase, Auto Trail etc...) that govern how these orders are managed. By pre-defining your personal trading strategy in NinjaTrader, you are free to concentrate on the trade and not on the management of orders and positions. NinjaTrader does this all for you automatically.

Do I have to use an ATM Strategy?

Absolutely not. NinjaTrader is incredibly flexible in that you can trade independent of an ATM Strategy and manually submit and manage all of your own orders. You can also choose to manage a portion of an open position by an ATM Strategy and leave another portion to be managed independently. It's completely up to you.

What are the advantages to using an ATM Strategy?

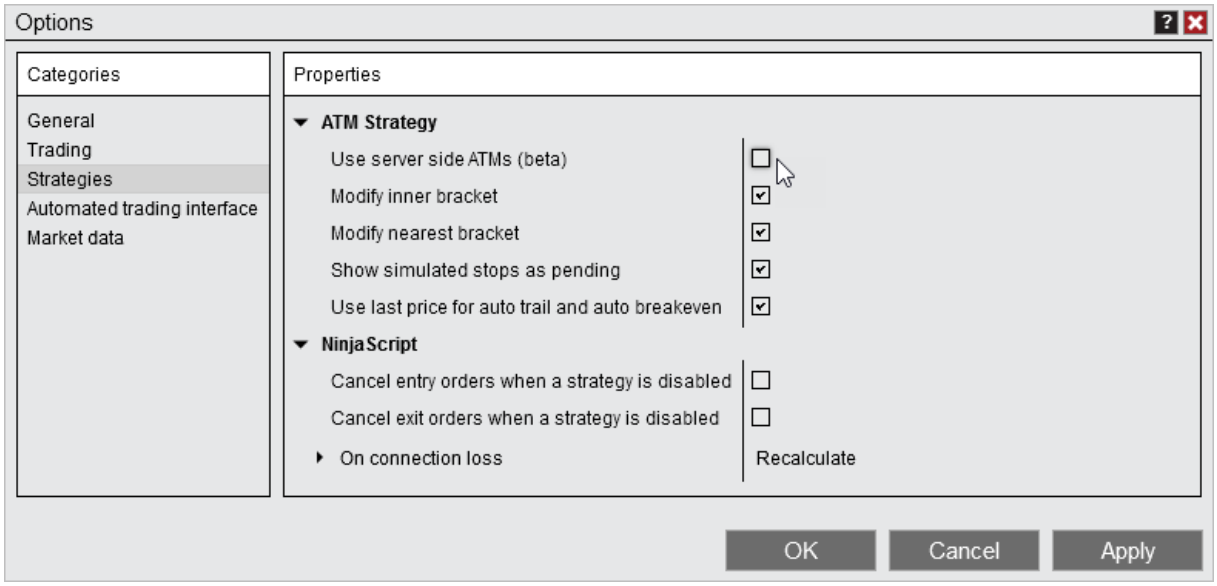
There are several:

- Reduce errors in order management
- Speed (orders are submitted and modified at PC speed instead of human speed)
- Discipline (less prone to applying 'too much' discretion)

- Consistency with your trading
- Reduces emotions

10.1.1 Server Side vs Local ATMs

Server side ATMs are the newly implemented feature, exclusive to the NinjaTrader connection. They simplify the local ATMs, while still including many of the primary features. Server side ATMs are currently in beta and will continue to be developed on to add additional functionality. However, there are some differences between server side ATMs and local ATMs that you must be aware of to ensure your orders function as expected.
Server side ATMs can be enabled/disabled under Tools> Options> Strategies> ATM Strategies



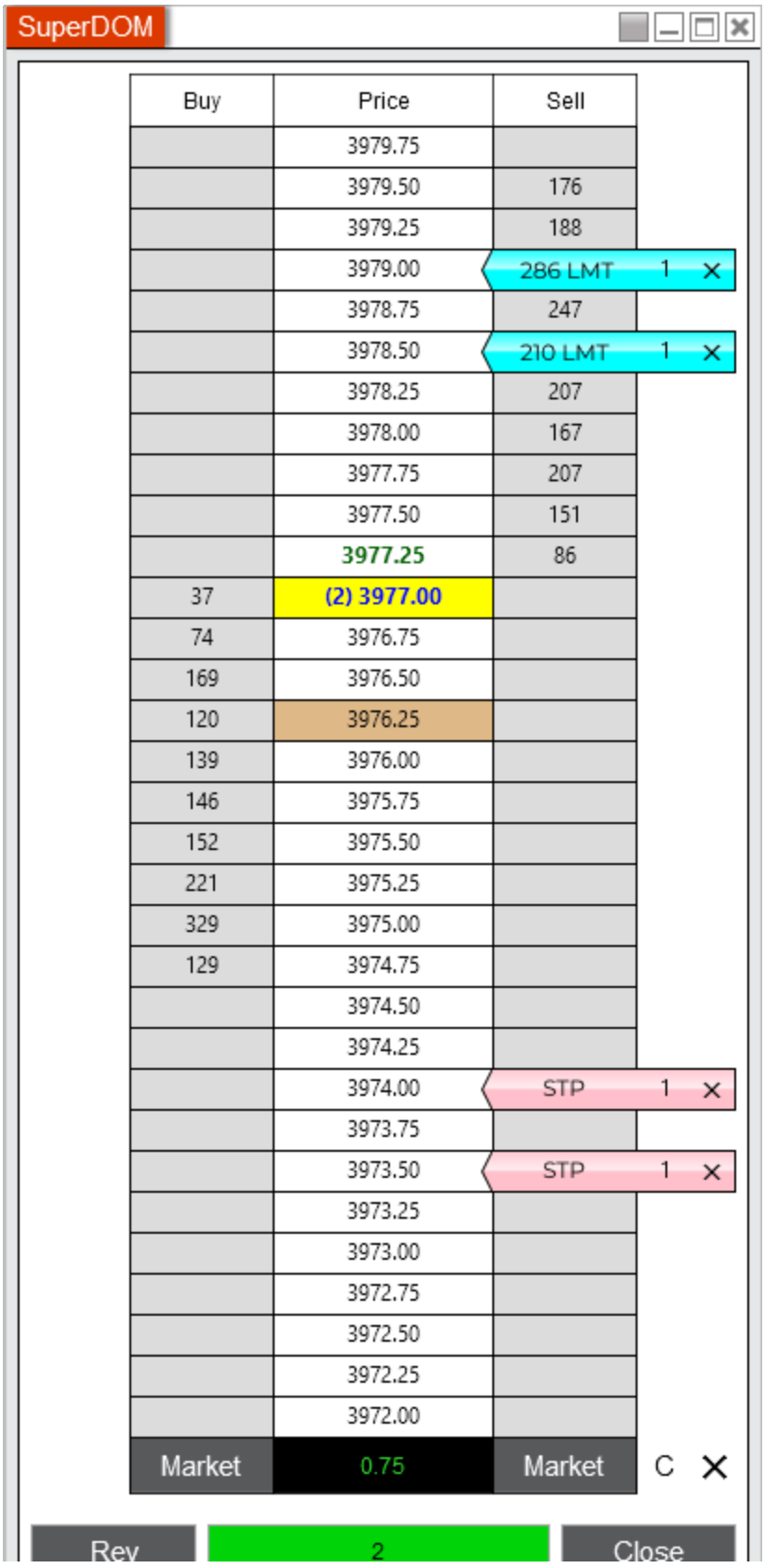
Warnings:
It is recommended to exit all ATM orders before switching between server side and local ATMs.

If Multi-provider mode is enabled, server side ATMs will be disabled

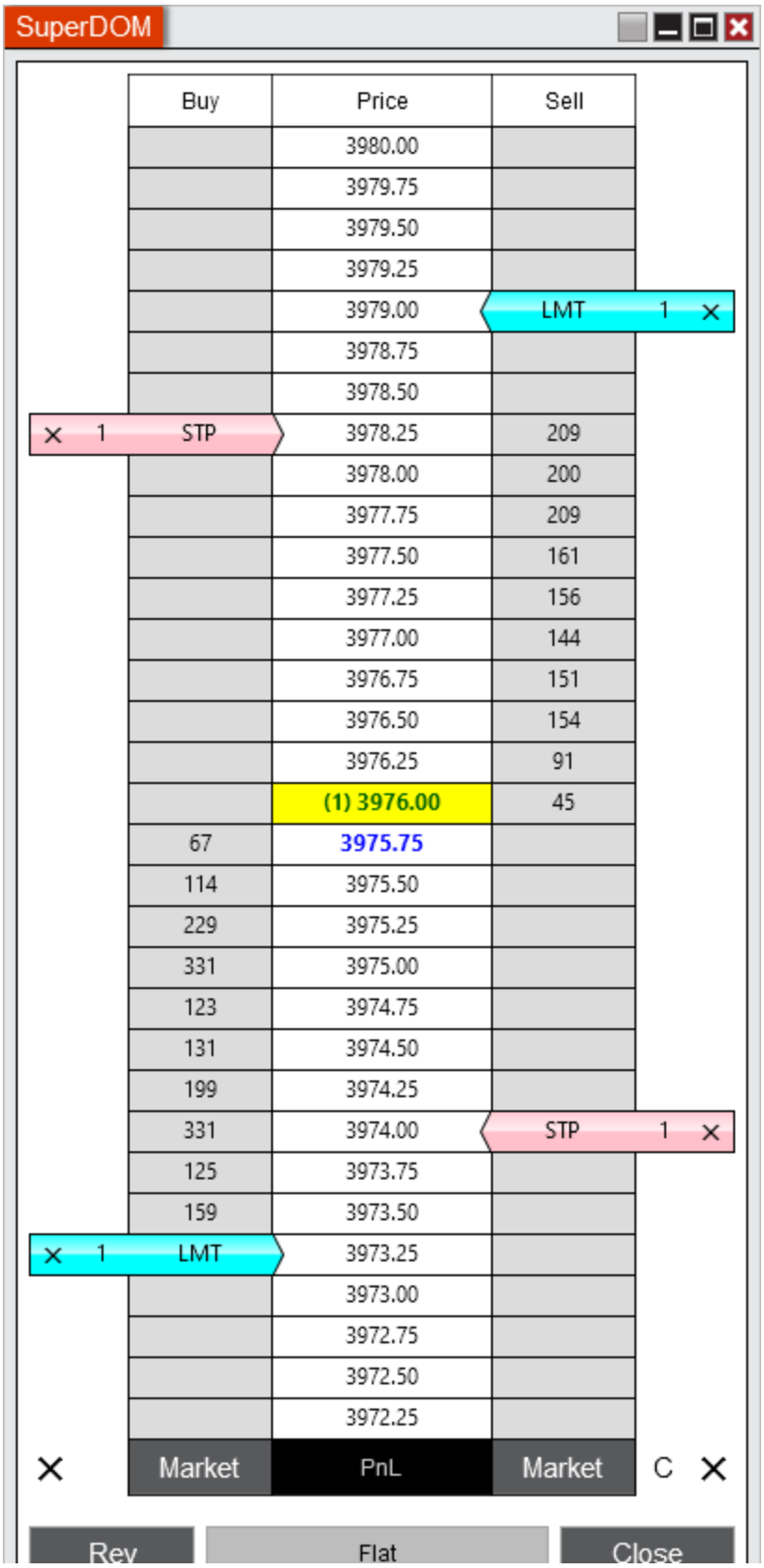
▼ Scaling in/out of an ATM position

Server Side ATM
Server side ATMs function independently of one and another. There is no active ATM strategy to select to scale in/out of. Every entry you make will place it's stops and profit in relation to your entry order.

Example: If you entered long 1 contract on the ES at 3976.50 with a stop loss and profit of 10, your stop loss would be placed at 3974.00 and your profit would be placed at 3979.00. If you then scaled in 1 contract with your ATM at 3976.00 a new stop loss would be placed at 3973.50 and a new profit would be placed at 3979.00



The same would be true if you were to enter long 1 contract with your ATM and then enter short 1 contract with your ATM. In this scenario you would be flat, but your stop losses and profits would still be running. You would want to click Close to exit your position and cancel the the stop loss/profit.
Example: If you entered long 1 contract on the ES at 3976.50 with a stop loss and profit of 10, your stop loss would be placed at 3974.00 and your profit would be placed at 3979.00. If you then entered short 1 contract with your ATM a new stop loss would be entered at 3978.25 and a new profit would be placed at 3973.25, but you would be flat.

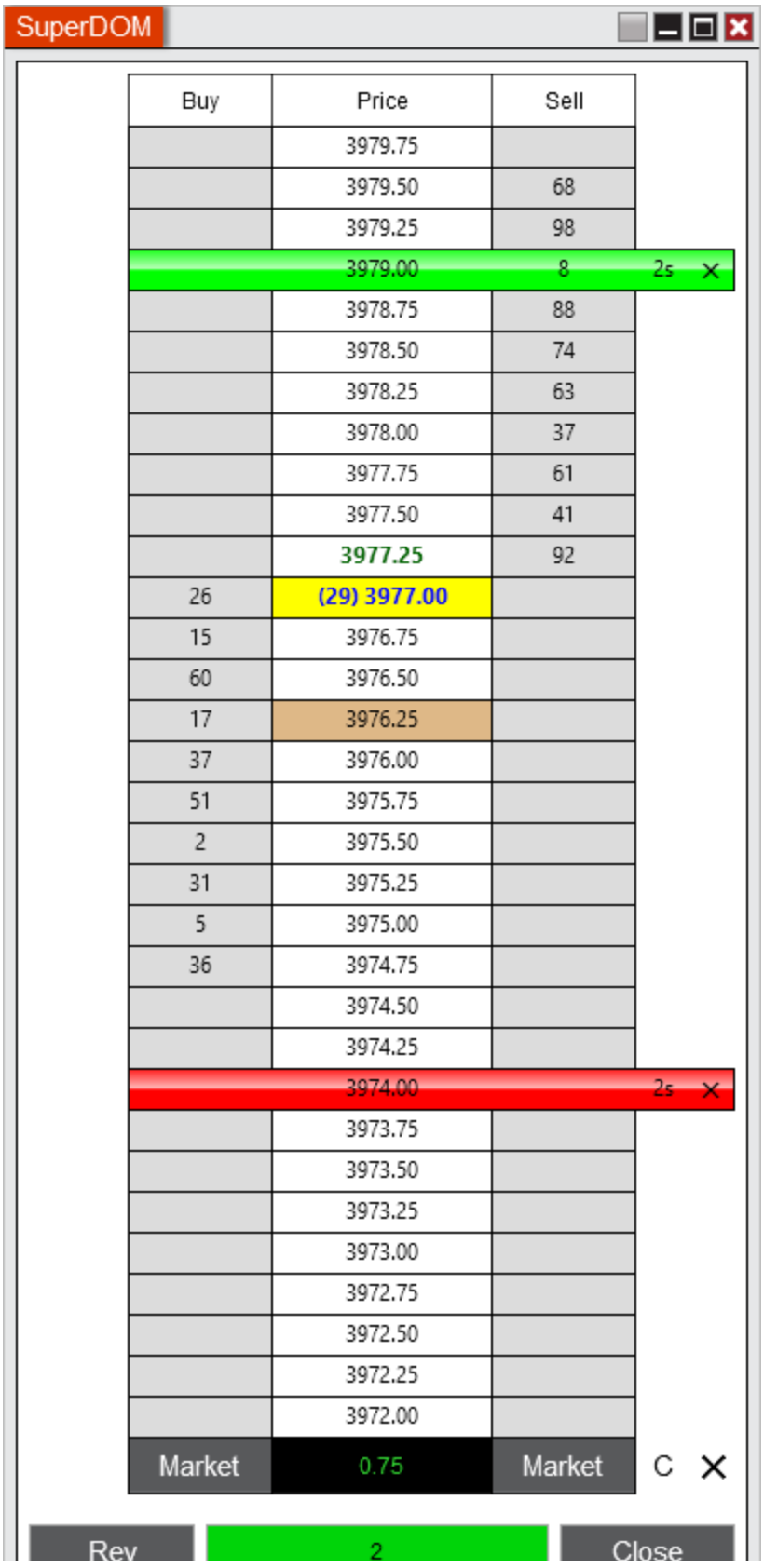


Local ATM

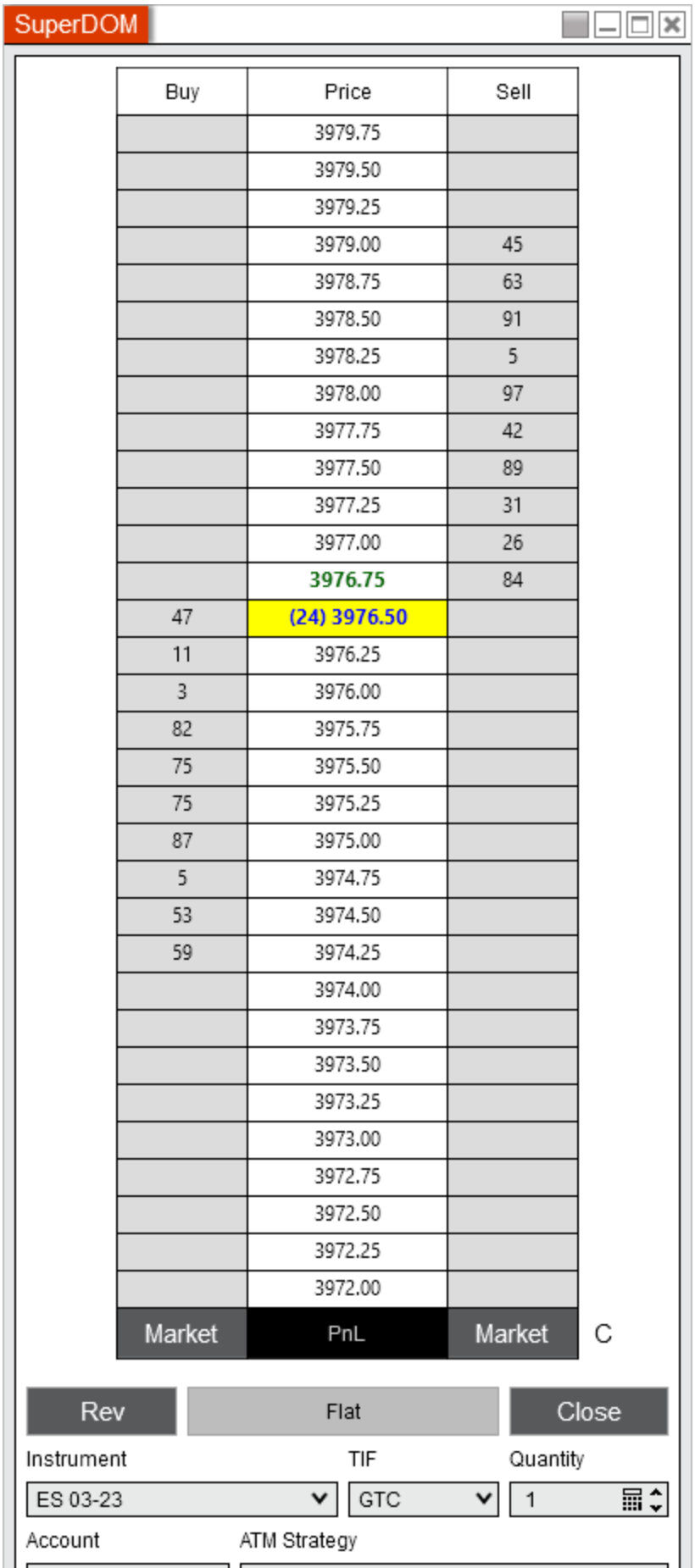
With local ATMs there is an active ATM which you can choose to scale in/out of. There is a property called ATM Strategy Selection Mod, which defaults if the active ATM should be selected or not. The Keep selected ATM strategy template on order submission selection functions similar to server side ATMs. However, the default that many people are familiar with works differently. The default is Select active ATM strategy on order submission. There is also a Display selected TM strategy only mode. For more information on all these modes, see the ATM Strategy Selection Mode section of the help guide. However, for this example we will focus on the default setting.

By default, after entering into a position with an ATM your active ATM will be selected. As you scale in/out, that will then add/subtract to the nearest stop loss/profit of your ATM.

Example: If you entered long 1 contract on the ES at 3976.50 with a stop loss and profit of 10, your stop loss would be placed at 3974.00 and your profit would be placed at 3979.00. If you then scaled in 1 contract with your ATM at 3976.00 your existing stop loss and target would get stacked with an additional order at the same price.



The same would be true if you were to enter long 1 contract with your ATM and then enter short 1 contract with your ATM. In this scenario you would be flat, but the scaling out of the ATM would also remove the stop loss and target.

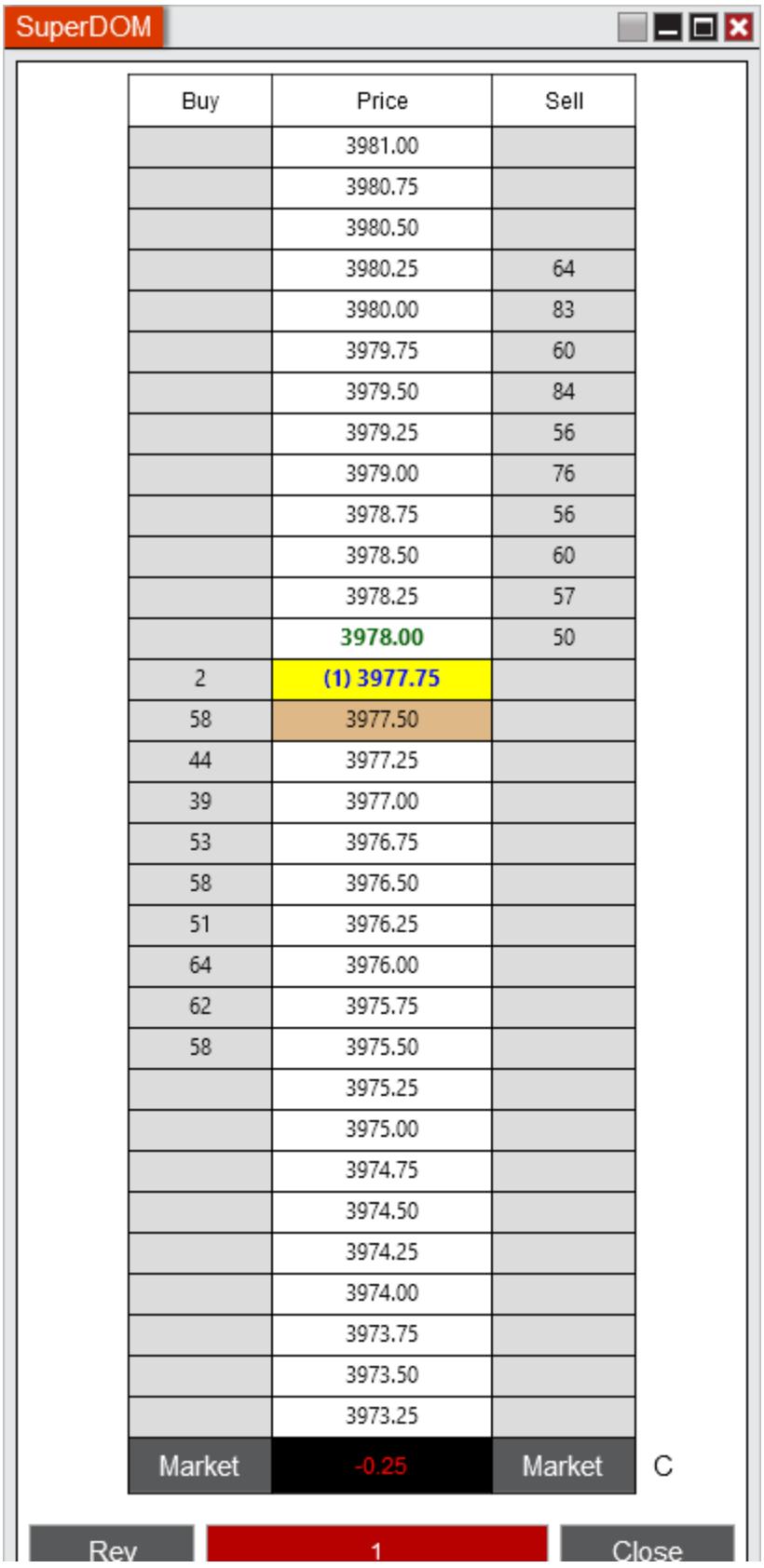

SuperDOM
Buy
Price
Sell
3979.75
3979.50
3979.25
3979.00
45
3978.75
63
3978.50
91
3978.25
5
3978.00
97
3977.75
42
3977.50
89
3977.25
31
3977.00
26
3976.75
84
47
(24) 3976.50
11
3976.25
3
3976.00
82
3975.75
75
3975.50
75
3975.25
87
3975.00
5
3974.75
53
3974.50
59
3974.25
3974.00
3973.75
3973.50
3973.25
3973.00
3972.75
3972.50
3972.25
3972.00
Market
PnL
Market
C
Rev
Flat
Close
Instrument
TIF
Quantity
ES 03-23
GTC
1
Account
ATM Strategy

Selecting Rev On an ATM Position

Server Side ATM

With server server side ATMs clicking **Rev** while in an ATM will exit your position, cancel your orders, and enter your into a position on the other side of the market without an ATM strategy.

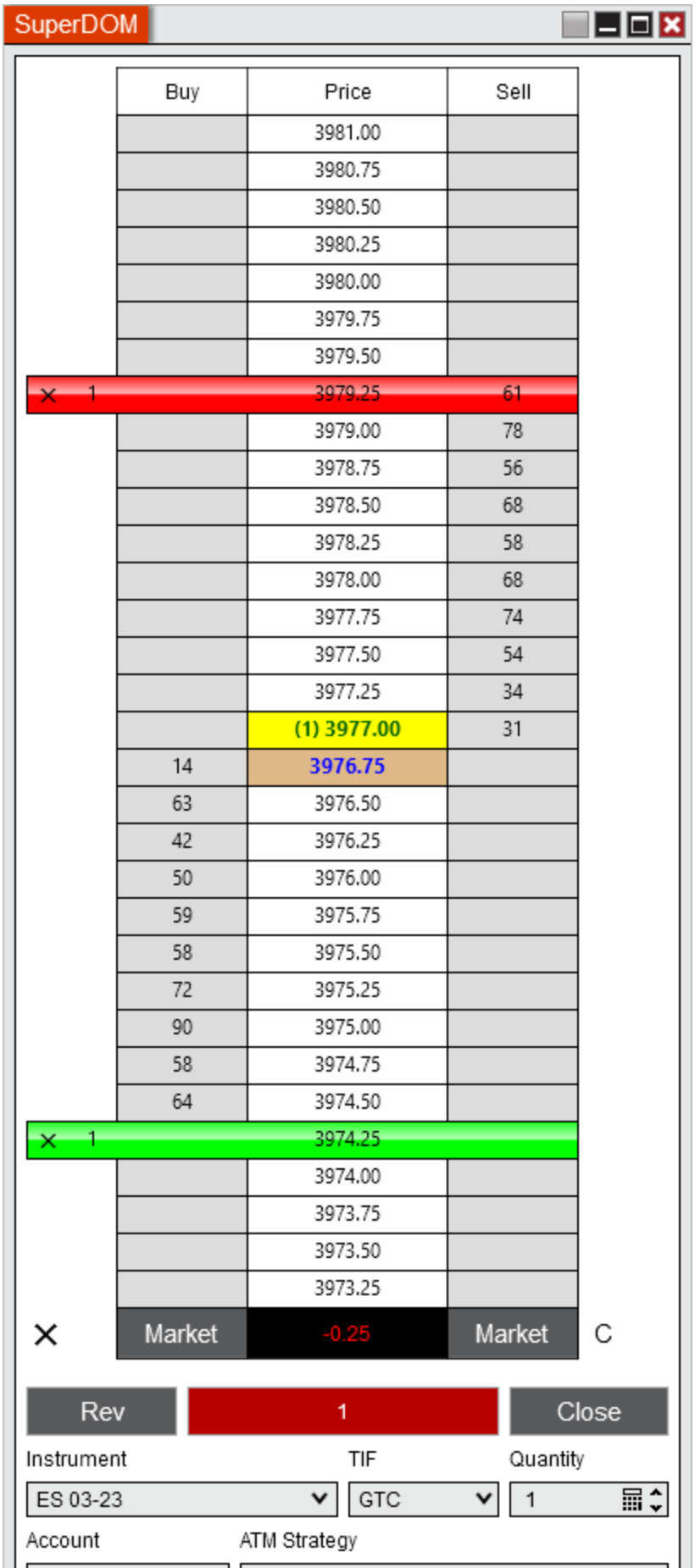
Example: Enter long 1 contract with an ATM, press **Rev**. Now you are short 1 contract without a protected stop loss and profit.



Local ATM

With local ATMs clicking **Rev** while in an ATM will exit your position, cancel your orders, and enter your into a position on the other side of the market with an ATM strategy.

Example: Enter long 1 contract with an ATM, press **Rev**. Now you are short 1 contract with a protected stop loss and profit.


SuperDOM
Buy
Price
Sell
3981.00
3980.75
3980.50
3980.25
3980.00
3979.75
3979.50
1
3979.25
61
3979.00
78
3978.75
56
3978.50
68
3978.25
58
3978.00
68
3977.75
74
3977.50
54
3977.25
34
(1) 3977.00
31
14
3976.75
63
3976.50
42
3976.25
50
3976.00
59
3975.75
58
3975.50
72
3975.25
90
3975.00
58
3974.75
64
3974.50
1
3974.25
3974.00
3973.75
3973.50
3973.25
Market
-0.25
Market
C
Rev
1
Close
Instrument
TIF
Quantity
ES 03-23
GTC
1
Account
ATM Strategy

Manually canceling a stop/target

Server Side ATMs

When canceling a stop loss or profit of a server side ATM the other order will stay active until it is also manually canceled or filled. However, if a stop loss or profit is filled, the other order will be canceled.

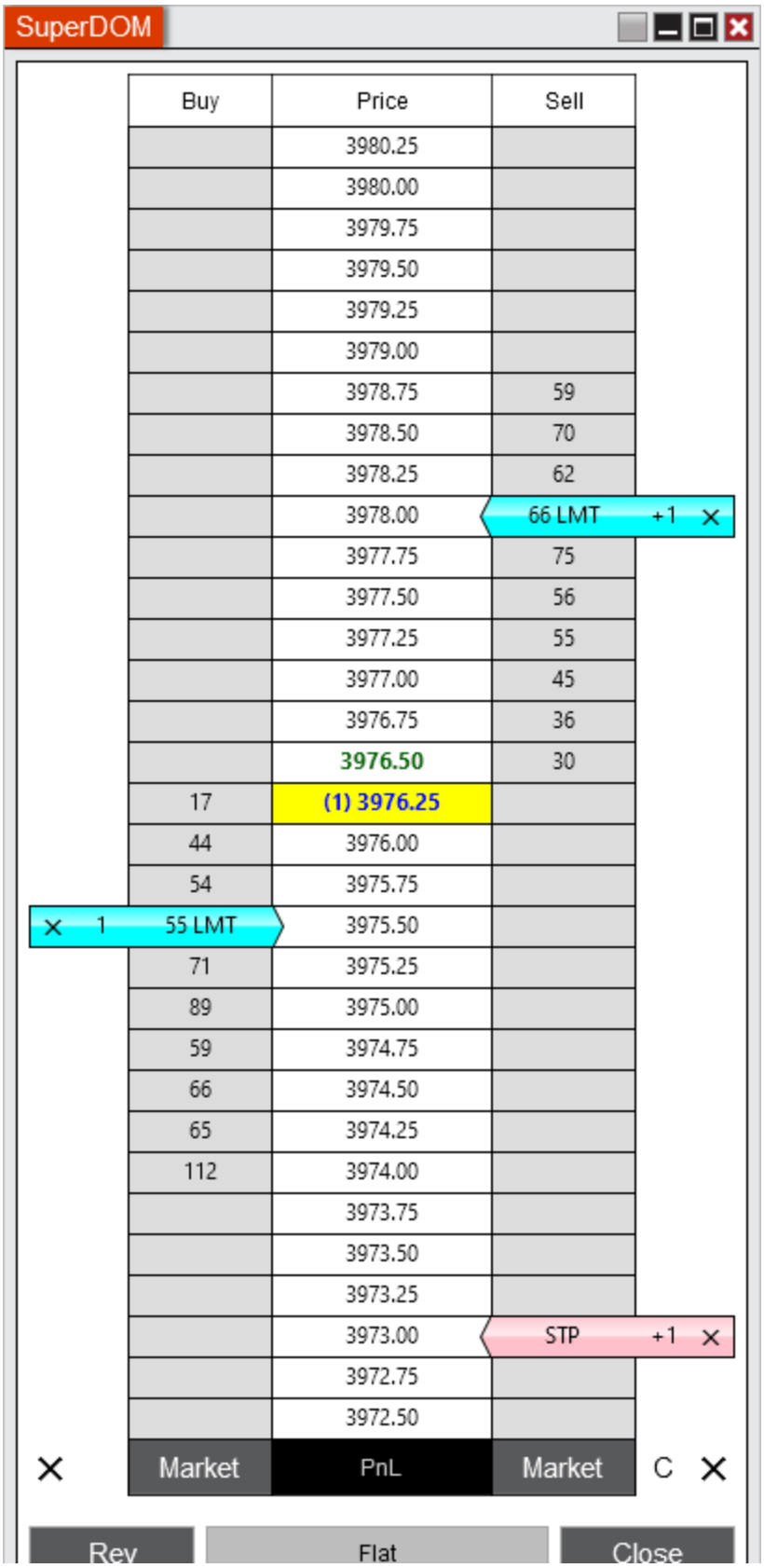
Local ATMs

When canceling a stop or target of a local ATM the bracket of the order will also be canceled. Additionally, if a stop loss or profit is filled, the other order will be canceled.

Visibility and modification of stop losses/profits of ATM entry orders

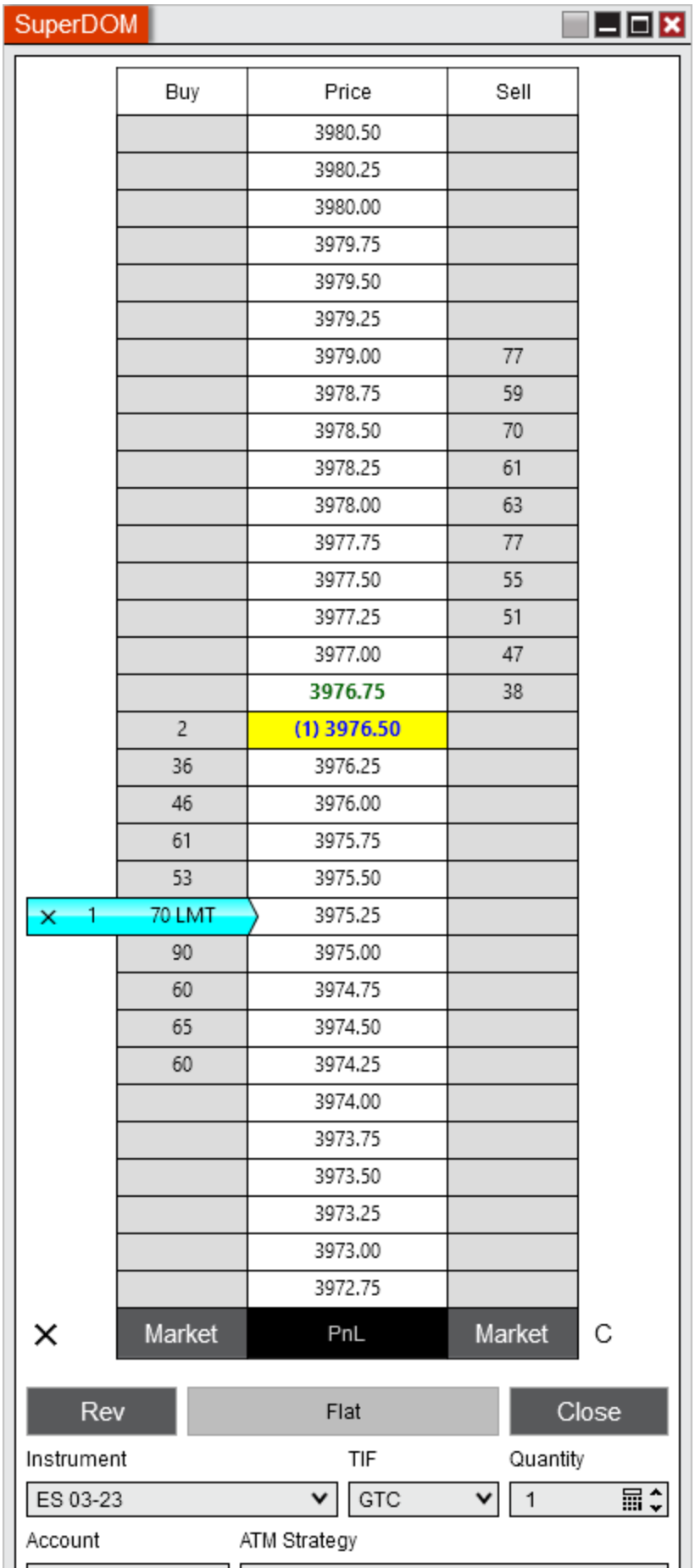
Server Side ATMs

When placing an entry order with a server side ATM, the stop loss and target will show also. They will be in a suspended state, indicating they will not be working until they are triggered by the entry order filling. You can identify them as suspended order by the + next to the order quantity. The prices of these orders can be modified before the entry is filled.



Local ATMs

When placing an entry order with a local ATM, the stop loss and profit will not show. They cannot be modified before the entry is filled.

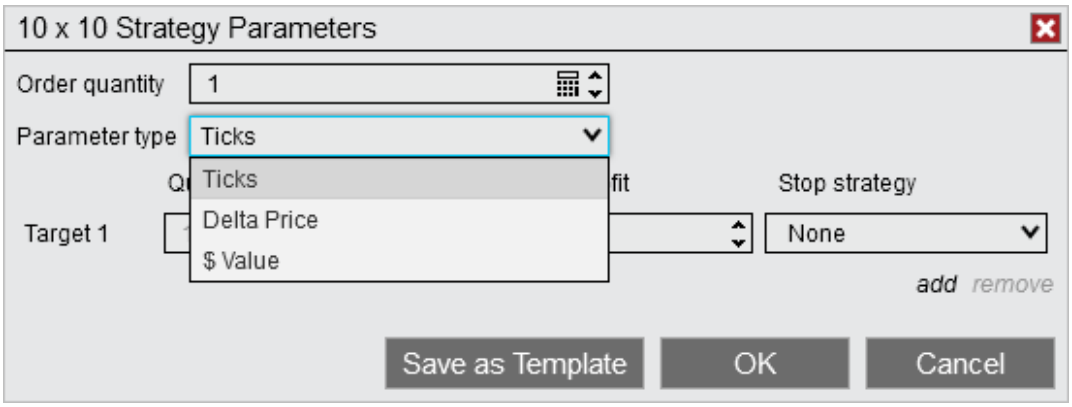

SuperDOM
Buy
Price
Sell
3980.50
3980.25
3980.00
3979.75
3979.50
3979.25
3979.00 77
3978.75 59
3978.50 70
3978.25 61
3978.00 63
3977.75 77
3977.50 55
3977.25 51
3977.00 47
3976.75 38
2 (1) 3976.50
36 3976.25
46 3976.00
61 3975.75
53 3975.50
1 70 LMT 3975.25
90 3975.00
60 3974.75
65 3974.50
60 3974.25
3974.00
3973.75
3973.50
3973.25
3973.00
3972.75
Market
PnL
Market
C
Rev
Flat
Close
Instrument
TIF
Quantity
ES 03-23
GTC
1
Account
ATM Strategy

Parameter Type functionality

Server Side ATMs

There are 3 Parameter type selection for server side ATMs. They will all function the same, but just display the values in the specified format.

Ticks	Ticks away from the average entry.
Delta Price	Price away from the average entry, based on the displayed price of the instrument. AKA Points away from the average entry.
$ Value	Cash value away from the average entry, based on to Tick value of the instrument.

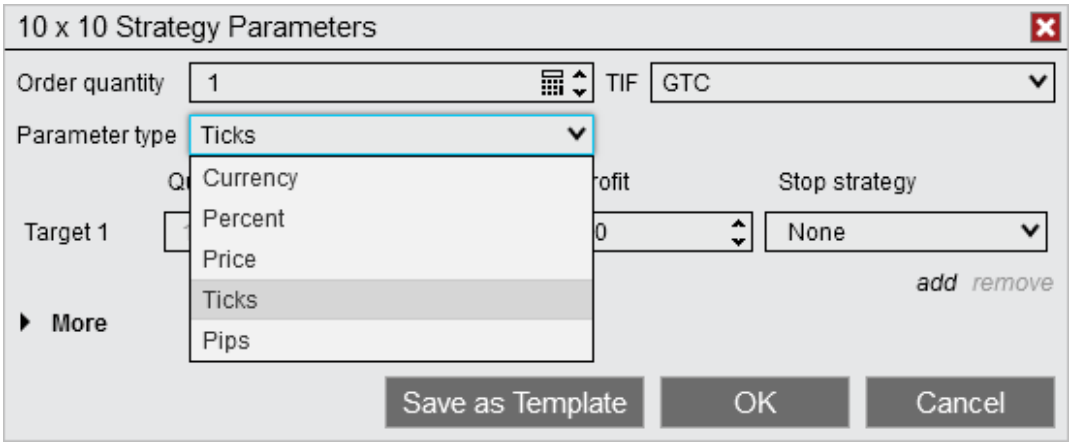


Local ATMs

There are 5 Parameter type selections for local ATMs. Each functions differently than the other.

Currency	PnL away from average entry. Calculated by the dollar per tick value for the order quantity used.

Percent	Percentage away from the average entry, based on the average entry price.
Pips	Pips away from average entry.
Price	The absolute price point specified.
Ticks	Ticks away from entry average entry.



Miscellaneous differences

Server side ATMs	**Local ATMs**
Can't be used with local orders (simulated and MIT orders)	Can be used with local orders (simulated and MIT orders)
Templates are saved to the server, for use on any system you connect to	Templates are saved to your local computer and are only available where saved

Templates are specific to the instrument	Templates can be used on any instrument
Order quantity and TIF are not part of the template	Order quantity and TIF are part of the template
Can be set up to have only a stop loss or only a profit	Can be set up to have just a stop loss. Cannot be set up to have just a profit.
Can't place OCO order with ATMs attached	Can place OCO orders with ATMs attached
Can't be used with NinjaScript	Can be used with NinjaScript
Can't add ATMs to an unprotected position	Can add ATMs to an unprotected position
No section called More with additional features	Includes a section called More with additional features
No simulated stop volume trigger feature	Includes simulated stop volume trigger feature
One step Auto Trail	Three step Auto Trail
Can't enable/disabled stop strategy on a working ATM	Can enable/disable stop strategy on a working ATM
No configurable settings under Options> Strategies> ATM Strategies	Configurable settings under Options> Strategies> ATM Strategies
Can't be used with Playback	Can be used with Playback

10.1.2 ATM Strategy

What is an ATM Strategy?

An ATM Strategy provides a semi-automated order management features to allow you to automate the management of a position. In trading, a position is defined as the total contracts/shares held long or short for a specific instrument in a specific account. An ATM Strategy can be thought of as:

"A collection of user defined rules/conditions that create and manage a set of Stop Loss and Profit Target orders that are used to govern a portion or an entire open position."

Let's assume the following:

- We want to go long the S&P E-Mini for 5 contracts
- We want a Stop Loss set 2.5 points from our entry price
- We want a Profit Target set 5 points from our entry price

We just defined a set of conditions for the management of a 5 contract long position, or in other words, we just defined an ATM Strategy. The ATM Strategy is the foundation for how positions (or partial positions) can be managed within NinjaTrader. ATM Strategies can be defined on the fly or you can pre-define them using templates that can be recalled for later use in a split second.

The Value of an ATM Strategy

Now that we understand what an ATM Strategy is, what exactly is the value of it? When trading, one develops ideas and methods for entry and further management of their position. The management of this position can be simple to complex and everything in between. The ATM Strategy allows the trader to define the rules and conditions that govern the management of the position. How many Profit Targets should there be and at what prices? What Auto Trail Stop Loss setting should be used? When should a Stop Loss be moved to breakeven? Should Profit Target orders chase the market if not filled? Should the Stop Loss order trigger immediately on trade through or should NinjaTrader's leading edge Simulated Stop order be used? An ATM Strategy also provides a layer of discretionary automation and intelligence that takes responsibility for mundane order modifications which can be inefficient, time-consuming and error prone. When scaling into a position, for example, all of the Stop Loss and Profit Target orders will be automatically updated to reflect the new position size. Changing order contract sizes will update the distribution of contracts on other orders. Decrease your first Profit Target order by one contract and your second Profit Target will automatically be increased by 1 contract. The bottom line is that an ATM strategy thinks the way a trader thinks about managing their trade only 100x faster. It performs a lot of the routine tasks for you allowing you to concentrate on what matters; the trade itself.

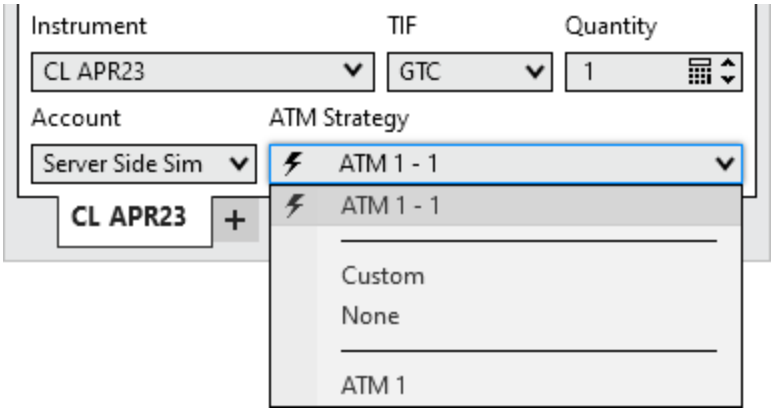
10.1.2.1 ATM Strategy Parameters

Majority of NinjaTrader's order entry interfaces house the same control for defining an ATM Strategy.

Understanding the ATM Strategy control list options

The Strategy Control List

The drop down list shown in the image below is very important to understand as it defines how your orders will be handled once submitted. There are three main categories of options that will be displayed in this drop-down list; **None**, **Custom** or strategy template names, and **Active ATM Strategy Name(s)**.



None

When this option is selected, any orders placed in the entry window will not be applied to an active ATM Strategy nor will it initiate a new ATM strategy.

Custom or ATM Strategy Template Names

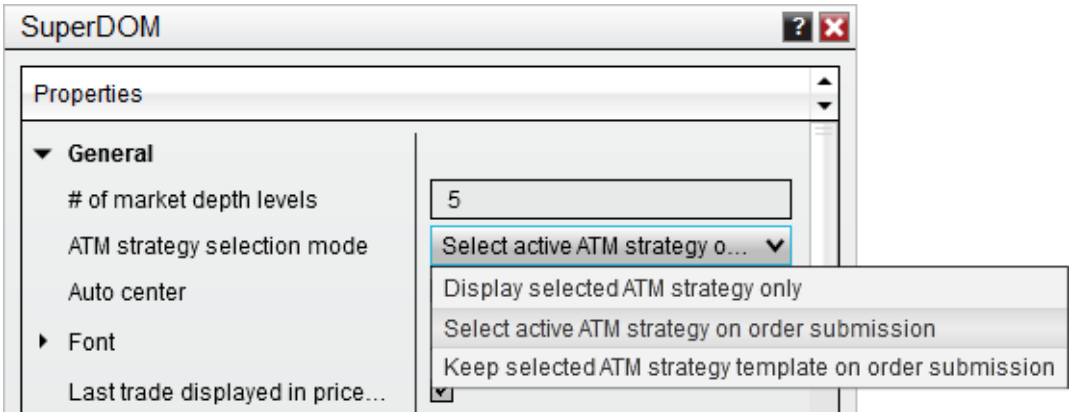
When an ATM Strategy template name is selected, all of the parameters will update to reflect your pre-defined ATM Strategy, or when **Custom** is selected, you have the ability to define a new ATM Strategy on the fly. Once an order is submitted, the ATM Strategy parameters specified will be initiated when the order is partially or completely filled.

Active ATM Strategy Names

All active (live and working) ATM Strategies will be displayed and indicated by a lightning bolt icon. If one is selected, any order submitted will be applied to the selected active ATM Strategy. For example, if you have an active ATM Strategy with a stop and target bracket for 1 contract, if you are filled on another contract, the fill is applied to this ATM Strategy and the stop and target bracket is automatically updated from 1 contract to 2 contracts.

Strategy Selection Mode Overview

The behavior of the strategy control list can be controlled automatically by selecting an ATM Strategy Selection Mode.



When it comes to the automatic submission of Stop Loss and Profit Targets and how subsequent order fills are handled, there are two approaches:

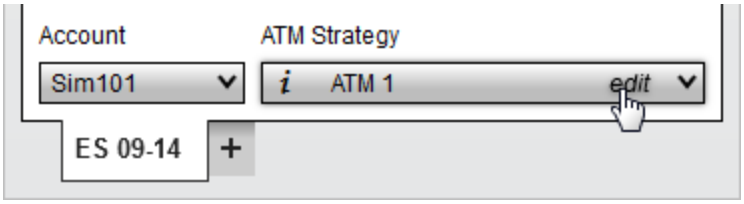
1. Scaling into a position or out of a position should automatically update the order sizes of existing stop and target brackets
2. Scaling into a position should create a new set of stop and target brackets based on the new order fill price

If you always want to operate with approach number 1, then you will always want to have the ATM Strategy control list set to your active ATM Strategy when one exists. This is accomplished by setting the ATM Strategy Selection Mode to "Select active ATM strategy on order submission". If you would rather have new stop and target brackets submitted on a new fill, then set the ATM Strategy Selection Mode to "Keep selected ATM strategy template on order submission" and the strategy control list will not automatically set to an active strategy when one is created.

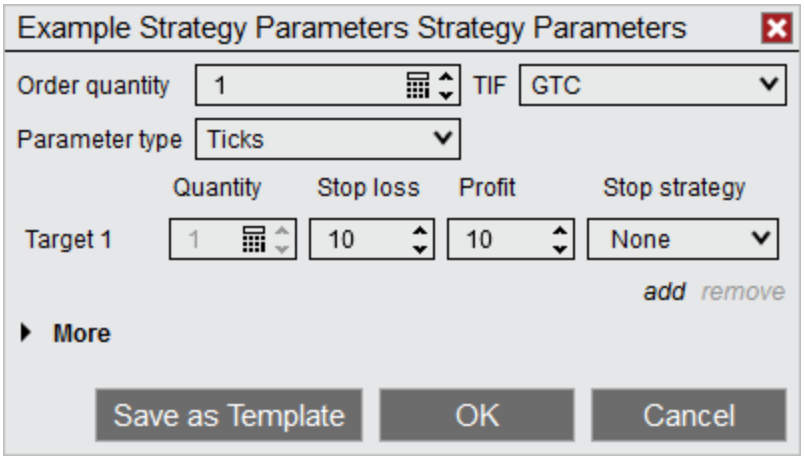
Understanding Stop Loss and Profit Target parameters (how to set your stop and target value

ATM Strategy Parameters

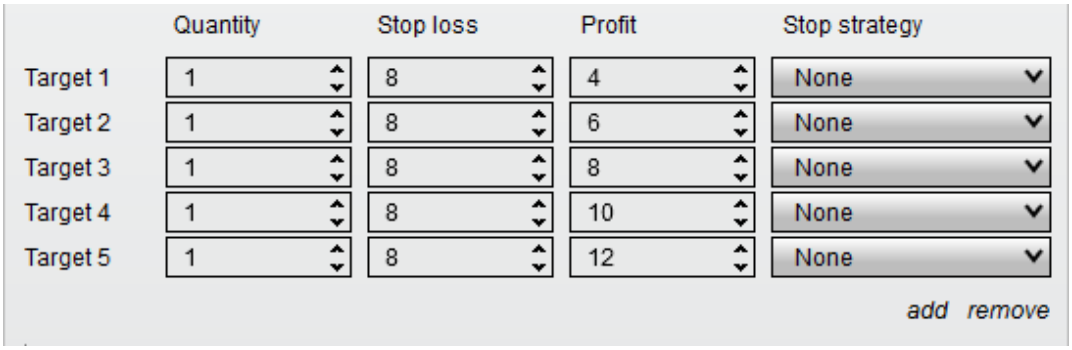
Select **Custom** to define a new ATM Strategy or select the saved **ATM Strategy** template and select "edit" in the **ATM Strategy** combo box as seen below.



In the image below there are parameters that define the **ATM Strategy**. This strategy is a single quantity strategy that will automatically place its target at 10 ticks above the average entry price and stop loss 10 ticks below.



Selecting the "***add***" will allow you to configure additional **Targets** for your **ATM Strategy**. You can add as many targets as you desire. Selecting "***remove***" will reduce the number of configured **Targets** that are configured.



Tip: You can also **add Targets** to your **ATM Strategy** by right-clicking on an active strategy in the **ATM Strategy** combo box and selecting "**Add Target**". More details are documented in the SuperDOM section Managing Positions.

Order quantity	Replicated from the order entry display and sets the initial quantity used for the entry order.
TIF (Time In Force)	Replicated from the order entry display and sets the TIF used for entry, profit target, and stop loss orders.
Parameter type	Sets the type of parameter used for defining where the stop loss and profit target will be placed. **Currency** – PnL away from average entry. Calculated by the dollar per tick value for the order quantity used. **Percent** – Percentage away from the average entry, based on the average entry price. **Pips** – Pips away from average entry. **Price** – The absolute price point specified. **Ticks** – Ticks away from entry average entry.
Quantity	Sets the quantity for the Stop Loss and Profit Target orders for this target
Stop Loss	Sets the value that determines the Stop Loss price. If the value is set to 4 (ticks) and your average entry for the initiating order is 1000 and you are long, your Stop Loss would be submitted at AvgEntry - Stop Loss = 1000 - 4 ticks = stop price of 999. This assumes each tick is valued at 0.25.

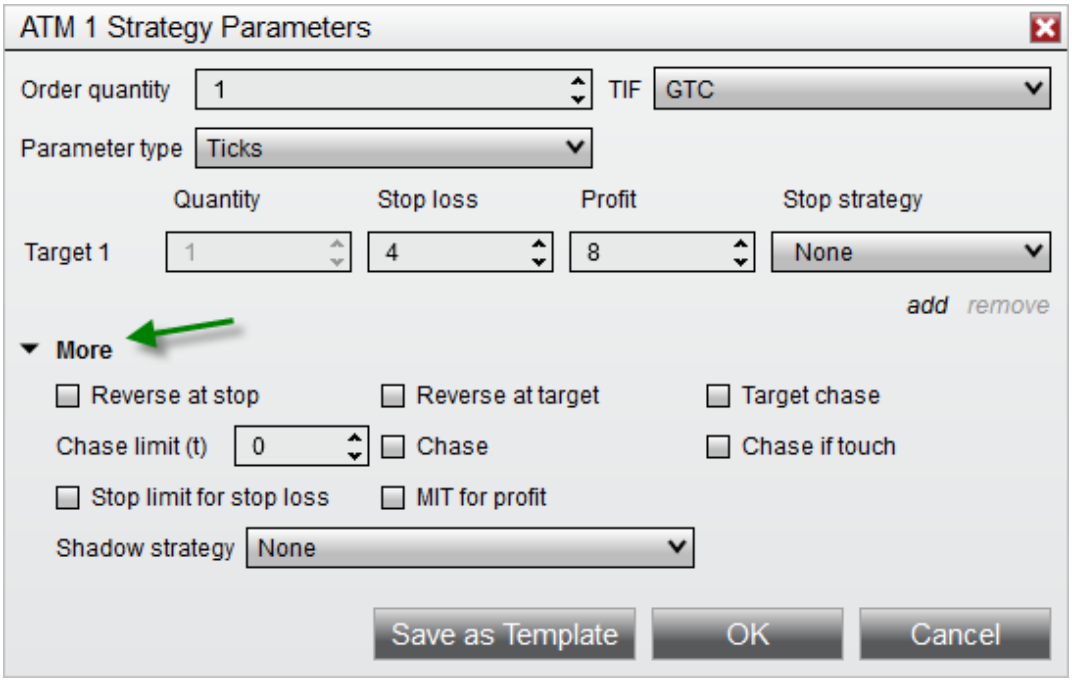
Profit	Sets the value that determines the Profit target price. If the value is set to 4 (ticks) and your average entry for initiating the order is 1000 and you are long, your Profit target would be submitted at AvgEntry + Profit target = 1000 + 4 ticks = 1001 Profit target. This assumes that each tick is valued at 0.25.
Stop Strategy	Sets the Stop Strategy

For further reference, please look at the Strategy Examples located within the "ATM Strategy" page.

Understanding advanced ATM parameters

More Options

To access the Advanced options, click on the **More** text which will expand these additional **ATM Strategy** features.

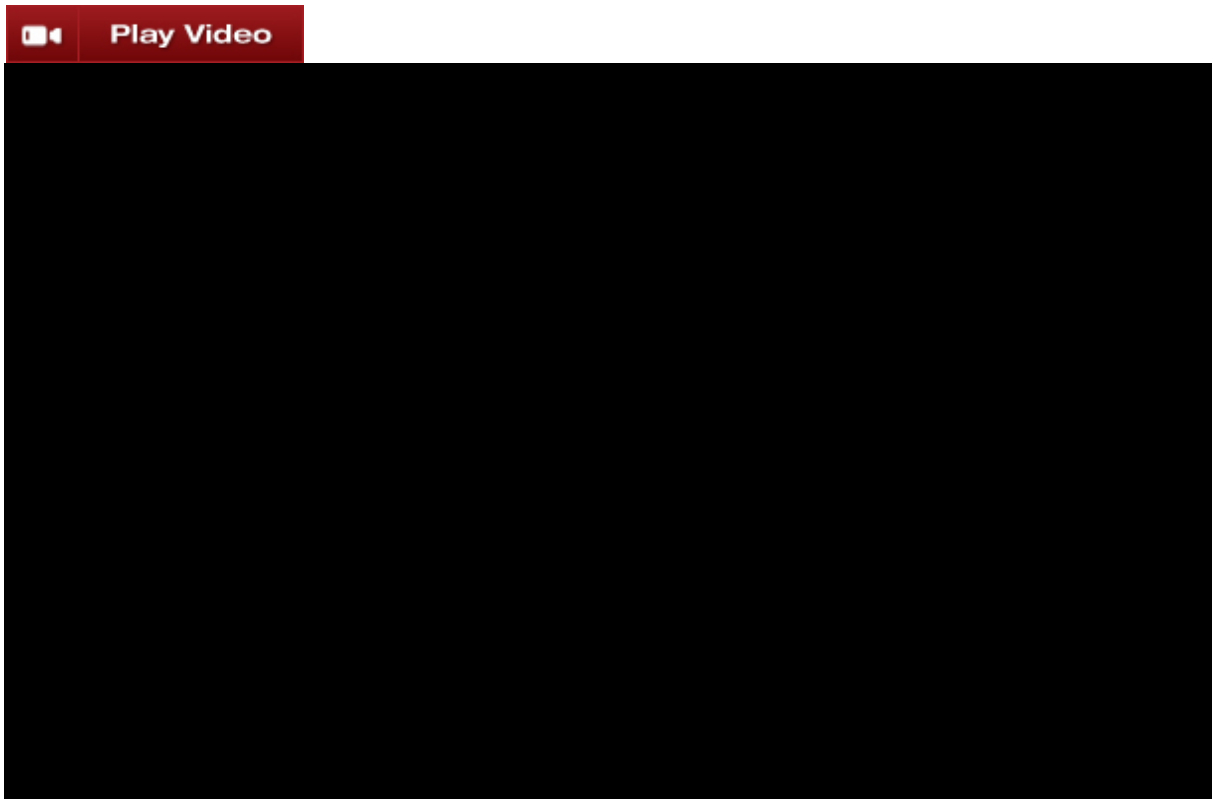


From the Advanced Options section you can enable the Shadow Strategy, Auto Reverse, or Auto Chase features.

10.1.2.2 ATM Strategy Selection Mode

Most of the NinjaTrader order entry screens have three modes that you can set to determine the behavior of the ATM Strategy selection control list upon submission of an order that enters the market/initiates an ATM Strategy. You can set this mode via the order entry screen's Properties dialog window that is accessible via the right mouse click context menu.

Before reviewing this section you should have a thorough understanding of how the strategy control list determines what actions (if any) to take when a submitted order is filled. Please review the video and content in the preceding page ATM Strategy Parameters.



As a quick reminder, when the strategy control list is set to:
<None> - Orders submitted take no action once filled or part filled (no stops or targets are placed)
<Custom> - Orders submitted will initiate the custom defined ATM Strategy (submission of stops and targets) once filled or part filled

<My Strategy Template Name> - Orders submitted will initiate your user defined ATM Strategy (submission of stops and targets) once filled or part filled

< My Strategy Template Name - X > - Existing ATM Strategy stop and target orders will be amended once the submitted order is filled or part filled

There are three available ATM Strategy Selection Modes:

- Select Active ATM Strategy on Order Submission
- Keep Selected ATM Strategy Template on Order Submission
- Display Selected ATM Strategy Only

Understanding the "Select Active ATM Strategy on Order Submission" mode

Select Active ATM Strategy on Order Submission

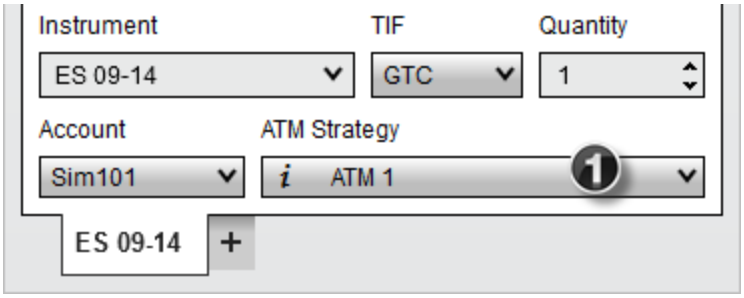
This mode will automatically select the newly created active ATM Strategy on entry order submission in the ATM Strategy control list. This is the default setting upon initial NinjaTrader installation.

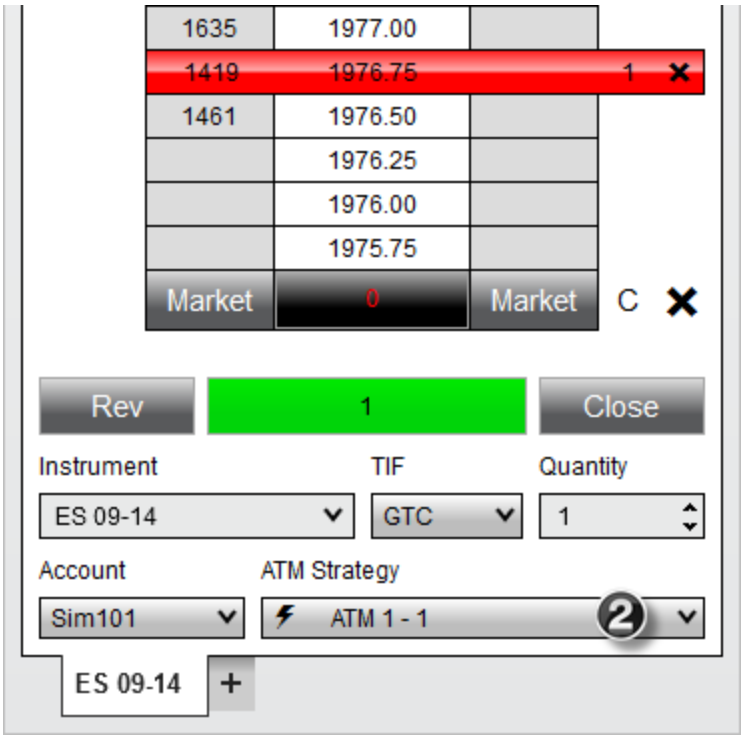
Who is this mode designed for?

This mode is designed for traders who want the existing strategy Stop Loss and Profit Targets to be automatically amended when they scale into or out of a position being managed by an ATM Strategy by default.

Example (see image below)

1. A user defined ATM Strategy is selected.
2. Once the entry order is submitted, the ATM Strategy selection control automatically selects the active ATM Strategy that you just created (**< My Strategy Template Name - X>**).





Note: When using multiple tabs, all tabs will select the same ATM or their own active ATM. With Select Active ATM Strategy on Order Submission, once you place an order your active ATM will be selected. Since the other tabs with different instruments would not be using that same active ATM, they will be set back to None or their own active ATM.

Understanding the "Keep Selected ATM Strategy Template on Order Submission" mode

Keep Selected ATM Strategy Template on Order Submission

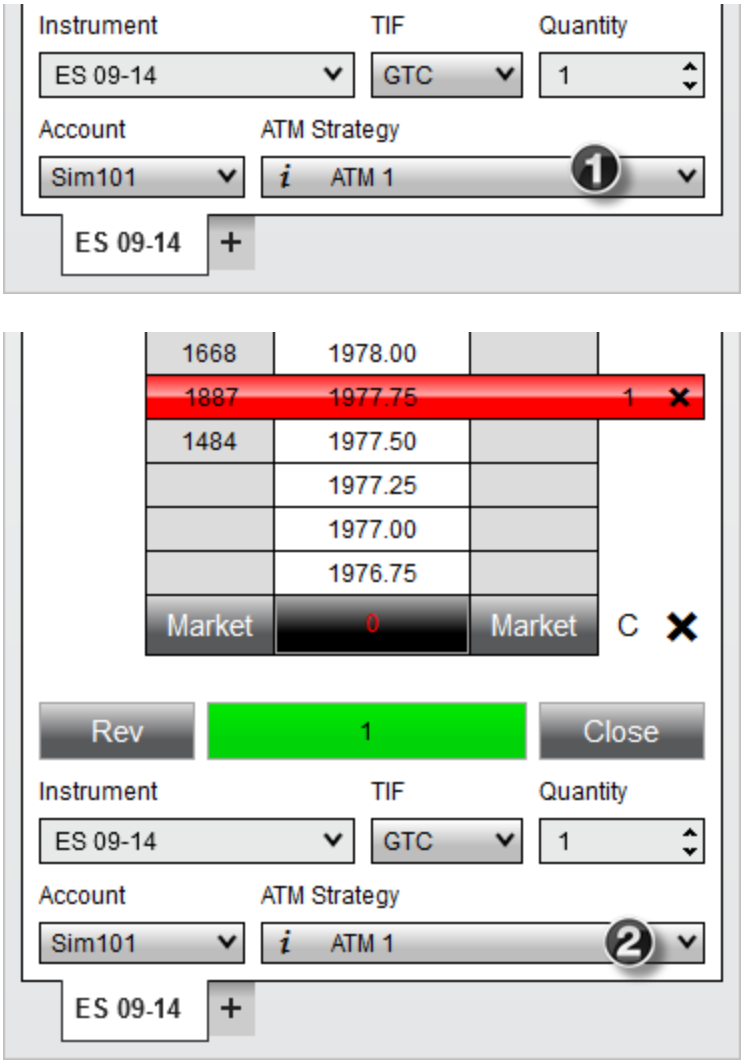
This mode will keep the currently selected ATM Strategy template selected in the strategy control list upon order submission.

Who is this mode designed for?

This mode is designed for traders who by default, want to always create a new set of Stop Loss and Profit Target brackets (new ATM Strategy) with each new order placed. An example of this would be a trader who wanted a single bracket placed with a Stop Loss of four ticks and a Profit Target of eight ticks. The trader wants to place two entry limit orders, the first at a price of X and the second at a price of X - 2 ticks. The purpose is to scale into an overall position but have the brackets be submitted and calculated from each individual fill price of the two orders.

Example (see image below)

1. A user defined ATM Strategy is selected.
2. Once the entry order is submitted, there will be no change in selection in the ATM Strategy control list. It will continue to look like the upper right image as the same ATM Strategy is automatically reselected after each order.



Understanding the "Display Selected ATM Strategy Only" mode

Display Selected ATM Strategy Only

This mode is an advanced mode and should only be used once you have become very familiar with the NinjaTrader application.

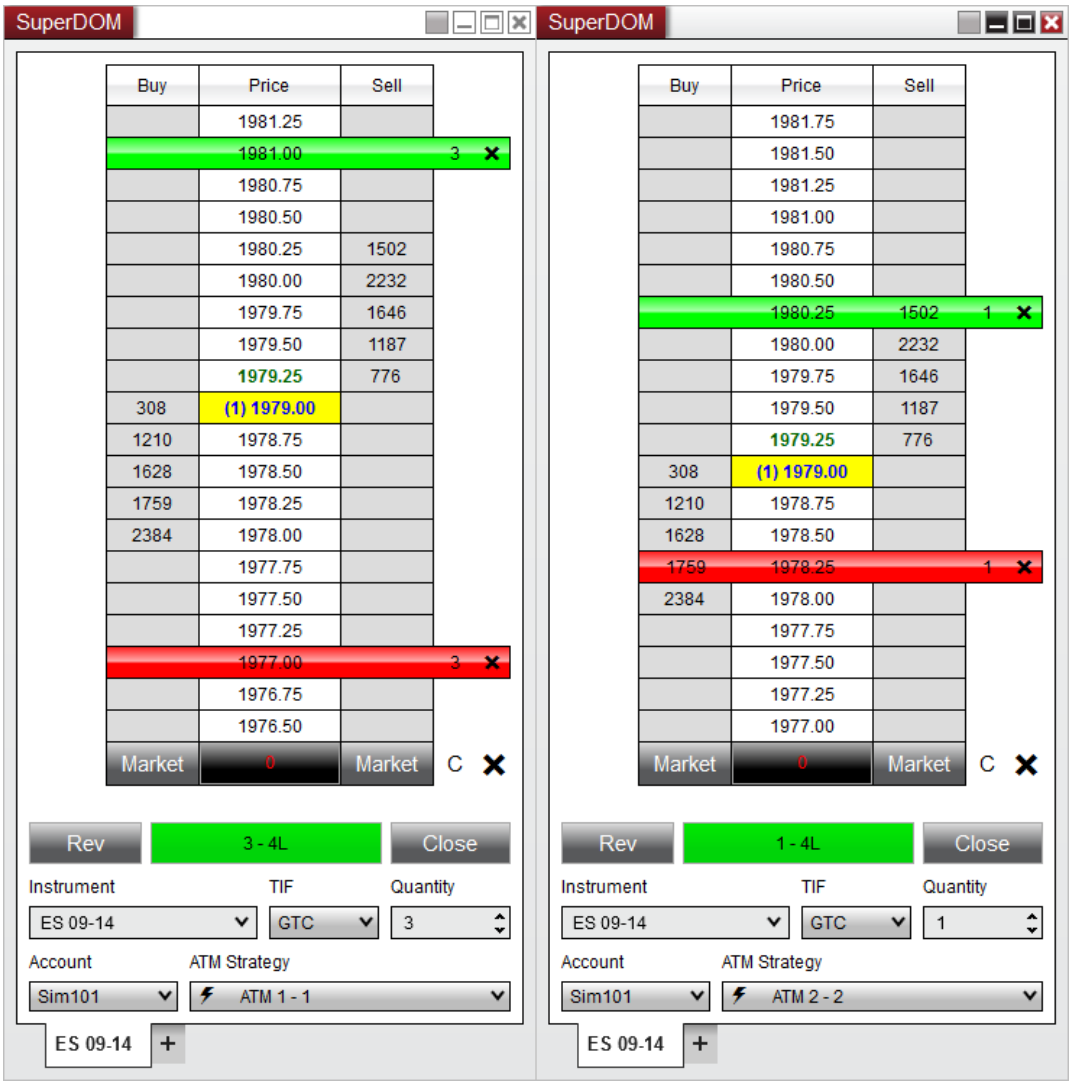
Who is this mode designed for?

This mode is designed for traders who want to run concurrent ATM Strategies (trades) in the same market. This mode will visually separate all concurrent running ATM Strategies thereby allowing you to have multiple SuperDOMs open, tracking the same market but displaying different trade strategies. A practical example might be that you have taken a day long intra day swing trade against a fifteen minute chart for five contracts. Throughout the day, you scalp the same market on a one minute time frame. This mode allows you to have two SuperDOMs open, one allocated to manage and only display your day long intra day swing trade, the other used to manage and only display your scalp trades.

Example (see image below)

In the image right, you can see two separate SuperDOMs monitoring the same ES 09-14 market. In the ATM Strategy control list, there are two different ATM Strategies running and each is displayed separately in an individual SuperDOM. Orders, positions, average entry and unrealized profit are displayed individually for each separate running ATM Strategy.

- When running multiple concurrent ATM Strategies by changing the selected active strategy in the strategy control list you can change which strategy will be displayed
- When you select **<None>** in the strategy list, all working orders and ATM Strategies will be displayed
- The position display will display the number of contracts being managed by the ATM Strategy and then your net position size. The box is color coded to the ATM Strategy's market position. So if your ATM Strategy is long, the box will be colored green. Using the first image on the right as an example, it shows "3 - 4L" in a green box. Green indicates the ATM Strategy is long, the number 3 indicates that there are 3 contracts being managed by that ATM Strategy and 4L indicates the account actually holds 4 contracts long. What it is saying is; that we are running a long ATM Strategy that is managing 3 of 4 contracts that are held long in my account. The image on the right is managing 1 of the 4 long contracts.
- When you do not have any active ATM Strategies selected, the SuperDOM position display will display your net account position
- Pressing the "CLOSE" button while an active ATM Strategy is selected will close only that ATM Strategy. If anything else is selected, it will close the entire account position including all other working ATM Strategies.



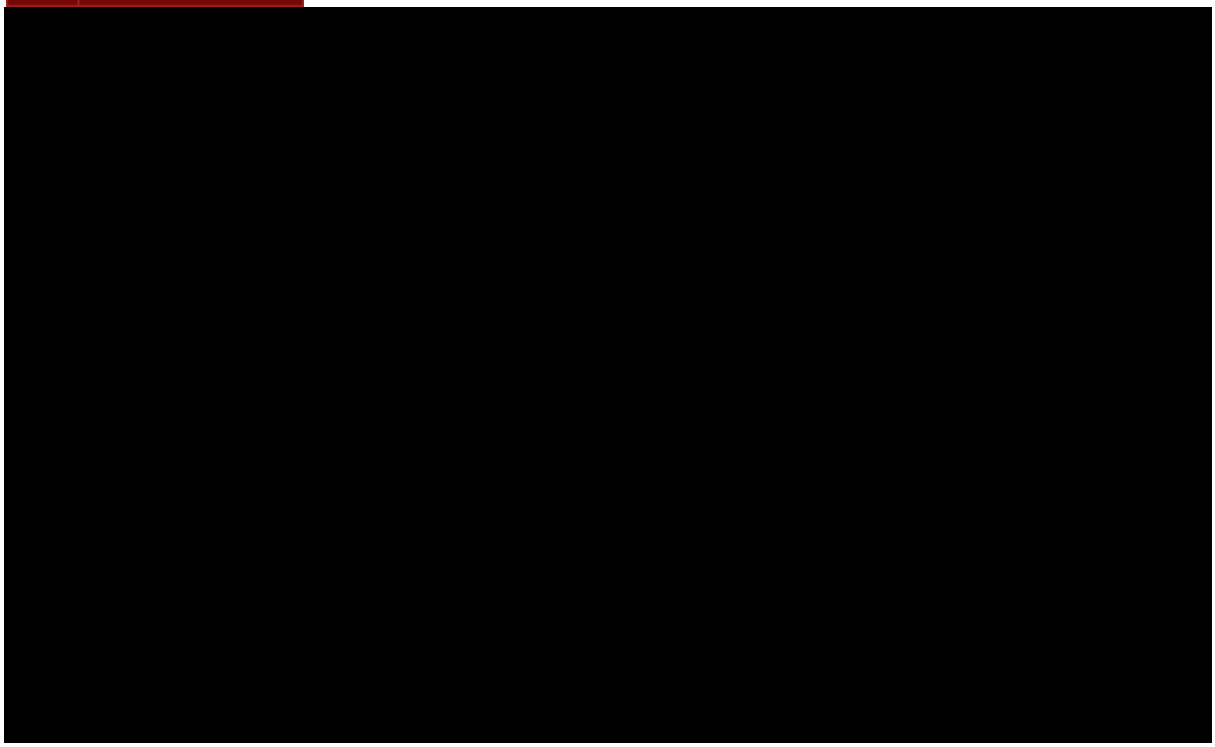
Critical: When a SuperDOM is set to this mode, it will only display orders associated to the active selected ATM Strategy in the ATM Strategy control list. This means that if there are other orders working in the selected market that are not associated to the ATM Strategy, you will not see them displayed. The risk is that you could have orders working, you forget about them or did not even know they were still working, they are filled and you could damage your trading account. Please fully understand how to use this powerful feature before putting it to use.

10.1.2.3 Stop Strategy

ATM Stop Strategies

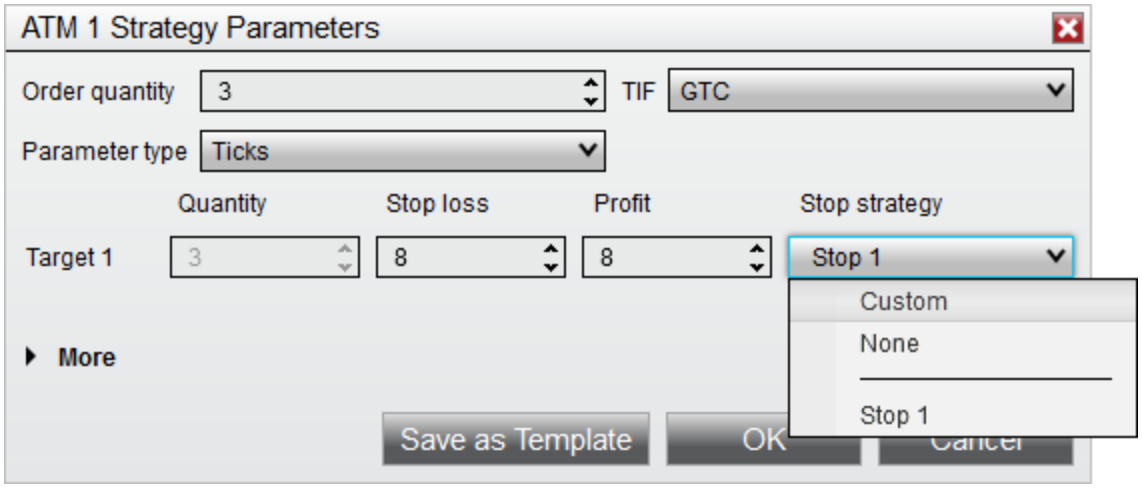
ATM Stop Strategies provide additional functionality for the stop losses placed by an ATM Strategy, including auto-breakeven, auto-trail, and Simulated Stop orders.

Play Video

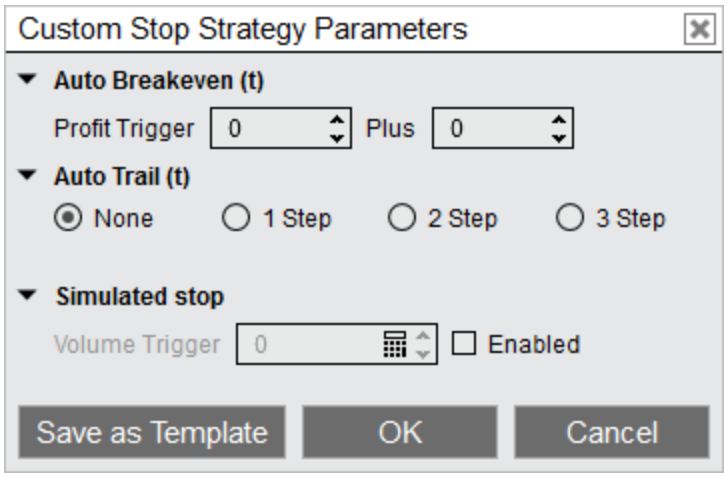


A Stop Strategy is an extension of an ATM Strategy. It allows you to combine Auto Breakeven, Auto Trail, and Simulated Stop strategies for the management and automatic adjustment of your Stop Loss orders.

When setting up an ATM Strategy, you can select either **<*Custom*>**, **<*None*>**, or any pre-defined Stop Strategy template from the Stop Strategy control list.



If **<*Custom*>** or any template is selected ("Stop 1" in the image below is a template) a Stop Strategy Dialog window will appear.



You can enter the appropriate values to enable any of the Stop Loss automation strategies. You can also save commonly used parameters as a Stop Strategy template.

Note: The parameter type of **Currency** or **Price** cannot be used for stop strategies. **Tick** will be used instead, which is indicated by the **(t)** next to **Auto Breakeven** and **Auto Trail** in the stop strategy parameters.

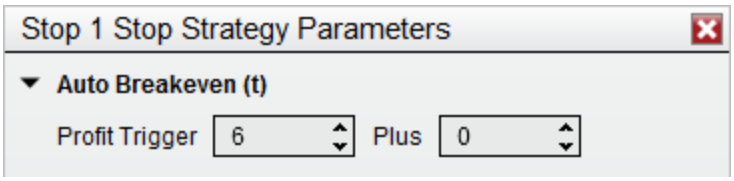
10.1.2.3.1 Auto Breakeven

The Auto Breakeven feature will adjust your Stop Loss order to breakeven (average entry price for the ATM Strategy position) once a user defined Profit Trigger has been reached.

▼ Understanding the Auto Breakeven parameters

Auto Breakeven Parameters

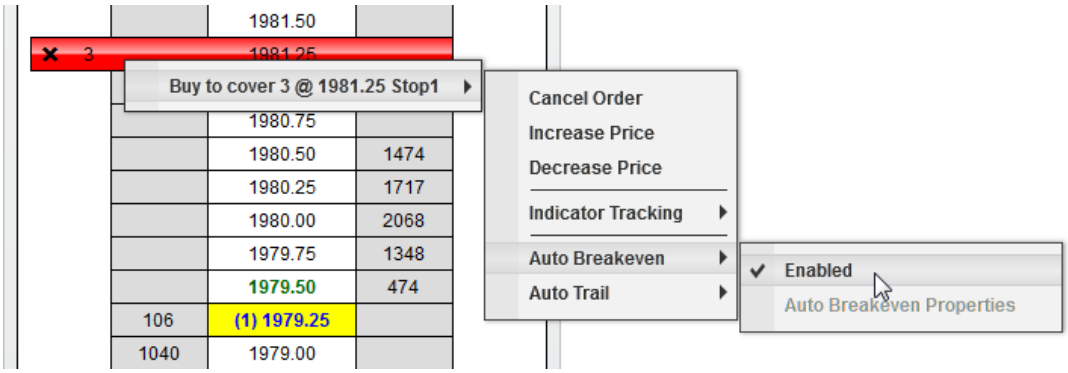
Profit Trigger	Sets the amount of profit required to move the Stop Loss to a breakeven value
Plus	Sets the amount added to the breakeven (average entry price for the ATM Strategy position) value



How to enable the Auto Breakeven

Auto Breakeven can be set before entering a position as part of a stop strategy, and you can also enable or disable it on a working Stop Loss order.

If you move your mouse over an active Stop Loss order in the buy cell for a buy order or sell cell for a sell order and press down on your right mouse button, you will see a menu of all working orders. Each working order menu has a sub menu that displays any applicable strategies that can be enabled or disabled. In the image below, you can see that Auto Breakeven is currently enabled. By selecting the "**Enabled**" menu item, you can enable or in this example disable the Auto Breakeven. You can change the parameters by selecting the "**Auto Breakeven Properties**" menu when Auto Breakeven is disabled.



Auto Breakeven Examples

Auto Breakeven Example #1

- Profit Trigger - 8 ticks
- Plus - 0 ticks
- Average Entry - 1000 Long (SP Emini contract)

As soon as the market trades at 1002 (Average Entry + Profit Trigger = 1000 + 8 ticks = 1002) NinjaTrader will move the Stop Loss order to 1000 (Average Entry + Plus = 1000 + 0 = 1000) and enter a log event in the Log tab.

Auto Breakeven Example #2

- Profit Trigger - 10 ticks
- Plus - 2 ticks
- Average Entry - 10200 Short (DOW Emini contract)

As soon as the market trades at 10190 (Average Entry - Profit Trigger = 10200 - 10 ticks = 10190) NinjaTrader will move the Stop Loss order to 10,198 (Average Entry - Plus = 10200 - 2 ticks = 10198) and enter a log event in the Log tab.

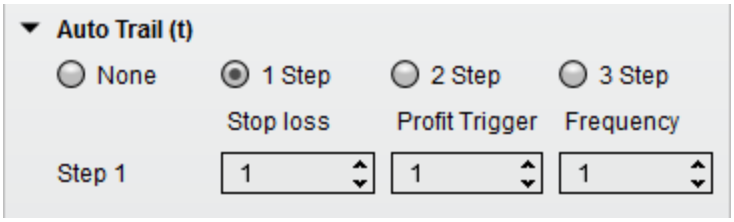
10.1.2.3.2 Auto Trail

Auto Trail is a powerful stop strategy that allows you to be more liberal with your Stop Loss at the early stage of your trade and tighten your Stop Loss as your profits in your trade increase.

▼ Understanding the Auto Trail parameters

Auto Trail Parameters

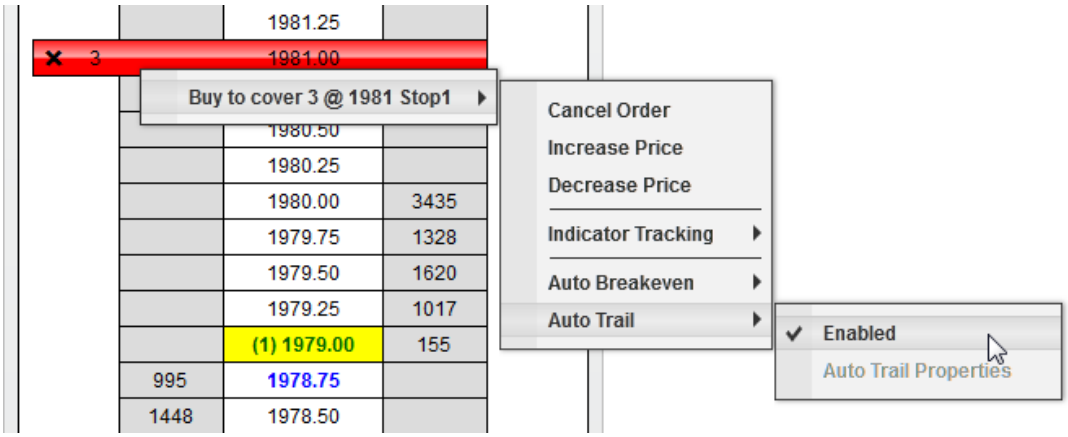
Stop Loss	Sets the value of the Stop Loss order as an offset behind the Profit Trigger/Frequency
Profit Trigger	Sets the amount of profit required to trigger the initial Stop Loss adjustment for the step
Frequency	Sets the value of how frequent the Stop Loss order is adjusted after the Profit Trigger



There are 3 available steps for Auto Trail parameters. Each step can have unique parameters providing you with the flexibility to tighten your Stop Loss automatically as your profits increase. Auto Trail can be set before entering a position as part of a Stop Strategy. You can also enable or disable it on a working Stop Loss order.

If you move your mouse over an active Stop Loss order in the buy cell for a buy order or sell cell for a sell order and press down on your right mouse button, you will see a menu of all working orders. Each working order menu has a sub menu

that displays any applicable strategies that can be enabled or disabled. In the image below, you can see that **Auto Trail** is currently enabled. By selecting the "**Enable**" menu item, you can enable or in this example disable the Auto Trail. You can change the parameters by selecting the "**Auto Trail Properties**" menu item when Auto Trail is disabled.

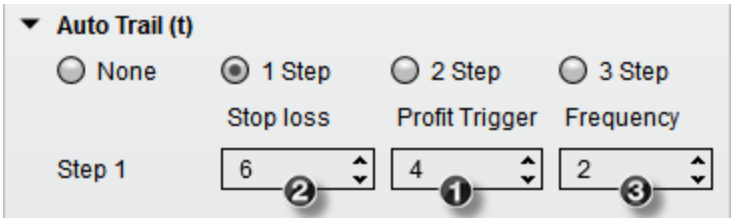


Auto Trail Examples

Auto Trail Example #1:

The settings in the image below are saying:

1. *"Once our trade has 4 ticks in profit..."*
2. *"...move our Stop Loss back 6 ticks..."*
3. *"...and also move it up for every additional 2 ticks in profit."*



Average Entry - 1000 Long (SP Emini contract)

The market moves up to 1001 and the Auto Trail is triggered **(Average Entry + Profit Trigger = 1000 + 4 ticks = 1001)** and the Stop Loss is adjusted to 999.50 **(1001 - Stop Loss = 1001 - 6 ticks = 999.50)**. For every additional 2 ticks **(Frequency of 2 ticks)** the Stop Loss will be adjusted by 2 ticks.

Auto Trail Example #2 building on top of Example #1:

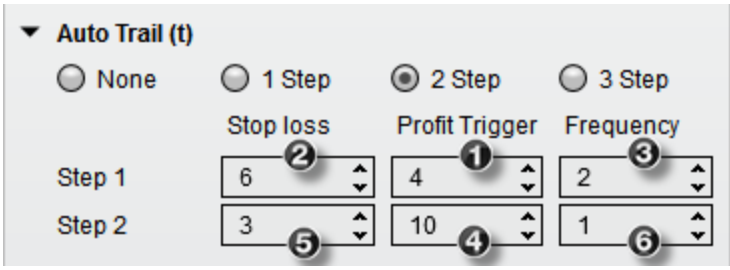
The settings in the image below are saying:

Step 1

1. *"Once our trade has 4 ticks in profit..."*
2. *"...move our Stop Loss back 6 ticks..."*
3. *"...and also move it up for every additional 2 ticks in profit."*

Step 2

4. *"Then once our trade has 10 ticks in profit...*
5. *"...tighten and move our Stop Loss back 3 ticks..."*
6. *"...and increase the rate at which the Stop Loss is adjusted and move it up for every additional 1 tick in profit."*



Average Entry - 1000 Long (SP Emini contract)

The market moves up to 1001 and the Auto Trail is triggered **(Average Entry + Profit Trigger = 1000 + 4 ticks = 1001)** and the Stop Loss is adjusted to 999.50 **(1001 - Stop Loss = 1001 - 6 ticks = 999.50)**. For every additional 2 ticks **(Frequency of 2 ticks)** the Stop Loss will be adjusted by 2 ticks (same as Example #1). Then the market moves to 1002.50 and the 2nd step of the Auto Trail strategy is triggered and the Stop Loss is adjusted to 1001.75 and moves up by 1 tick with every additional tick in profit.

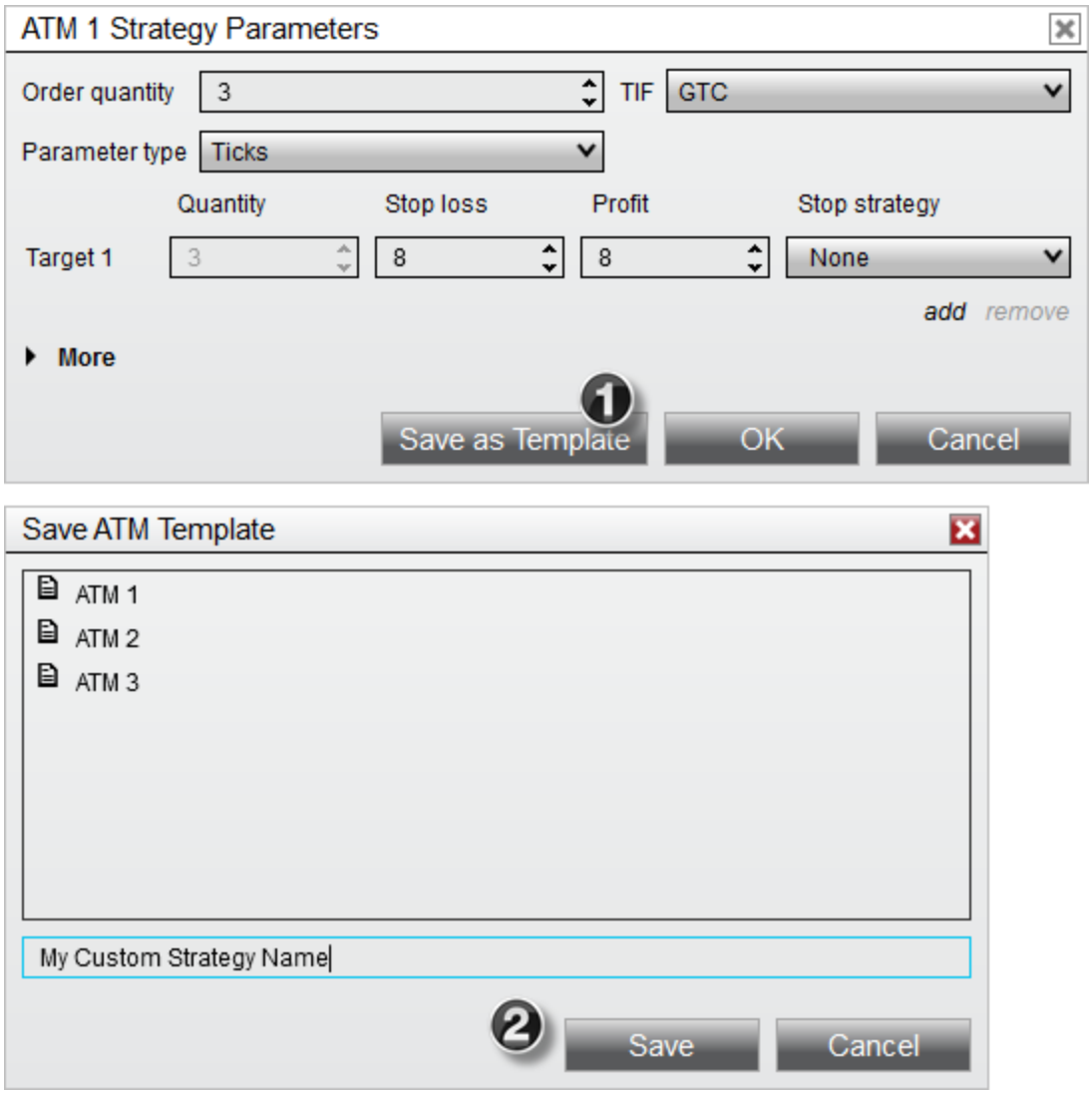
10.1.2.4 Manage ATM Strategy Templates

An ATM Strategy is defined by the parameters you enter into the ATM Strategy parameters section on any of the order entry screens. The collection of parameters that make up a strategy can be saved as a template that you can recall at a later date to automatically populate all of the ATM Strategy parameters.

Saving ATM Strategy Templates

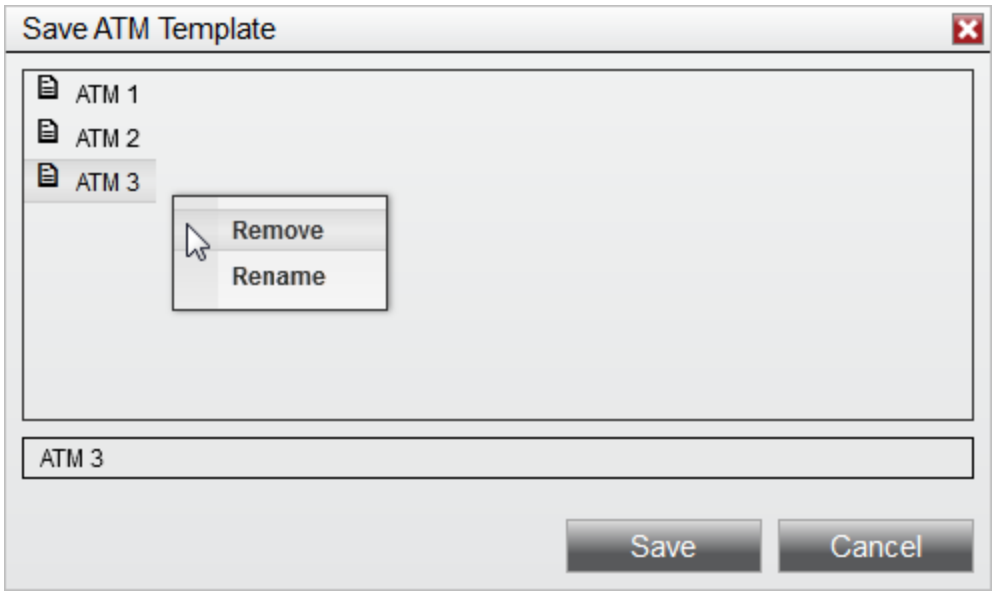
To save your current **ATM Strategy parameters** in a template:

1. Select the **Save as Template** button
2. From the presented file dialog give the template a custom name



Removing or Renaming ATM Strategy Templates

Right clicking on an existing **ATM Strategy template** will give you the option to either **Remove** or **Rename** the strategy template.



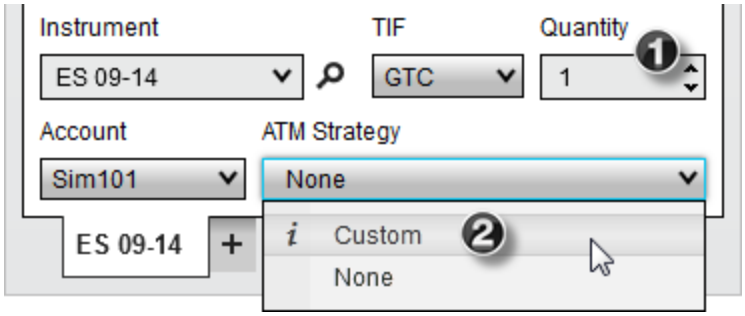
See ATM Strategy Example #1 and ATM Strategy Example #2 for further reference on how to create and save an ATM Strategy template.

10.1.2.5 Tutorial: ATM Strategy Example #1

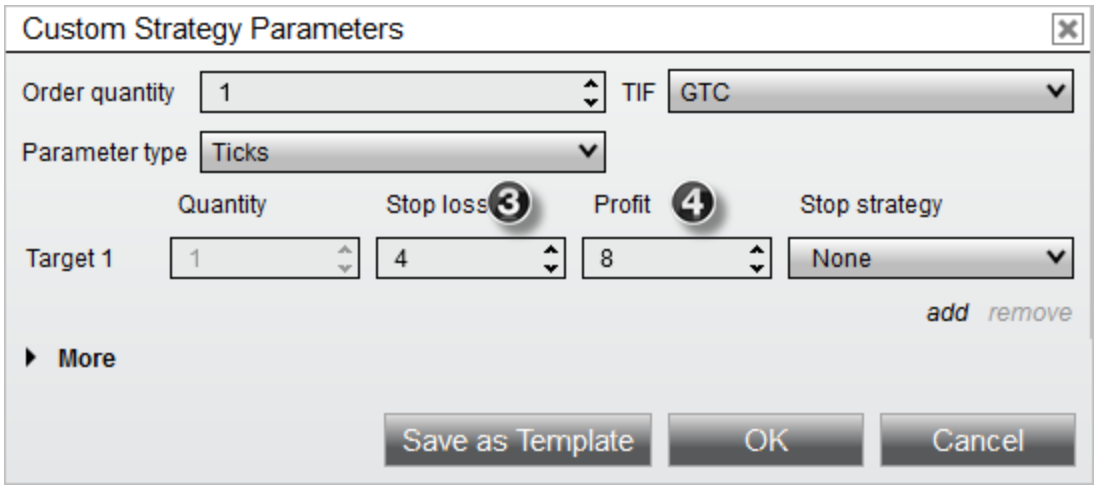
ATM Strategy Example

Following is an example of how to create a simple 1 stop/1 target ATM Strategy and save the strategy as a template. You can do this via any NinjaTrader order entry window (excluding the Order Ticket window)

Play Video



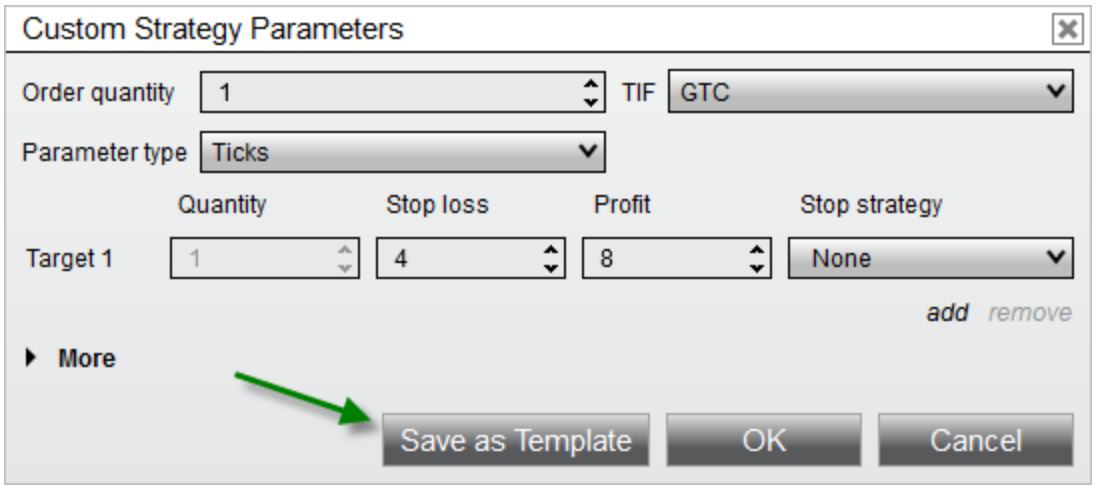
1. Set the order quantity to 1 contract
2. From the ATM Strategy control list select **<Custom>** which will open the **Custom Strategy Parameters** window

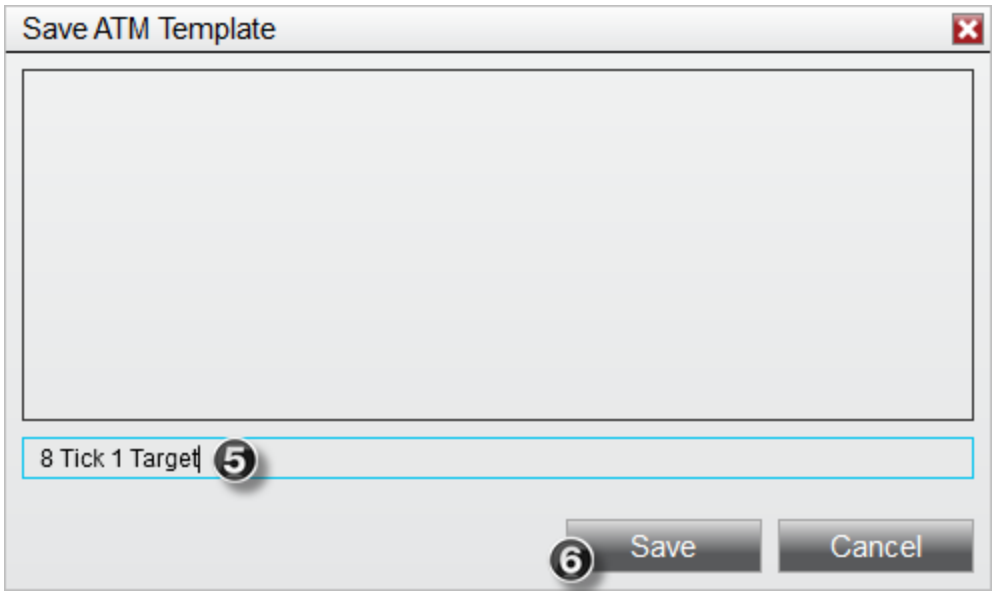


3. Set the Stop Loss value to 4 ticks
4. Set the Profit Target value to 8 ticks

This simple ATM Strategy will automatically submit a Stop Loss order 4 ticks from entry and a Profit Target order 8 ticks from entry.

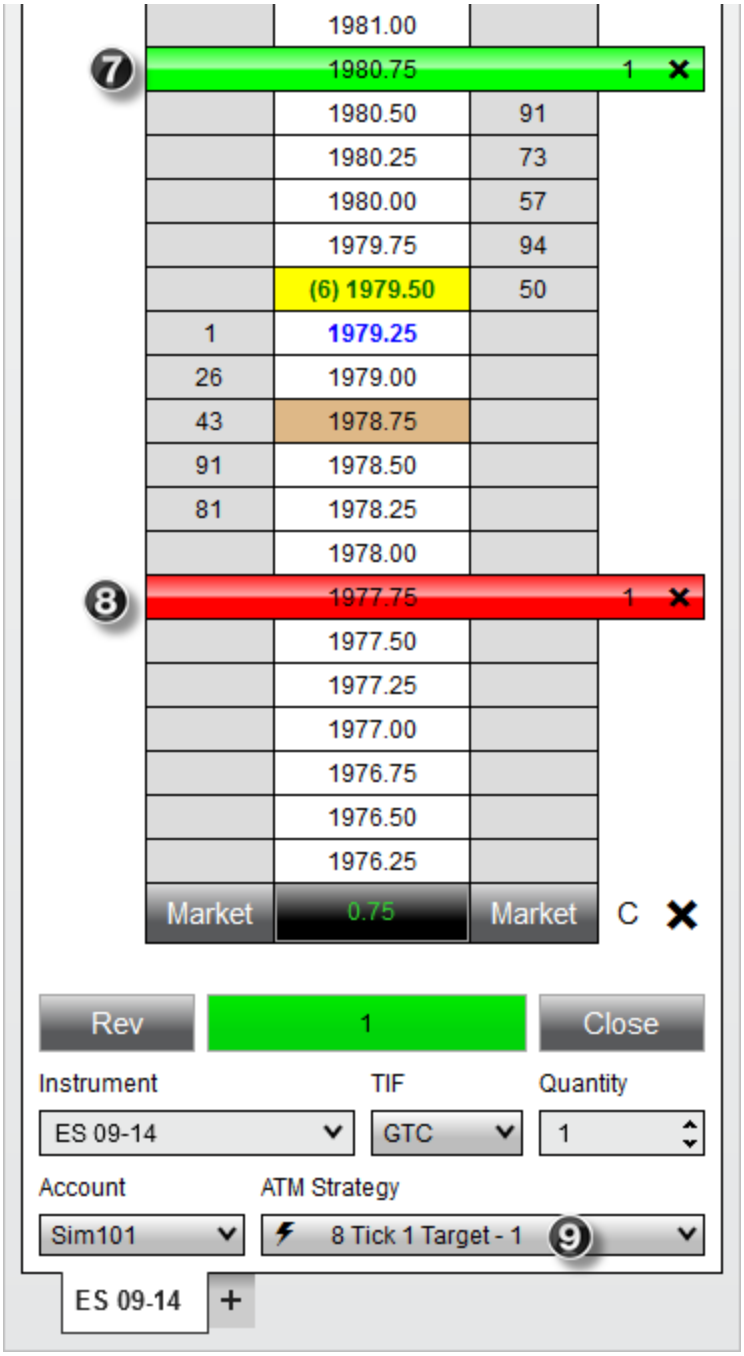
You can save this ATM Strategy as a template by clicking the **Save as Template** button from the **Custom Strategy Parameters** window





5. Enter the name "8 Tick 1 Target"
6. Press the "Save" button

Once you press the **save** button, a template is created for this ATM Strategy, and it will become available in the strategy control list of all order entry windows. You can now place an order which, once filled, will automatically trigger the ATM Strategy to submit the Stop Loss and Profit Target. In the image below, an order was submitted and filled at 1978.75 as depicted by the brown colored cell.



7. A Profit Target was submitted at 1980.75 which is 8 ticks from our entry price of 1978.75
8. A Stop Loss was submitted at 1977.75 which is 4 ticks from our entry price of 1978.75
9. An active strategy named " 8 Tick 1 Target - 1" is created and listed under the ATM Strategy control list.

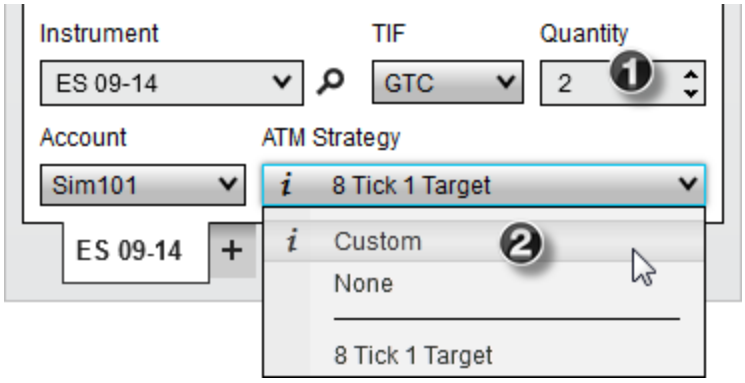
If under SuperDOM Properties you have the "ATM Strategy selection mode" set to "SelectActiveATMStrategyOnOrderSubmission", NinjaTrader will automatically set the ATM Strategy control list to the newly created ATM Strategy. The importance of this is if you place

another order, any fills resulting from the order will be applied to the existing Stop Loss and Profit Target orders. As an example, if we were filled on an additional contract, our Stop Loss and Profit Target would automatically be modified from 1 contract to 2 contracts. Both Stop Loss and Profit Target orders are tied via OCO which means if one of the orders is filled, the other will automatically be cancelled. If the option in the first sentence was not checked, the ATM Strategy control list would be set to the "8 Tick 1 Target" ATM Strategy template we just created. Any subsequent orders would create an additional ATM Strategy that would submit another set of Stop Loss and Profit Target orders.

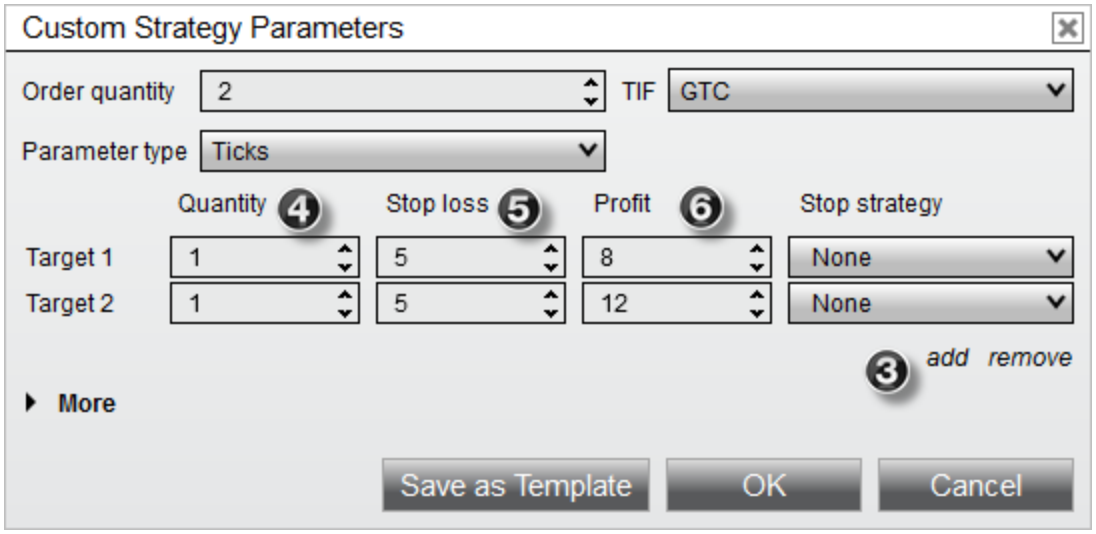
10.1.2.6 Tutorial: ATM Strategy Example #2

ATM Strategy Example

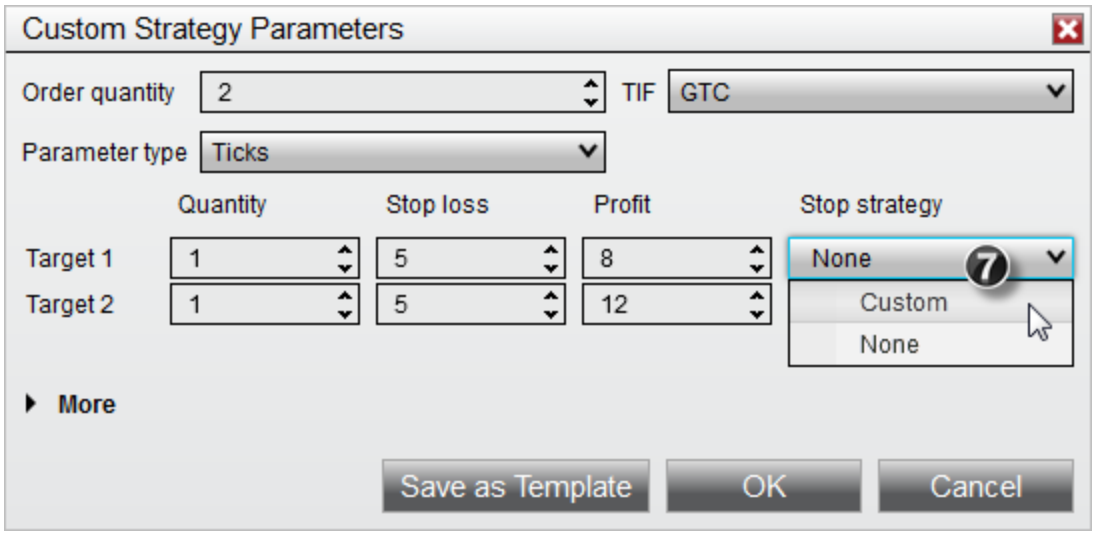
Following is an example of an ATM Strategy that will automatically submit 2 Stop Loss and Profit Target brackets once the originating entry order is filled. This ATM Strategy includes a Stop Strategy that will automatically adjust the Stop Loss orders using Auto Breakeven and Auto Trail strategies.



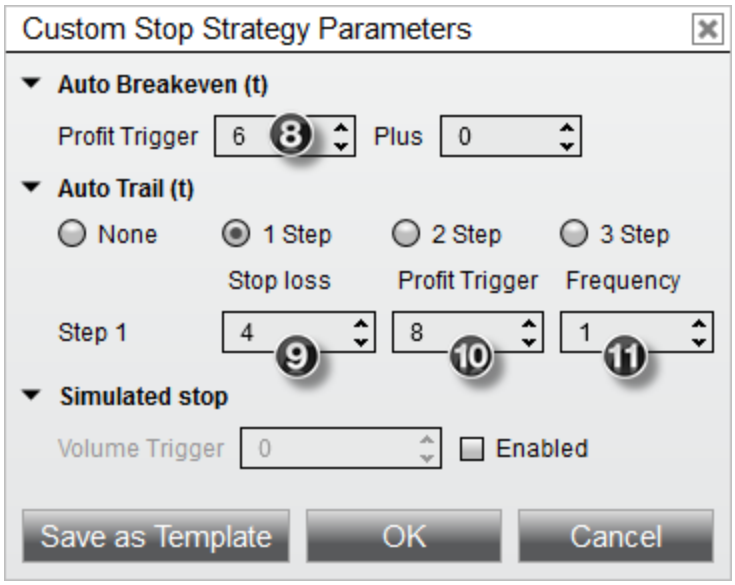
1. Set the order quantity to 2 contracts
2. From the ATM Strategy control list select **<Custom>**



3. Select "*add*" once to enable a 2nd target
4. Set "**Quantity**" fields to 1 contract each (that represents 1 contract for the first Stop Loss/Profit Target bracket and 1 for the 2nd)
5. Set the Stop Loss values to 5 ticks (you can set the 2nd Stop Loss to a wider value)
6. Set the first Profit Target to 8 ticks and the 2nd Profit Target to 12 ticks



7. Select **<Custom>** from the Stop Strategy control list under the first target.

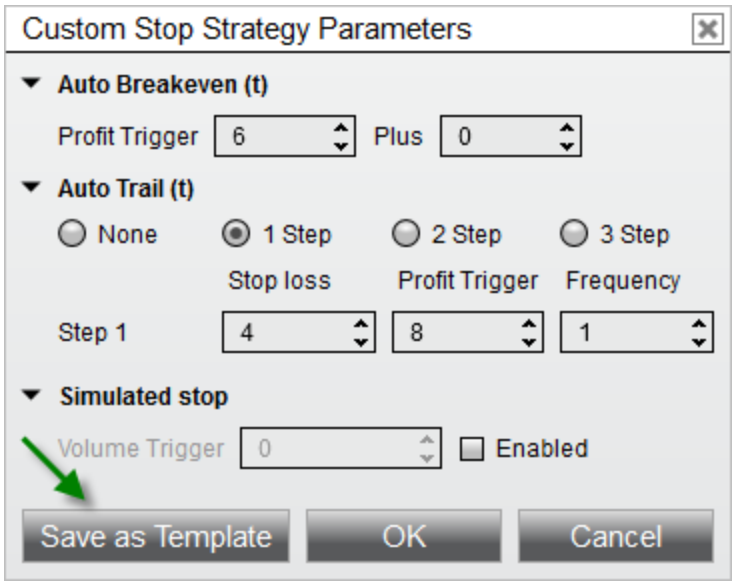


A Stop Strategy parameters dialog window will appear. This is where you will define the automation strategies for automatic Stop Loss adjustment.

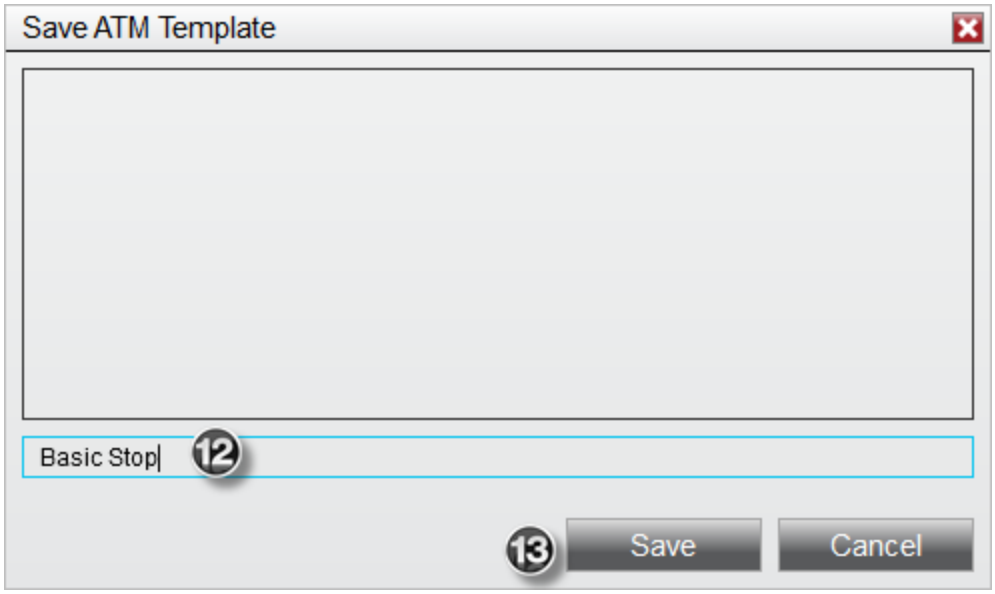
8. Set the Auto Breakeven "Profit trigger" value to 6 ticks. This will automatically adjust our Stop Loss order to breakeven once the ATM Strategy has 6 ticks in profit.
9. Set the Auto Trail "Stop loss" to 4 ticks

10. Set the Auto Trail "Profit trigger" to 8 ticks
11. Set the Auto Trail "Frequency" to 1 tick

The auto trail parameters will automatically start adjusting our Stop Loss order once we have 8 ticks in profit (9) to 4 ticks back (10) and adjust it for every 1 tick (11) in profit gain.

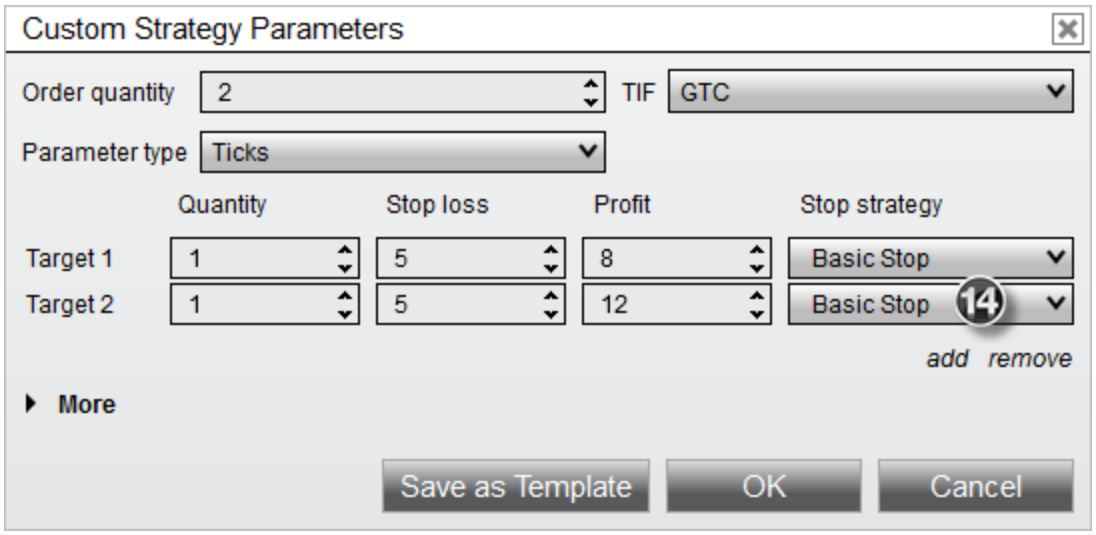


You can save the Stop Strategy as a template by clicking the **Save as Template** button



12. Enter the name "Basic Stop"
13. Press the "Save" button

Once you press the Save button, a template is created for this Stop Strategy and it will become available in all Stop Strategy control lists. Press the "OK" button on the Stop Strategy parameters dialog window to exit.

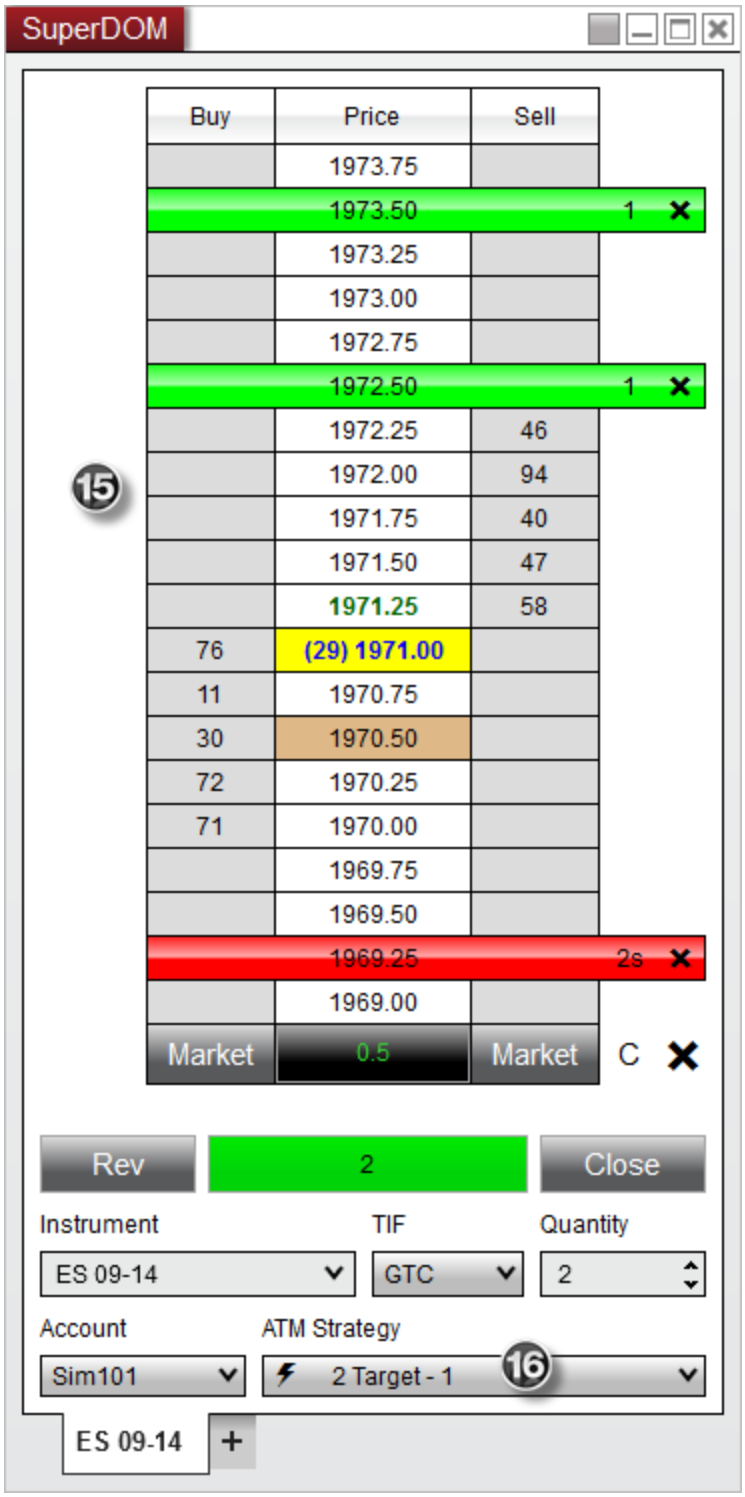


14. Select the Stop Strategy we just created (Basic Stop) in the 1st and 2nd Stop Strategy control lists. This sets the 1st and 2nd Stop Loss orders to the same Stop Strategy so that Stop Loss 1 and Stop Loss 2 will adjust in unison.

You can now save this ATM Strategy (Stop Strategies included) as a template by pressing the **Save as Template** button.

Type in the Name "2 Target" and click the "Save" button. We now have a 2 target strategy template that can be selected from the ATM Strategy control list at any time. Doing so will update all of the parameter fields automatically based on the information we have entered in this example.

You can now place an order which once filled will automatically trigger the ATM Strategy to submit the Stop Loss and Profit Target brackets. In the image below, an order was submitted and filled at 1970.50 as depicted by the brown colored cell.



15. The first Profit Target order was submitted at 1972.50 which is 8 ticks from our entry, the 2nd Profit Target was submitted at 1973.50 which is 12 ticks from our entry and finally, our 2 Stop Loss orders were submitted at 1969.25 which is 5 ticks from our entry. You can tell we

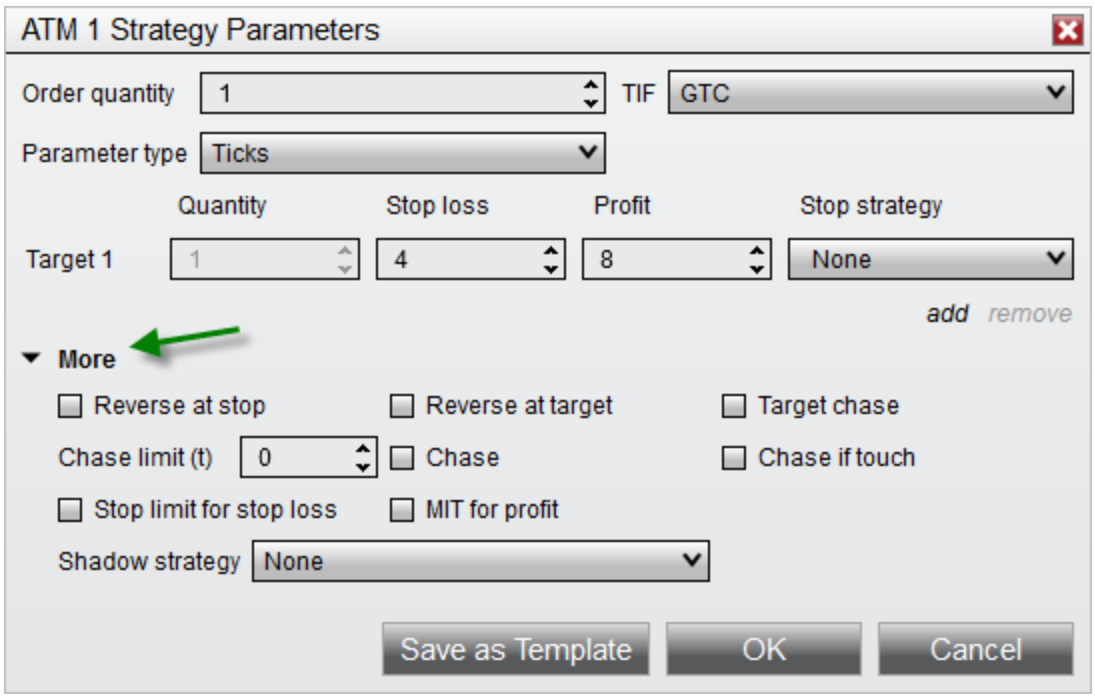
have 2 orders at the Stop Loss level because the Size Marker has the "s" suffix indicating that we have multiple orders consolidated at the price 1969.25.

16. An active ATM Strategy named "**2 Target - 1**" is created and listed under the ATM Strategy control list. The significance of 1 is that this is the only instance of the strategy that has been executed.

If under SuperDOM properties you have "ATM Strategy selection mode" set to "SelectActiveATMStrategyOnOrderSubmission", NinjaTrader will automatically set the ATM Strategy control list to the newly created ATM Strategy. The importance of this is if you place another order, any fills resulting from the order will be applied to the existing Stop Loss and Profit Target orders. As an example, if we were filled on an additional contract, our Stop Loss and Profit Target orders would automatically be modified from 1 contract to 2 contracts. Both Stop Loss and Profit Target orders are tied via OCO which means if one of the orders is filled, the other will automatically be cancelled. If the option in the first sentence was not checked, the ATM Strategy control list would be set to the original ATM Strategy template we created. Any subsequent orders would create an additional ATM Strategy that would submit another set of Stop Loss and Profit Target orders.

10.1.2.7 Advanced Options

All NinjaTrader order entry windows that offer **ATM Strategies** also include the **Advanced Options**. The Advanced Options include: Shadow Strategy, Auto Chase, and Auto Reverse features. You will find the **Advanced Options** from the ATM Strategy Parameters window by clicking on the **More** text (see the green arrow in the image below) which will expand these additional features.



Reverse at Stop	This will enter a position in the opposite direction, using the same ATM parameters, when a stop loss is hit
Reverse at Target	This will enter a position in the opposite direction, using the same ATM parameters, when a profit target is hit
Target Chase	This will cause a profit target to move towards the market price, as price moves away from the target
Chase	This will cause a Limit entry order to move towards the market price as it moves away
Chase if Touch	Enables the Chase function only if the order has been touched
Stop Limit for Stop Loss	When enabled, this will cause Stop Limit orders to be used for stop losses (default is unchecked, so Stop Market used)
MIT for Profit	When enabled, this will cause MIT orders to be used for profit targets (default is unchecked, so Limit is used)

Note: The "Stop Limit for Stop Loss" will not apply to Equities or Forex instruments - if this property is enabled when trading a stock or forex instrument, Stop Market orders will be used, instead.

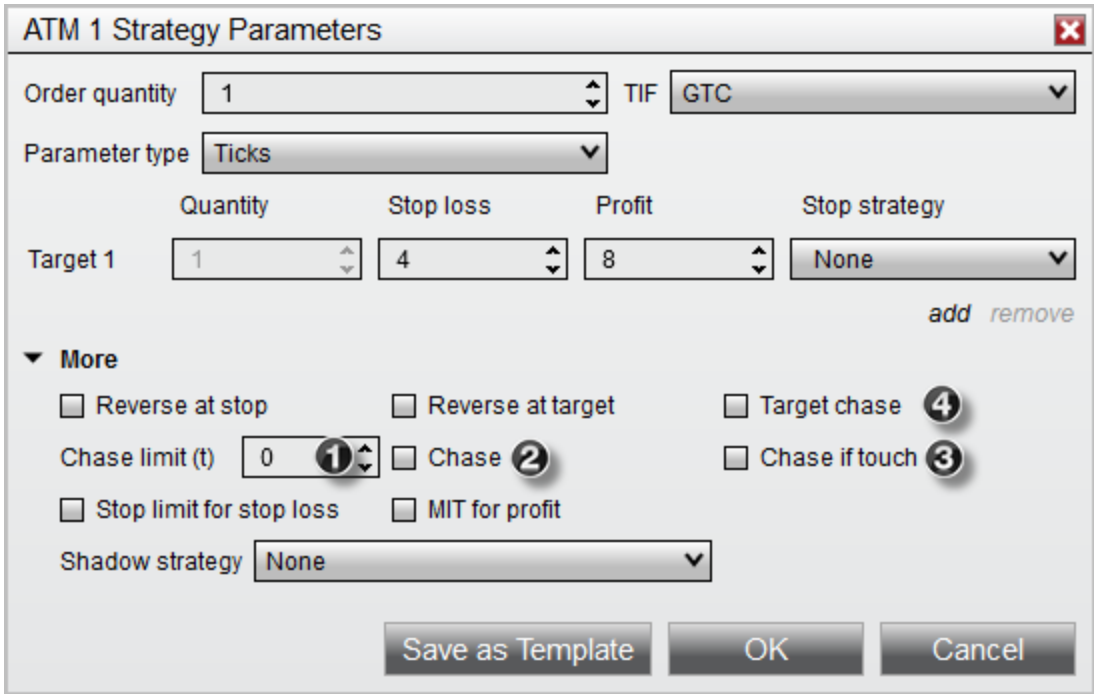
10.1.2.7.1 Auto Chase

Auto Chase will automatically adjust the price of a limit order as the market moves away from it.

Auto Chase Parameters

1. Chase Limit	The maximum amount that Auto Chase will adjust your limit order price
2. Chase	Enables Auto Chase on your entry orders
3. Chase if touch	Enables Auto Chase if touched on your entry orders

4. Target Chase	Enables Auto Chase if touched on your Profit Target orders



How does Chase work?

NinjaTrader will automatically adjust the price of your limit order with each tick the market moves away from your order up until the Chase Limit amount is reached.

How does Chase if touched work?

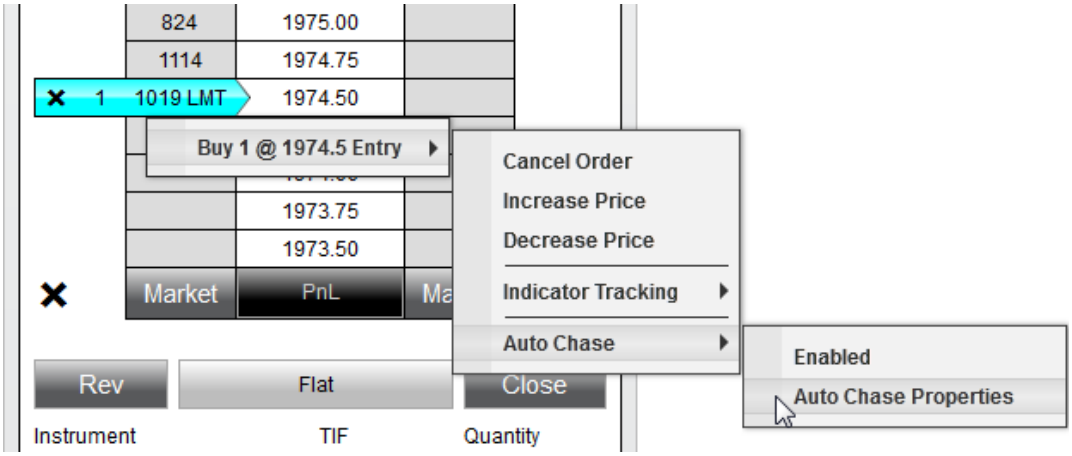
The difference between Chase and Chase if touched is that Chase if touched does not start chasing until your limit price has been touched. This works well for Profit Target orders. Your Profit Targets will rest at their respective limit price, if the market moves to the target and backs off but the target order does not fill, NinjaTrader would then start adjusting the target order to chase the market up until the Chase Limit amount.

How to enable the Auto Chase features

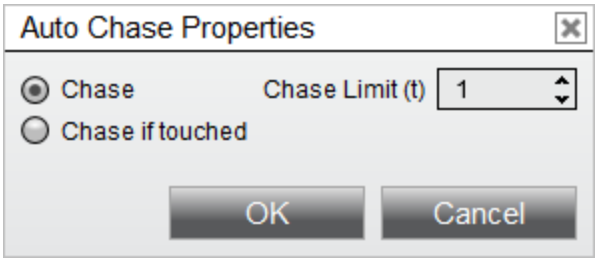
Auto Chase can be set as part of an ATM Strategy (set the parameters you want

use before entering the ATM Strategy). However, you can also enable or disable Auto Chase on working limit orders.

If you move your mouse over an active limit or Profit Target order in the buy cell for a buy order or sell cell for a sell order and press down on your right mouse button, you will see a menu of all working orders. Each working order menu has a sub menu that displays any applicable strategies that can be enabled or disabled. In the image below, you can see that Auto Chase is currently disabled. By selecting the "**Auto Chase**" menu, you can enable or disable it. You can change the parameters by selecting the "**Auto Chase Properties**" menu when Auto Chase is disabled.



The **Auto Chase Properties** window will allow you to select either **Chase** or **Chase If Touched** as well as the **Chase Limit** offset. Once the **Auto Chase Properties** have been configured, you will be able to navigate back to the **Auto Chase** sub-menu and check **Enabled** to turn on the Auto Chase features for the current strategy.



Auto Chase examples

Auto Chase Example #1

Chase Limit - 5
Buy Limit Price - 1000 (SP Emini contract)
Chase - Enabled
Current Bid - 1000.25

In this example, if the bid moves up to 1000.50, Auto Chase will adjust the buy limit price to 1000.25, subsequently each additional tick rise in price on the bid will adjust the buy limit price accordingly to a maximum price of 1001.25 which is Buy Limit Price + Chase Limit = 1000 + 5 ticks = 1001.25.

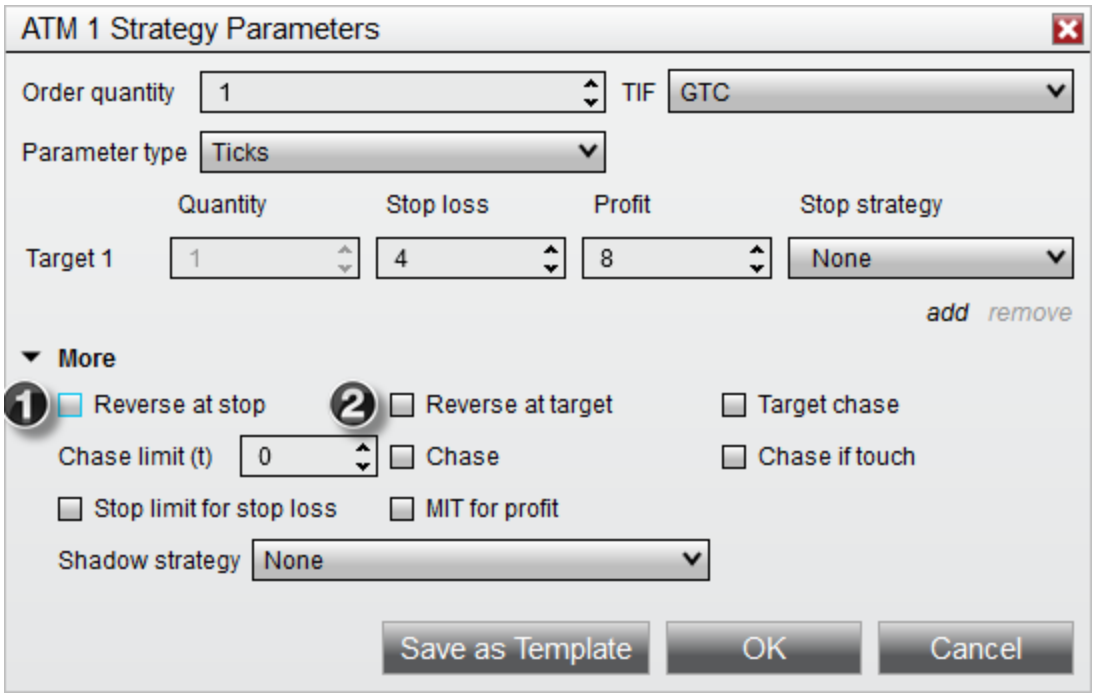
Auto Chase Example #2

Chase Limit - 5
Buy Limit Price - 1000 (SP Emini contract)
Chase if touched - Enabled
Current Bid - 1000.25

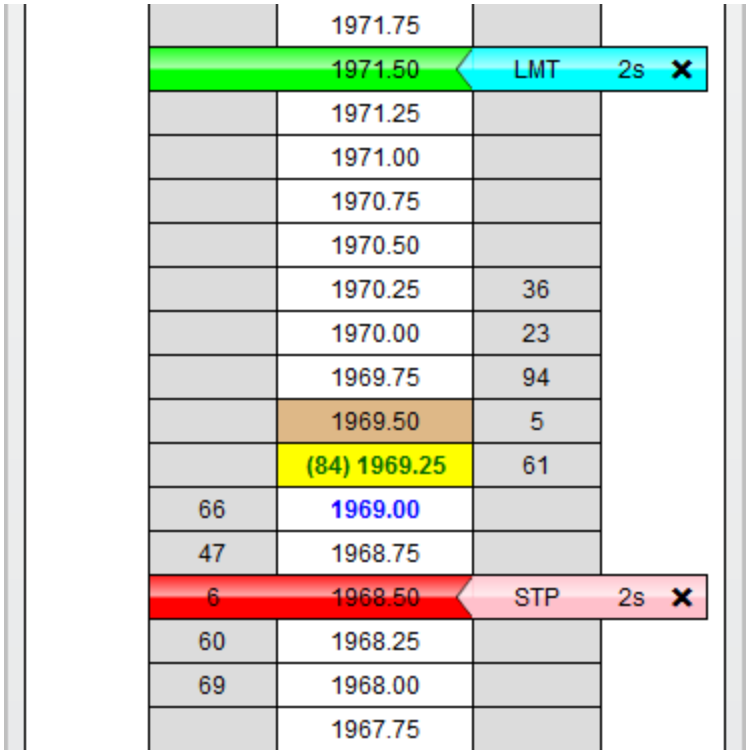
This example works in the same manner as example #1 with the exception that chasing does not start until the bid has touched the limit price of 1000.

10.1.2.7.2 Auto Reverse

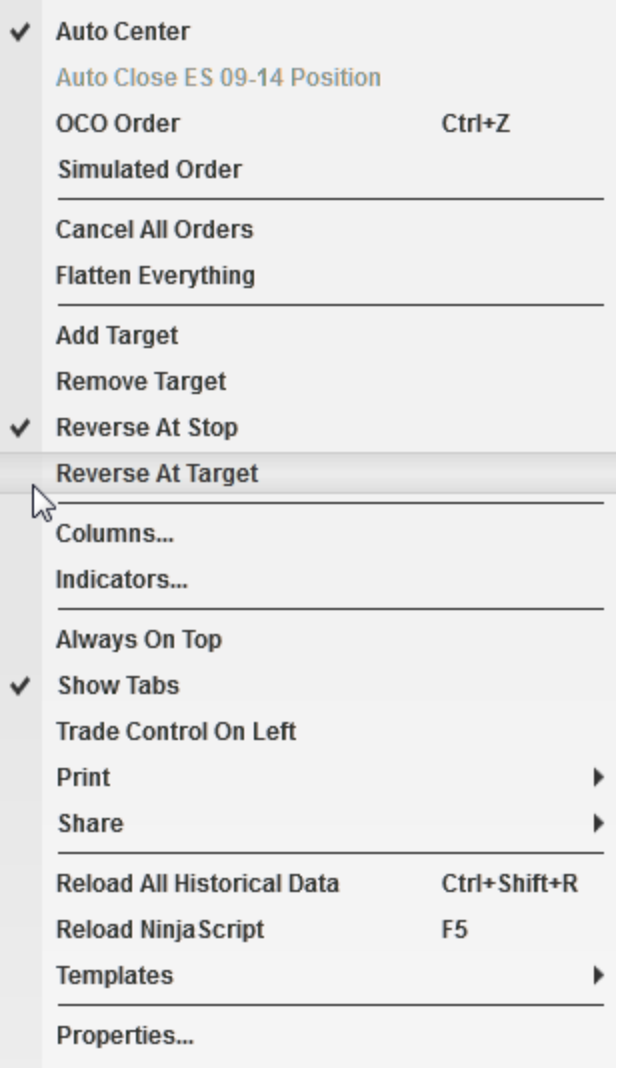
Auto Reverse simply reverses your position at either your Stop Loss or Profit Target. You can optionally enable (1) "Reverse at stop" or (2) "Reverse at target" with any ATM Strategy. The reverse ATM Strategy used will be the same as the position ATM Strategy you are reversing from.



When Auto Reverse is enabled, entry orders for the reverse ATM Strategy will be placed at either your Stop Loss or Profit Target orders. The image below shows a 1 stop/1 target ATM Strategy with Auto Reverse enabled for both the stop and target.



Modifying the price of either your Stop Loss or Profit Target will result in the modification of the reverse order as well. You can also enable or disable Auto Reverse of an active ATM Strategy at any time by selecting the "**Reverse At Stop**" or "**Reverse At Target**" menus via the right mouse click context menu in either the SuperDOM or Basic Entry windows.

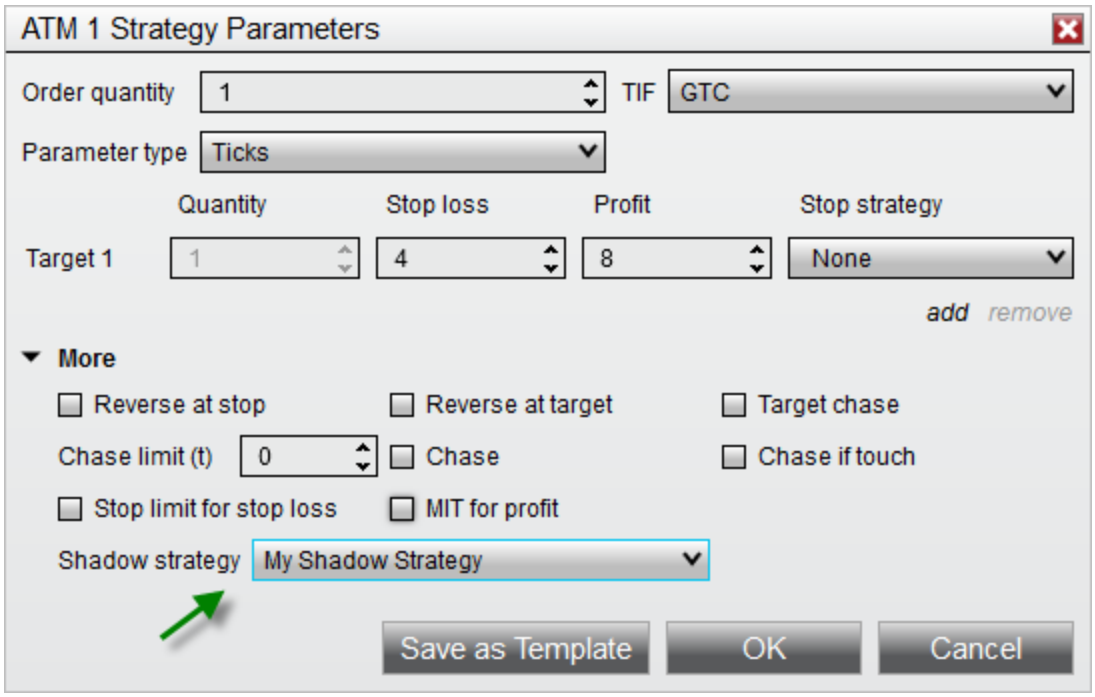


10.1.2.7.3 Shadow Strategy

What is a Shadow Strategy?

Initiating a Shadow Strategy is a method for forward testing alternate trade management ideas. As an example, you may have a method that is profitable, but you have some ideas on how to increase its profitability. Maybe hold on to a few contracts for a higher target? With a Shadow Strategy, you can set up an alternate ATM Strategy and link that to an ATM Strategy that will be used for live trading. Every time you enter a position using your live strategy, NinjaTrader opens a simulated position (e.g. Sim101 account) managed by your Shadow Strategy. This allows you to forward test your concepts using the same entry signals that trigger your live trades. Over time, a historical database of actual (live) and Shadow

(simulated) Strategies are compiled. You can then compare the live trades to the shadow trades under the Performance Tab. The end result is that you will be shown what ATM Strategy (over time) is more profitable. Changing your trade management logic without truly understanding the impact of the changes is a risky shot in the dark. Shadow Strategies give you the proof of concept needed to feel confident that your ATM Strategy changes make sense.



Warning: Shadow strategies can only be used when Multi-provider mode is enabled, so that orders can be submitted to the local simulation account. If Multi-provider is disabled, that Shadow strategy will be set to Disabled and will not function.

Tips

- Intelligently name Shadow Strategies by including a prefix such as "Shadow - My Strategy"
- When using the Performance Tab, you can filter your reports to include or exclude your Shadow Strategy

10.1.2.8 FAQ

Listed below are some common questions concerning building and implementing **ATM Strategies**.

▼ Do I need to turn on OCO order to use the ATM Strategies?

No, the Stop Loss and Profit Target orders submitted automatically through an ATM Strategy are OCO by default meaning that when your target is filled the stop will automatically be cancelled. The OCO function in each of the order entry windows can be used to manually link orders you place.

Please see the Submitting Orders section for more information and examples of the OCO function, or attend one of our free live training events to see further examples.

Does NinjaTrader need to be connected for ATMs to work?

Yes, for an ATM to activate and for it's functions to operate, NinjaTrader needs to be connected.

Is it possible to run concurrent ATM Strategies in the same market and the same account?

Absolutely, NinjaTrader's Strategy Selection Modes allow you to limit the display in the SuperDOM so that you can run concurrent ATM Strategies. One of the great features of NinjaTrader is its ability to manage multiple virtual positions in the same market. For example, this allows you to manage a long and short position in the same market simultaneously.

Here is how this is accomplished:

- Open 2 SuperDOMs and set them both to the same market
- Right click in one of the SuperDOMs and select the menu "**Properties**"
- Set the 'ATM Strategy selection mode' parameter to "DisplaySelectedAtmStrategyOnly"
- Repeat the last two instructions on the second SuperDOM
- Submit a buy order to open a long position in the first SuperDOM
- Submit a sell order to open a short position in the second SuperDOM

For more information please see the ATM Strategy Selection Mode section of the user help guide, or attend one of our free live training events.

Can I have my Auto Trail loosen as I gain ticks in profit?

Your Stop Strategy will never move your Stop Loss backward. The Stop Strategy

will only move your stop closer to the current trading price.

Example:
If the first step of your Auto Trail has the Stop Loss trailing by 5 ticks and then the second step of the Auto Trail tells the Stop Loss to trail by 10 ticks the Stop Loss will simply stay at its current price point until there is a 10 tick spread between the Stop Loss and the current trading price and then begin to trail by 10 ticks. The Stop Loss will not move backwards when the second step of the Auto Trail is activated.

For more information on the Auto Trail feature please see the Auto Trail section of the user help guide or attend one of our free live training events.

Why don't the following ATM Strategy parameters work?

	1 Target	2 Target	3 Target
Qty:	1	1	1
Stop Loss:	10	8	6
Profit Target:	10	8	6

When building an ATM Strategy each Profit Target must be greater than the Profit Target before it. Example: The Profit Target for 1 Target must be less than the Profit Target for 2 Target. Also each Stop Loss must be must be equal to or greater than the Stop Loss before it. Example: The Stop Loss for 1 Target must be equal to or less than the Stop Loss for 2 Target. The Parameters listed below show the correct way to enter the values listed above.

	1 Target	2 Target	3 Target
Qty:	1	1	1
Stop Loss:	6	8	10

Profit Target:	6	8	10

Please see the ATM Strategy Parameters section of the user help guide for further information.

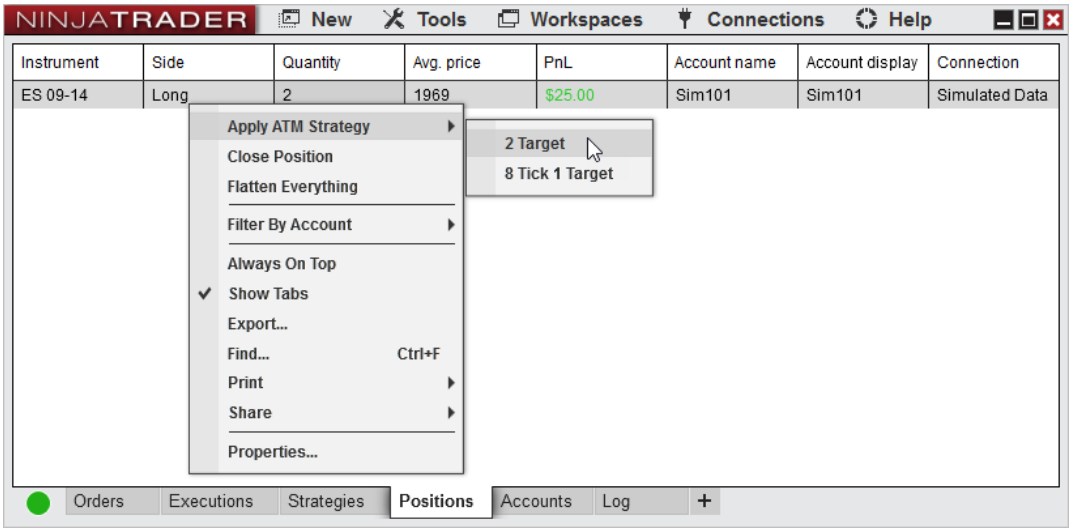
Can I use the Auto Breakeven and Auto Trail strategies together?

Absolutely, NinjaTrader gives you the flexibility to use these strategies alone or to combine them. However, when using these features together please be aware of the following:

- The Stop Strategy will not move your Stop Loss backward it will only move it closer
- The Profit Trigger for your Auto Trail must be higher then the Profit Trigger for your Auto Breakeven

How do I add an ATM Strategy to an open position?

If you have opened a position which is currently unprotected by an **ATM Strategy**, you can easily add an pre-defined ATM Strategy to that position from the Positions tab by right clicking on the instrument row, selecting **Apply ATM Strategy** and selecting the desired pre-defined ATM Strategy Template from this sub-menu.

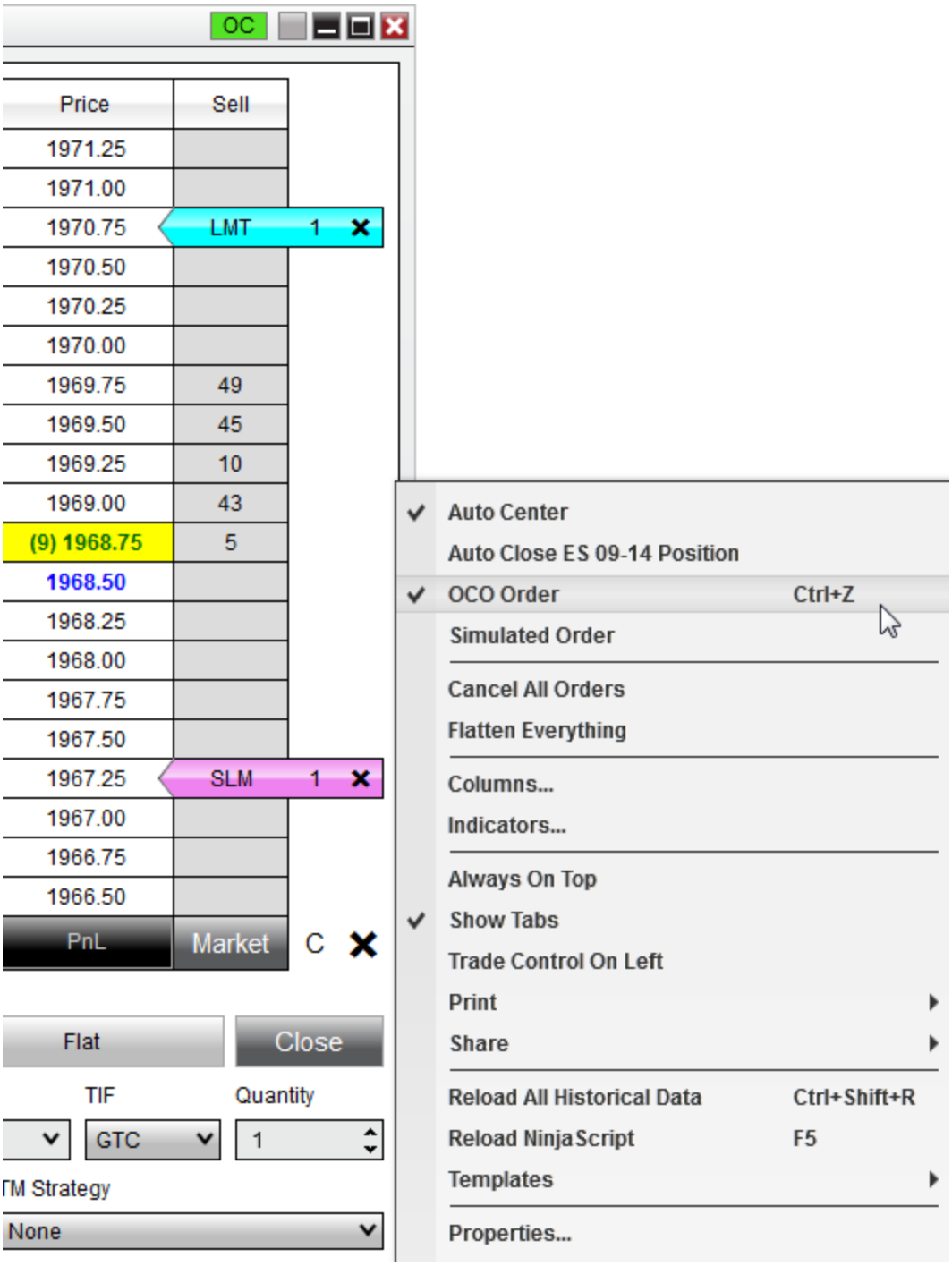


Can I manually bracket a position without using an ATM Strategy?

Of course! You are not required to use a pre-set ATM Strategy if you do not want to. If you have an open position without an ATM Strategy attached, and you wish to add limit and stop orders to protect the position follow these steps:

- Set the ATM strategy in the ATM Strategy selection drop down box to a value of <None>
- Right click in the SuperDOM and enable OCO order placement by selecting the menu name "**OCO Order**"
- Then place a limit order where you want to exit at a profit
- Then place a stop order where you want to exit at a loss
- Lastly, right click again and select the menu item "**OCO Order**" to disable the OCO order placement

Now you have a target and a stop placed protecting your open position, and when one of these orders is filled the other will be cancelled automatically.



How do I make one target a "runner" so that it has a Stop Loss only and no Profit Target?

If you want a target in your ATM Strategy to have a Stop Loss only then set the

Profit Target to zero.

What happens as one of my ATM orders are rejected?

If an ATM Strategy Entry is rejected, the stops and targets for the Entry will not be entered.
If a Stop Loss or Profit Target order is rejected, all Stop Losses and Profit Targets for the ATM will be canceled. This includes all Stop Losses and Profit Targets to an ATM you are scaling into.

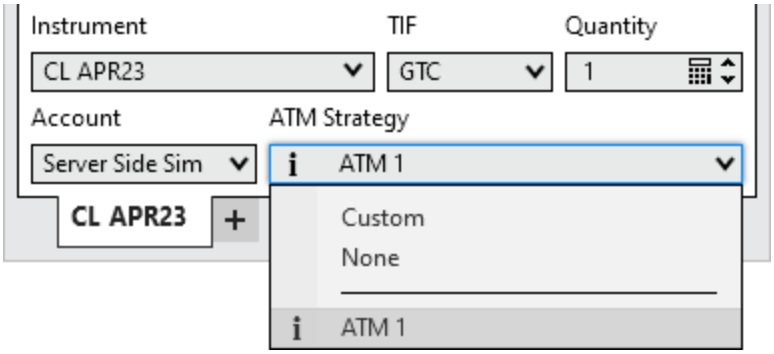
10.1.3 Server Side ATMs

Majority of NinjaTrader's order entry interfaces house the same control for defining an ATM Strategy.

Understanding the Server side ATM Strategy control list options

The Strategy Control List

The drop down list shown in the image below is very important to understand as it defines how your orders will be handled once submitted. There are two main categories of options that will be displayed in this drop-down list; **None**, **Custom** or strategy template names. Strategy templates are specific to your account and instrument.



None

When this option is selected, any orders placed in the entry window will not be applied to an active ATM Strategy nor will it initiate a new ATM strategy.

Custom or ATM Strategy Template Names

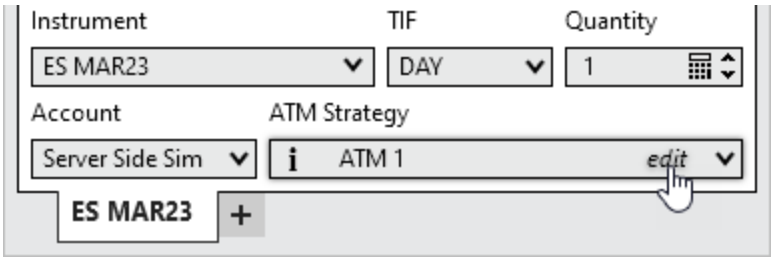
When an ATM Strategy template name is selected, all of the parameters will update to reflect your pre-defined ATM Strategy, or when **Custom** is selected, you have the ability to define a new ATM Strategy on the fly. Once an order is submitted, the ATM Strategy parameters specified will be initiated when the order is partially or completely filled.

> **Note**: If a server side ATM template was created on Web, it may include features not available on Desktop. The template will still submit and function as it was saved.

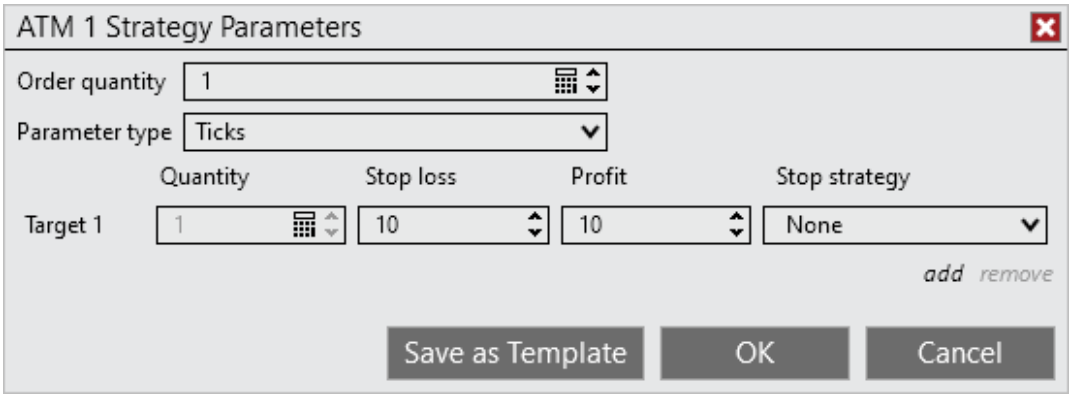
Understanding Server Side Stop Loss and Profit Target parameters

ATM Strategy Parameters

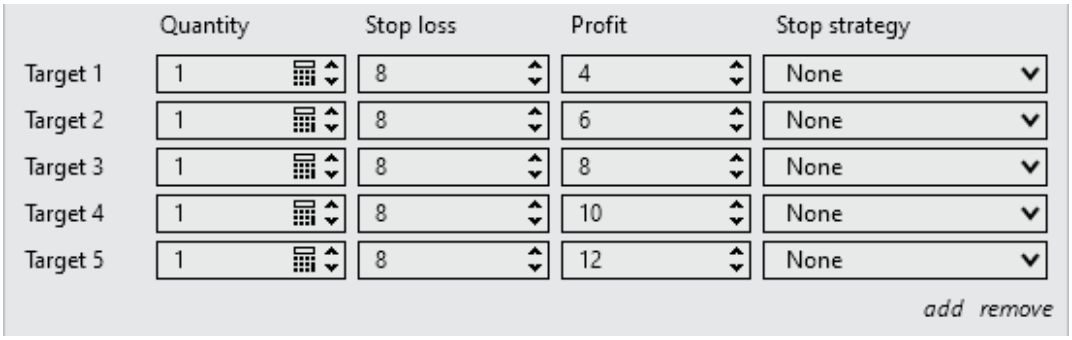
Select **Custom** to define a new ATM Strategy or select the saved **ATM Strategy** template and select "edit" in the **ATM Strategy** combo box as seen below.



In the image below there are parameters that define the **ATM Strategy**. This strategy is a single quantity strategy that will automatically place its target at 10 ticks above the average entry price and stop loss 10 ticks below.



Selecting the "***add***" will allow you to configure additional **Targets** for your **ATM Strategy**. You can add as many targets as you desire. Selecting "***remove***" will reduce the number of configured **Targets** that are configured.



Order quantity	Replicated from the order entry display and sets the initial quantity used for the entry order.
TIF (Time In Force)	Replicated from the order entry display and sets the TIF used for entry, profit target, and stop loss orders.
Parameter type	Sets the type of parameter to display the ATM values in. All Parameter types function the same Ticks — Ticks away from the average entry.

	<table><tr><td>Delta Price</td><td>Price away from the average entry, based on the displayed price of the instrument. AKA Points away from the average entry.</td></tr><tr><td>$ Value</td><td>Cash value away from the average entry, based on to Tick value of the instrument.</td></tr></table> Example: Long 1 contract on the ES at 4,000.00. If you wanted your Profit target to be at 4,001.00 you would use: Ticks - 4, Delta Price 1.00, or $ Value - 50.00
Quantity	Sets the quantity for the Stop Loss and Profit Target orders for this target
Stop Loss	Sets the value that determines the Stop Loss price. If the value is set to 4 (ticks) and your average entry for the initiating order is 1000 and you are long, your Stop Loss would be submitted at AvgEntry - Stop Loss = 1000 - 4 ticks = stop price of 999. This assumes each tick is valued at 0.25.
Profit	Sets the value that determines the Profit target price. If the value is set to 4 (ticks) and your average entry for initiating the order is 1000 and you are long, your Profit target would be submitted at AvgEntry + Profit target = 1000 + 4 ticks = 1001 Profit target. This assumes that each tick is valued at 0.25.
Stop Strategy	Sets the Stop Strategy

For further reference, please look at the Server Side Strategy Examples located within the "Server Side ATM Strategy" page.

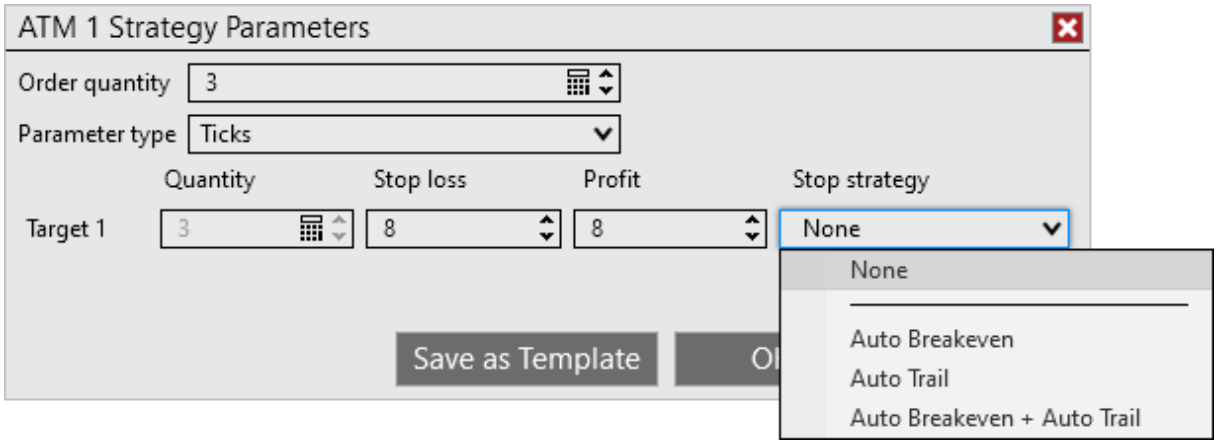
10.1.3.1 Server Side Stop Strategy

ATM Stop Strategies

ATM Stop Strategies provide additional functionality for the stop losses placed by an ATM Strategy, including Auto Breakeven, Auto Trail, or using both with Auto Breakeven + Auto Trail

A Stop Strategy is an extension of an ATM Strategy. It allows you to combineAuto Breakeven, Auto Trail, or using both with Auto Breakeven + Auto Trail for the management and automatic adjustment of your Stop Loss orders.

When setting up an ATM Strategy, you can select either **None**, **Auto Breakeven**, **Auto Trail**, or **Auto Breakeven + Auto Trail** before entering a position as part of a stop strategy.



When selecting **Auto Breakeven**, **Auto Trail**, or **Auto Breakeven + Auto Trail** the related Stop Strategy Dialog window will appear.

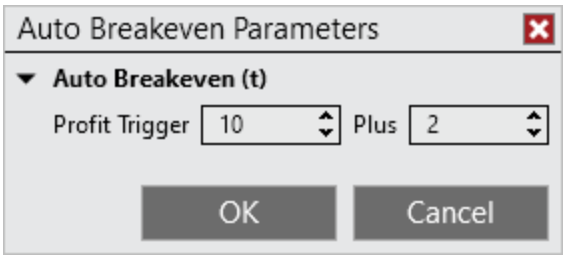
Understanding the Server Side Auto Breakeven parameters

The Auto Breakeven feature will adjust your Stop Loss order to breakeven (average entry price for the ATM Strategy position) once a user defined Profit Trigger has been reached.

Auto Breakeven Parameters

Profit Trigger	Sets the amount of profit required to move the Stop Loss to a breakeven value
Plus	Sets the amount added to the breakeven (average entry price for the ATM Strategy position) value

Auto Breakeven Example



- Profit Trigger - 10 ticks
- Plus - 2 ticks

Average Entry - 10200 Short (DOW Emini contract)
As soon as the market trades at 10190 (Average Entry - Profit Trigger = 10200 - 10 ticks = 10190) NinjaTrader will move the Stop Loss order to 10,198 (Average Entry - Plus = 10200 - 2 ticks = 10198) and enter a log event in the Log tab.

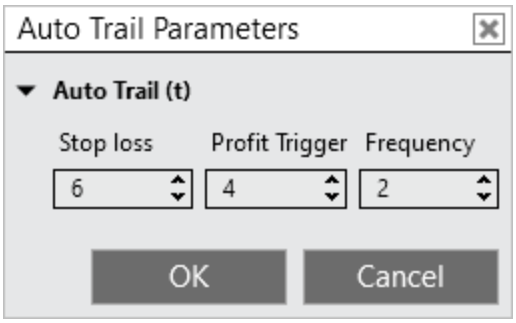
Understanding the server side Auto Trail parameters

Auto Trail is a powerful stop strategy that allows you to be more liberal with your Stop Loss at the early stage of your trade and tighten your Stop Loss as your profits in your trade increase.

Auto Trail Parameters

Stop Loss	Sets the value of the Stop Loss order as an offset behind the Profit Trigger/Frequency
Profit Trigger	Sets the amount of profit required to trigger the initial Stop Loss adjustment for the step
Frequency	Sets the value of how frequent the Stop Loss order is adjusted after the Profit Trigger

Auto Trail Example:



- Stop loss - 6 ticks
- Profit Trigger - 4 ticks
- Frequency - 2 ticks

Your Stop loss will move 6 ticks behind the current price...
... after the price has moved 4 ticks in your favor...
... and your Stop loss will continue to trail 6 ticks behind the current price with ever additional 2 ticks in your favor.

Average Entry - 1000 Long (SP Emini contract)
The market moves up to 1001 and the Auto Trail is triggered **(Average Entry + Profit Trigger = 1000 + 4 ticks = 1001)** and the Stop Loss is adjusted to 999.50 **(1001 - Stop Loss = 1001 - 6 ticks = 999.50)**. For every additional 2 ticks **(Frequency of 2 ticks)** the Stop Loss will be adjusted by 2 ticks.

Understanding the server side Auto Breakeven + Auto Trail parameters

Auto Trail is a powerful stop strategy that allows you to be more liberal with your Stop Loss at the early stage of your trade and tighten your Stop Loss as your profits in your trade increase.

Auto Breakeven Parameters

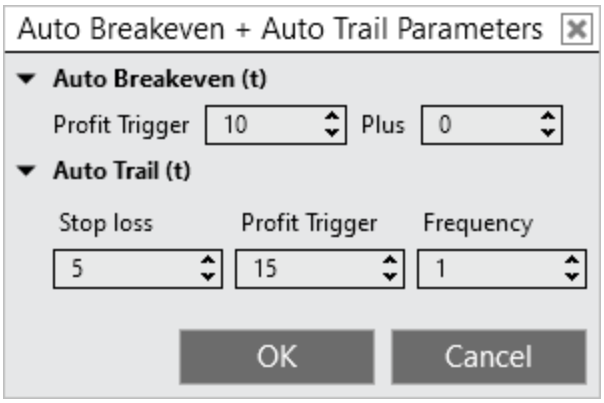
Profit Trigger	Sets the amount of profit required to move the Stop Loss to a breakeven value
Plus	Sets the amount added to the breakeven (average entry price for the ATM Strategy

	position) value

Auto Trail Parameters

Stop Loss	Sets the value of the Stop Loss order as an offset behind the Profit Trigger/Frequency
Profit Trigger	Sets the amount of profit required to trigger the initial Stop Loss adjustment for the step
Frequency	Sets the value of how frequent the Stop Loss order is adjusted after the Profit Trigger

Auto Breakeven + Auto Trail Example:



Auto Breakeven
- Profit Trigger - 10 ticks
- Plus - 2 ticks

Auto Trail
- Profit Trigger - 4 ticks
- Frequency - 2 ticks

Average Entry - 1000 Long (SP Emini contract)
The market moves up to 1002.50 and the Auto Breakeven Profit Trigger is activated **(Average Entry + Auto Breakeven Profit Trigger = 1000 + 4 ticks = 1002.50)** and the Auto Breakeven Plus adjusts the Stop Loss to 1000 **(Average Entry + Auto Breakeven Plus = 1000 + 0 = 1000)**.

Next, the market moved up to 1003.75 and the Auto Trail Profit Trigger is activated **(Average Entry + Auto Trail Profit Trigger = 1000 + 15 ticks = 1003.75)** and the Stop loss is adjusted to 1002.50 **(1003.75 - Stop loss = 1003.75 - 5 ticks = 1002.50)**. For every additional tick **(Frequency of 1)** the Stop loss will be adjusted by 1 tick.

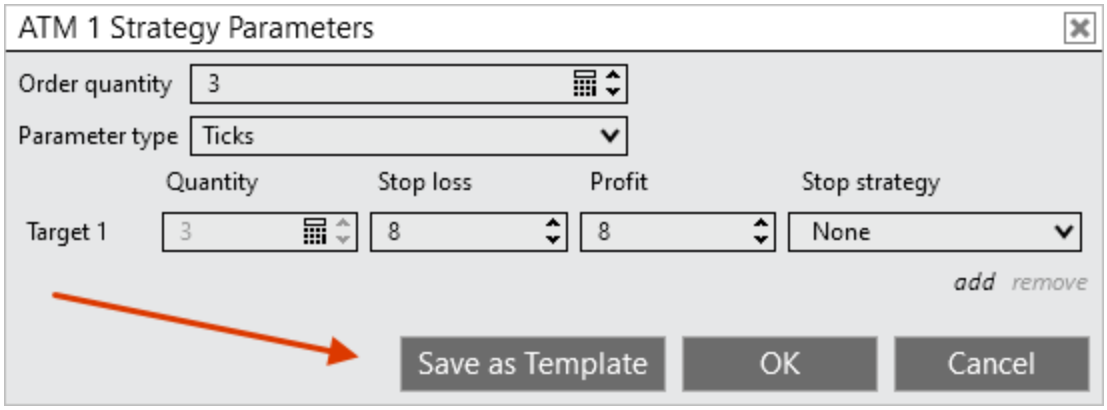
10.1.3.2 Manage Server Side ATM Templates

An ATM Strategy is defined by the parameters you enter into the ATM Strategy parameters section on any of the order entry screens. The collection of parameters that make up a strategy can be saved as a template that you can recall at a later date to automatically populate all of the ATM Strategy parameters. Templates are specific to your account and the selected instrument.

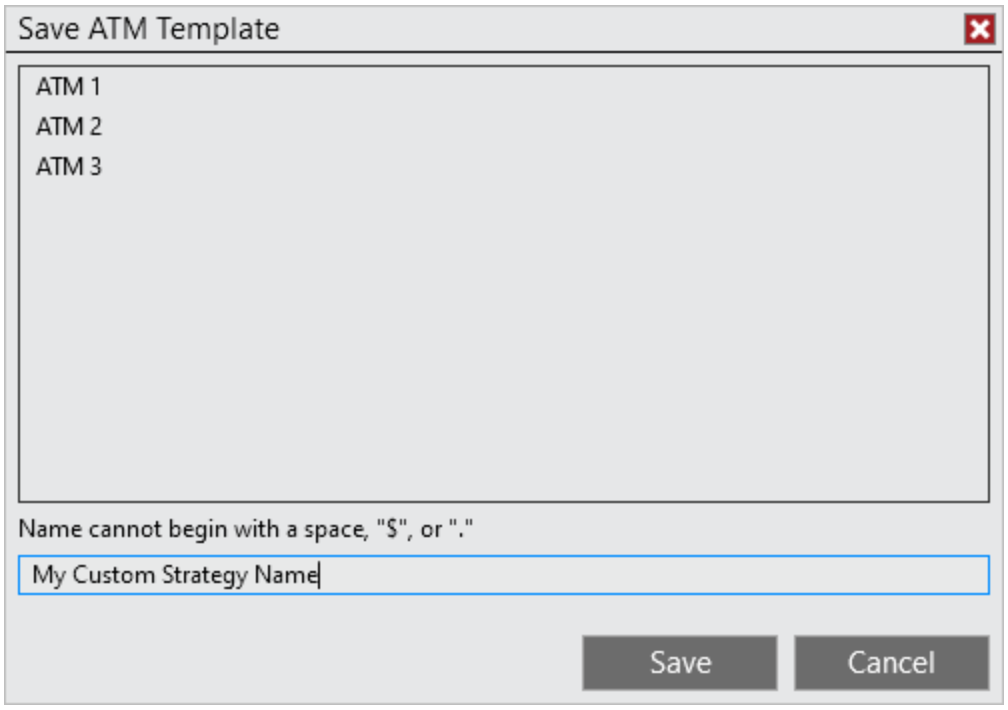
Saving ATM Strategy Templates

To save your current **ATM Strategy parameters** in a template:

Select the **Save as Template** button

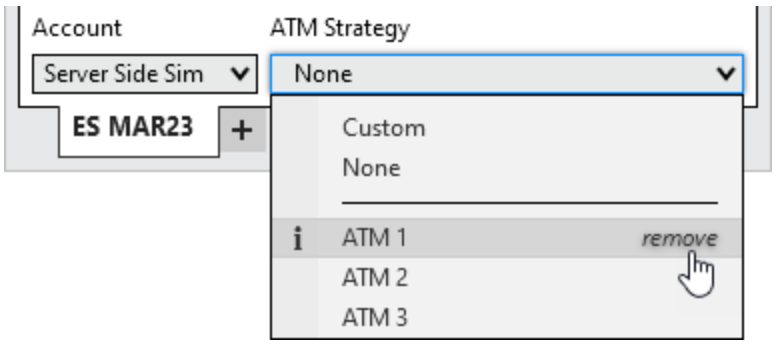


From the presented file dialog give the template a custom name, then press Save.



Removing an ATM Strategy Templates

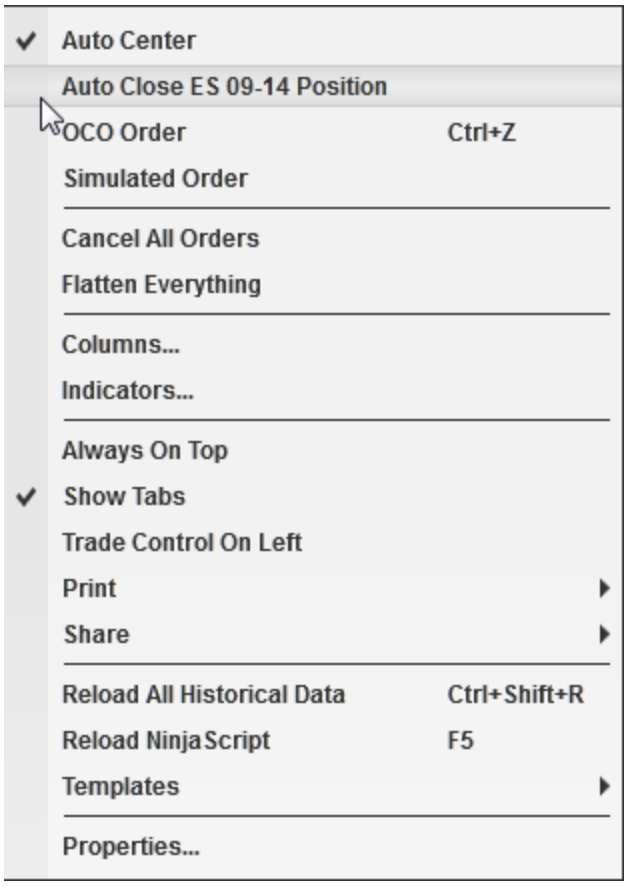
Within the **ATM Strategy** dropdown menu, hover your mouse over the ATM Strategy you want to remove, then select **remove**.



10.1.4 Auto Close Position

Automatically Closing Positions at a Specific Time

Auto Close Position is a strategy that will automatically close your position at an user defined time. A position will be closed using the NinjaTrader close algorithm. The user defined close time can be set via the "Auto Close Position - Time" property located in the Trading category of the General Options menu. You can enable or disable this strategy via any NinjaTrader order entry screen's right mouse click context menu.



Notes:
This feature not available to Direct Edition license users and will be disabled. Please contact platformsales@ninjatrader.com for upgrade options.
If auto close is configured for all instruments in the **Options**, disabling **Auto close** on an entry window will have no affect since it is globally set to close for all instruments.

10.2 Alerts

Alerts Overview

Alerts allow you to define custom triggers based on various conditions in the Market Analyzer, Hot List Analyzer, or Charts. Unique and complex conditions can be built around existing market data components, indicators, or drawing tools. The configuration of an **Alert** in NinjaTrader are completely achieved through a point and click interface, requiring no programming experience of any kind!

10.2.1 Using Alerts

What can an alert do?

When an **alert condition** is triggered, you can define exactly how the **alert** behaves allowing you to:

- Display a custom message on the Alerts Log
- Play a sound
- Share content to a specific Sharing Service
- Display a Pop Up Dialog with a custom message
- Submit a custom order*

Notes:
1) While an **alert** will give you the ability to **submit custom orders**, they are natively limited in the type of account and order management information that is available. If you are interested in developing a more complex system for an automated trading approach, please see our Help Guide articles on developing an Automated NinjaScript Strategy.
2) **Alerts** are not intended to be used with Playback while using an increased speed. **Alerts** are checked every 200 milliseconds, so using them in Playback could result in an alert not triggering.

What kind of information can be used for an alert?

Alerts will always work in real-time, giving you access to a wide variety of information you have currently setup in your workspace. However, the type of information that is available for

an **alert** will depend on where the **alert** was setup. For example, a **Market Analyzer** is real-time only and *does not* display historical values, therefore it *would not* be possible to create an **alert** based on a historical bar or indicator value. In contrast, a **Chart** *does* display historical bar data, therefore a chart **alert** *would* be able to use historical bar data to be considering in a specific **alert** condition used for indicators and other data series.

Chart Alerts

- Access to historical data allowing you to compare real-time indicator and data series values to previous values (bars ago)
- Manually configured chart objects and drawing tools
- Multiple data series such as additional instruments and time frames
- Ability to make time comparisons

Market / Hot list Analyzer Alerts

- Real-time data only
- Fundamental data such as Earnings Per Share, 52 week high/low, Settlement Price, etc can be used
- Access to account information such as instrument and account position information such Realized/Unrealized PnL, Position size, Position avg price, etc.

10.2.2 Alerts Dialog

The **Alerts Dialog** will list any alerts that are currently configured for the window the dialog was launched from as well as allow you to configure any running alerts. Alerts run on a per tab basis for each window. This means if you have two charts open in your workspace, the alerts dialog will only list alerts running from that specific tab.

▼ Accessing the alerts dialog

Accessing Alerts from the Market Analyzer

- Right click on a **Market Analyzer**
- Select **Alerts**

An **Alert Dialog** launched from the **Market Analyzer** will list any columns you have configured on your Market Analyzer to be used as alert condition objects.

Accessing Alerts from a Chart

- Right click on a **Chart**
- Select **Alerts**

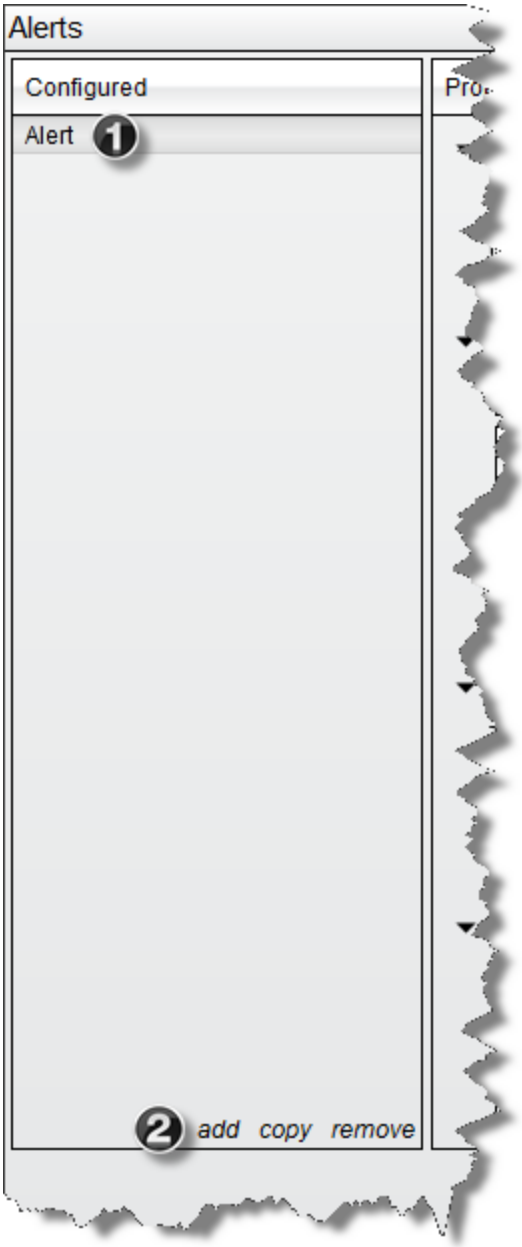
An **Alert Dialog** launched from the **Chart** will list any chart objects (data series, indicators, drawing tools) you have configured on your Chart to be used as alert condition objects.

Understanding the alerts dialog

The **Alerts Dialog** will list any configured alerts for the current tab as well as allow you to configure the alerts listed.

Configure Panel

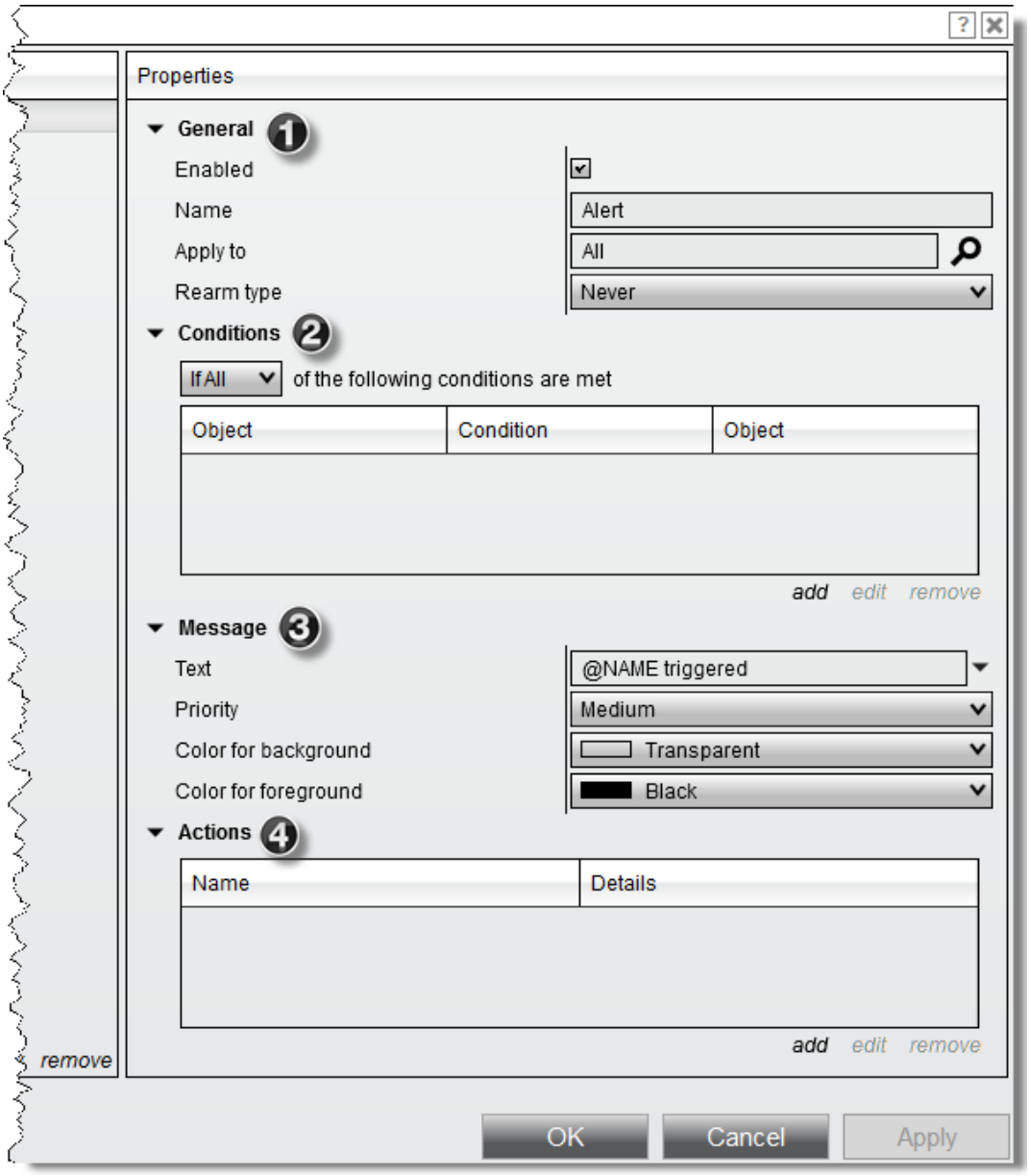
1. List of any configured alerts
2. Menu to add, remove or copy alerts



Properties Panel

The **properties panel** will allow you to define and modify each configured alert. For information specific to customizing an alert property, please see our help guide article on Configuring Alerts

1. General	Define general alert settings
2. Conditions	Define alert conditions to monitor
3. Message	Define alert message details generated when alert condition is triggered
4. Actions	Define alert actions to take. Possible actions are: • Play sound • Share • Pop Up Dialog • Submit an order



10.2.3 Configuring Alerts

Alerts can be created using conditions which monitor various "objects" which exist on the chart, or market analyzer display. Possible **Condition Objects** include a chart's data series, indicators, drawing tools, or any Market Analyzer column value.

Play Video

How to add a condition object

Adding an alert from the alert dialog

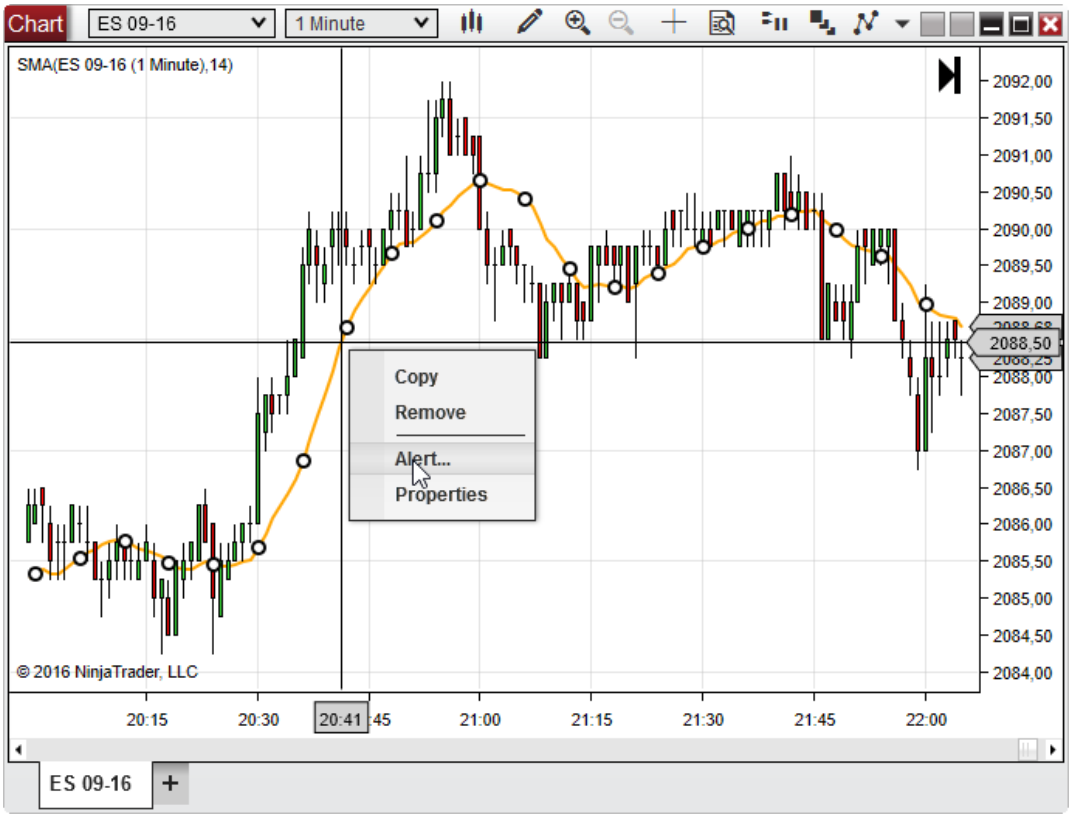
You can create a new generic alert by first accessing the Alerts Dialog window, and selecting the **"add"** text which will add a new alert to your configured alerts panel.

Any suitable object which currently exists on the window or tab will be available to use as a **Condition Object** for the alert. For example, if you have a **Market Analyzer** with several customized columns added, you will be able to use any of those columns as a **Condition Object**. A **Chart** will work the same way in that

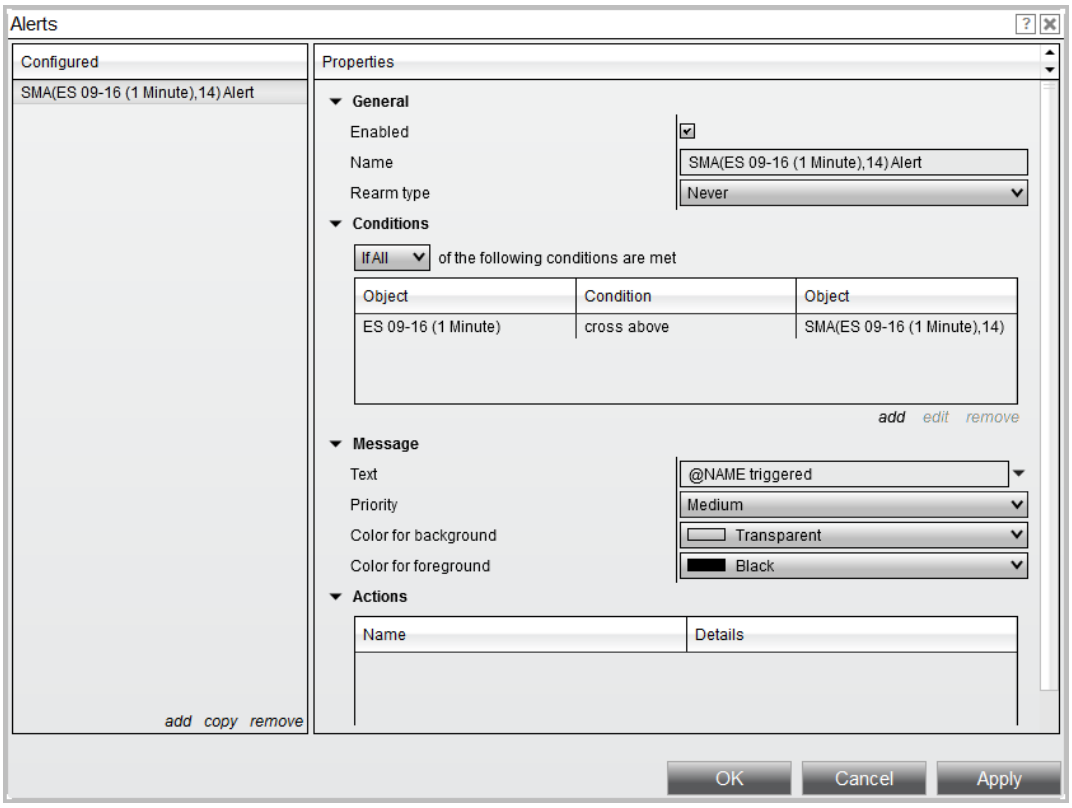
any data series, indicator, or drawing object that currently exists on the chart will be available as a **Condition Object**.

Creating an alert from a chart indicator

If you have an indicator configured on an existing chart you wish to use in your alert condition, you can easily access this by first left mouse clicking on the indicator plot to select the indicator, and then selecting **Alert**

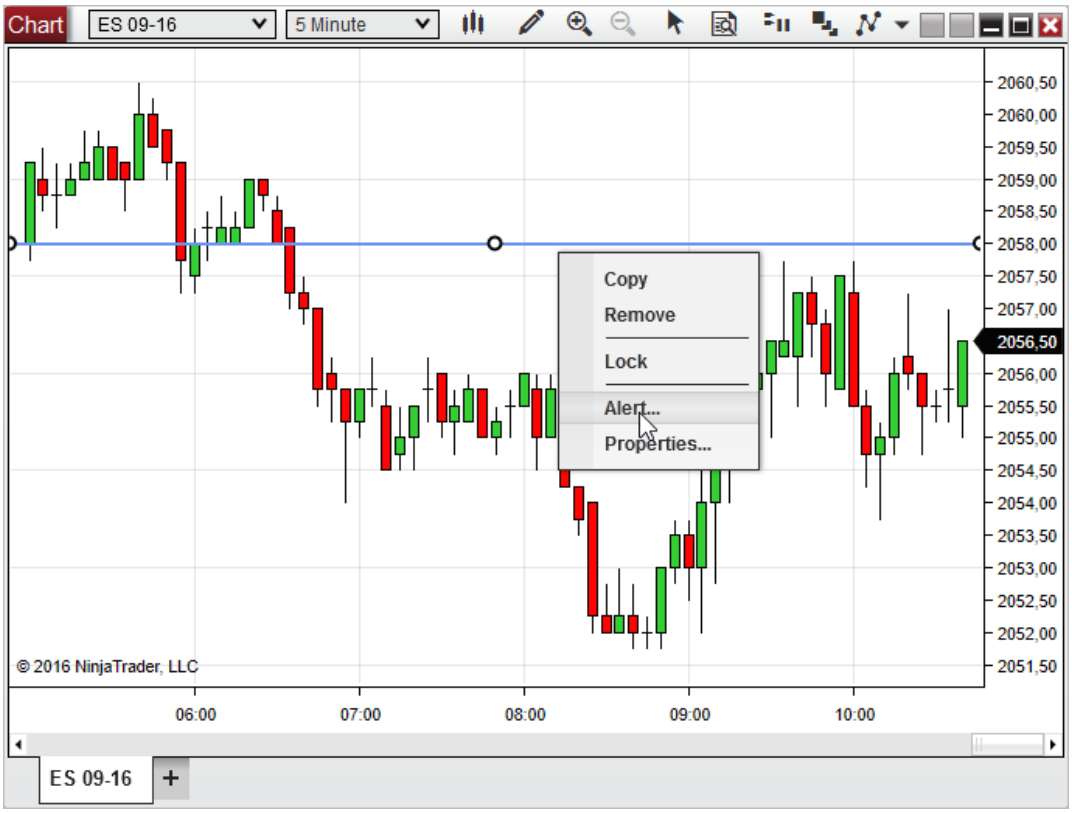


Doing so will automatically add the selected indicator as an object that is used in the **Conditions** properties.

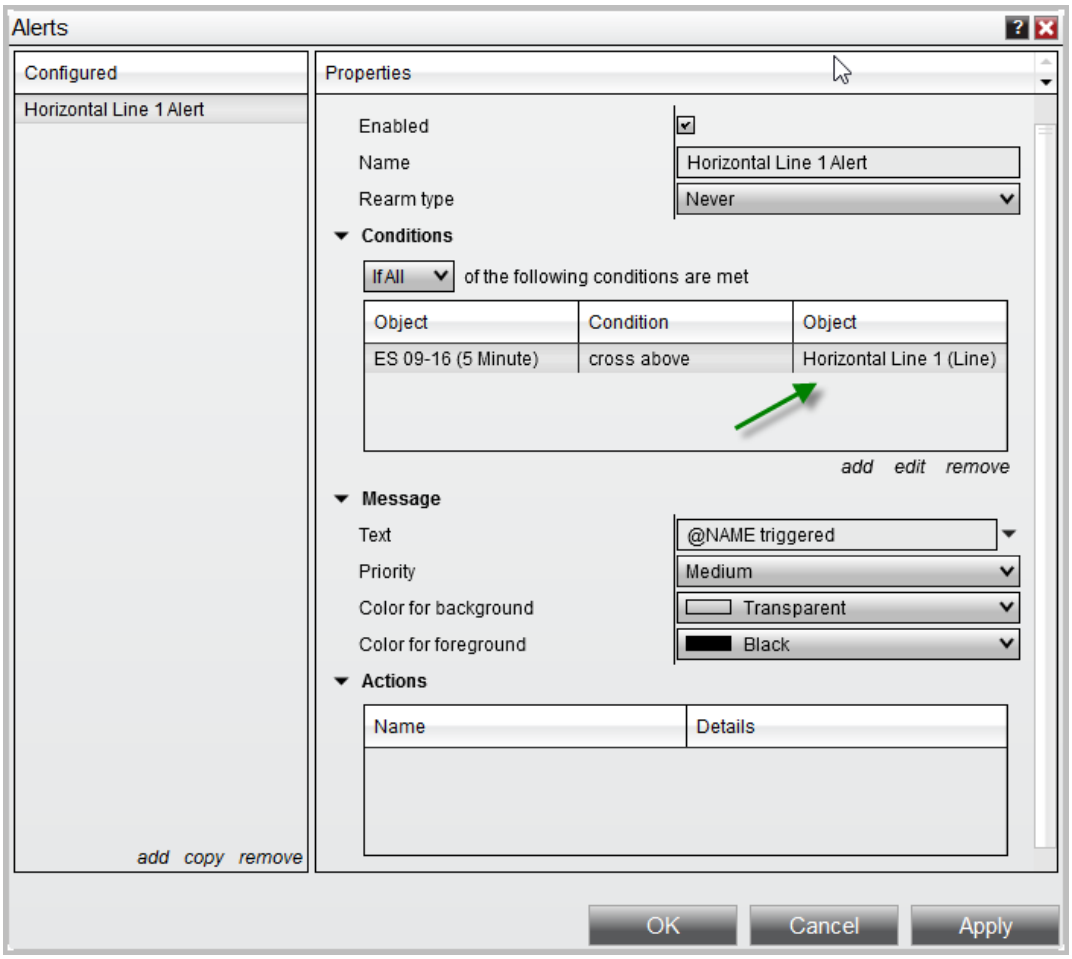


Adding an alert from a chart drawing object

Drawing tools which exist on a chart can also be added as a condition object by first left clicking on the drawing tool, then selecting **Alert**.



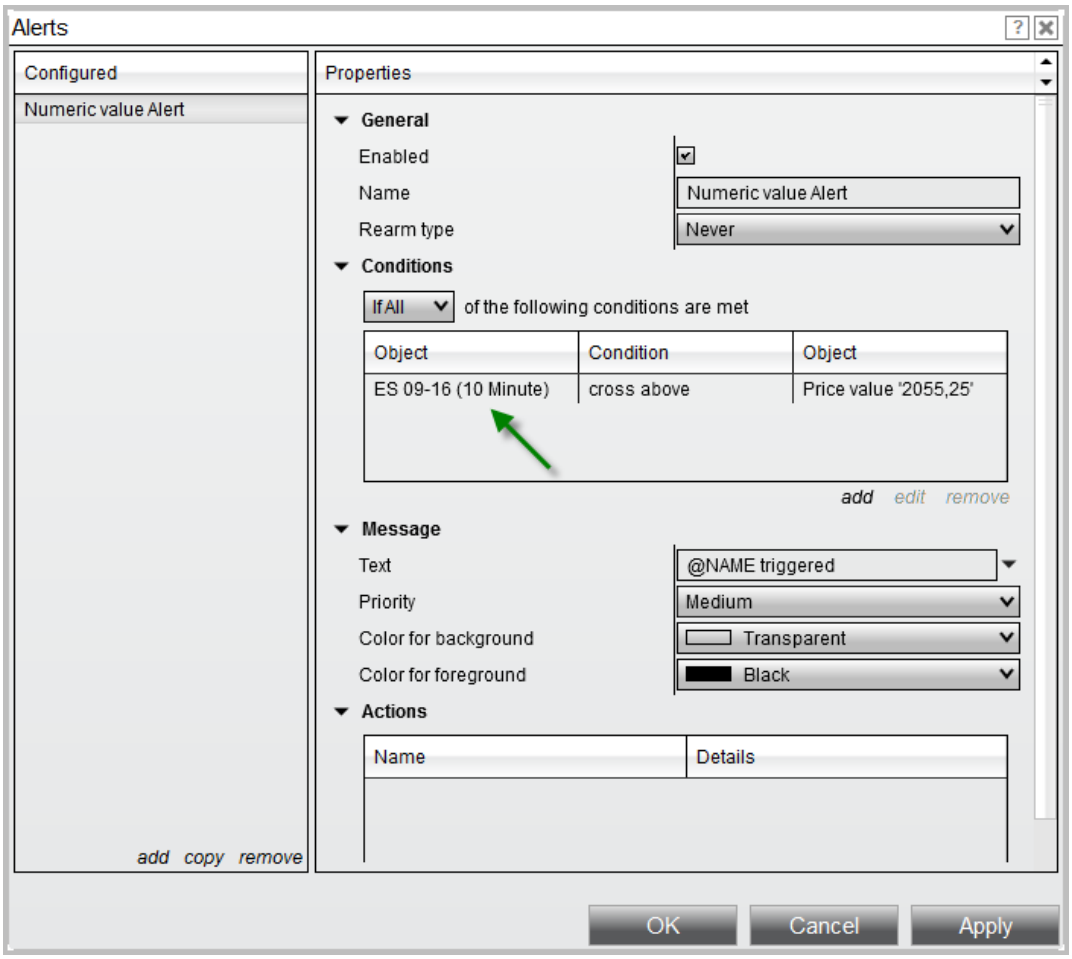
Doing so will automatically add the selected drawing tool as an object that is used in the **Conditions** properties.



Adding an alert from a chart data series

If you have multiple data series on your chart (e.g., 5 minute and 10 minute data series), you can select one of these series to be used as an condition object. Simply left mouse click on the chart data series itself, and select **Alert**

Doing so will automatically add the selected data series as an object that is used in the **Conditions** properties.



Understand the general alert properties

General Alert Properties

The **General** section allow you to configure the following alert properties:

Enabled	When checked, the current alert will be active
Name	Sets the displayed name of the alert
Apply to	Determines which instrument(s) are used to be monitored by the alert. This property is just for the Market Analyzer and Hot List Analyzer.

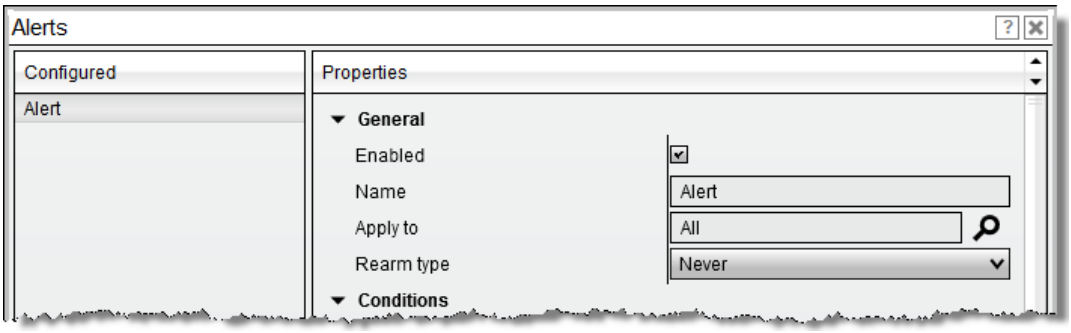
<table>
<tr><td>Rearm type</td><td>Sets under what condition the alert will rearm. Possible options are listed in the table below:

<table>
<tr><td>Never</td><td>The alert will only trigger once and never rearm</td></tr>
<tr><td>On timer</td><td>Rearm after a specific number of seconds defined in the rearm seconds property</td></tr>
<tr><td>On bar close</td><td>Rearm on the next bar close for the data series used in the alert condition. Only columns which have a data series are eligible for OnBarClose alerts.</td></tr>
<tr><td>On condition reversed</td><td>Rearm once the condition is no longer true</td></tr>
<tr><td>On connect*</td><td>Rearm after NinjaTrader has been manually connected to a data provider</td></tr>
<tr><td>Remove</td><td>Once triggered the alert will never rearm and be removed.</td></tr>
</table></td></tr>
<tr><td>Rearm seconds</td><td>Sets the number of seconds an alert will rearm. If the same alert is called within a time window of the time of last alert Rearm seconds, the alert will be ignored (only visible if Rearm type option 'On timer' is selected)</td></tr>
</table>

*The **On connect** rearm type is aware of 4 different types of connection events which will define how the **On connect** rearm type behaves:

Event	Description	Rearm behavior

Manual disconnect	When user has selected to disconnect from the data provider, or shut down NinjaTrader	Alert will be disabled
Manual reconnect	When a user has selected to connect to a data provider	Alert will be rearmed
Disconnect	When the data provider has been temporarily disconnected due to connectivity issues	Alert will just stay in active state and will wait for reconnect.
Reconnect	When the data provider connection has recovered from the lost connection	Alert will resume in active state



Applying alerts to specific instruments

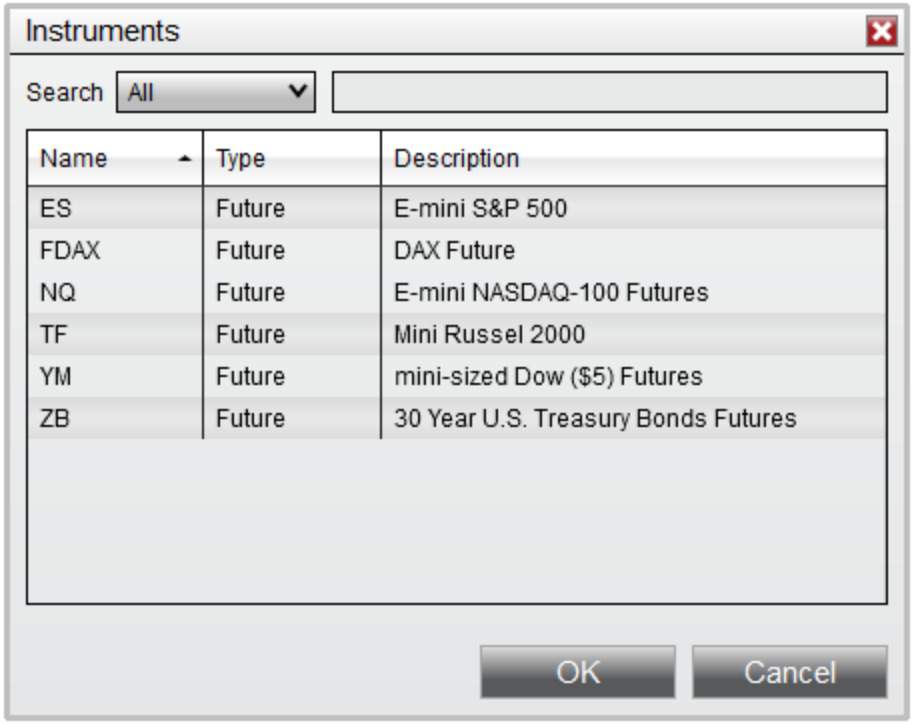
If you are using a window which has multiple instruments, such as a **Market Analyzer**, or a **Chart** with multiple data series, the default behavior will be monitor "**All**" instruments contained in that window tab.

By selecting the **Magnify glass** icon next to the **Apply to** property, a new window will appear which will list all of the configured instruments and allow you to select a specific set of instruments to be monitored by the alert:

1. From the newly opened **Instruments window**, select the instruments you wish to apply the condition to

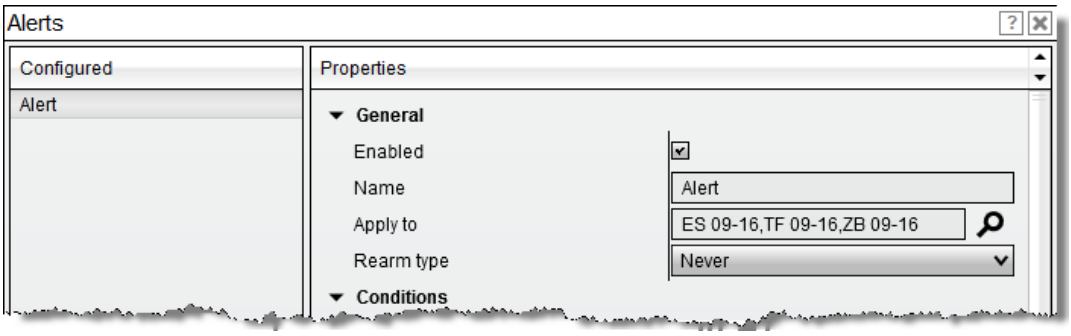
Tip: Multi-select is supported in the Instrument window:

- To select a consecutive instruments, click the first instrument, press and hold down the Shift key, and then click the last instrument.
- To select non-consecutive instruments, press and hold down the Ctrl key, and then click each instrument that you want to select.



2. Press ***OK*** on the **Instruments window**

3. Your **Apply to** field will now list the instrument names you selected earlier, indicating that alerts will only be triggered on instruments contained in this list.



Tip: You can also simply type in the Apply to field to add a specific instrument, or use your backspace key to delete a specific instrument

Understanding the conditions properties

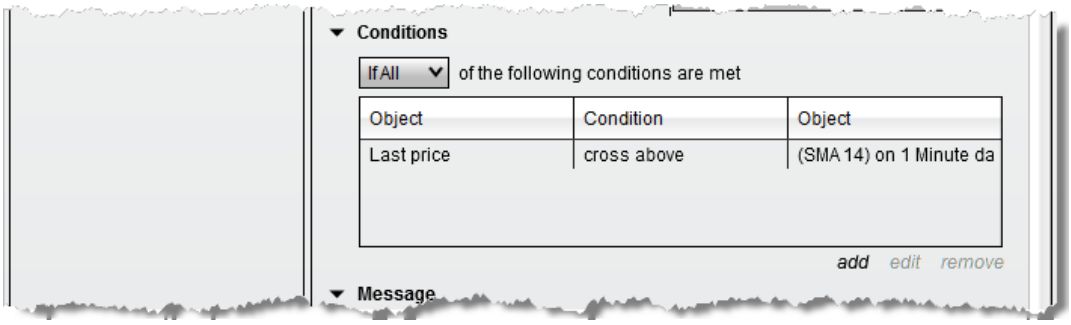
Alert Conditions

The **Alert Conditions** allow you to define the exactly what the alert will monitor.

1. Set Conditions to match "If Any" or "If All" of the following conditions are met:

If Any	Alert if any of the listed conditions are true
If All	Alert only if all of the listed conditions are true

2. A list of the current objects and conditions to monitor
3. Add a new condition, or edit and remove existing conditions



Managing Alert Conditions

To define a new alert condition, select the ***"add"*** text which will open a **Condition Builder** window where you can specify exactly which condition to monitor. Please see our Help Guide article on the Condition Builder for information on defining an alert condition.

The ***"edit"*** text will allow you to edit a selected condition.

Selecting ***"remove"*** will remove the selected condition.

Understanding the alert message properties

Alert Message

The alert message properties allow you to define general settings for how the alert is treated when the condition is satisfied. All alerts that are generated are sent to to the Alerts Log window and will display the message you configured in this section. You will also be able to control the priority of the alert, as well as background and foreground colors used.

Message	The text that is displayed in the Alerts Log window. There are a number of keywords which begin with an "@" symbol which will work as variables to fill in information related to the alert, such as the Instrument, Time, Price, etc.
Priority	The priority of the alert for filtering and sorting in the Alerts Log window
Color for background	The color used for the grid background in the **Alerts Log**
Color for foreground	The text color used in the **Alerts Log**

Understanding the action properties

Actions

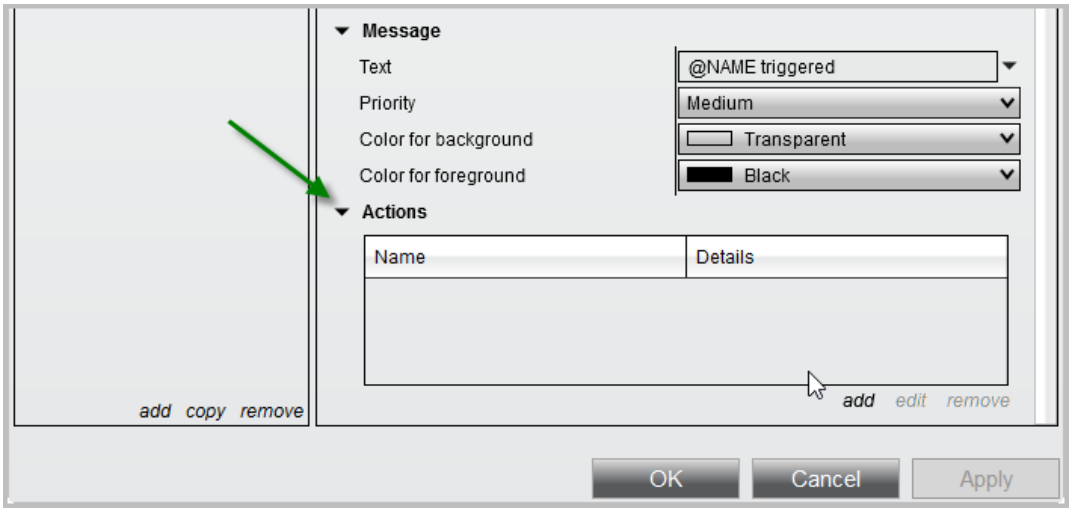
When an alert has been triggered, there are a number of customizable actions that can be taken at that time. These actions include the following:

Play Sound	Play a user defined sound file. Sounds can include system default sounds, or custom sound files
Share	Share a message or screenshot to a selected Sharing Service

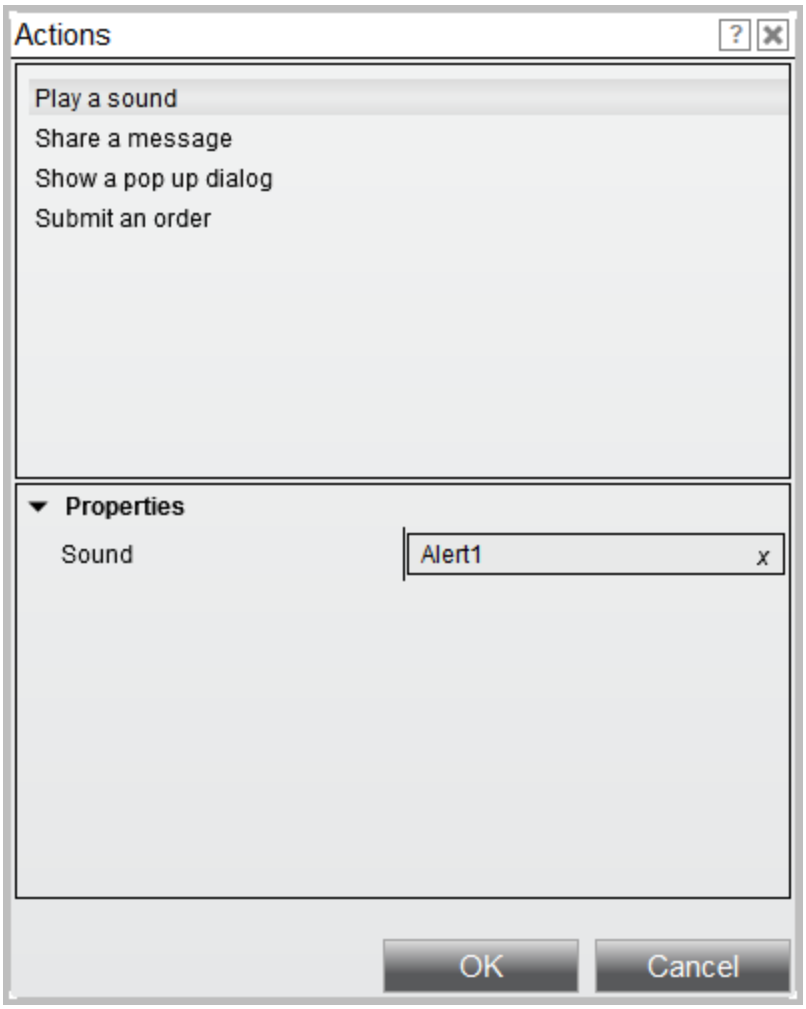
Pop Up Dialog	Display a Pop Up Dialog with a custom message
Submit Order	Submit a custom order

Configuring Actions

To access these actions, you will need to make sure the **Actions** group is expanded by selecting the arrow next to this field in the Alerts properties menu as per the screen shot below:

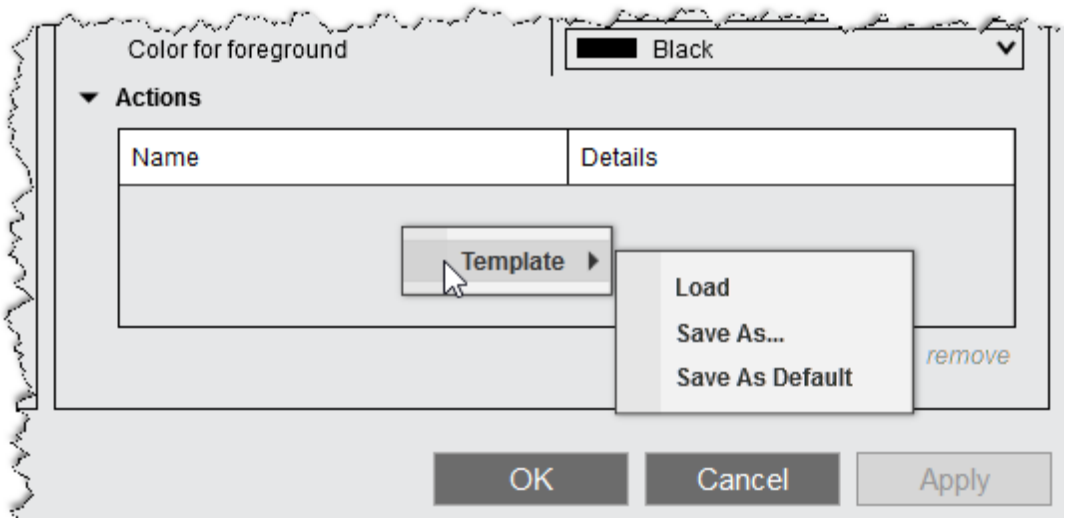


Selecting the "***add***" text will open the Actions window where you can define the custom actions. You can setup as many custom actions as you would like. This means you can have an alert do more than one custom action as you want. For example, you can set an alert to Play a Sound and Share a message to a Sharing Service under he same condition.



Working with Actions Templates

After configuring Actions, you can right click within the Actions columns, select **Template**, and **Save As** to create a template. Within the right click menu is also where you would be able to load any saved templates



Actions Property Definitions

PlaySound	
Sound	Sets the location of the sound file to be played
Share	
Message	Sets the text that is sent to the sharing service
Screenshot type	Optional image to be attached with the message. Possible screen shot types are: • None - No screen shot is included with the message • Chart - A screen shot of the chart that generated the alert will be sent • Tab - Screenshots are taken from the currently active tab. Therefore changing the tab will result in a screenshot of the current tab and not the original. • Window - A screen shot of the entire window that generated the alert will be sent
Share to	Selects the Sharing Service that the alert message is sent to.

Pop Up Dialog	
	Note: There is no property for this section and is intentionally left blank. The pop up dialog will use the **"Message"** that is configured on the **Alert Message** section of the Alerts window
Submit Order	
Account	Selects the account the order is submitted
Instrument	Selects the instrument to submit the order to. Using @INSTRUMENT will submit to the primary instrument.
Limit Price	Sets the limit price used for the order
Order action	Selects the type of action used. Possible order actions are: • Buy • Buy to cover • Sell • Sell short
Order type	Selects the type of order used. Possible order types are: • Limit • Market • MIT • Stop-Limit • Stop-Market
Quantity	Sets the order quantity that is used for the order
Stop Price	Sets the stop price used for the order

Time in force	Selects the TIF (Time in Force) used for the order. Possible TIF setting are: • Day • GTC • GTD
ATM Strategy	Selects the ATM strategy you would like applied to the order

Notes:
When applying an **ATM Strategy** to a **Submit Order** Alert, the ATM Strategy must be saved.
If the saved ATM Strategy is modified before the alert is triggered, the order will submit with the modified ATM.
If the ATM Strategy is removed before the alert is triggered, no ATM Strategy will be applied to the order and you will receive an error.
All TIF options will display. If the broker does not support the TIF selected, there may be an error.

10.2.4 Condition Builder

The **Condition Builder** is a very powerful feature that allows you to define complex conditions for your alerting systems without having to know how to program. The sections below assume you have read and understood how to configure the alerts dialog and understand how to select a **Condition Object** to be used in the condition. If you have not yet, please be sure to review the material under Configuring Alerts.

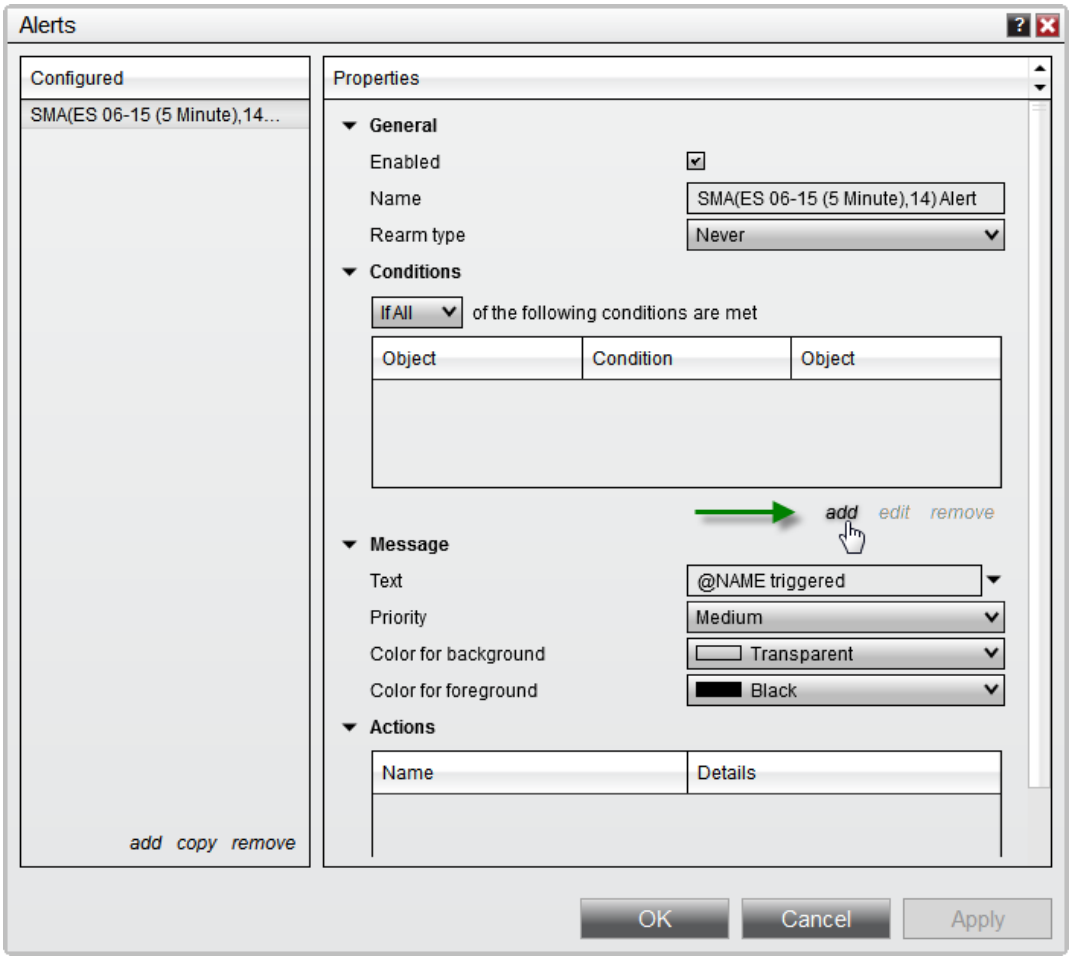
▼ Understanding the Condition Builder

Condition Builder

Most if not all trading system code wizards are limited in scope in that they provide canned pre-defined expressions and only allow you to change a few parameters on those expressions. The NinjaTrader **Condition Builder** is advanced in that you can develop powerful expressions with extensive configurations. Due to its power

and flexibility, it is extremely important that you read through and understand its capabilities.

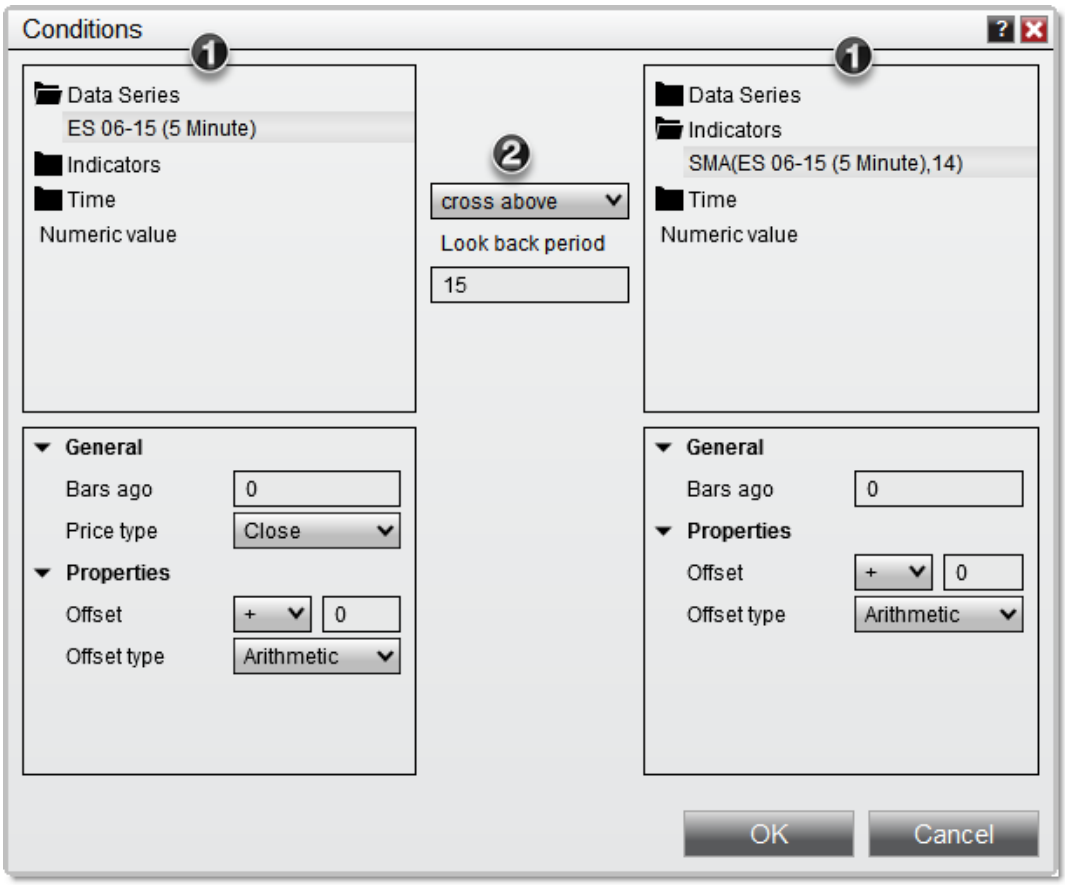
The **Condition Builder** can be accessed via the Alerts Dialog screen by selecting the "***add"*** text



Basic Operation

The general concept of the **Condition Builder** to generate a Boolean expression also known as comparison expressions or conditional expressions. What does that mean? It is simply an expression that results in a value of either TRUE or FALSE. For example, the expression **2 < 7 (2 is less than 7)** is a Boolean expression because the result is TRUE. All expressions that contain relational operators are Boolean. Boolean expressions or "Conditions" as they are known in NinjaTrader is used to determine when to take a specified action such as submitting an order or drawing on the chart.

Looking at the image below, you can instantly see that the **Condition Builder** is set up like a Boolean expression. Select an item from the left window, select the relational operator (2) and compare it to a selected item in the right window.

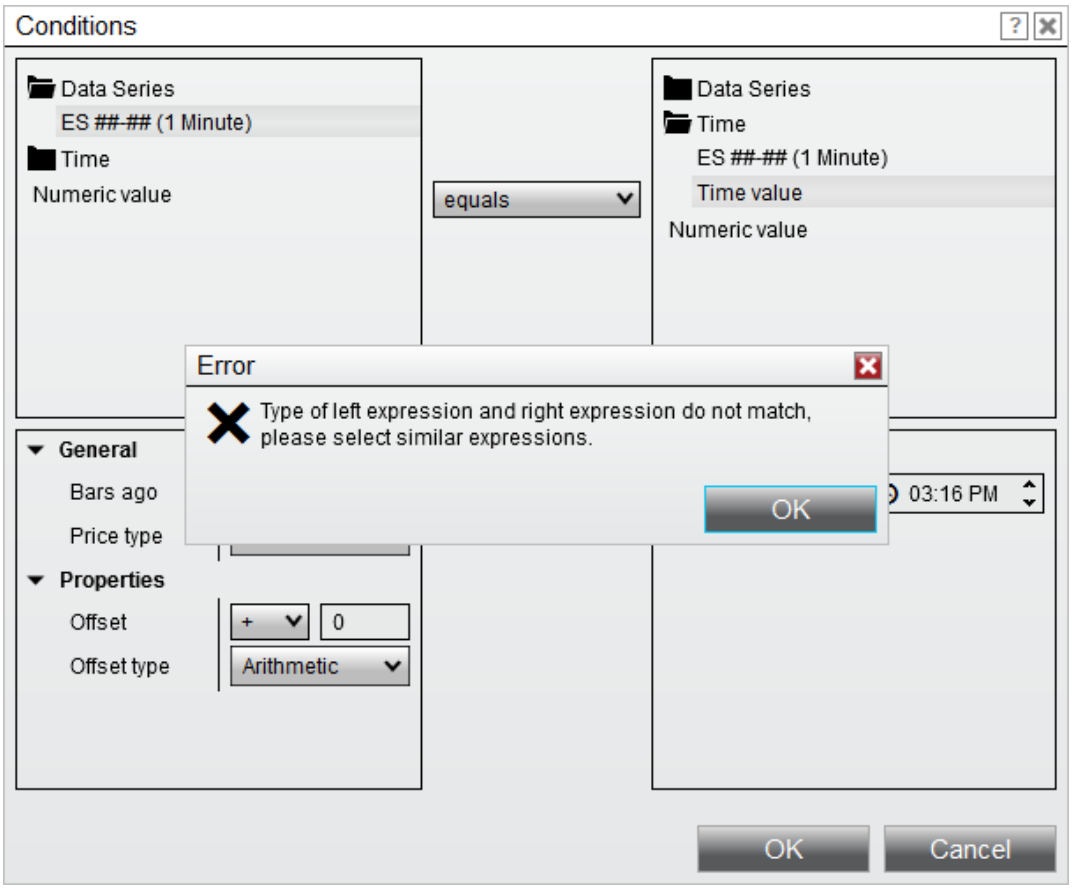


1. Available items such as indicators, price data, etc. to use for the comparison
2. List of relational operators

Relational operator invalid comparisons

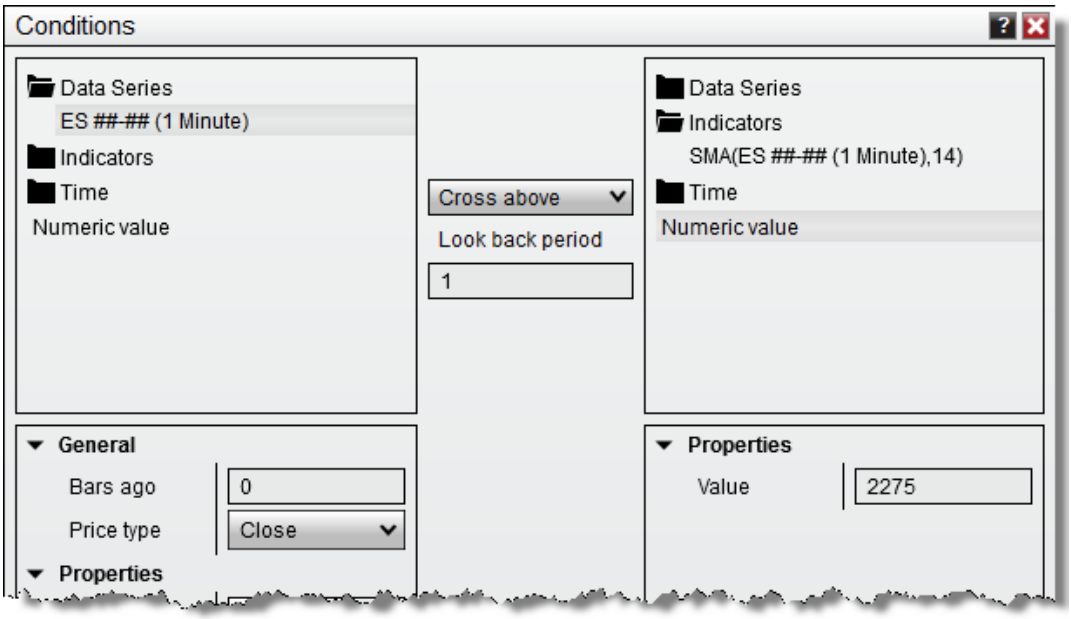
Since the relational operator will let you select any items from the left to compare to the right in the Condition Builder, you need to be mindful what you attempt comparing. For example comparing a price based value like the **ES ##-##** Data Series to the Time category **Time Value** would not be possible, and prompt the Condition Builder to issue an error like shown below -

"Type of left expression and right expression do not match, please select similar expressions"



To work around, you would need to select expressions with a similar return value that would allow for a programmatic comparison. In the example used above, the **ES ##-##** Data Series provides an double value in return that is attempted to be compared to a time span value, which **Time Value** would return.

The correct approach is shown below, the **ES ##-##** price would return a double value which would be compared to the **Numeric Value** 2275 to see if the price will **Cross Above** that.



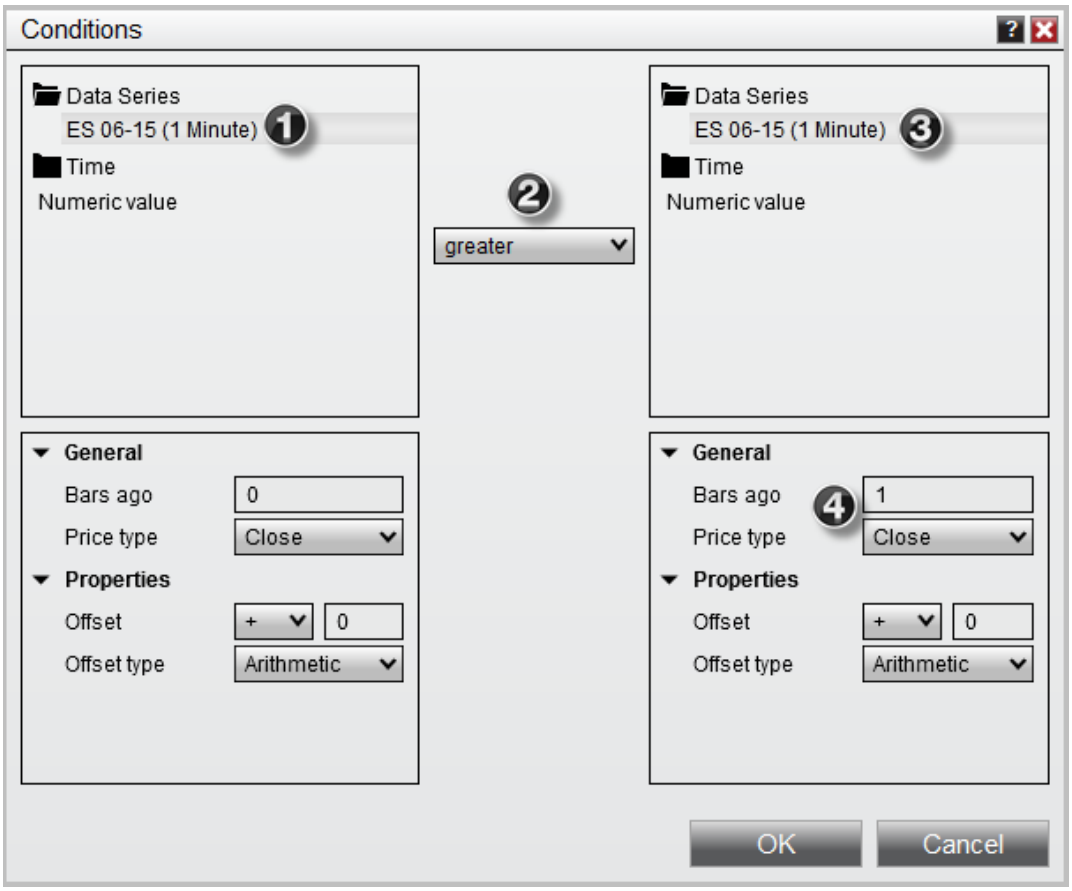
How to make chart price data comparisons

Price Data Comparisons

You can compare a chart's bar price data such as checking for a higher close. In order to compare the current bar value, to a previous bar value, we will need to use a Chart's **Data Series** as our condition object. In our example, we are using the ES 12-14 (1 minute) **Data Series** as our condition object.

The following is an an example and represents one of many possible combinations.

1. Select the **Data Series** and set the **Price type** to Close.
2. Select the "greater" relational operator
3. Select the **Data Series** and set the **Price type** to Close.
4. Set the Bars ago parameter to a value of "1"



Once the OK button is pressed, a condition is created that would translate to the following:

"Current closing price is greater than the closing price of 1 bar ago"

How to offset an item value

Offsetting an Item Value

You can offset the value of most items available in the **Condition Builder**. An offset is a value that is added or subtracted from the actual item's value. When an item is selected such as an indicator or price data, the **Offset type** and **Offset** parameters become visible in the window directly below the item selected. This is shown as numbers 5 and 6 in the image below.

> **Note**: Offsetting a condition **CANNOT** be applied directly to **Drawing Tools**. Should you wish to be alerted once a value is within *N-value* of the drawing

tool, apply the offset calculation to the data series or indicator condition.

Offset type can be set to:

Arithmetic	Performs simple math functions such as adding, subtracting, multiplying, or dividing from the items value
Pips	An absolute pip value (for forex) from the item's value
Percent	A percentage value of the item's value. A value of 10 is equal to 10%
Ticks	The number of ticks (0.01 for stocks and the tick size for futures) from the item's value

Once the **Offset type** is selected, you must set the value **Offset**.

The following is an example and represents one of many possible combinations:

1. Select the **Data Series** and set the **Price type** to Close
2. Select the "greater" relational operator
3. Select the **Data Series** and set the **Price type** to High
4. Set the **Bars ago** parameter to a value of "1"
5. Set the **Offset type** parameter to Ticks
6. Set the **Offset** parameter to a value of "1"

Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current closing price is greater than the high price of 1 bar ago + 1 tick"

How to make indicator to value comparisons

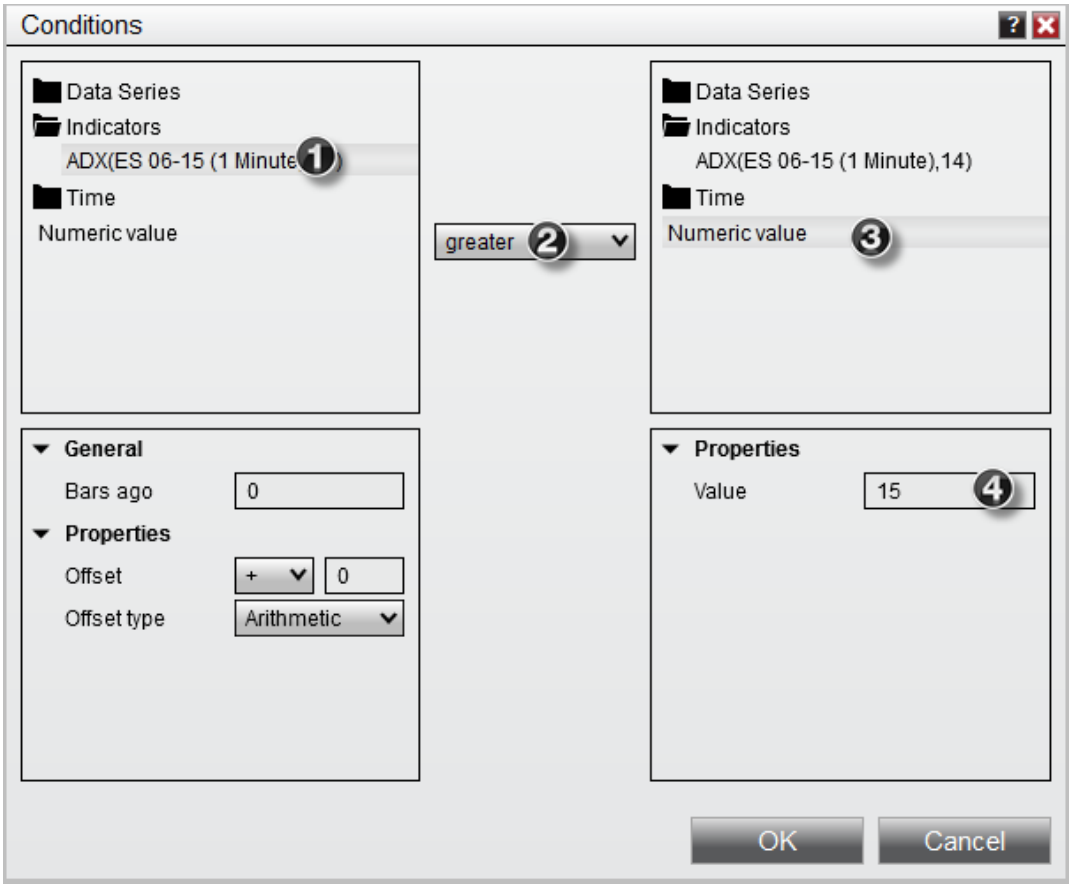
Indicator to Value Comparisons

You can compare an indicator's value to a numeric value. This can come in handy if you wanted to check if ADX is over a value of 30 (trending) or if Stochastics is under a value of 20 (oversold) or any other conditions you can think of.

The following is an an example and represents one of many possible combinations. We have already added the ADX indicator to our chart so it is available as condition object.

1. Under the **Indicators** category, select the **ADX** indicator

2. Select the "greater" relational operator
3. Select the **Numeric value** category
4. Enter the numeric value



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current value of a 14 period ADX is greater than 15"

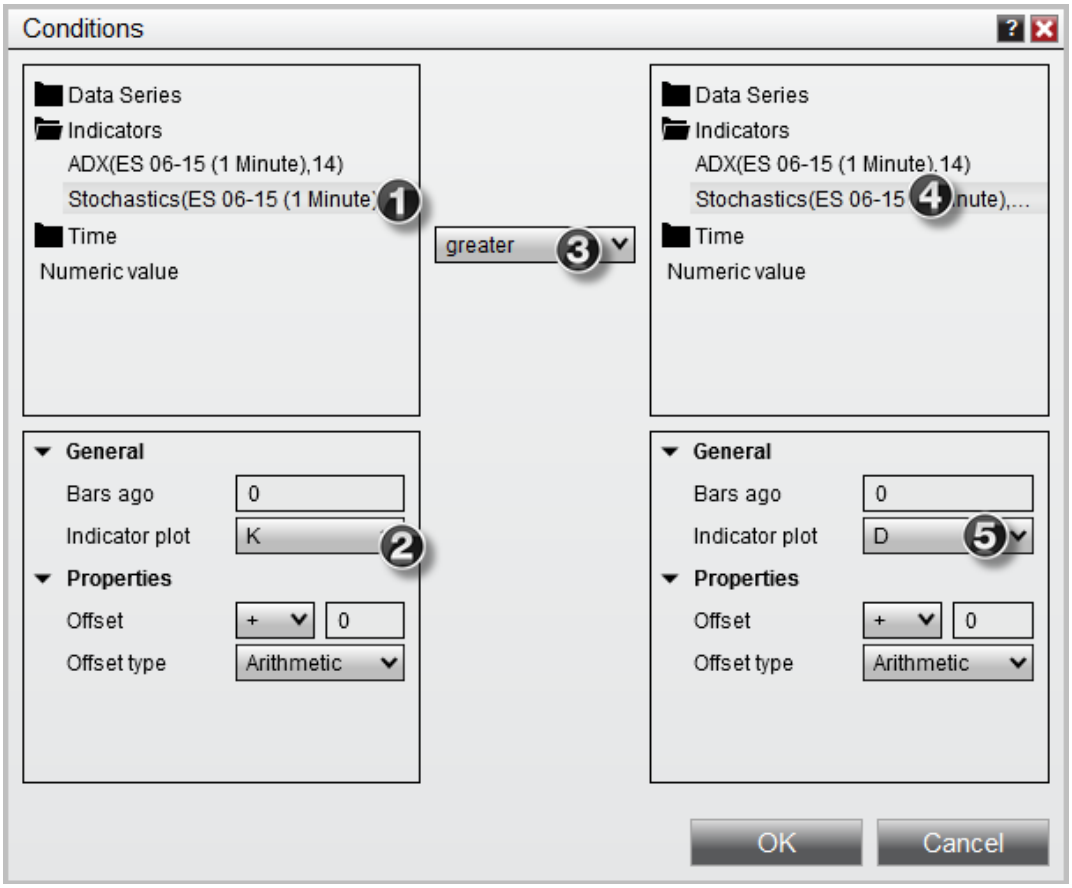
How to compare plot values of multi-plot indicators

Comparing Plot Values of Multi-Plot Indicators

You can compare plots in the same indicator or select any individual plot within an indicator to create a condition.

The following is an example and represents one of many possible combinations. We have already added the Stochastics indicator to our chart so it is available as condition object.

1. Under the **Indicators** category, select the **Stochastics** indicator
2. Set the **indicator plot** and select the **K** plot
3. Select the "greater" relational operator
4. Under the **Indicators** category, select the **Stochastics** indicator
5. Set the indicator input parameters and select the D plot



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current K plot value of a Stochastics indicator is greater than the current D plot value of the same Stochastics indicator"

How to create a cross over condition

Cross Over Conditions

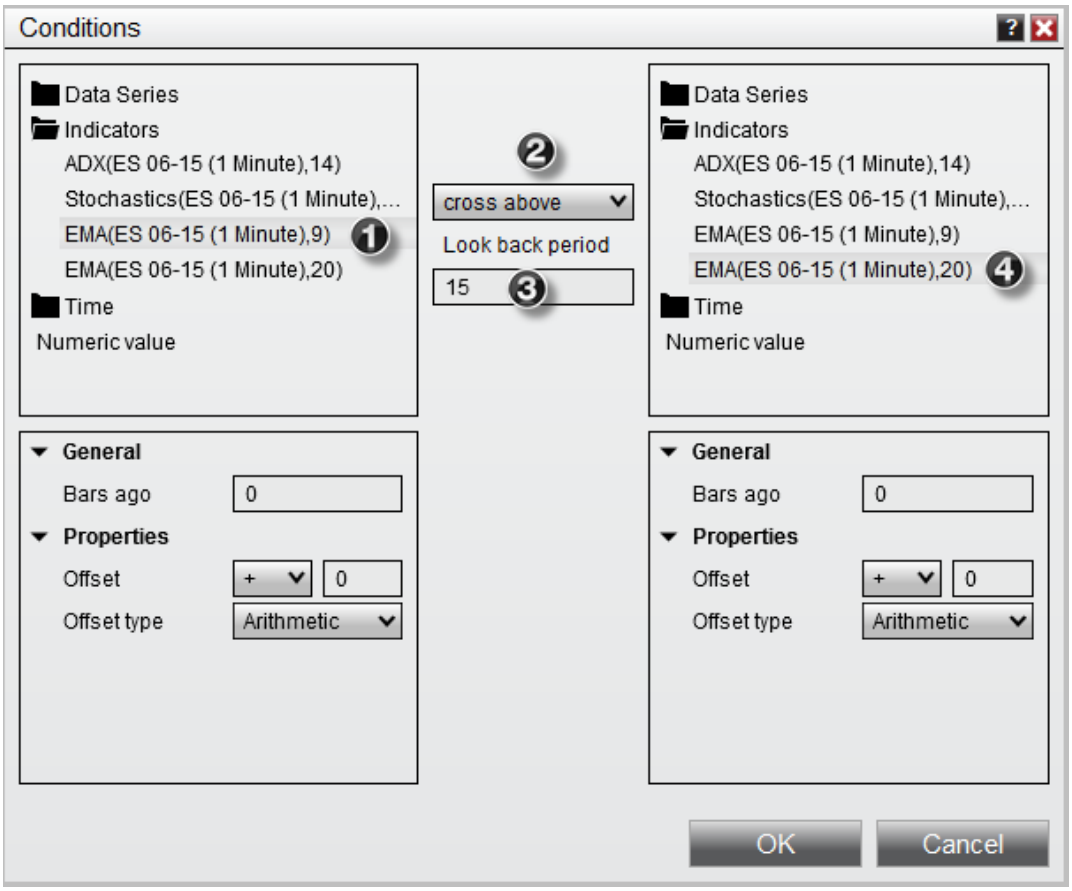
You can check for either a **Cross Above** or **Cross Below** condition with a user defined look back period. The look back period sets the number of bars to look back to check for the cross over condition.

The following is an an example and represents one of many possible combinations. We have already added two EMA indicators (9 period EMA and 20 period EMA) to our chart so they are both available as condition objects.

1. Under the **Indicators** category, select the **9 period EMA** indicator
2. Select "cross above" relational operator
3. Set the **Look back period**
4. Under the Indicators category, select the **20 period EMA** indicator

Notes:

- The **Look back period** must be at least 1 to function.
- **CrossInside** and **CrossOutside** are reserved for drawing objects, setup examples could be found on the next section Alerts Examples



Once the **OK** button is pressed, a condition is created that would translate to the following:

"9 period exponential moving average crosses above the 20 period exponential moving average in the last 15 bars"

How to compare account position information

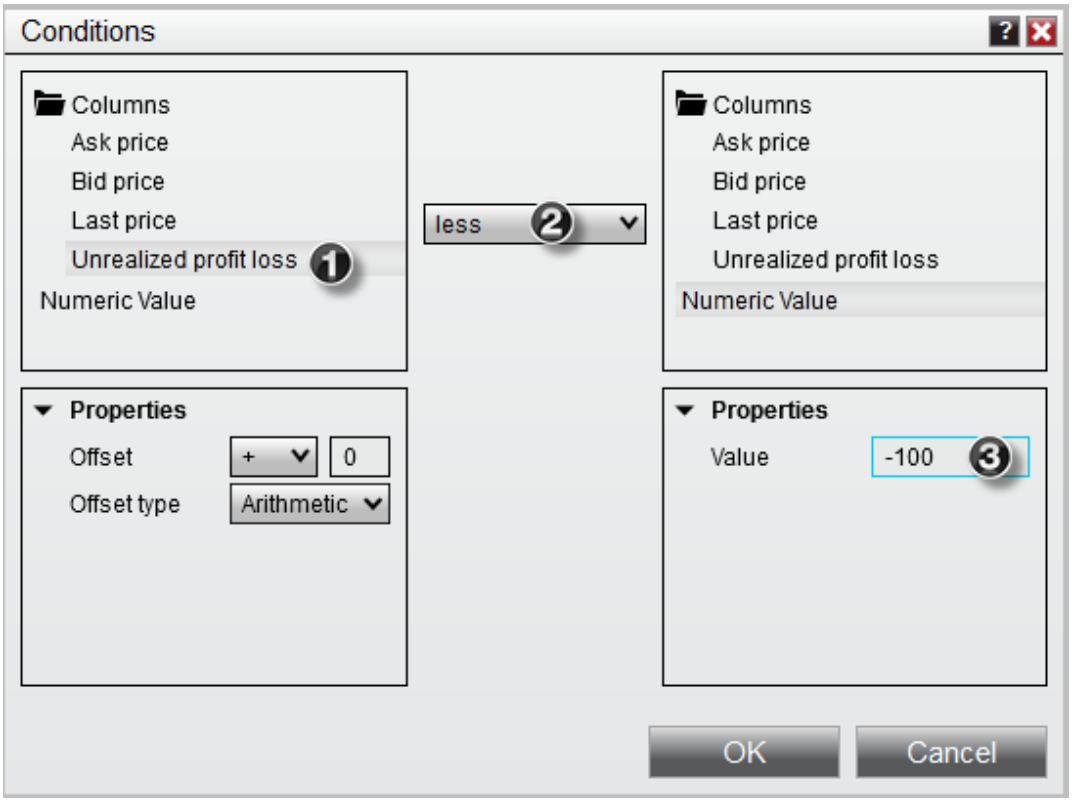
Creating Account Position Comparisons

You can compare your current account state information such as but not limited to account PnL or position size using the Market Analyzer.

The following is an an example and represents one of many possible combinations. We have already added the **Unrealized profit loss** column to our **Market Analyzer** so it is available as [condition object](#).

1. Under the **Columns** category, select the **Unrealized profit loss** column

2. Select the "less" relational operator
3. Under the **Columns** category, select the **Number Value** category
4. Set the Value



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current Unrealized profit loss is less than -$100"

How to create time comparisons

Creating Time Comparisons

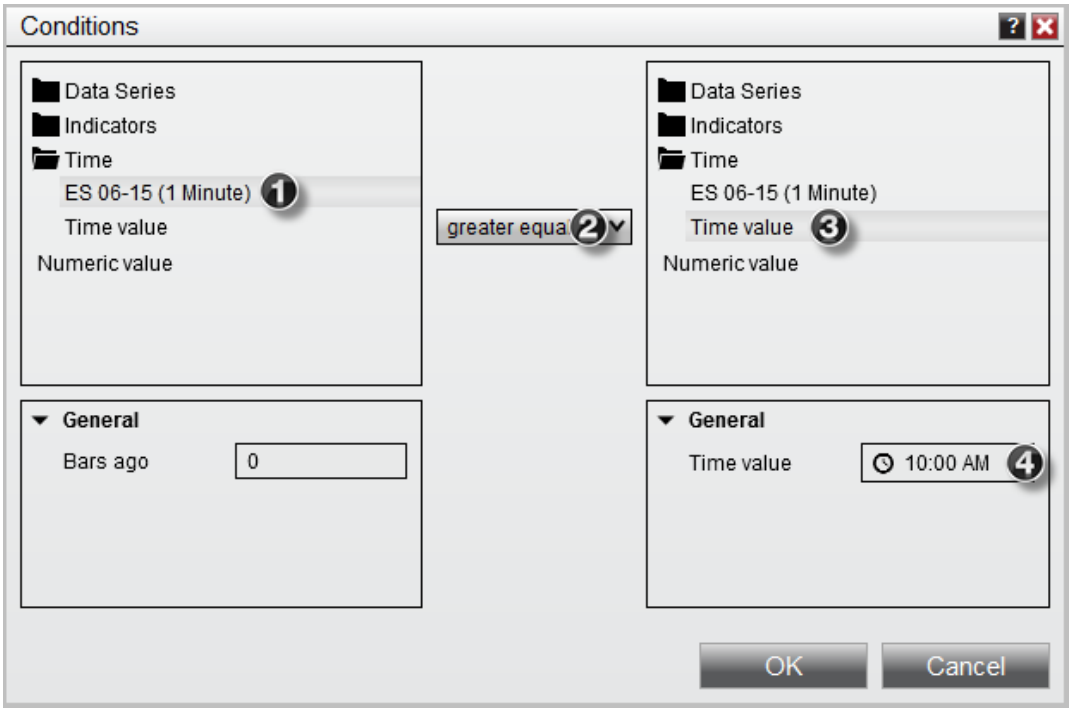
You can compare a chart bar's time data to a user defined time or date value.

The following is an an example and represents one of many possible combinations.

> **Note**: Time series represents a collection of bar Date/Time values of a bar

series which are available from a chart

1. Select the **Time** category and select the Data Series series
2. Select the "greater equal" relational operator
3. Expand the **Time value** category
4. Set the **Time value** parameter to a user defined value of "10:00 AM"



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current bar's time is greater or equal to 10:00 AM"

10.2.5 Alerts Examples

Following are a few examples of Alerts which can be set up on charts or Market Analyzer windows. Each example shows a different type of alert condition, along with a different action or combination of actions. Feel free to copy and modify these examples for your own uses, or simply use them as a guide to reinforce the material covered on the previous pages.

Play Video

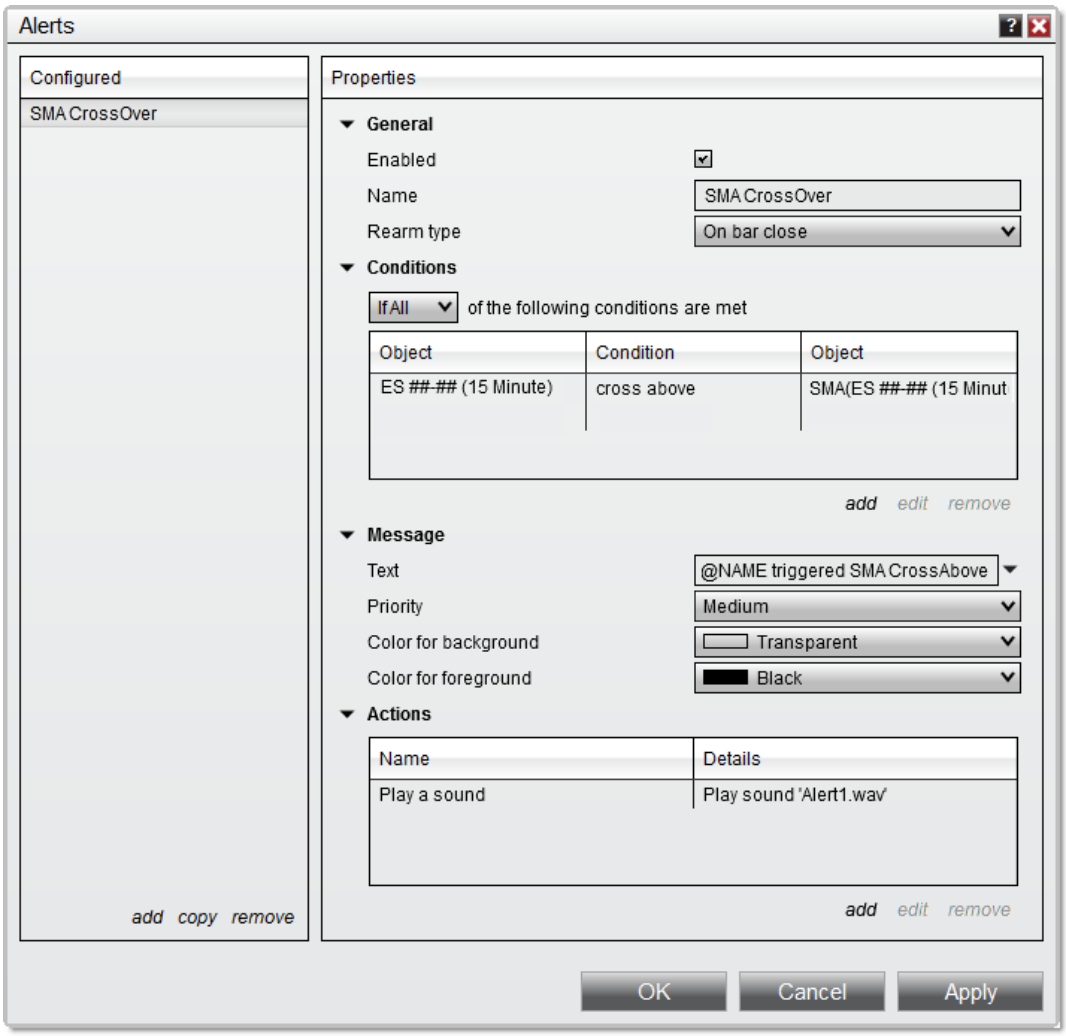
Simple Moving Average Crossover

Preparation

- Open a chart
- Apply an SMA indicator to the chart

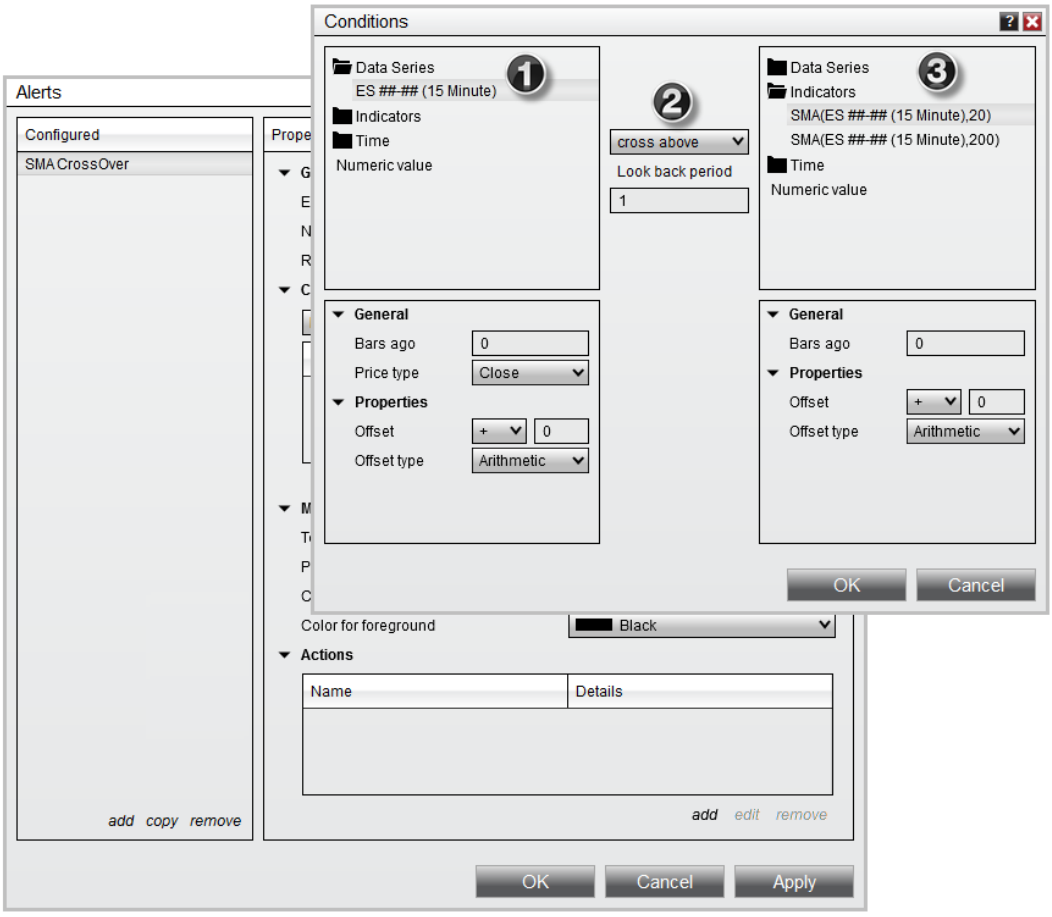
Overview

This basic alert triggers when the current market price crosses above a 20-period Simple Moving Average. The image below shows the fully configured alert.



Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

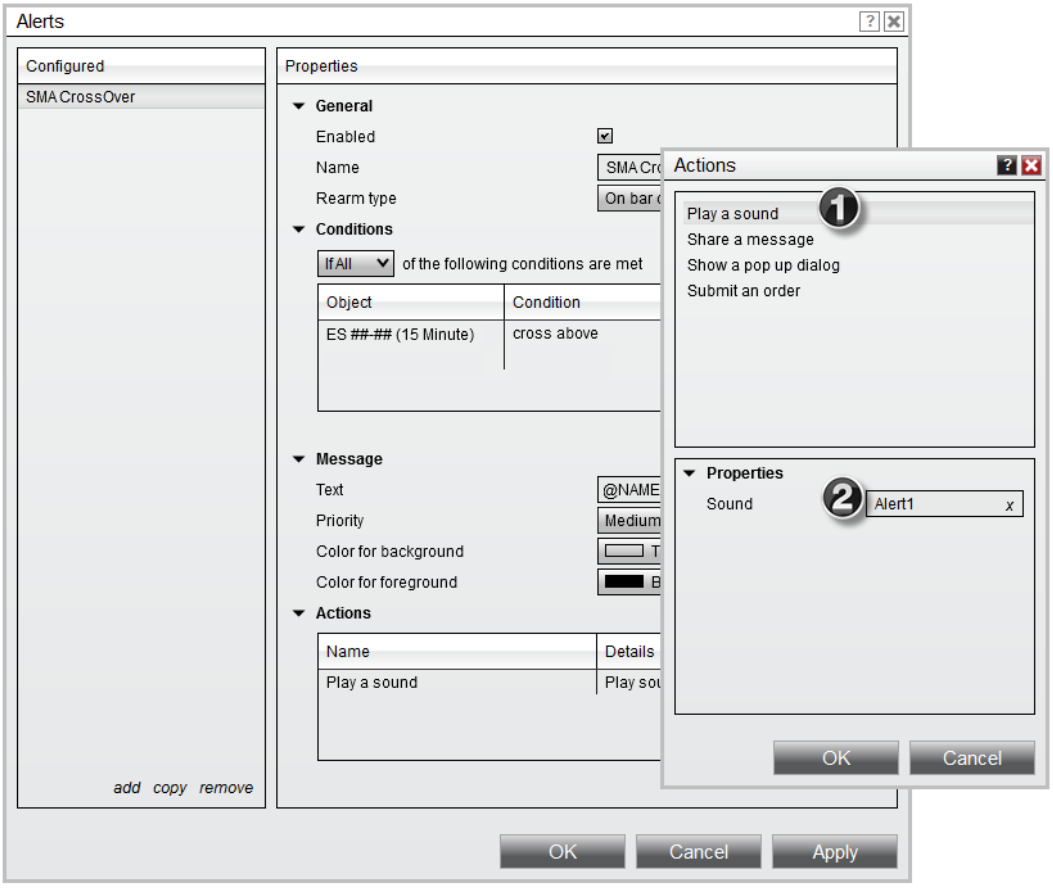


1. In the Conditions window pictured above, the ES data series is selected in the left panel

2. The "CrossAbove" condition is selected

3. The 20-period SMA (one of two SMAs applied to the chart) is selected in the right panel

We now have a condition which translates to **"When the current price crosses above the 20 SMA."**



1. In the Actions window, the "Play a Sound" option is selected

2. A sound named "Alert1" is selected to be played when the alert triggers

Multi-Plot Indicator Crossover

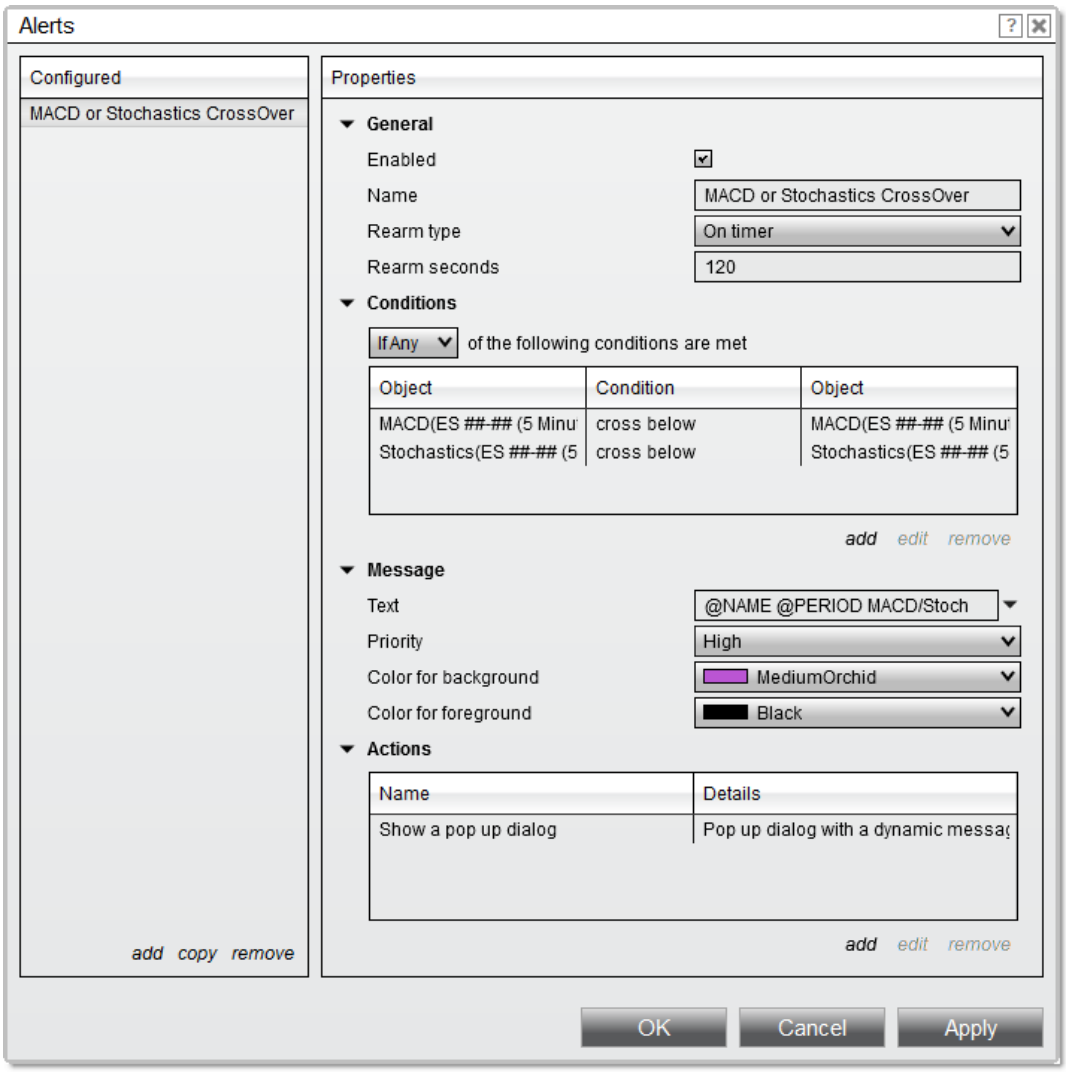
Preparation

- Open a chart
- Apply a MACD indicator to the chart
- Apply a Stochastics indicator to the chart

Overview

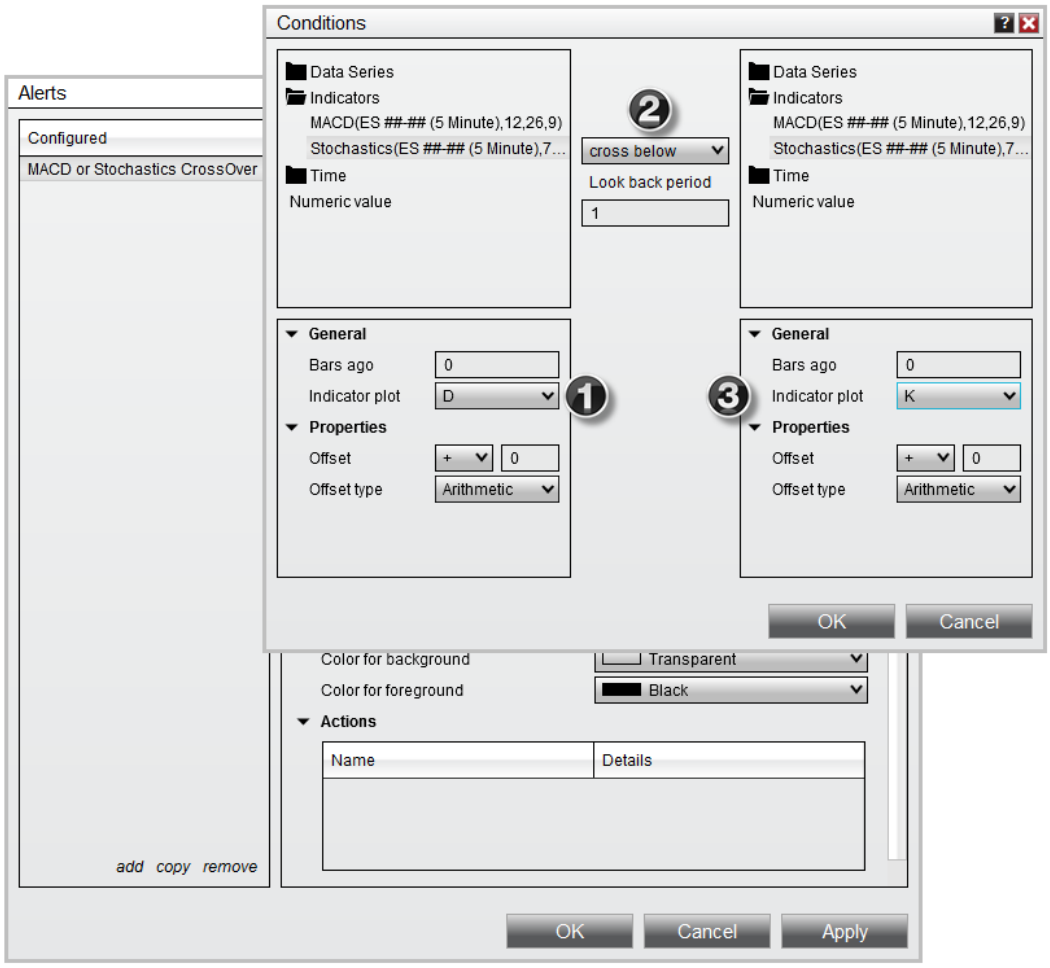
This alert is a bit more advanced than the example above. This alert demonstrates a multi-plot crossover scenario, detecting when one specific plot of an indicator crosses a different plot of the same indicator. In this example, plots within the

MACD and the Stochastics indicators must cross other plots within the same indicators. The image below shows the fully configured alert.



Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

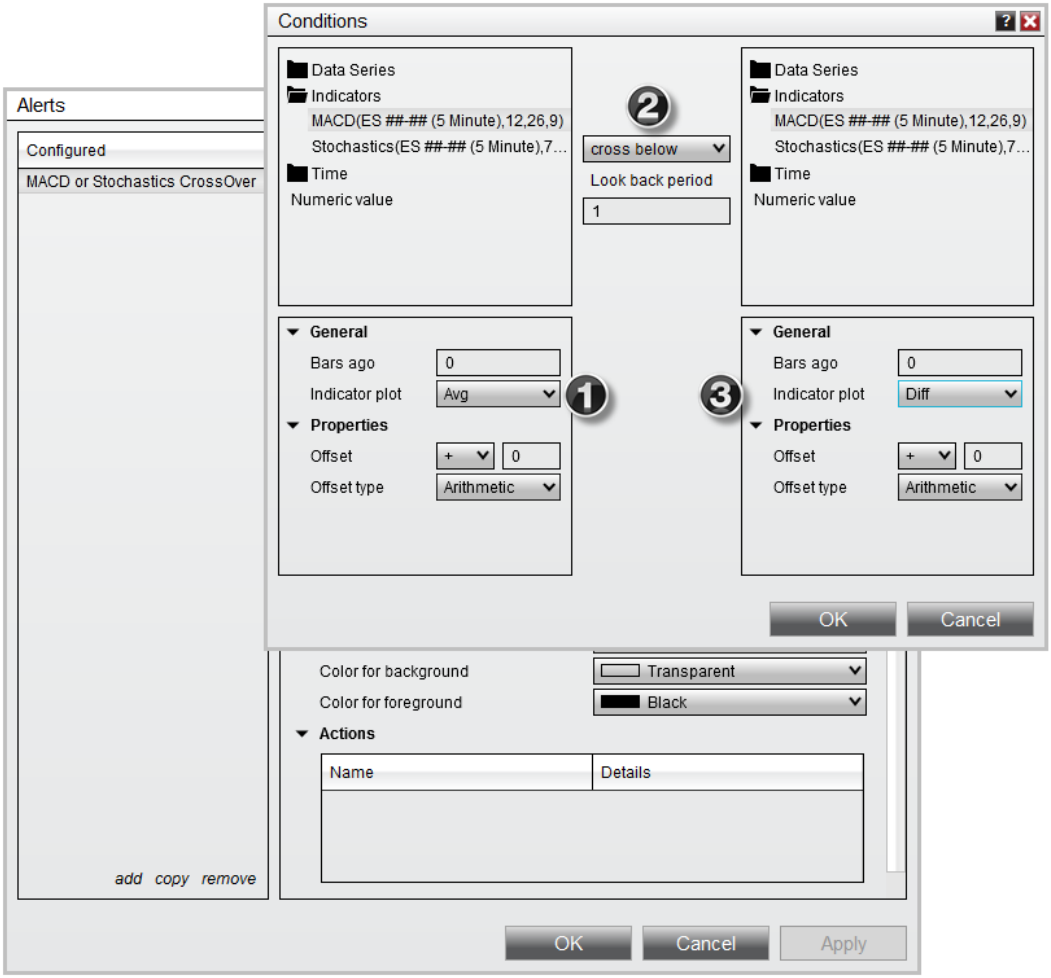


1. In the Conditions window, the D plot of the Stochastics indicator is selected in the left panel

2. The CrossBelow condition is selected, and a value of 1 is entered for the look-back period

3. The K plot of the Stochastics indicator is selected in the right panel

We now have a condition that translates to "When Stochastics D crosses below Stochastics K within the last one bar."

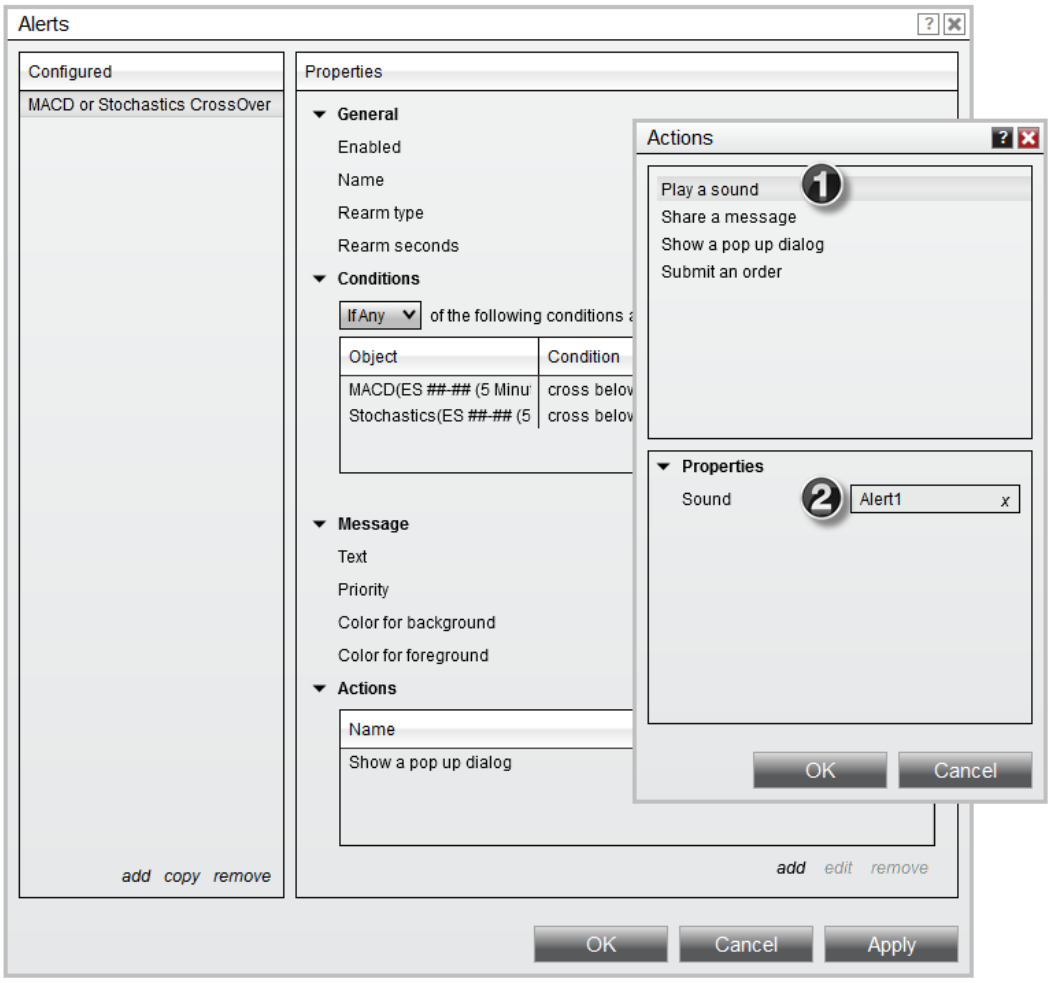


1. In the Conditions window for the second condition, the Avg plot of the MACD indicator is selected in the left panel

2. Just like the previous condition, the CrossBelow operator is used with a look-back period of 1

3. The Diff plot of the MACD indicator is selected in the right panel

We now have a second condition that translates to **"When MACD Avg crosses below MACD Diff within the last 1 bar."**



1. In the Actions window, the "Play a Sound" option is selected

2. A sound named "Alert1" is selected to be played when the alert triggers

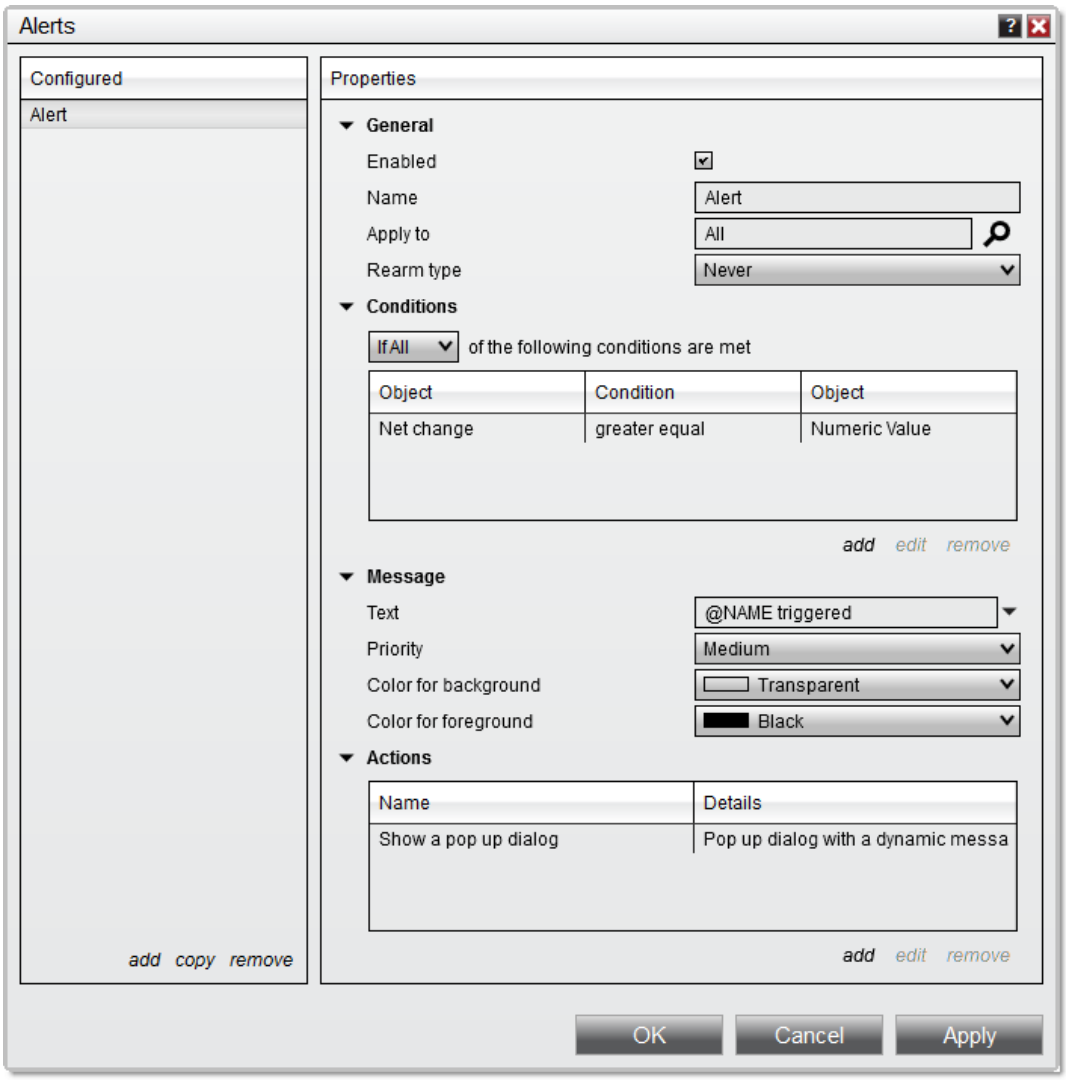
Hot List Analyzer Net Change

Preparation

- Open a Hot List Analyzer
- Populate a Hot List in the window

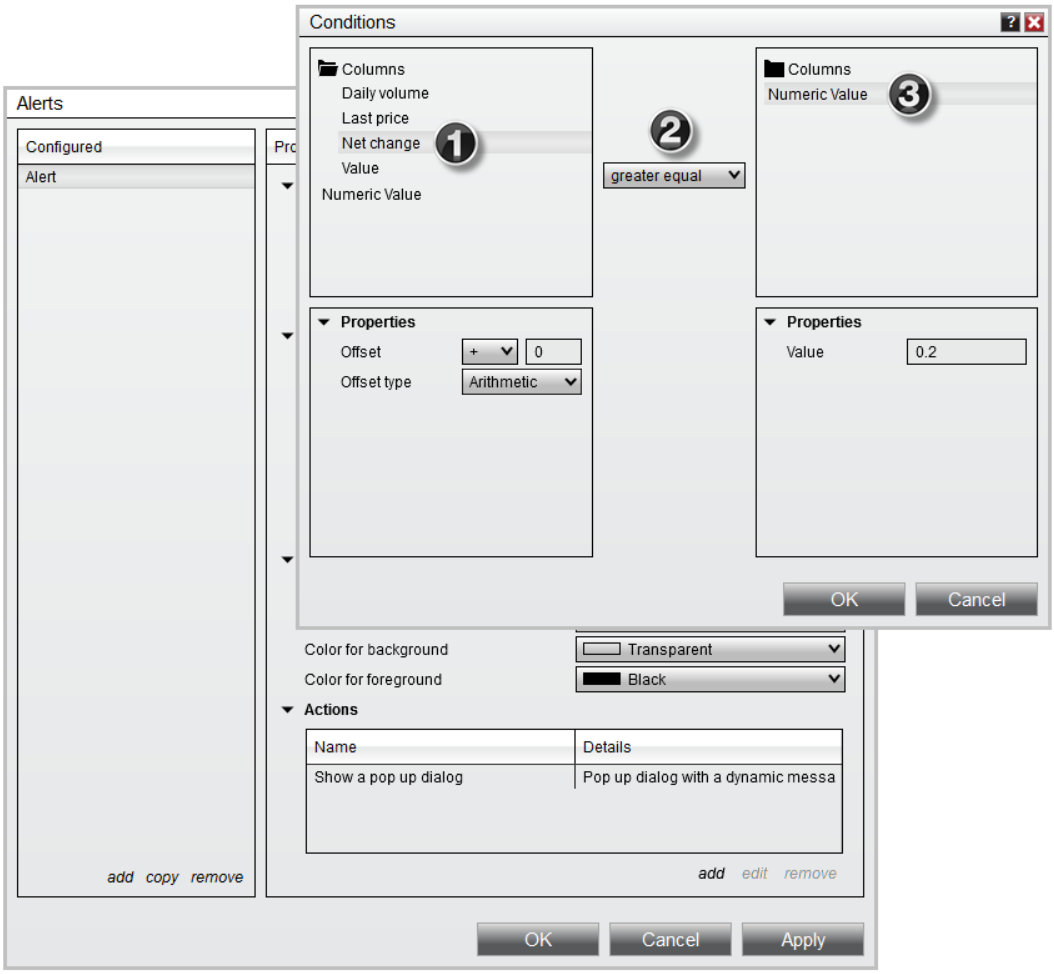
Overview

This alert is set up on the Hot List Analyzer, and specifically relates to the Net Change column. Alerts can be set up for other Hot List Analyzer columns in a similar way. The image below shows the fully configured alert.



Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

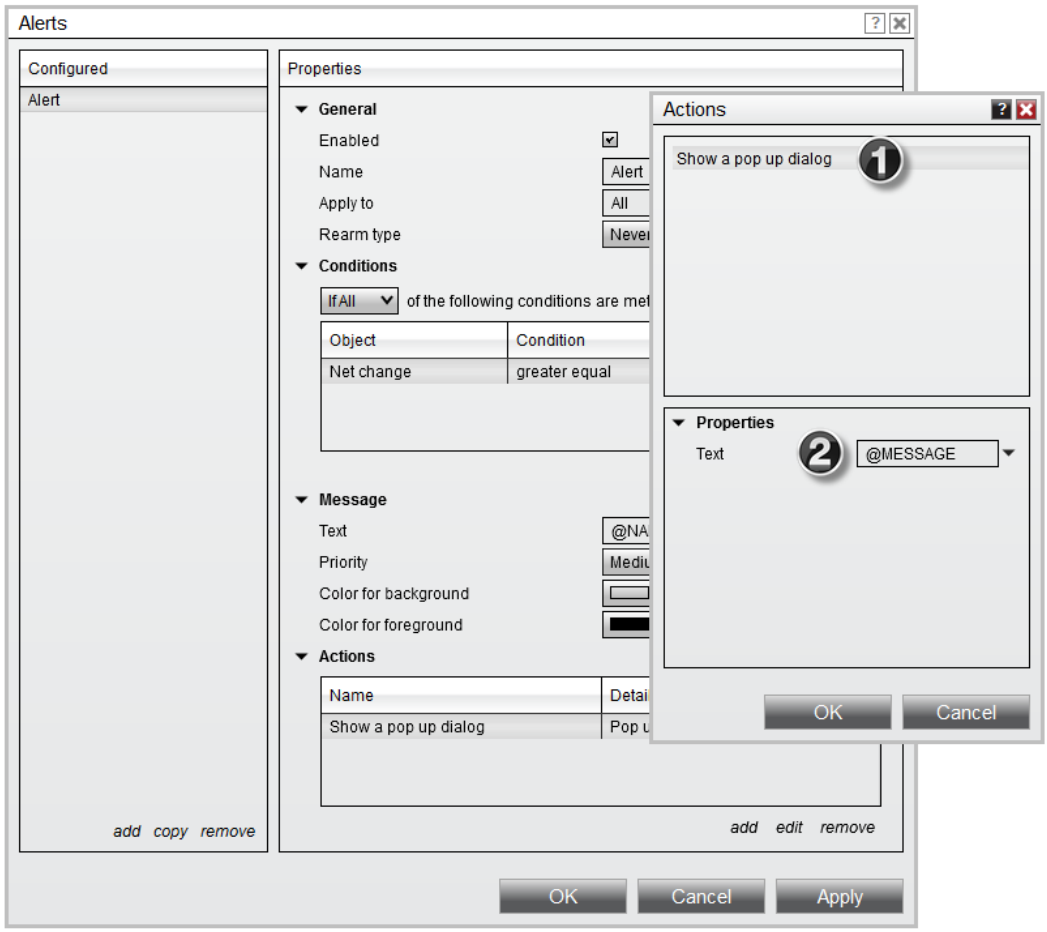


1. In the Conditions window, the Net Change column is selected in the left panel.

2. The Greater/Equal condition is selected

3. The Numeric Value property is selected in the right panel, with a value of 0.2

We now have a condition that translates to **"When the value of the Net Change column is greater than or equal to 0.2."**



1. In the Actions window, the "Show a pop up dialog" action is selected

2. "@MESSAGE" is entered for the text to be displayed in the dialog. This will populate the dialog with the message entered in the "Message" section of the Alerts window.

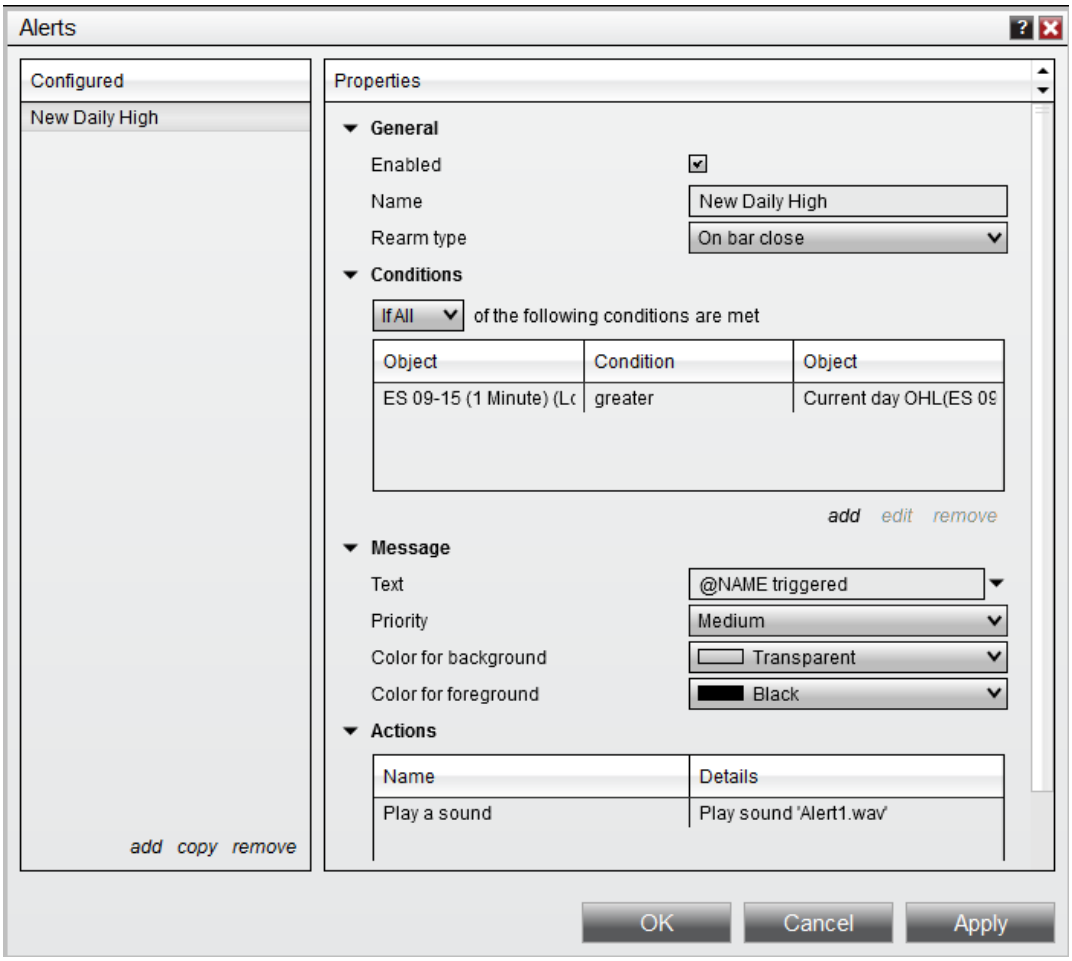
New Intraday High

Preparation

- Open a chart
- Apply a Current Day OHL indicator to the chart

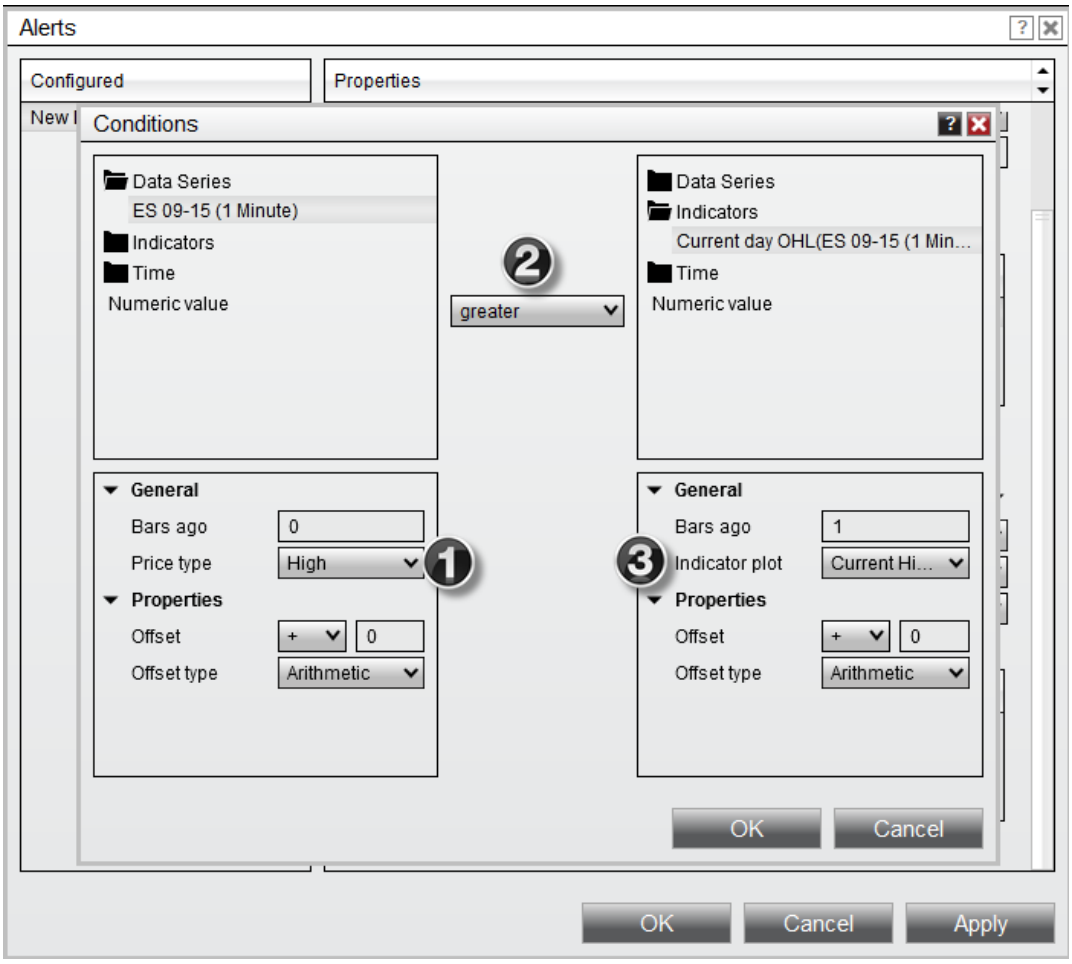
Overview

This alert will trigger when a new High is formed intraday. The image below shows the fully configured alert.



Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

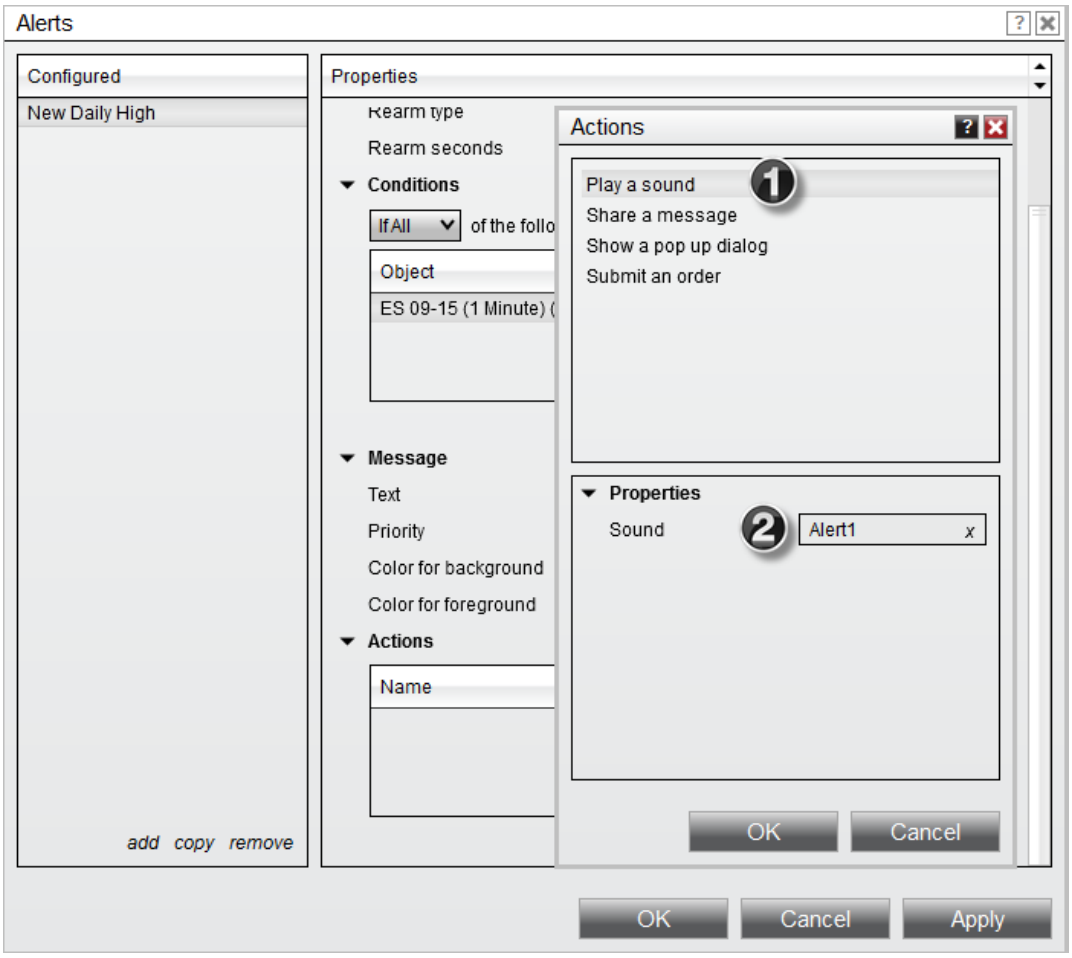


1. The "High" price type is selected for the ES 09-15 instrument in the left panel, with 0 used as the "BarsAgo" parameter

2. The "Greater" condition is selected

3. The Current Day OHL indicator is selected in the right panel, and the "Current High" plot is selected

We now have a condition that translates to **"When the current bar's High price is greater than the current day's High (prior to the current bar."**



1. In the Actions window, the "Play a Sound" option is selected

2. A sound named "Alert1" is selected to be played when the alert triggers

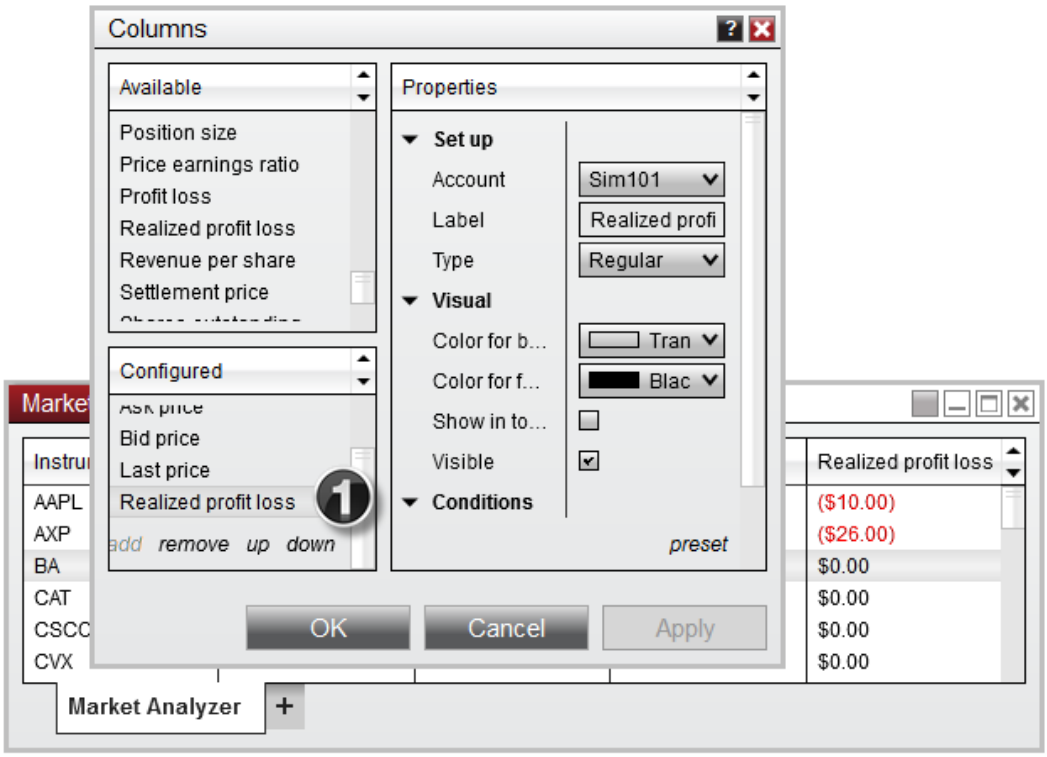
PnL Risk Management

Preparation

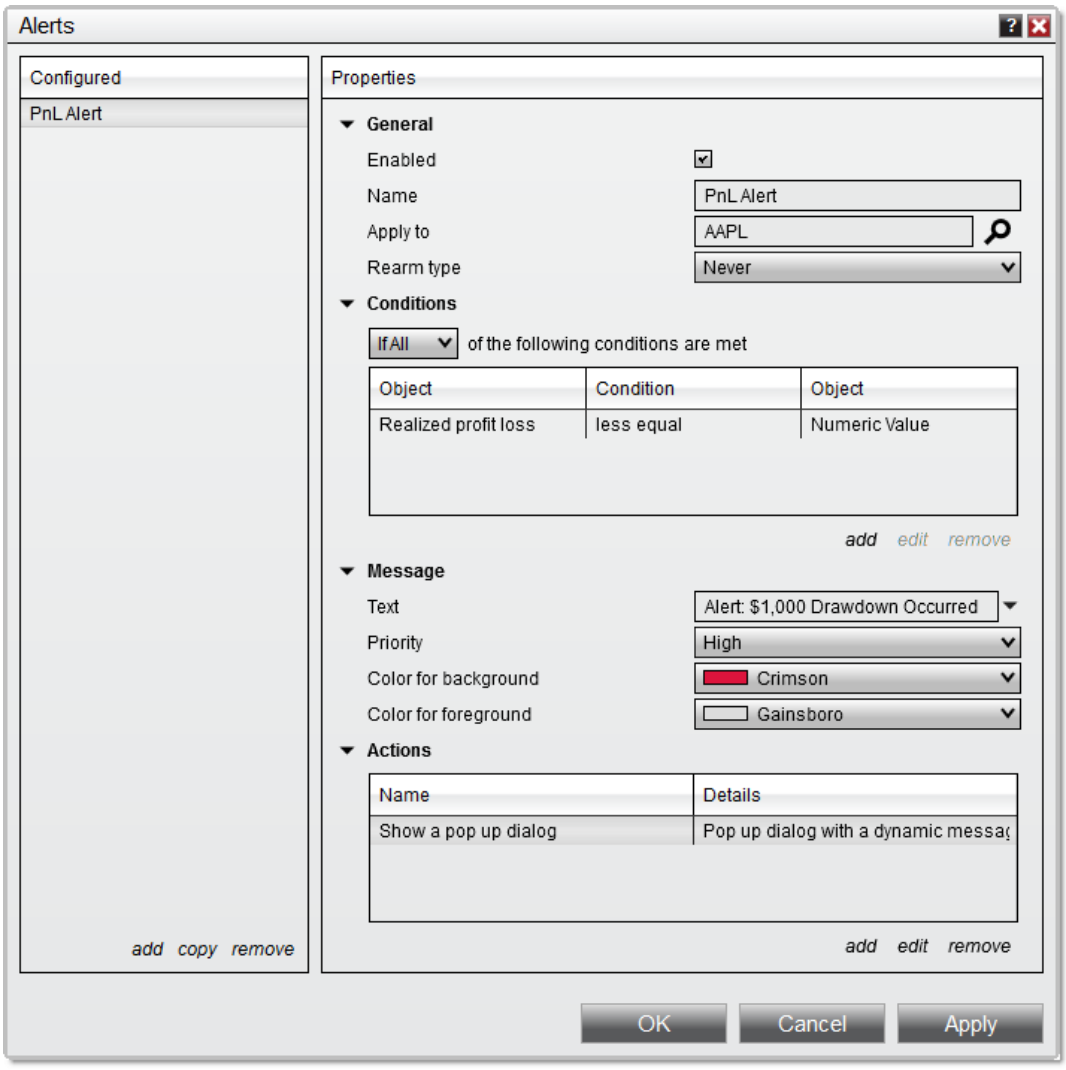
- Open a Market Analyzer
- Apply a "Realized Profit/Loss" column to the Market Analyzer (see image below)

Overview

This alert uses the Market Analyzer's "Realized Profit/Loss" column to trigger an alert when a certain level of loss has occurred. In the image below, the "Realized Profit/Loss" column is configured in a Market Analyzer window.

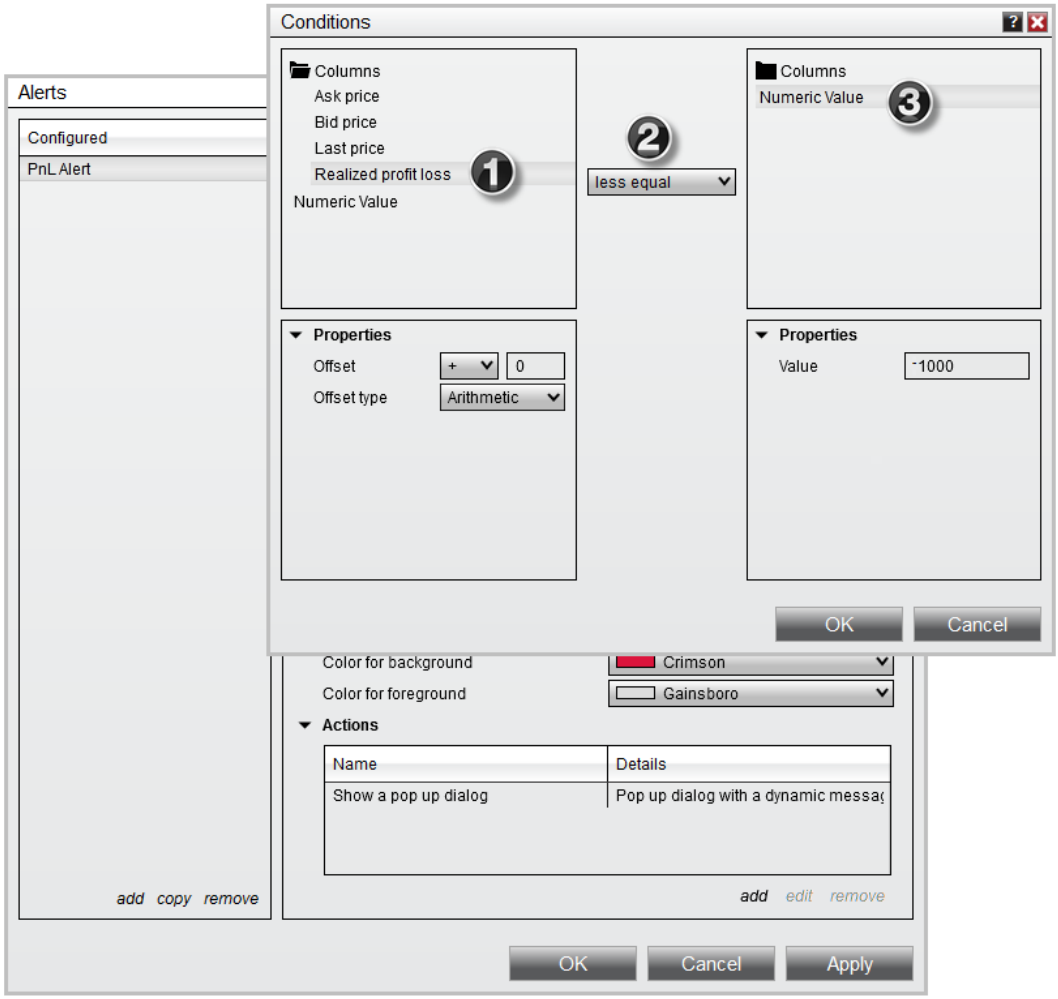


The image below shows the fully configured alert.



Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

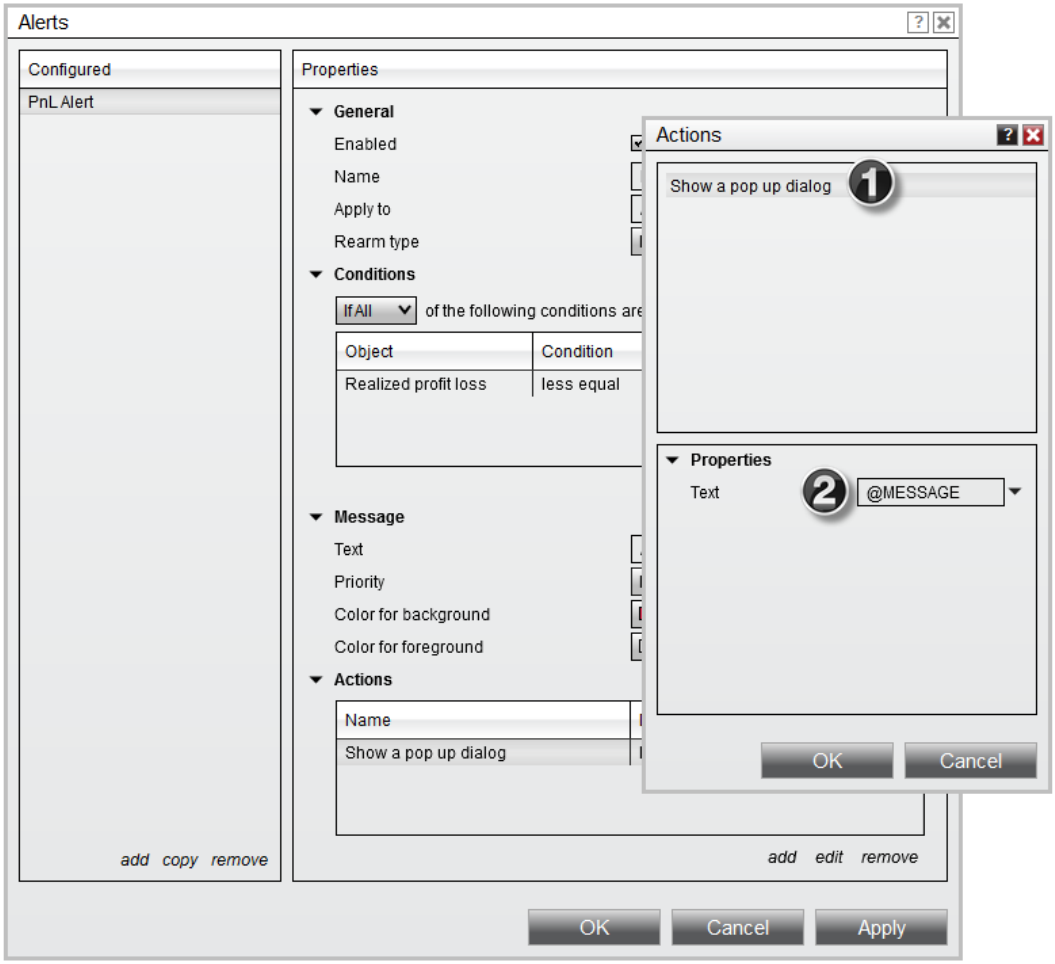


1. In the Conditions window, the "Realized Profit/Loss" column is selected in the right panel. Note that this column was manually added to the Market Analyzer before opening the Alerts window.

2. The Less Equal condition is selected.

3. The Numeric Value property is selected in the right panel, and a value of -1,000 is entered.

We now have a condition that translates to **"When the value of the "Realized Profit/Loss" column is -1,000 or less."**



1. In the Actions window, the "Show a pop up dialog" action is selected

2. "@MESSAGE" is entered for the text to be displayed in the dialog. This will populate the dialog with the message entered in the "Message" section of the Alerts window.

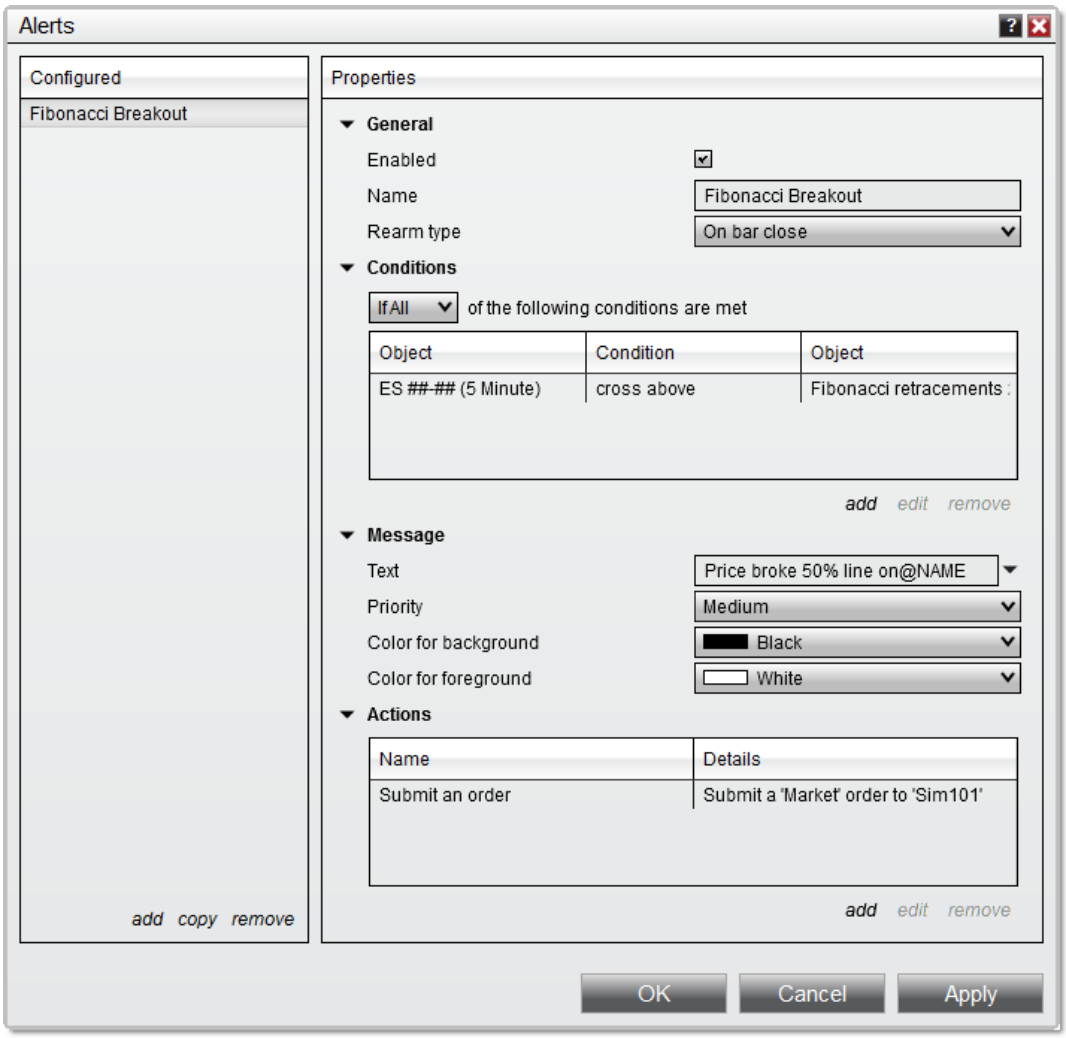
Price and Fibonacci Retracements

Preparation

- Open a chart
- Draw a Fibonacci Retracements object anywhere on the chart

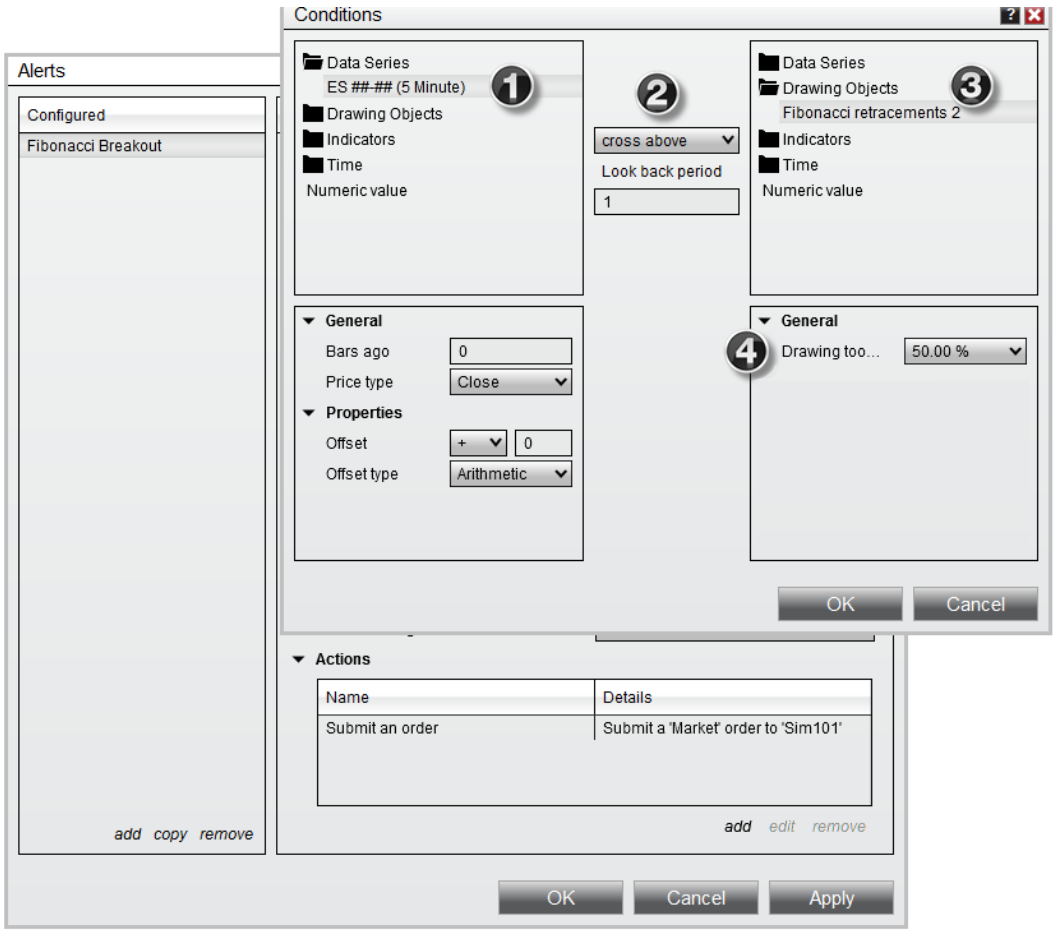
Overview

This alert compares the current market price to the 50% line drawn by a Fibonacci Retracements Drawing Tool. The image below shows the fully configured alert.



Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:



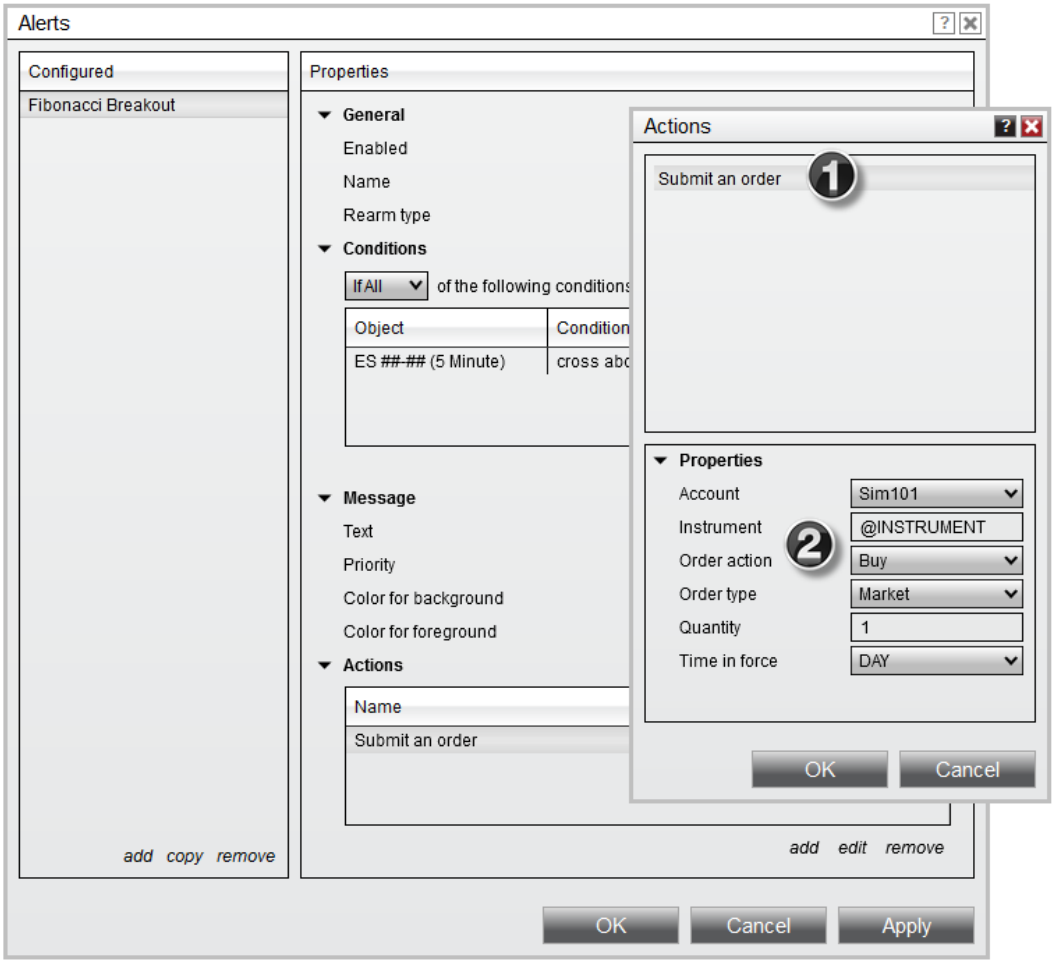
1. The ES ##-## data series is selected in the left panel, and the "Close" price type is selected (which will contain the Last price on the current bar)

2. The CrossAbove condition is selected

3. The **Drawing Tool** named "Fibonacci Retracements 2" is selected in the left panel. This is a specific drawing object which has already been drawn on the chart to which this alert is attached

4. The 50% line of the Fibonacci drawing object is selected

We now have a condition that translates to **"When the current price of ES crosses above the 50% line of the Fibonacci Retracements object on the chart."**



1. In the Actions window, the "Submit an Order" option is selected

2. Parameters for the order are set in the Actions window, as well

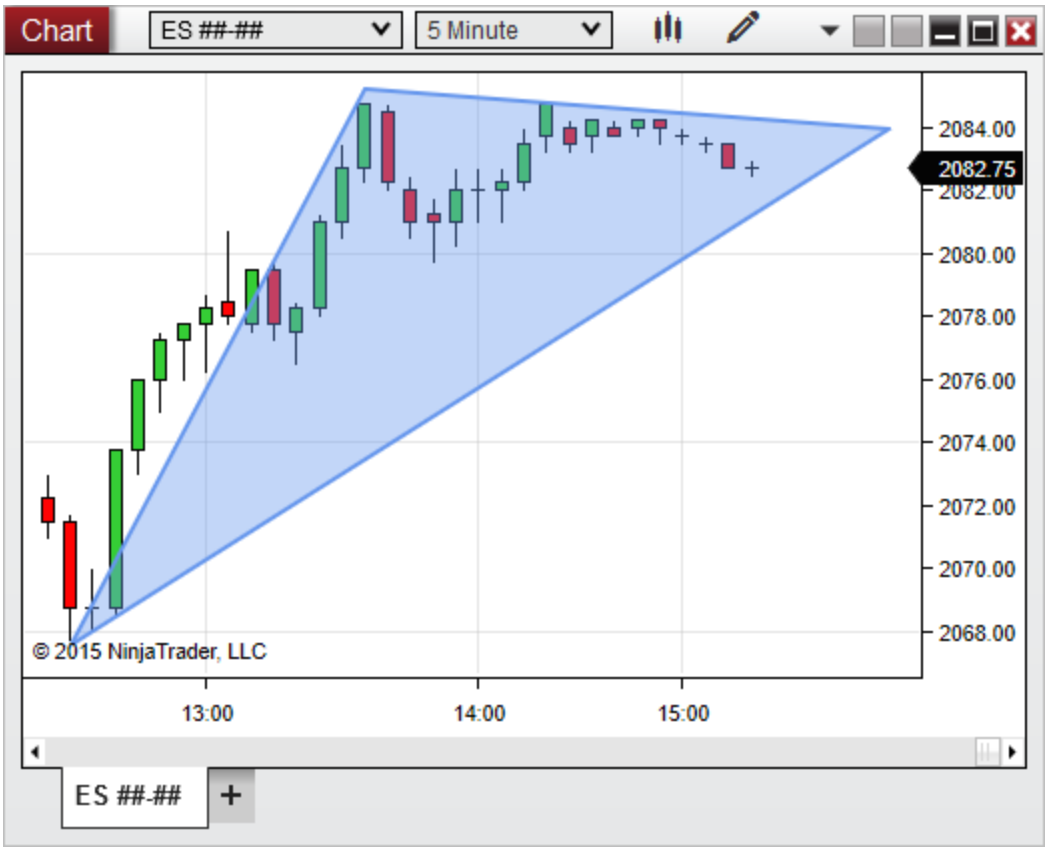
Price and User-Drawn Objects

Preparation

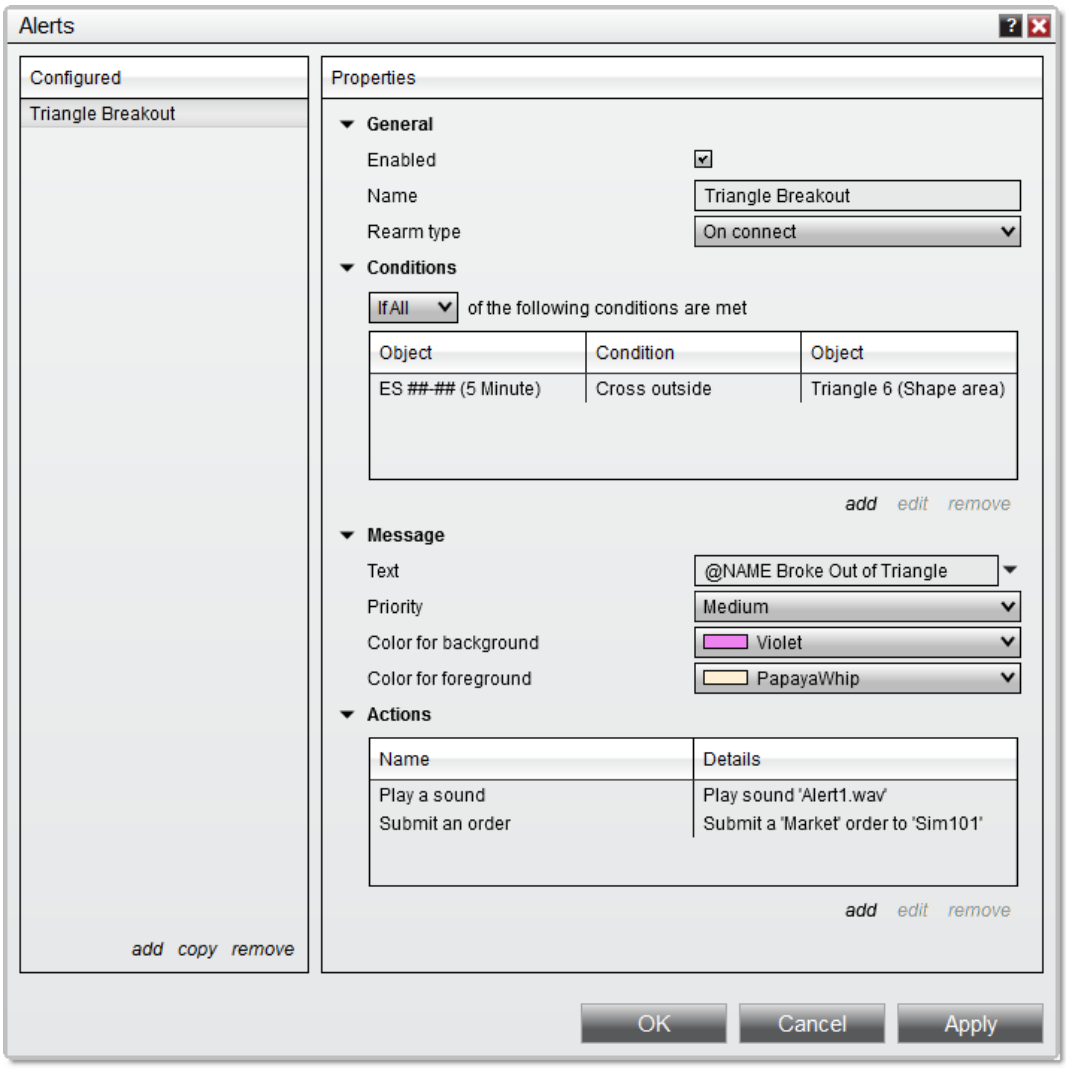
- Open a chart
- Use the Triangle **Drawing Tool** to draw a triangle on the chart. Make sure that the current market price is within the bounds of the triangle.

Overview

This alert detects when the current market price breaks outside of a user-drawn shape on the chart. The shape used in this alert can be seen below:

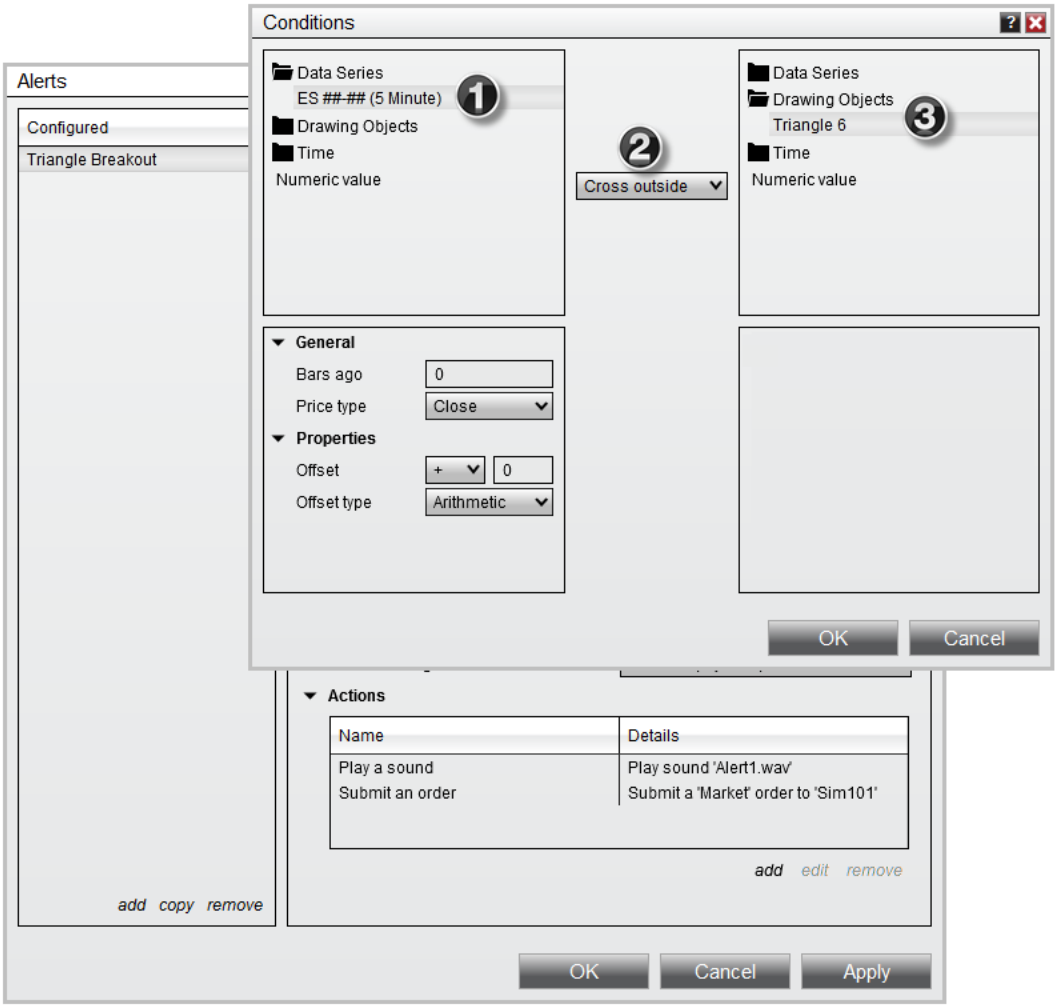


The image below shows the fully configured alert.



Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

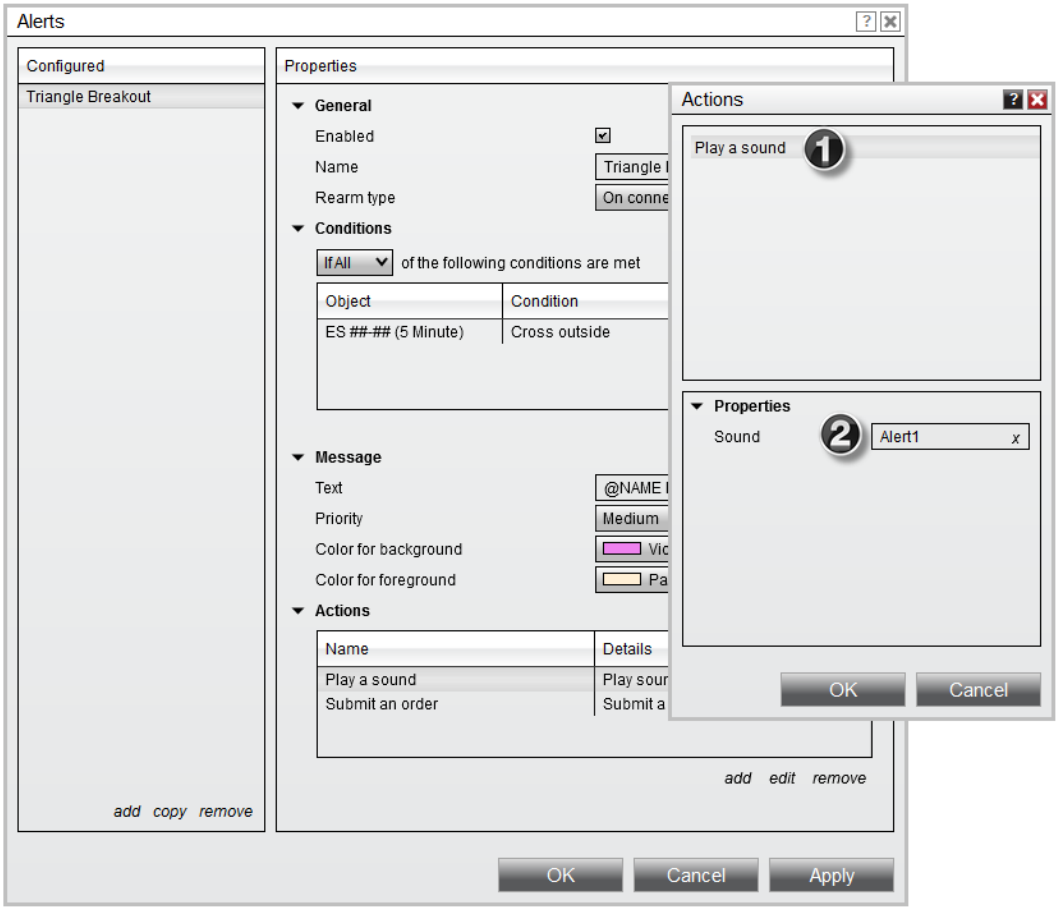


1. The ES ##-## data series is selected in the left panel

2. The Cross Outside condition is selected. This condition populates when a Drawing Object is selected in either the left or right panel

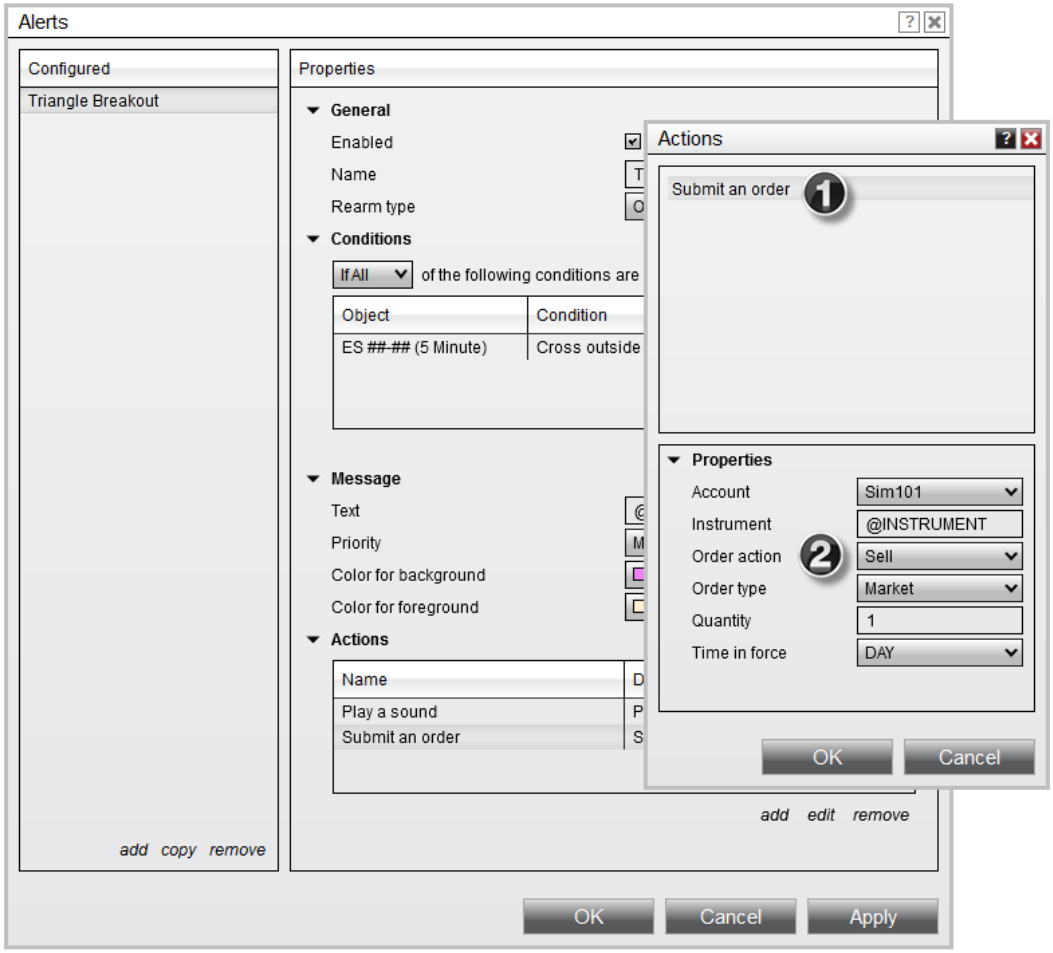
3. The custom-drawn triangle drawn on the chart is selected in the right panel (your object may have a difference name)

We now have an alert that translates to **"When the current market price breaks outside of the Drawing Object named 'Triangle 6'."**



1. In the Actions window, the "Play a Sound" option is selected

2. A sound named "Alert1" is selected to be played when the alert triggers



1. In the Actions window, the "Submit an Order" option is selected for a second action

2. Parameters for the order are set in the Actions window, as well. We now have two actions associated with this **Alert**.

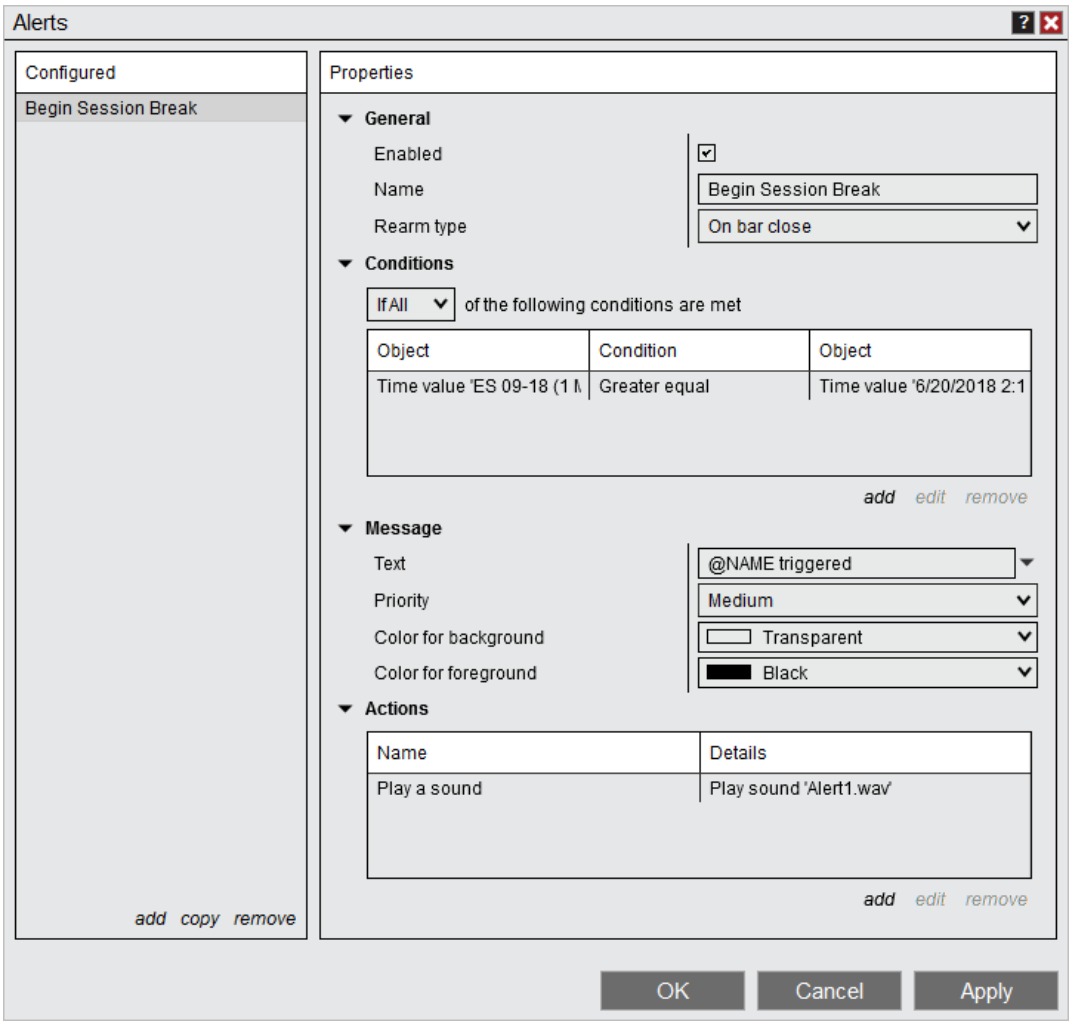
Time-Based Alert

Preparation

- Open a chart

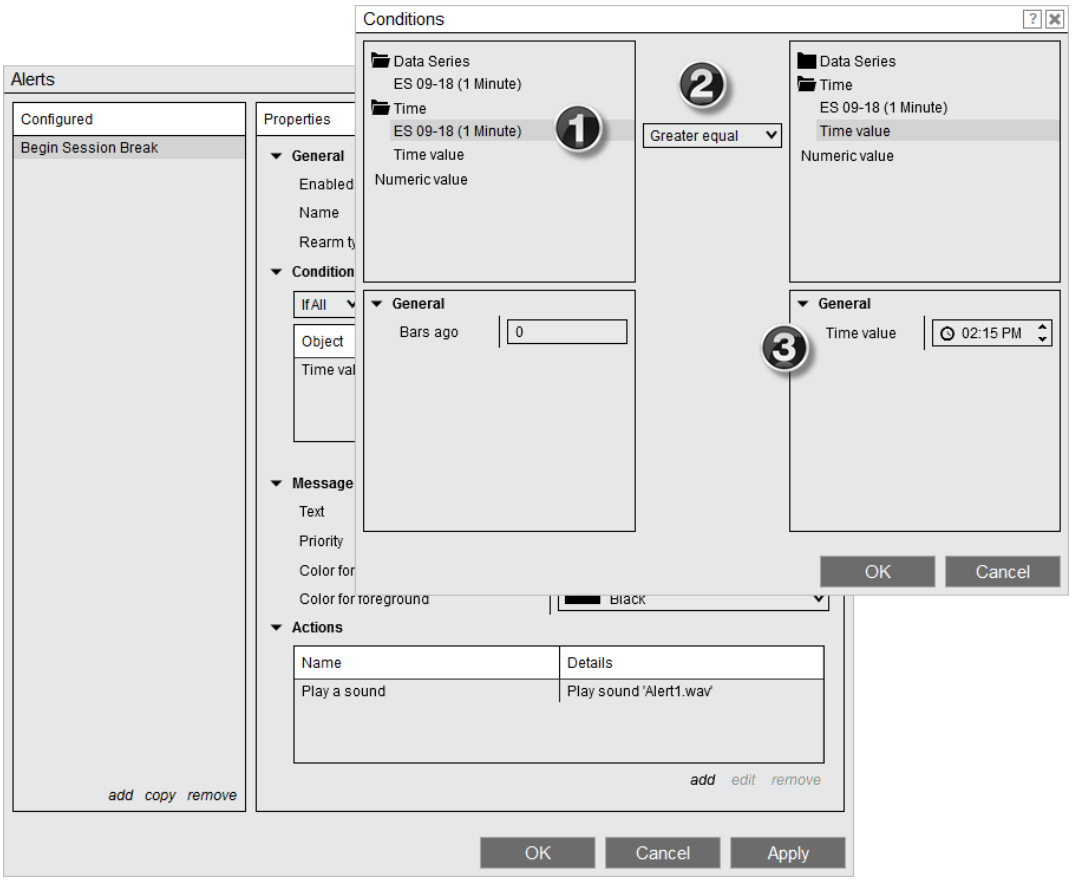
Overview

This alert is based upon the timestamps of bars on a chart. The alert will trigger when the timestamp of the current bar is greater equal to 2:15pm. The image below shows the fully configured alert.



Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

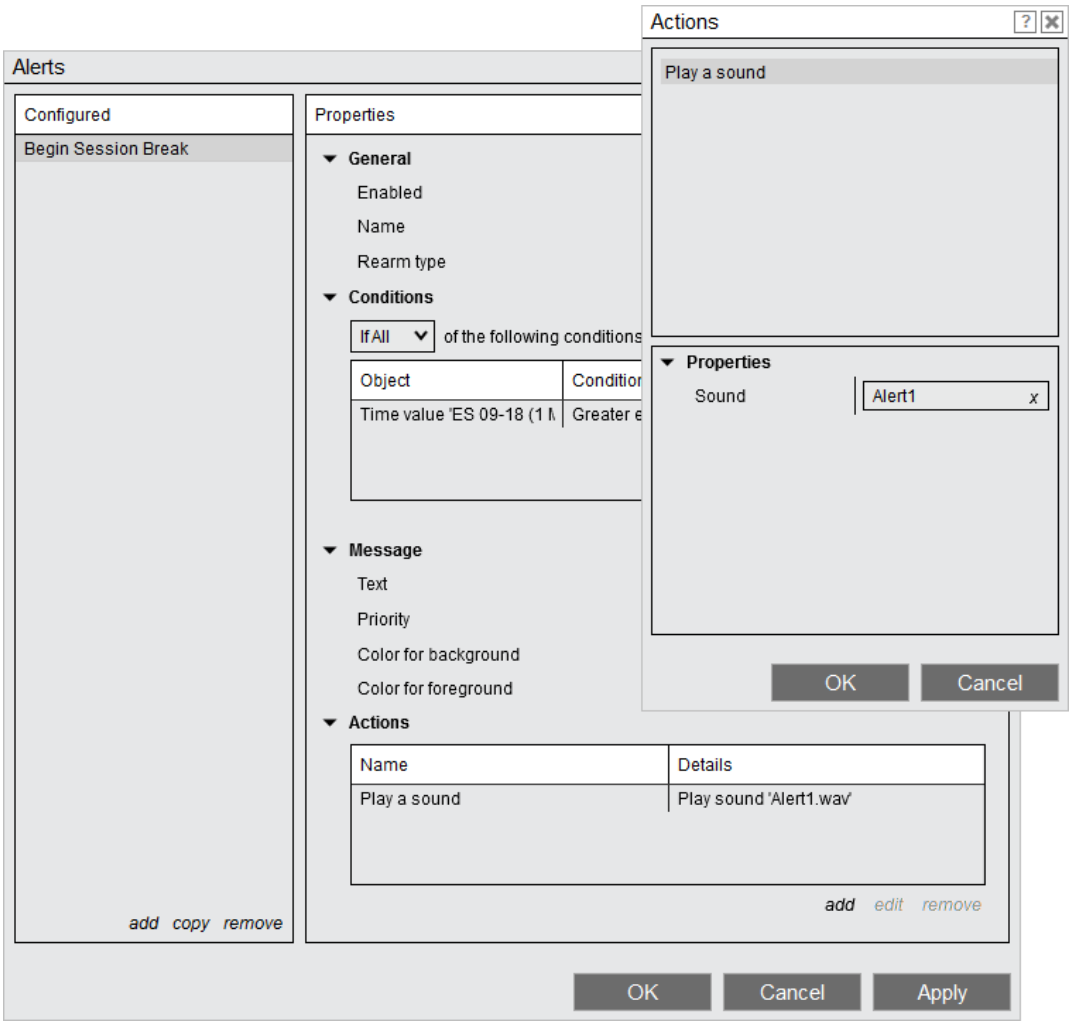


1. The ES 09-18 data series is selected in the left panel under the Time folder

2. The "Greater equal" condition is selected

3. The "Time Value" property is selected in the right panel, and a time of 2:15pm is entered directly below

We now have a condition that translates to **"When the time stamp of the current bar on the ES data series is greater equal to 2:15pm."** This means the alert would trigger at about 2:14:01pm since bars are timestamped on the bar close time.



1. In the Actions window, the "Play a Sound" option is selected

2. A sound named "Alert1" is selected to be played when the alert triggers

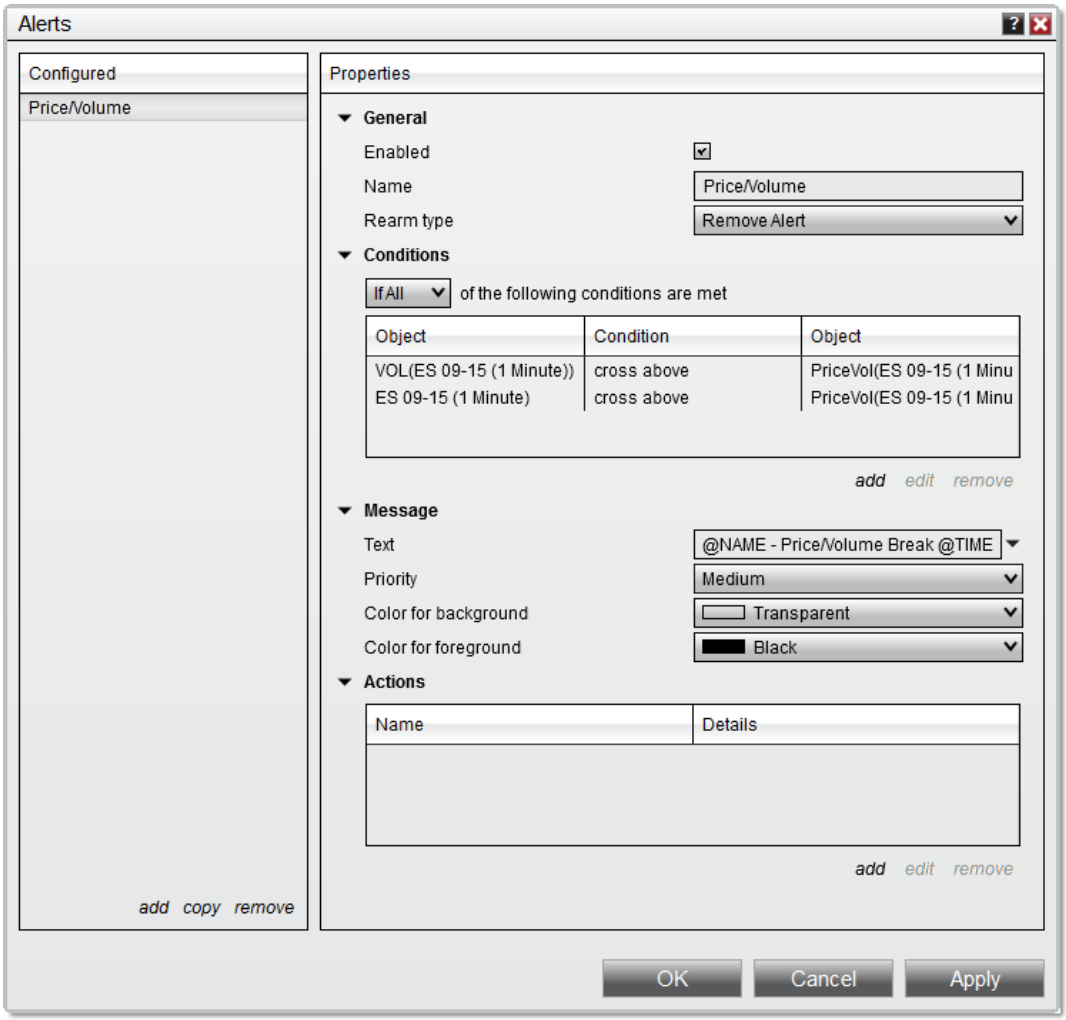
Conditions for custom indicators

Preparation

- Open a chart
- Apply a custom indicator to the chart
  - This example will use a custom indicator which is not pre-loaded in NinjaTrader. You will not have access to the PriceVol indicator in your installation, but this process can be used with any custom indicator that you havee developed

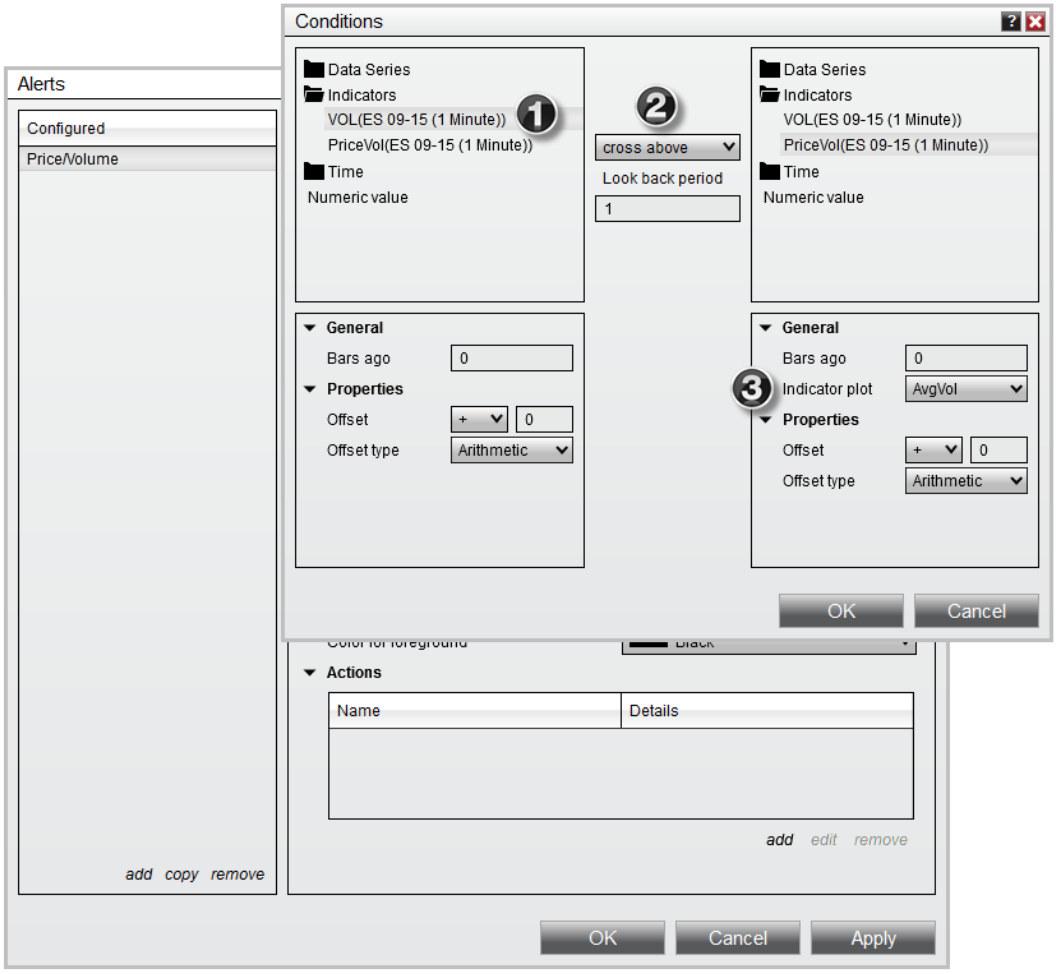
Overview

This alert compares the current market price and the pre-built VOL indicator to different plots of a custom indicator developed with NinjaScript. In this example, the custom indicator is named "PriceVol." This alert will trigger when the current value of the VOL indicator crosses above a historical average of volume calculated by PriceVol, if the current market price is greater than the instrument's 52-week High (also calculated by PriceVol). The image below shows the fully configured alert.



Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

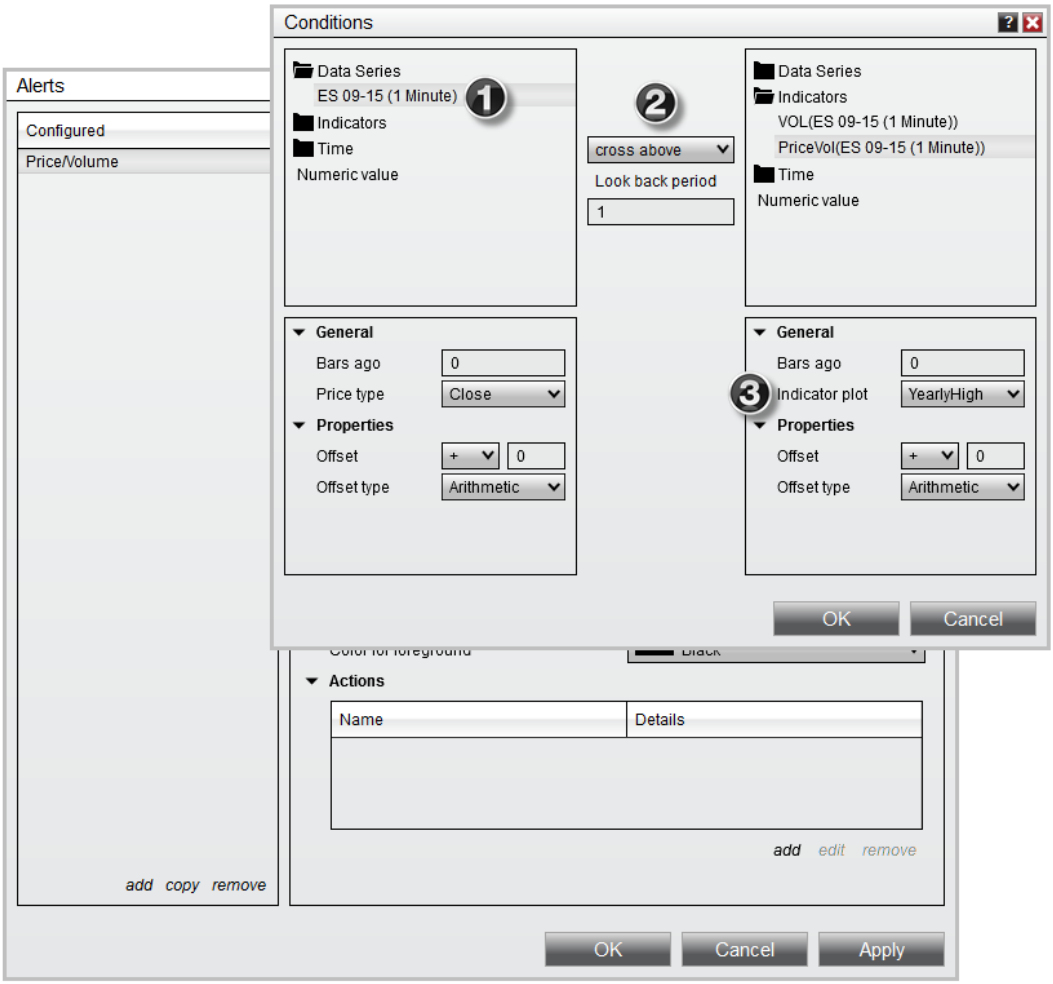


1. In the Conditions window, the VOL indicator is selected in the left panel

2. The Cross Above condition is selected.

3. The AvgVol plot of the PriceVol indicator is selected in the right panel. For this custom indicator, the AvgVol plot contains a 14-period average of volume

We now have a condition that translates to **"When the current volume crosses above the 14-period average of volume."**



1. In the Conditions window for the second condition, the primary data series applied to the chart is selected in the left panel

2. The Cross Above condition is selected, just like the first condition

3. The YearlyHigh plot of the PriceVol custom indicator is selected in the right panel. This contains the 52-week High for the instrument.

We now have a second condition that translates to **"When the current market price crosses above the instrument's 52-week High."**

Since this alert does not define any actions, it will simply display the specified message in the **Alerts Log** window.

10.3 Alerts Log

Alerts Log Overview

The **Alerts Log** window can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting the menu item **Alerts Log**.

The **Alerts Log** window displays triggered user defined alerts. Alert conditions can be defined within Charts, the Market Analyzer window, News window or alerts can be triggered within a custom NinjaScript indicator or strategy.

10.3.1 Using the Alerts Log Window

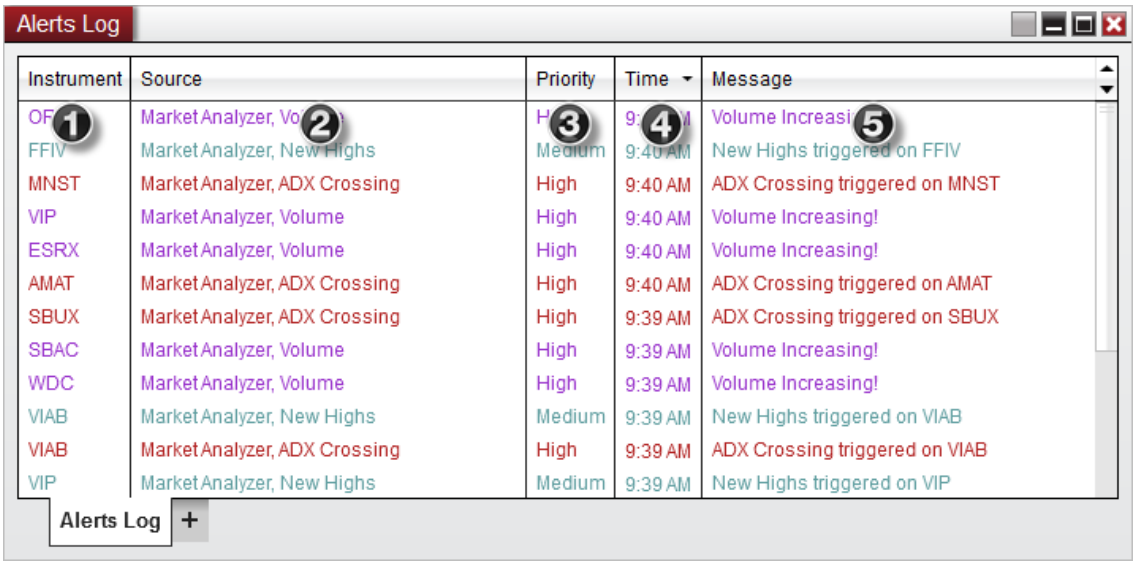
The **Alerts Log** window displays information for each alert that is triggered within NinjaTrader.

▼ Understanding the Alerts Log window

Alerts Log Window Display

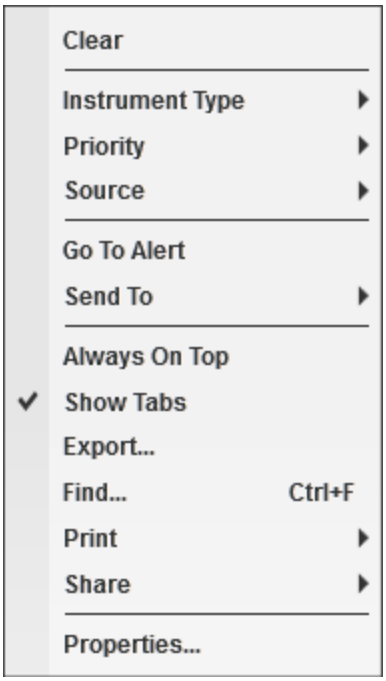
When an alert is triggered, the following information is available in the **Alerts Log** window:

1. Instrument name
2. Source of alert
3. Priority of alert
4. Time of the alert
5. User defined message



Right Click Menu

Right mouse clicking within the **Alerts Log** window will bring up the following menu options:



Clear	Clears the **Alerts Log** history

Instrument Type	Filters alerts by type of instrument
Priority	Filters alerts by user defined priority
Source	Filters alerts by originating source
Go To Alert	Brings source of alert in focus
Send To	Loads the selected instrument into another NinjaTrader window
Always On Top	Sets the **Alerts Log** window to always be on top of other windows
Show Tabs	Set if the window should allow for tabs
Export	Exports the **Alerts Log** contents to "CSV" or "Excel" file format
Find...	Search for a term in the **Alerts Log**
Print	Displays Print options
Share	Displays Share options
Properties	Set the **Alerts Log** properties

Setting up alert filters

Filtering Alerts

By default, all alerts triggered in the workspace will be displayed in the **Alerts Log** window. However, each **Alerts Log** window and tab has the capability to only display certain alerts based on a number of alert attributes.

The following alert filter attributes will be available from the **Alerts Log** right click menu:

Instrument Type	Forex, Future, Index, Option, Stock, CFD
Priority	High, Medium, Low
Source	Chart, Market Analyzer, Hot List Analyzer, News, NinjaScript

To enable or disable these filters, simply right click on the **Alerts Log** window and check or uncheck the attribute you wish to configure.

When checked, only alerts which meet the alert attribute description will be displayed on the current **Alerts Log** window or tab. You can create multiple tabs, or multiple windows to setup varying filters for each attribute you desire to help organize the type of alerts that are displayed.

Using Alerts Logs and multiple workspaces

Finding an Alert

From the **Alerts Log**, you can quickly locate the source window or tab in which the alert was generated.

1. Double clicking an alert entry row will bring the the source window or tab to front and focus.
2. You can also ***right click*** on the alert entry row and select **Go To Alert**.

Alerts Logs in Multiple Workspaces

The default behavior of the **Alerts Log** is to only receive alerts from its parent workspace. However, you can configure an individual **Alerts Log** window or tab to receive alerts from other workspaces by right clicking on the **Alert Log**, selecting **Properties** and checking Receive alerts from all active workspaces (Please note that NinjaScript objects are excluded from this check).

With this configuration, should you attempt to **Go To Alert** which was generated from another workspace, you will receive a prompt asking if you would like to navigate to the source workspace. Selecting Yes on this prompt will then switch the active workspace and set the source window or tab into view.

10.3.2 Alerts Log Properties

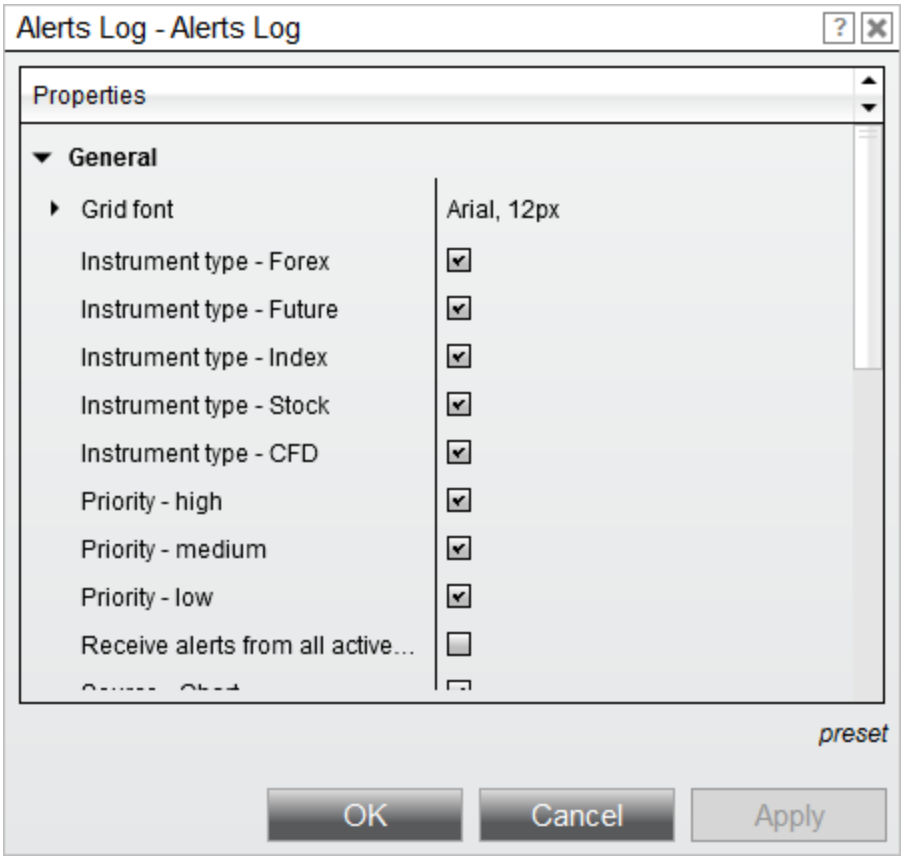
The **Alerts Log** window can be customized through the **Alerts Log Properties** window.

How to access the Alerts Log Properties window

You can access the Alerts Log properties dialog window by clicking on your right mouse button and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the Alerts Log Properties window:



Property Definitions

General	

Grid font	Sets the font options
Instrument type - Forex	Filter by Forex instrument type
Instrument type - Future	Filter by Future instrument type
Instrument type - Index	Filter by Index instrument type
Instrument type - Option	Filter by Option instrument type
Instrument type - Stock	Filter by Stock instrument type
Instrument type - CFD	Filter by CFD instrument type
Priority - high	Filter by high priority
Priority - medium	Filter by medium priority
Priority - low	Filter by low priority
Receive alerts from all active workspaces	Sets if the window receives alerts from other active workspaces.
Source - Chart	Filter from alerts triggered by Charts
Source - Market Analyzer	Filter from alerts triggered by the Market Analyzer
Source - Hot List Analyzer	Filter from alerts triggered by the Hot List Analyzer
Source - News	Filter from alerts triggered by the News Window

Source - NinjaScript	Filter from alerts triggered from custom NinjaScript files
Time format	Sets the format used for the time column
Tab name	Sets the name of the tab, please see Managing Tabs for more information.
Columns	
Instrument	Sets if the Instrument name column is displayed
Instrument type	Sets if the Instrument type column is displayed
Message	Sets if the Message column is displayed
Priority	Sets if the Priority column is displayed
Source	Sets if the Source column is displayed
Time	Sets if the Time column is displayed
Window	
Always on top	Sets if the window will be always on top of other windows.

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

10.3.3 Window Linking

Please refer to the Window Linking section of the Help Guide for more information on linking chart windows. Many windows in NinjaTrader can be linked by instrument allowing synchronous changing of instruments and / or intervals in all linked windows.

10.4 Automated Trading

Automated Trading Overview

NinjaTrader provides methods for automated trading through NinjaScript or from an outside source via the Automated Trading Interface (ATI).

- ATI Interface
- File Interface
- DLL Interface
- TradeStation Integration
- Running NinjaScript Strategies

10.4.1 Automated Trading Interface (ATI)

You can enable the AT Interface on under Automated trading interface category from the General Options menu.

Automated Trading Interface (ATI) Overview

NinjaTrader's ATI (Automated Trading Interface) provides efficient protocols to communicate trading signals from various external sources to NinjaTrader for the automation of order execution.

- Popular charting applications such as but not limited to TradeStation, eSignal, NeoTicker, and Investor RT
- Custom applications written in but not limited to Visual Studio .NET, Visual Basic, Delphi, and MS Excel
- Black box trading systems

NinjaTrader provides methods for automated trading through NinjaScript or from an outside source via the Automated Trading Interface (ATI).

Note: This interface is **ONLY** used for processing trade signals generated from **external applications** and is **NOT** a full blown brokerage/market data API. If you are interested in automated trading using native NinjaScript strategies please proceed to the following help guide section.

10.4.1.1 What can I do and how?

What can I do through automation?

- Place orders
- Initiate a NinjaTrader ATM Strategy
- Change orders
- Cancel orders
- Close ATM Strategies and positions
- Flatten accounts
- Cancel all orders
- Retrieve information on positions and orders

NinjaTrader provides three options for communicating from an external application to NinjaTrader for trade automation. The Email Interface requires absolutely no programming experience whatsoever, other options require various levels of programming/scripting experience.

Understanding the three interface options

TradeStation Email Interface

The TradeStation Email Interface allows you to take advantage of TradeStation's email notification capabilities right out of the box. Run your TradeStation strategy in real time, order signals are emailed within your computer (never leaves your PC) to NinjaTrader which processes the order through to your broker.

File Interface

The File interface uses standard text files as input. These files are called order instruction files (OIF) and have specific format requirements. NinjaTrader processes the OIF the instant the file is written to the hard drive and subsequently deletes the file once the processing operation is complete.

DLL Interface

NinjaTrader provides a DLL named NtDirect.dll that supports various functions for automated trading.

Which interface option should I use?

TradeStation Systems

- If you are not running your own strategies or you have limited or no programming experience you should use the TradeStation Email Interface
- If you are running your own system and you are comfortable with EasyLanguage and want to have bi-directional control of your real-time order processing you should use the DLL interface.

Other Charting Applications

- You should use the DLL if your charting application supports that interface type or use the File Interface

Custom Applications

- You should use the DLL interface

10.4.1.2 Commands and Valid Parameters

The following section is only relevant for the File and DLL interfaces. Both interfaces share common interface functions/methods that take as arguments the parameters defined in the tables below. You can automate your trading through eight different commands. Command definitions are also provided below.

▼ Understanding parameters and valid values

Available Parameters and Valid Values

Parameters	Values
COMMAND	CANCEL, CANCELALLORDERS, CHANGE, CLOSEPOSITION, CLOSESTRATEGY, FLATTENEVERYTHING, PLACE, REVERSEPOSITION
ACCOUNT	The name of the account the command is to be processed (This will reflect the Account's **Name** property as opposed to the **Display Name** property.)
INSTRUMENT	Instrument name
ACTION	BUY, SELL
QTY	Any integer value
ORDER TYPE	MARKET, LIMIT, STOPMARKET, STOPLIMIT
LIMIT PRICE	Any decimal value (use decimals not commas 1212.25 for example)
STOP PRICE	Any decimal value

TIF	DAY, GTC
OCO ID	Any string value
ORDER ID	Any string value (must be unique for each line/file)
STRATEGY	Strategy template name (must exist in NinjaTrader)
STRATEGY ID	Any string value (must be unique for each line/file)

Understanding the parameters available to commands

Available Commands

The following table displays required (R) and optional (O) values for each different command value.

Command	Account	Instrument	Actions	Qty	OrderTYpe	LimitPrice	StopPrice	TIF	OCOID	OrderID	Strategy	StrategyID
CANCEL										R		O
CANCELALLO												

RDERS												
CHANGE				O		O	O			R		O
CLOSEPOSITION	R	R										
CLOSESTRATEGY												R
FLATTENEVERYTHING												
PLACE	R	R	R	R	R	O	O	R	O	O	O	O

REVERSEPOSITION	R	R	R	R	R	O	O	R	O	O	O	O

Understanding the commands

Following are the descriptions of each available command.

CANCEL COMMAND

This command will cancel an order and requires an order ID value and an optional strategy ID value. The order ID value must match either the order ID value given to an order placed through the PLACE command or, an order name such as ENTRY*, EXIT*, STOP*, SIMSTOP* or TARGET*. The star (*) represents an integer value such as TARGET1 or TARGET2. Order names are only valid if a valid strategy ID value is passed. The strategy ID value must match a strategy ID value given to a strategy in the PLACE command.

CANCELALLORDERS COMMAND

This command will cancel all active orders across all accounts and broker connections.

CHANGE COMMAND

This command will change the parameters of an order and requires an order ID value, optional price and quantity values and an optional strategy ID value. The order ID value must match either the order ID value given to an order placed through the PLACE command or, an order name such as ENTRY*, EXIT*, STOP*, SIMSTOP* or TARGET*. The star (*) represents an integer value such as TARGET1 or TARGET2. Order names are only valid if a valid strategy ID value is passed. Pass in zero (0) values for price and quantity if you do not wish to change these order parameters. Price values must be in US decimal format (1212.25 is correct while 1212,25 is not).

CLOSEPOSITION COMMAND

This command will close a position and requires an account name value and an instrument name value. The instrument name value is the name of the NinjaTrader instrument including the exchange name. For equities, the symbol is sufficient. This command will cancel any working orders and flatten the position.

CLOSESTRATEGY COMMAND

This command will close an ATM Strategy and requires a strategy ID value. The strategy ID value must match a strategy ID given to a strategy in the PLACE command. This command will close the specified strategy.

FLATTENEVERYTHING COMMAND

This command will cancel all active orders and flatten all positions across all accounts and broker connections.

PLACE COMMAND

This command will place orders, place orders that initiate a NinjaTrader ATM Strategy, or place orders that are applied to an active NinjaTrader position ATM Strategy. Providing the optional strategy name field with a valid ATM Strategy template name will result in execution of that ATM Strategy once the order is partially or completely filled. Pass in an optional unique string value for the strategy ID in order to reference that ATM Strategy via other commands. To apply an order to an active ATM Strategy (existing strategies Stop Loss and Profit Target orders are amended) pass in the active strategy ID value and leave the strategy name field blank. Pass in an optional unique string value for the order ID in order to reference that order via other commands. If specifying an ATM Strategy template name, there is no need to pass in an order ID as the strategy based orders can be referenced by their internally generated names such as TARGET1, STOP1 and so on.

REVERSEPOSITION COMMAND

This command will close the current position and place an order in the opposite direction. The field requirements are identical to the PLACE command.

10.4.1.3 Initialization

If using the DLL based interface, it is important to understand how the ATI is initialized with respect to referencing account names. The ATI is initialized to the first account name used in the first calling function.

Some functions accept an account name as a parameter. In most if not all functions, these parameters can be left blank in which case the "Default" account will be used. You can set the Default account by left mouse clicking on the Tools menu in the NinjaTrader Control

Center and selecting the menu item **Options**, once in the Options window select the Automated trading interface category and select the account you want to use from the Default account menu. If your default account is set to 'Sim101' and you call functions and leave the account parameter blank, the Sim101 account will be automatically used.

Example:

- Default account = Sim101
- A function call is made with "" empty string as the account name argument
- Sim101 account is automatically used
- Subsequent function calls must use empty string if you want to reference the Sim101 account
- If you call a function and pass in the argument "Sim101", invalid information will be returned

10.4.1.4 File Interface

File Interface Overview

The **File** interface is an easy way you can instruct NinjaTrader to place and manage orders. To use this interface, just create Order Instruction Files (OIFs) in "My Documents\<NinjaTrader Folder>\incoming" and when NinjaTrader sees the instructions they will be processed immediately. This interface allows you the flexibility to create order instructions to NinjaTrader from any application that allows you to create text files.

› Order Instruction Files (OIF)
› Information Update Files

10.4.1.4.1 Order Instruction Files (OIF)

OIFs must be written to the folder "My Documents\<NinjaTrader Folder>\incoming" and be named oif*.txt. You can simply send an oif.txt file however, it is suggested that you increment each OIF so that you end up with unique file names such as oif1.txt, oif2.txt, oif3.txt. The reason is that if you send a lot of OIFs in rapid succession, you do run the risk of file locking problems if you always use the same file name. This will result in a situation where your file is not processed.

Each file must also contain correctly formatted line(s) of parameters. You may stack the instruction lines so that each file contains as many instruction lines as you desire. The delimiter required is the semicolon and this section is a good reference for generating correctly formatted OIF. Files are processed the instant they are written to the hard disk without delay.

Please reference the Commands and Valid Parameters section for detailed information on available commands and parameters.

Warning: Move or directly write OIF files to the incoming folder. Copying OIF files to the incoming folder can cause file locking problems.

The following are examples of the required format for each of the available commands. Required fields are embraced by <> where optional fields are embraced by [].

CANCEL COMMAND
```
CANCEL;;;;;;;;;;<ORDER ID>;;[STRATEGY ID]
```

CANCELALLORDERS COMMAND
```
CANCELALLORDERS;;;;;;;;;;;;
```

CHANGE COMMAND
```
CHANGE;;;;<QUANTITY>;;<LIMIT PRICE>;<STOP PRICE>;;;<ORDER ID>;;[STRATEGY ID]
```

CLOSEPOSITION COMMAND
```
CLOSEPOSITION;<ACCOUNT>;<INSTRUMENT>;;;;;;;;;;
```

CLOSESTRATEGY COMMAND
```
CLOSESTRATEGY;;;;;;;;;;;;<STRATEGY ID>
```

FLATTENEVERYTHING COMMAND
```
FLATTENEVERYTHING;;;;;;;;;;;;
```

PLACE COMMAND
```
PLACE;<ACCOUNT>;<INSTRUMENT>;<ACTION>;<QTY>;<ORDER TYPE>;[LIMIT PRICE];[STOP
PRICE];<TIF>;[OCO ID];[ORDER ID];[STRATEGY];[STRATEGY ID]
```

REVERSEPOSITION COMMAND
```
REVERSEPOSITION;<ACCOUNT>;<INSTRUMENT>;<ACTION>;<QTY>;<ORDER TYPE>;[LIMIT PRICE];[STOP
PRICE];<TIF>;[OCO ID];[ORDER ID];[STRATEGY];[STRATEGY ID]
```

10.4.1.4.2 Information Update Files

NinjaTrader provides update information files that are written to the folder "My Documents\<NinjaTrader Folder>\outgoing". The contents of this folder will be deleted when the NinjaTrader application is restarted.

▼ Understanding order state files

Order State Files

Orders that are assigned an order ID value in the "PLACE" command will generate an order state update file with each change in order state. The file name is 'orderId.txt' where orderId is the order ID value passed in from the "PLACE" command. Possible order state values can be found here. The format of this file is:

Order State;Filled Amount;Average FillPrice

Understanding position update files

Position Update Files

Position update files are generated on every update of a position. The name of the file is Instrument Name + Instrument Exchange_AccountName_Position.txt. An example would be ES 0914 Globex_Sim101_Position.txt. The format of the file is:

Market Position; Quantity; Average Entry Price

Valid Market Position values are either LONG, SHORT or FLAT.

Understanding connection state files

Connection State Files

Connection state files are written with each change of connection state. The name of the file is ConnectionName.txt where connectionName is the name of the connection given in the Connection Manager. The format of the file is:

```
Connection State
```

Valid connection state values are CONNECTED or DISCONNECTED.

10.4.1.5 DLL Interface

DLL Functions Overview

The .net managed DLL Interface functions are contained in NTDirect.dll located in the C:\Program Files(X86)\NinjaTrader 8\bin\NinjaTrader.Client.DLL.

› Functions

10.4.1.5.1 Functions

DLL Interface Functions

int Ask(string *instrument*, double *price*, int *size*)
Sets the ask price and size for the specified instrument. A return value of 0 indicates success and -1 indicates an error.

int AskPlayback(string *instrument*, double *price*, int *size*, string *timestamp*)
Sets the ask price and size for the specified instrument for use when synchronizing NinjaTrader playback with an external application playback. A return value of 0 indicates success and -1 indicates an error. The *timestamp* parameter format is "yyyyMMddHHmmss".

double AvgEntryPrice(string *instrument*, string *account*)
Gets the average entry price for the specified instrument/account combination.

double AvgFillPrice(string *orderId*)
Gets the average entry price for the specified orderId.

int Bid(string *instrument*, double *price*, int *size*)
Sets the bid price and size for the specified instrument. A return value of 0 indicates success and -1 indicates an error.

int BidPlayback(string *instrument*, double *price*, int *size*, string *timestamp*)
Sets the bid price and size for the specified instrument for use when synchronizing NinjaTrader playback with an external application playback. A return value of 0 indicates success and -1 indicates an error. The *timestamp* parameter format is "yyyyMMddHHmmss".

double BuyingPower(string *account*)
Gets the buying power for the specified account. *Not all brokerage technologies support this value.

double CashValue(string *account*)
Gets the cash value for the specified account. *Not all brokerage technologies support this value.

**int Command(string *command*, string *account*, string *instrument*, string *action*, int *quantity*, string *orderType*, double *limitPrice*, double *stopPrice*,
string *timeInForce*, string *oco*, string *orderId*, string *strategy*, string *strategyId*)**
Function for submitting, cancelling and changing orders, positions and strategies. Refer to the Commands and Valid Parameters section for detailed information. The Log tab will list context sensitive error information.

int ConfirmOrders(int *confirm*)
The parameter confirm indicates if an order confirmation message will appear. This toggles the global option that can be set manually in the NinjaTrader Control Center by selecting the

Tools menu and the menu item **Options**, then checking the "Confirm order placement" checkbox. A value of 1 sets this option to true, any other value sets this option to false.

```
int Connected(int showMessage)
```
Returns a value of zero if the DLL has established a connection to the NinjaTrader server (application) and if the ATI is currently enabled or, -1 if it is disconnected. Calling any function in the DLL will automatically initiate a connection to the server. The parameter *showMessage* indicates if a message box is displayed in case the connection cannot be established. A value of 1 = show message box, any other value = don't show message box.

```
int Filled(string orderId)
```
Gets the number of contracts/shares filled for the orderId.

```
int Last(string instrument, double price, int size)
```
Sets the last price and size for the specified instrument. A return value of 0 indicates success and -1 indicates an error.

```
int LastPlayback(string instrument, double price, int size, string timestamp)
```
Sets the last price and size for the specified instrument for use when synchronizing NinjaTrader playback with an external application playback. A return value of 0 indicates success and -1 indicates an error. The *timestamp* parameter format is "yyyyMMddHHmmss".

```
double MarketData(string instrument, int type)
```
Gets the most recent price for the specified instrument and data type. 0 = last, 1 = bid, 2 = ask. You must first call the SubscribeMarketData() function prior to calling this function.

```
int MarketPosition(string instrument, string account)
```
Gets the market position for an instrument/account combination. Returns 0 for flat, negative value for short positive value for long.

```
string NewOrderId()
```
Gets a new unique order ID value.

```
string Orders(string account)
```
Gets a string of order ID's of all orders of an account separated by '|'. *If a user defined order ID was not originally provided, the internal token ID value is used since it is guaranteed to be unique.

```
string OrderStatus(string orderId)
```
Gets the order state (see definitions) for the orderId. Returns an empty string if the order ID value provided does not return an order.

```
double RealizedPnL(string account)
```
Gets the realized profit and loss of an account.

```
int SetUp(string host, int port)
```

Optional function to set the host and port number. By default, host is set to "localhost" and port is set to 36973. The default port number can be set via the General tab under Options. If you change these default values, this function must be called before any other function. A return value of 0 indicates success and -1 indicates an error.

```
string StopOrders(string strategyId)
```
Gets a string of order ID's of all Stop Loss orders of an ATM Strategy separated by '|'. Internal token ID value is used since it is guaranteed to be unique.

```
string Strategies(string account)
```
Gets a string of strategy ID's of all ATM Strategies of an account separated by '|'. Duplicate ID values can be returned if strategies were initiated outside of the ATI.

```
int StrategyPosition(string strategyId)
```
Gets the position for a strategy. Returns 0 for flat, negative value for short and positive value for long.

```
int SubscribeMarketData(string instrument)
```
Starts a market data stream for the specific instrument. Call the MarketData() function to retrieve prices. Make sure you call the UnSubscribeMarketData() function to close the data stream. A return value of 0 indicates success and -1 indicates an error.

```
string TargetOrders(string strategyId)
```
Gets a string of order ID's of all Profit Target orders of an ATM Strategy separated by '|'. Internal token ID value is used since it is guaranteed to be unique.

```
int TearDown()
```
Disconnects the DLL from the NinjaTrader server. A return value of 0 indicates success and -1 indicates an error.

```
int UnsubscribeMarketData(string instrument)
```
Stops a market data stream for the specific instrument. A return value of 0 indicates success and -1 indicates an error.

10.4.1.6 TradeStation Email Integration

The TradeStation Email Interface is targeted toward individuals who are familiar with programming in EasyLanguage and want to run TradeStation strategies and automate order flow to any supported NinjaTrader broker.

The interface works as follows:

1. You apply a strategy in your TradeStation chart that generates buy/sell orders
2. TradeStation will send email notification for Strategy Orders Activated, Filled, Canceled and Replaced to NinjaTrader
3. NinjaTrader will process these emails and execute them as orders either to the NinjaTrader simulator or your live brokerage account

Email Interface

10.4.1.6.1 Running concurrent strategies in the same market

NinjaTrader uses a number of different properties in the TradeStation generated email to identify unique orders as they are sent to NinjaTrader.

These properties include

- Instrument name
- Action (Buy, Sell etc...)
- Signal name
- Workspace name

If you are running concurrent strategies on the same market you should ensure that you either

- Make all signal names unique or
- Run the concurrent strategies in different TradeStation workspaces

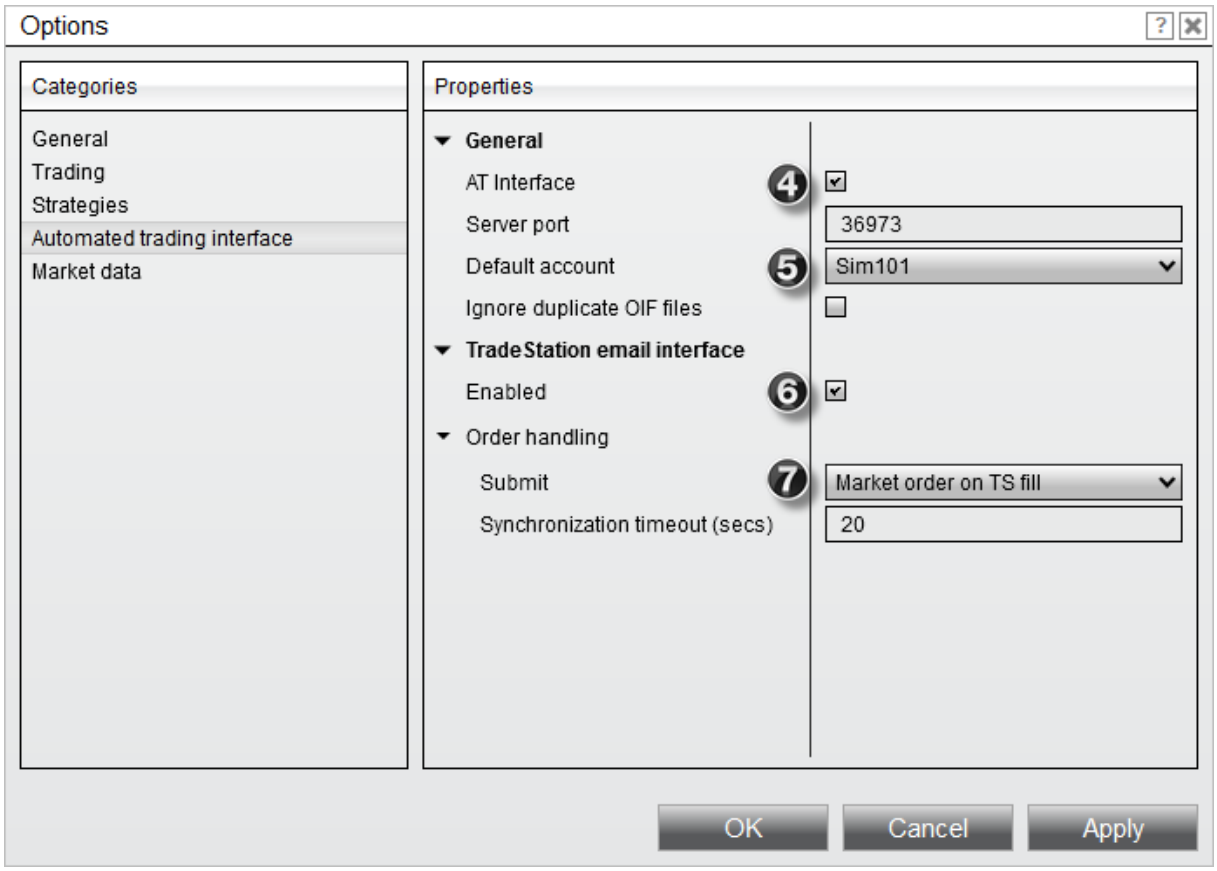
This will ensure accurate processing of your automated signals.

10.4.1.6.2 Set Up

The following set up is for TradeStation Version 9.XX. This section will walk you through the set up in both NinjaTrader and TradeStation as well as allow you to send a test email through the Email Interface you have created.

Setting up NinjaTrader

1. Start NinjaTrader
2. Select the Tools menu and then the menu item Options from the Control Center window
3. Once in the Options window select the Automated trading interface category



4. Ensure that **AT Interface** has been checked
5. Set the default account to Sim101 (you can always set this to your live brokerage account later but we recommend leaving it to Sim101)
6. Check the "**Enabled**" option under the **TradeStation email interface** category
7. Set your Order Handling options
8. Connect to your broker by selecting the File menu and then the menu item **Connect** within the Control Center window (make sure you have set up a connection to your broker)

Note: If upon restarting the SMTP server does not initialize, go to **Tools** --> **Options** and click **OK**

Symbol Mapping for Futures Contracts (Stocks and Forex traders may skip this step)

9. Set your symbol mapping for futures contracts

Setting Up Antivirus Software

10. Antivirus software which scans outgoing emailing can impair the link between TradeStation and NinjaTrader. If your PC has Antivirus software installed and scans

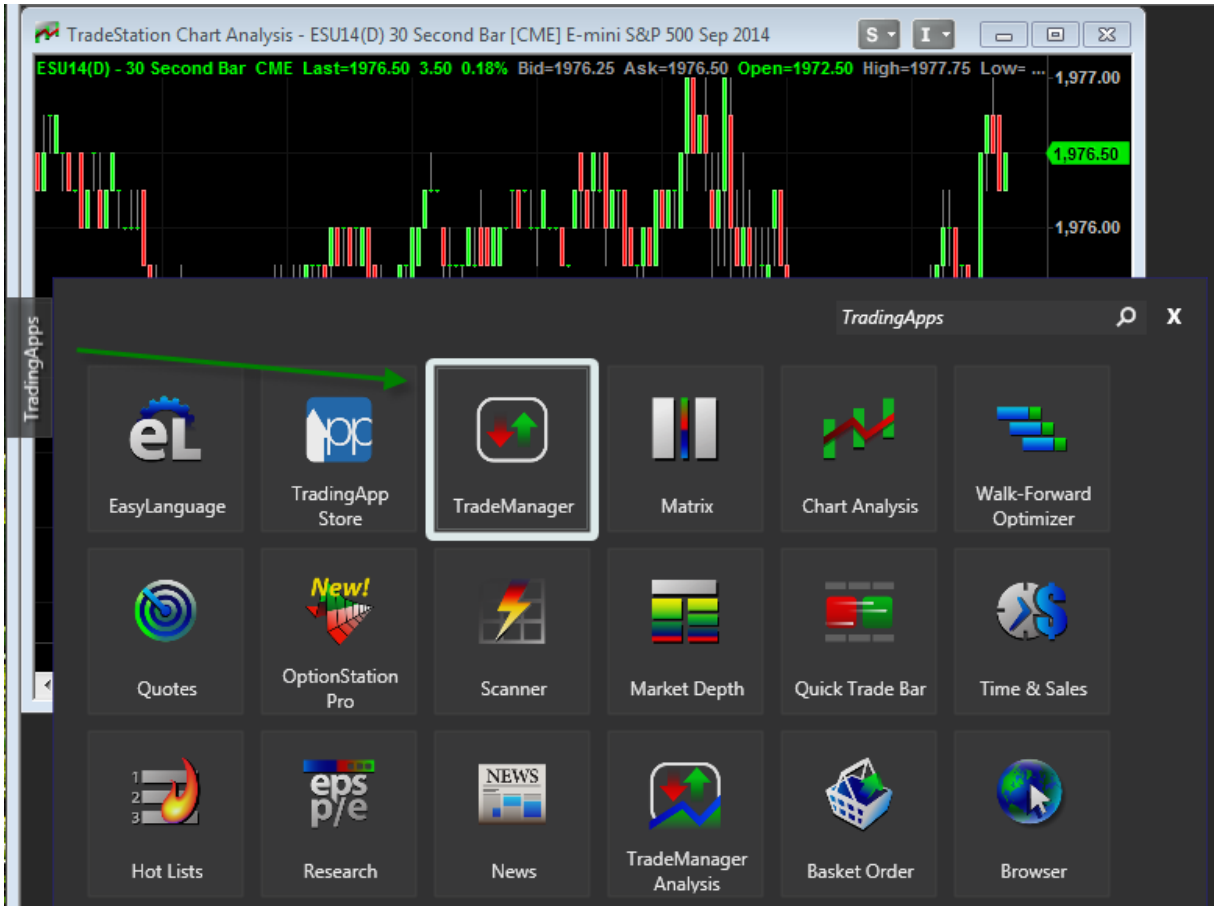
outgoing mail, each mail notification sent from TradeStation to NinjaTrader will be scanned and therefore add significant delay in automatically processing your trading signals. Please consult your Antivirus software Help Guide to determine how to disable the scanning of outgoing email.

Setting Up TradeStation Workspace

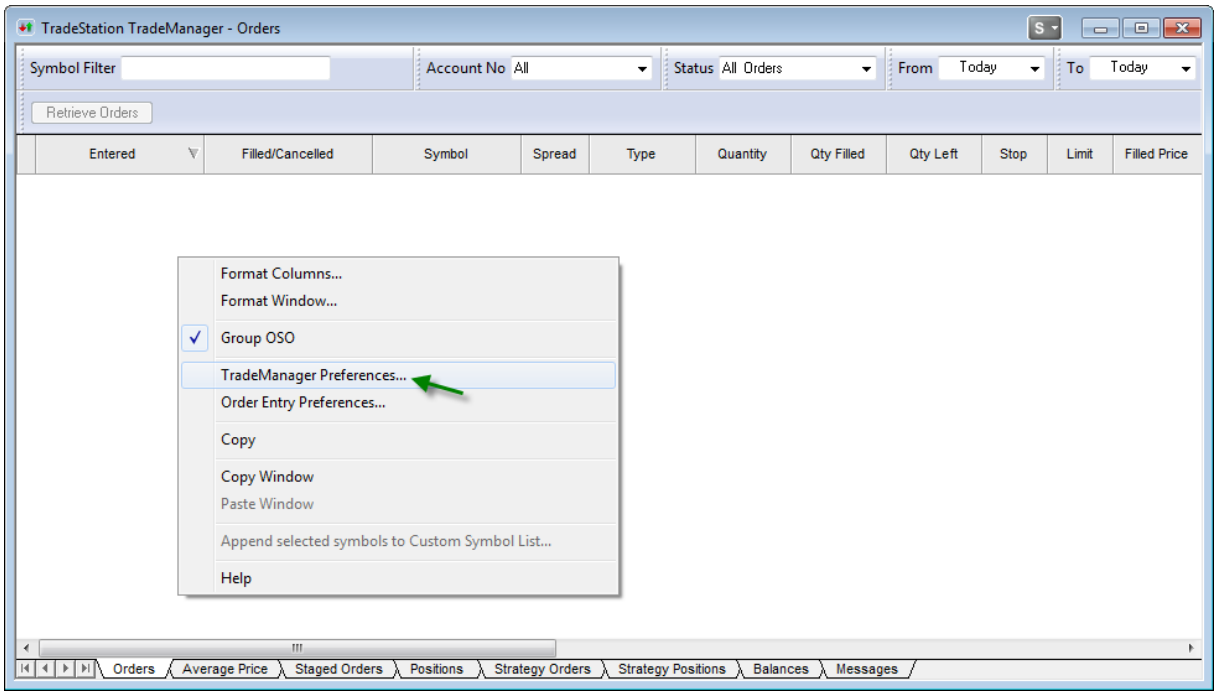
11. Start TradeStation
12. Set up your workspace options

Setting Up TradeStation Email Notification

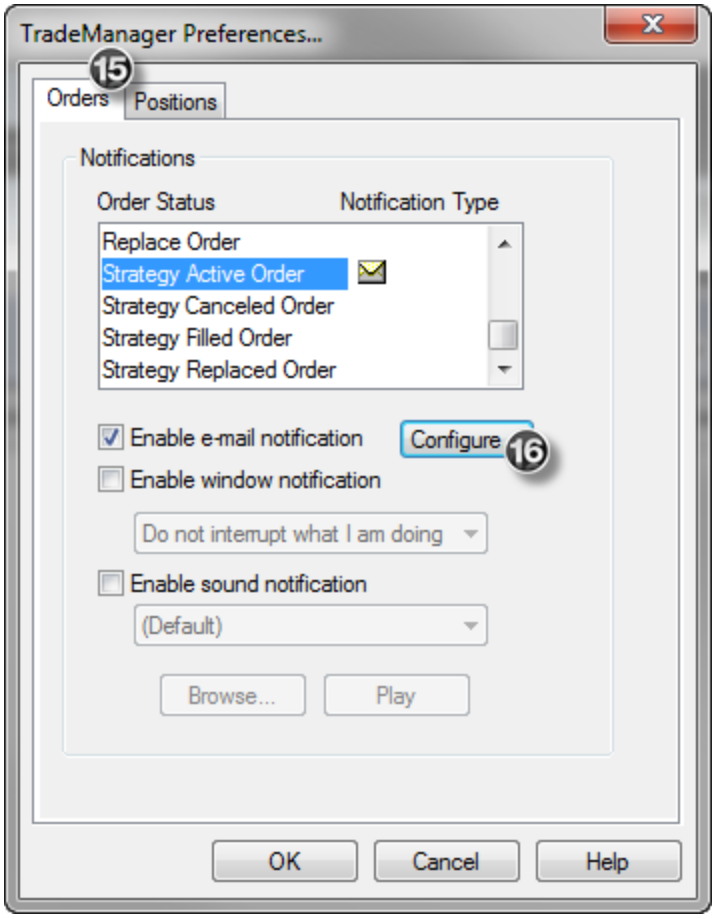
13. Add a **TradeManager** window to your workspace by clicking on the TradingApps panel on left pane as per the image below.



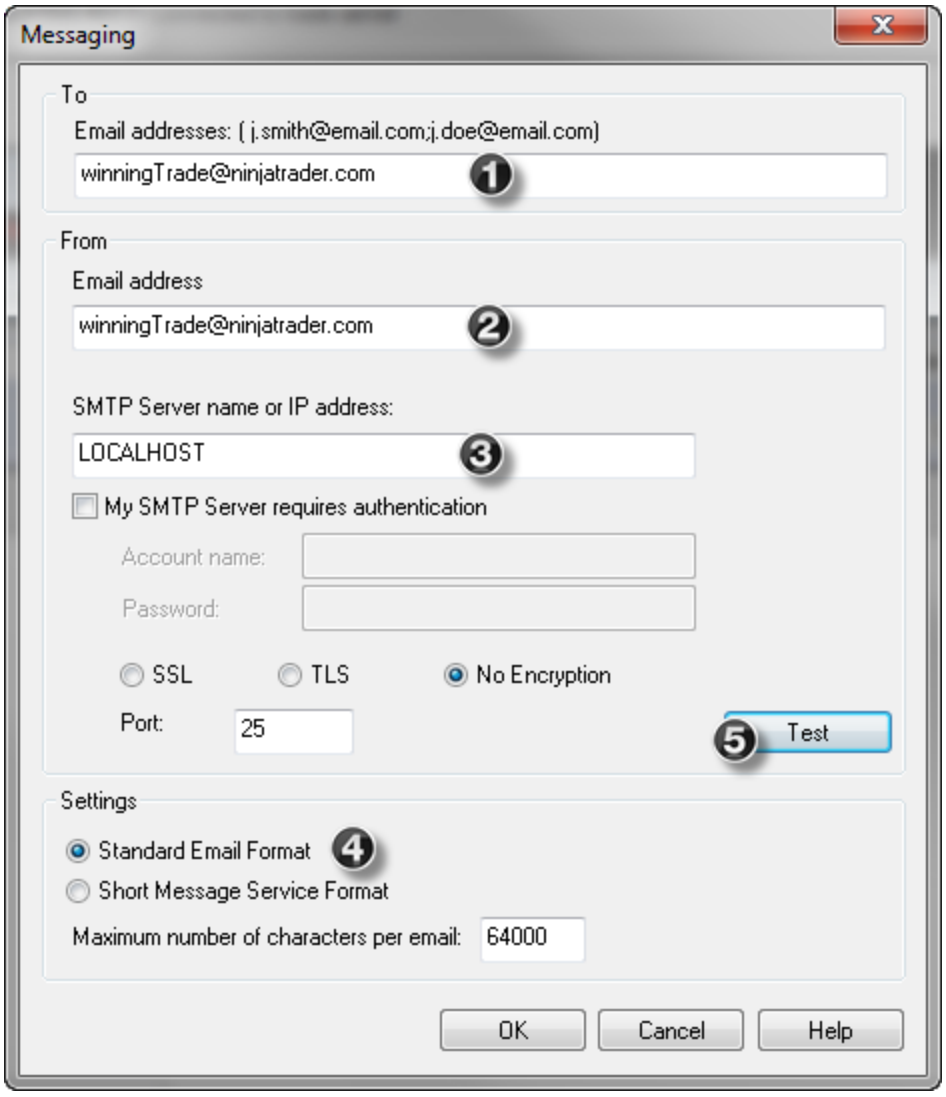
14. Once the TradeManager window appears, right click on this window, and then select the menu name TradeManager preferences



15. Select the "Orders" tab as per the image below and then select "Strategy Active Order"



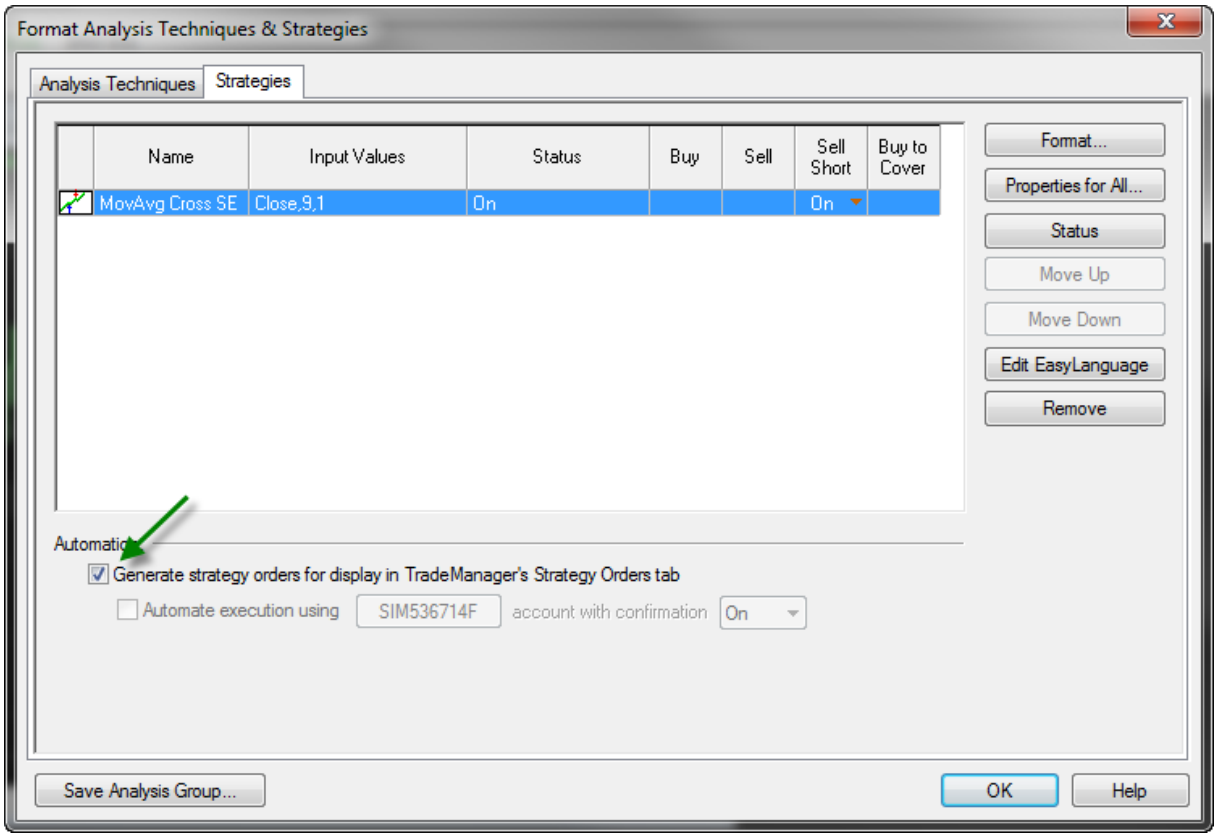
16. Press the "Configure..." button to bring up the "Messaging" window



17. Enter the information exactly as shown above in items 1 through 4; you can press the "Test" button which will send a test message to NinjaTrader and show up in the Control Center Log tab. If you receive an error when attempting to send a test message, please ensure that you have no other SMTP server running on your PC and make sure that any competitive products are uninstalled.
18. Press "OK"
19. Repeat steps 15 through 17 for "Strategy Canceled Order", "Strategy Filled Order" and "Strategy Replaced Order"

Setting Up a TradeStation Strategy

20. Open a chart(s) of the instrument that you will run your strategy on
21. Right click in the chart and select the menu name "**Insert Strategy...**" and select a strategy



22. Your strategy will appear in the "Format Analysis Techniques & Strategies" window as shown above
23. Check the "Generate strategy orders for display in TradeManager's Strategy Orders tab" box and press "Close"

> **Note:** Following this set up procedure, orders will **NOT** be sent to any live TS brokerage account, only to NinjaTrader.

That's it! Your strategy will now be automated for execution through NinjaTrader!

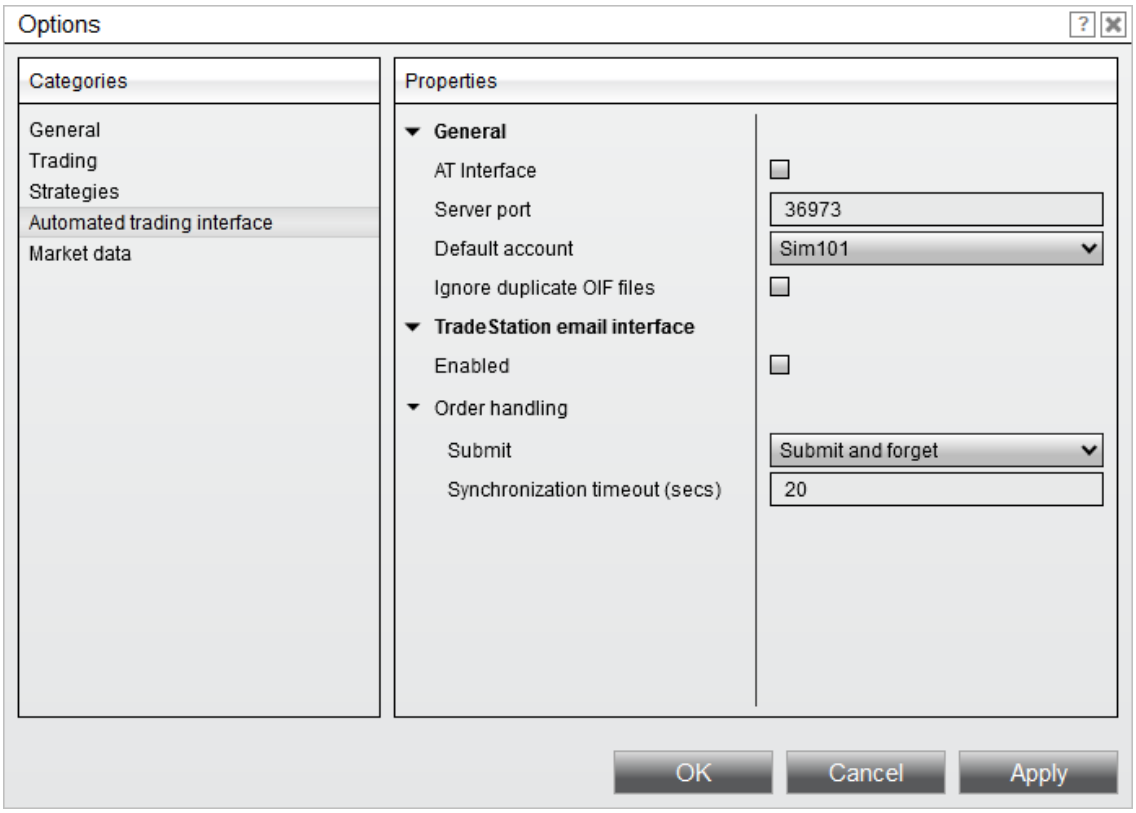
10.4.1.6.3 Symbol Mapping

Please see the TradeStation Symbol Mapping section.

10.4.1.6.4 Order Handling Options

There are several Order Handling options available for the signals sent from TradeStation. All Order Handling options are available by selecting the Tools menu in the Control Center, selecting the menu name **Options**, and then selecting the

Automated trading interface category. Please review all of the following Order Handling options to ensure your orders are managed as expected.



Understanding submit market order on TradeStation fill

Submit market order on TS fill
Submits a market order when NinjaTrader receives a "strategy filled order" notification email from TradeStation. This is the recommended option.

Understanding submit "as-is"

Submit "as-is"
Submits orders as specified (limit, market, stop, stop-limit) when NinjaTrader receives a "strategy active order" notification email from TradeStation. Upon receiving the subsequent "strategy filled order" notification email from TradeStation, NinjaTrader will convert any unfilled shares/contracts to either market order or marketable limit order (substantially higher than inside market if

buying or below market if selling) depending on the instrument type after a user defined number of seconds.

> **Note**: If trading currencies (**Forex**) it is advised to start a market data stream (any order entry window) for the market you are trading. Since limit buy orders above the offer or limit sell below the bid are invalid orders that are **rejected** from your broker, NinjaTrader will check the TradeStation requested limit price against the current market price and if it would result in a rejected order, it will convert to a market order.

Understanding submit and forget

Submit and forget

Submits orders as specified (limit, market stop, stop-limit) when NinjaTrader receives a "strategy active order" notification. There is a high probability that your TradeStation strategy position size will be out of synchronization with your live brokerage account using this option. It requires manual user interaction and is **NOT** recommended.

Understanding synchronization time out

Synchronization Time Out

Excluding the "Submit and forget" option, NinjaTrader will notify you after the specified number of seconds if an order is out of sync with TradeStation's reported order fill amount. An example would be if TradeStation reported a market order fill of 1 contract, NinjaTrader submits a market order but the order is not filled for some reason after the specified amount of time, you will be notified.

How to enable order confirmation

Order Confirmation

You can choose to have NinjaTrader prompt you for approval before submitting your order to your brokerage account. To enable this feature start in the NinjaTrader Control Center and select the Tools menu, then select the menu name **Options**, once in the Options window , click on the **Trading** category and check "Confirm order placement".

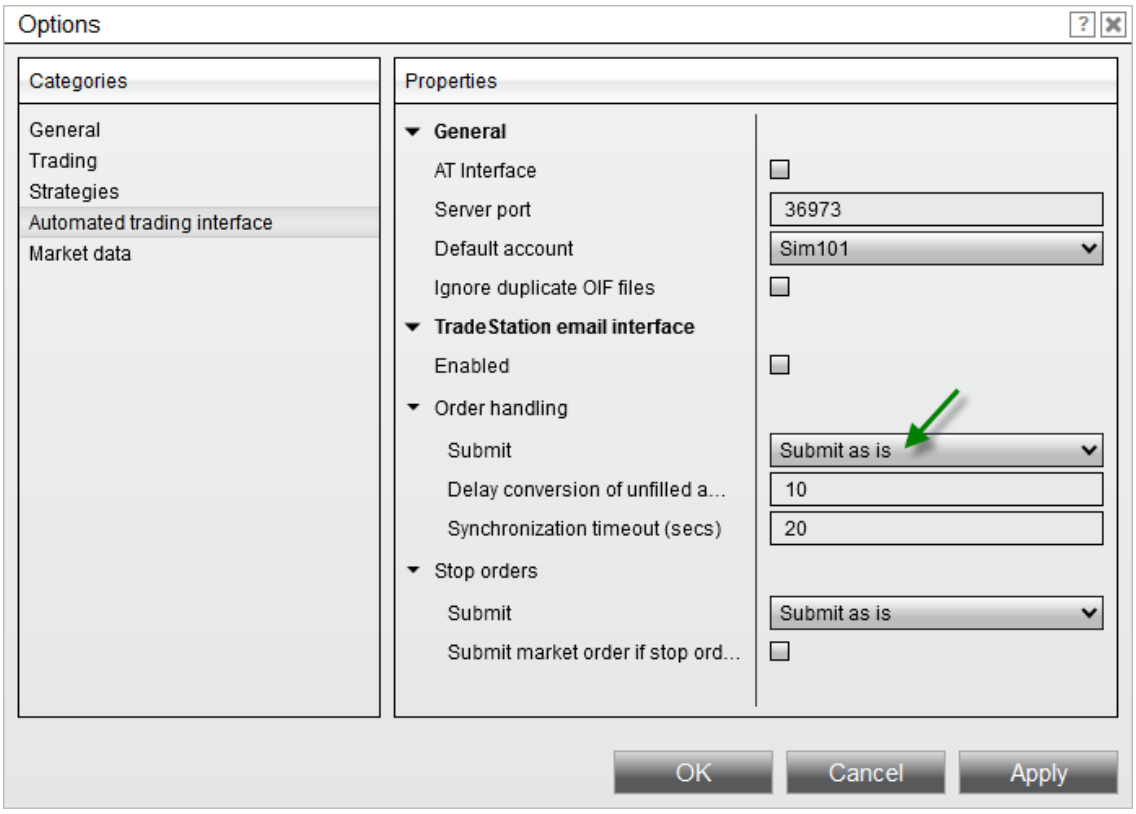
Understanding special handling for FX through FOREX.com/City Index

Special Handling for FX through FOREX.com/City Index

FOREX.com will reject a limit order to buy at the offer or above or to sell at the bid or below. NinjaTrader can check the current market rate on limit order submission and automatically convert to market if the limit price is invalid according to FOREX.com but in your favor resulting in a fill. To have NinjaTrader check for these conditions, you must be subscribed to rate data for the currency pair being traded. We suggest opening a Market Analyzer (to open a Market Analyzer window select the File menu and then the menu name **New**) and adding all traded currency pairs to this grid. This ensures that there is available rate data for NinjaTrader to cross check an incoming limit price against.

10.4.1.6.5 Stop Order Handling

There are several Stop Order Handling options available for the signals sent from TradeStation. All Stop Order Handling options are available by selecting the Tools menu in the Control Center, selecting the menu name **Options**, selecting the Automated Trading interface category, and setting the **Order handling "Submit"** option to "**Submit as is**".



If you have "Submit market order on TS fill" or **"Submit and forget"** enabled via Order Handling Options, the following Stop Order Handling is ignored.

> ***Warning***
> *Please review all of the following **Stop Order Handling** options on this page to ensure your stop orders are managed as expected.*

How to submit stop orders "as-is"

Submit "as-is"
Submits the stop order as specified.

How to convert to stop-limit orders

Convert to stop-limit

Will convert a stop order to a stop-limit order. When this option is selected, you will have a property displayed to set the limit price is calculated based on the user defined "Limit price offset as ticks" value.

▼ How to convert to simulated stop-market

Convert to Simulated Stop-Market
Submits a simulated stop-market order which is a local PC held order that submits a market order once the stop price is hit.

▼ How to submit market order if stop order was rejected

Submit market order if stop order was rejected
Submits a market order in the event that a stop order is rejected for any reason.

Behavior as follows:
1. Stop order worse than current last traded price --> Market order submitted (desired outcome)
2. Stop order rejected due to insufficient funds --> Market order submitted and also rejected (not desired but no risk)
3. Stop order rejected due to price outside of range --> Market order submitted and likely filled (risky)
4. Stop order rejected due to limit price worse than stop price --> Market order submitted and likely filled (risky)

> ***Risk***
> *If this option is enabled, it is your responsibility to ensure that your TS EL code is sending valid stop prices to NinjaTrader otherwise you risk getting filled when you may not want to.*

10.4.1.6.6 Workspace Options

Creating a workspace within TradeStation with the correct naming convention is critical to enabling TradeStation to properly communicate with NinjaTrader.

▼ How to create a new TradeStation workspace

From within TradeStation, you can create a new workspace by left mouse clicking on the menu File, selecting the menu name **New**, and then selecting the menu

name **Workspace**. This will create an untitled workspace in TradeStation. You must then left mouse click on the menu File and select the menu name **Save Workspace As...** to provide a workspace name following the naming conventions listed below.

Understanding workspace naming convention functions

The workspace name must contain "NinjaTrader;" (one word, without the quotations) otherwise NinjaTrader will NOT process any trade signals received from TradeStation.

Account Name

You can optionally add your brokerage account name(s) to the workspace name to identify an account that NinjaTrader will route orders to. If the account name is missing NinjaTrader will route orders to the default account (to set the default account from NinjaTrader left mouse click on the Tools menu, select the menu name **Options**, left click on the ATI tab, and then select the General tab). The account name must be specified as "Account=YourAccount" (without quotations) and where "YourAccount" is the name/number of your brokerage account.

Multiple Accounts

You can add multiple accounts in the workspace name to inform NinjaTrader to replicate the TradeStation order across more than one account. To do this add a comma "," (without the quotations) after each account name. For example; "Account=Account1,Account2"

Quantity Multiplier

You can optionally associate a quantity multiplier with each account that you have specified in the workspace name. This optional value will be multiplied by the TradeStation's strategy quantity amount. For example; if your TradeStation strategy has a quantity amount of 1 contract and you want to trade 2 contracts and you do not want to modify this amount in the strategy itself you can add "=2" after the account name in the workspace which would multiple the strategy contract amount by 2. The text would look like "Account=YourAccount=2"

Aliases - Chart Instrument "A" then Execute Orders in Instrument "B"

You can redirect orders to a different instrument than the instrument that your TradeStation strategy is actually running on. For example, you can run a strategy on $SPX.X but have orders actually placed to the S&P Emini contract. The text would look like "Map=$SPX.X,ESH09" where $SPX.X is the TradeStation chart instrument followed by a comma and then ESH09 which is the S&P Emini March 2009 contract which is the contract that will be traded. Since you can have multiple charts running in a workspace, you can add multiple mapping relationships. For

example "Map=$SPX.X,ESH09,$COMPX,MSFT" would map the $SPX.X to ESH09 and $COMPX would map to MSFT.

Workspace name examples

Workspace Name Examples

Following are samples of valid TradeStation workspace names. Remember, you will separate functions with a semi colon ";".

The following workspace name routes orders to the Default account specified under Tools --> Options --> ATI tab
NinjaTrader

The following workspace name routes orders to account # 1235
NinjaTrader;Account=1235

The following workspace name routes an order to account #7777 and another order to account #1311 with the original strategy quantity multiplied by a factor of 2
NinjaTrader;Account=7777,1311=2

The following workspace name routes orders to account #123 and maps trade signals generated by the $SPX.X chart to the S&P Emini March 2009 contract
NinjaTrader;Account=123;Map=$SPX.X,ESH09

Understanding multiple workspaces

Multiple Workspaces

You may create multiple workspaces provided that they each contain "NinjaTrader;" (without the quotations) in their name. For example, you could have two workspaces named "NinjaTrader1;" and "NinjaTrader2;"

10.4.2 Running NinjaScript Strategies

Running NinjaScript Strategies Overview

The following section explains how to automate a NinjaScript strategy. Please keep in mind that a strategies real-time performance can and will vary from your backtest results.

10.4.2.1 Setting Real-Time Strategy Options

Prior to running a NinjaScript strategy against a live account, you must first understand and set the options for a NinjaScript strategy. These options can be found in the Control Center under Tools > Options > Strategies.

10.4.2.2 Strategy Position vs. Account Position

An important concept to understand prior to using NinjaScript strategies in a real-time trading environment (live brokerage account, for example) is the difference between a **Strategy Position** and an **Account Position**.

Strategy Position

A **Strategy Position** is a virtual position that is created by the entry and exit executions generated by a strategy and is independent from any other running strategy's position or an **Account Position**.

Account Position

An **Account Position** is the position you actually hold in a real-time trading account, whether it is a NinjaTrader internal simulation account (Sim101) or your live real-money brokerage account.

In most cases, a trader would want their **Strategy Position's** size and market direction to be equal (in sync) to their **Account Position,** but there are situations when this may not be the case.
For example:

- You want to run multiple strategies in the same market simultaneously where strategy A holds a LONG 1 position, strategy B holds a LONG 2 position resulting in an account that should hold a LONG 3 position in order to be in sync with both strategies

- You want to run a strategy and at the same time trade the same market the strategy is running on using discretionary tactics through one of NinjaTrader advanced order entry window such as the SuperDOM or Chart Trader

An extremely common scenario…

An extremely common scenario is starting a NinjaScript strategy in the middle of a trading session, such as one hour after the session has begun. The NinjaScript strategy is run on each historical bar for the 1st hour of the session (it will actually run on all historical data loaded in a chart) to determine the current position state it would be in if it had been running live since the start of the session. This position state then becomes the **Strategy Position** for your strategy. Let us assume that during the historical hour your strategy would have entered a LONG 1 position and the position is still open. This would mean the **Strategy Position** is LONG 1 and since this trade was not actually executed on an account, your **Account Position** is FLAT.

What can you do in this case?

If you want your **Account Position** to match your **Strategy Position**, you will need to place a manual order into the account the strategy is running on. Continuing from the above example, you would need to place a 1-lot market order for the market being traded into the account the strategy is running on. Alternatively, NinjaTrader has the ability to have your account automatically synced to your strategy position on strategy startup by setting the desired **Start Behavior.** New to NinjaTrader is the ability to sync your **Strategy Position** to an **Account Position**. For more information on Strategy **Start Behavior**, please see the article here about syncing account positions.

What if I do not sync my account?

The resulting behavior when the **Strategy Position** and **Account Position** are out of sync is when your strategy (continuing with the example above) closes the long position with a sell order it would bring the **Strategy Position** to flat and your **Account Position** to SHORT

10.4.2.3 Syncing Account Positions

It is critical to understand the various options available to you that determine how the strategy will behave on startup through the **Start Behavior** parameters. NinjaTrader provides several option combinations that can be used in different scenarios depending on what your requirements are. Please first review the information about strategy position vs account positions as this article builds on that concept.

The **Start Behavior** settings can be set from the Strategy Parameters when you are adding a strategy.

> **Note**: Please be aware that these options will only help you sync your **Account Position** to your **Strategy Position** once on startup. These options will *not* guarantee your

Account Position remains in sync afterward. Any active orders you may have had on your account prior to strategy start that was *not* generated by your strategy would not have been cancelled on start and can lead to your **Account Position** being out of sync from your **Strategy Position.** Placing manual trades or running multiple strategies on the same instrument can also lead to your **Account Position** being out of sync from your **Strategy Position**.

Warnings:
- Using **Synchronize account** can close or place live trades to your account
- If you have existing **historical order references** which have transitioned to real-time, you **MUST** update the **order object reference** to the newly submitted **real-time** order; otherwise errors may occur as you attempt to cancel the order. You may use the GetRealtimeOrder() helper method to assist in this transition.

Wait until flat

These are the default settings for your strategies and are the least disruptive in terms of handling your current **Account Position**. It assumes your **Account Position** is in a flat state.

When your strategy starts it will check for any active orders previously generated by the strategy on your account and cancel those first. Should the strategy be unable to cancel and receive confirmation on the cancellation of these orders within 40 seconds the strategy will not start and an alert will be issued.

- If the **Strategy Position** is flat, then the **Account Position** and **Strategy Position** are assumed to be in sync with each other. The next order placed by your strategy would be placed as a live order to your account.
- If the **Strategy Position** is *not* flat, the strategy will place all trades in a "virtual" sense until the **Strategy Position** reaches or crosses a flat state. Once a flat state is achieved the **Strategy Position** will be assumed to be in sync with the **Account Position** and all future orders will be placed live.

Critical: Should your **Account Position** *not* be flat at the point in time the **Strategy Position** reaches a flat state your **Account Position** and **Strategy Position** will **NOT** be in sync.

Wait until flat, synchronize account

This combination should be used when you want to begin trading your strategy off a flat state with minimal user interaction to sync your **Account Position** prior to start.

When your strategy starts it will check for any active orders previously generated by the strategy on your account and cancel those first. Should the strategy be unable to cancel and receive confirmation on the cancellation of these orders within 40 seconds the strategy will not start and an alert will be issued. After the strategy is successful in cancelling any orders that required action it will check your current **Account Position** and compare it to a flat state. On multi-instrument strategies it will perform this check for all instruments used by the strategy.

- If the **Account Position** is flat already, no reconciliatory order will be submitted. The strategy will then wait for the **Strategy Position** to reach a flat state as well before submitting any orders live.
- If the **Account Position** is *not* flat, NinjaTrader will submit a market order(s) to reconcile the **Account Position** to a flat state. The strategy will then wait for the **Strategy Position** to reach a flat state before submitting live orders.

> **Note**: The reconciliatory market order is submitted outside of the strategy so your strategy will not be able to manage it from methods like OnOrderUpdate(), OnExecution(), etc.

Immediately submit

This combination should only be used when you are sure your **Account Position** is the way you want it to be in relation to the **Strategy Position** prior to strategy start.

On startup the strategy will begin executing orders immediately.

- Any active orders on the account previously generated by the strategy that does not match* an active strategy order will be cancelled. Should the strategy be unable to cancel and receive confirmation on the cancellation of these orders within 40 seconds the strategy will not start and an alert will be issued.
- The matching active orders on the account will then be mapped to the active strategy orders

- Any remaining active strategy orders that cannot be successfully paired will be submitted live and the strategy will begin managing your **Strategy Position** assuming your **Account Position** is in sync with it.

* A previously generated order is considered to match an active strategy order when the order action, order type, quantity, limit price, and stop price are exactly identical.

Immediately submit, synchronize account

This combination should be used when you want to begin trading with your strategy immediately while not worrying about your **Account Position** prior to start.

On startup the strategy will begin executing orders immediately.

- Any active orders on the account previously generated by the strategy that does not match* an active strategy order will be cancelled. Should the strategy be unable to cancel and receive confirmation on the cancellation of these orders within 40 seconds the strategy will not start and an alert will be issued.
- The matching active orders on the account will then be mapped to the active strategy orders
- Any remaining active strategy orders that cannot be successfully paired will be submitted live and the strategy will then try to sync your **Account Position** to your **Strategy Position** through the process below.

After the strategy is successful in cancelling and submitting any orders that required action it will check your current **Account Position** and compare it to your **Strategy Position**. On multi-instrument strategies it will perform this check for all instruments used by the strategy.

- If the **Account Position** matches your **Strategy Position**, no reconciliatory order will be submitted. The strategy will then begin managing your **Strategy Position** immediately.
- If the **Account Position** does *not* match your **Strategy Position**, NinjaTrader will submit a market order(s) to reconcile the **Account Position** to match your **Strategy Position**. The strategy will then begin managing your **Strategy Position** immediately.

Note: The reconciliatory market order is submitted outside of the strategy so your strategy will not be able to manage it from methods like OnOrderUpdate(), OnExecution(), etc.

***** A previously generated order is considered to match an active strategy order when the order action, order type, quantity, limit price, and stop price are exactly identical.

Adopt account position

This setting should be used if you would like your strategy to disregard the historical virtual **Strategy Position** and to start in the same position as the real-world **Account Position**.

On startup the strategy will begin executing orders immediately.

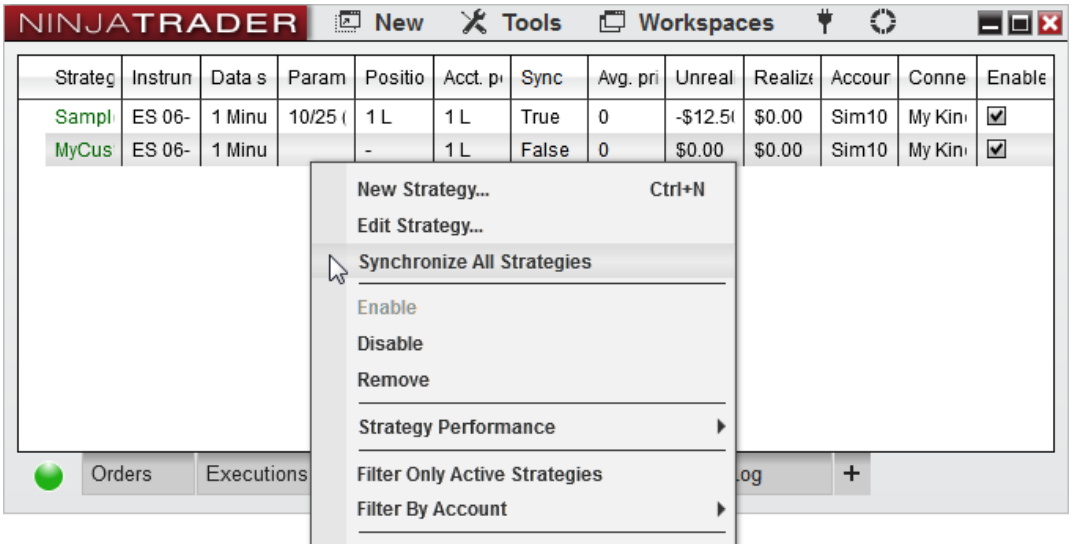
- Any active orders on the account previously generated by the strategy that does not match* an active strategy order will be cancelled. Should the strategy be unable to cancel and receive confirmation on the cancellation of these orders within 40 seconds the strategy will not start and an alert will be issued.
- The matching active orders on the account will then be mapped to the active strategy orders
- Any remaining active strategy orders that cannot be successfully paired will be submitted live and the strategy will then try to sync your **Account Position** to your **Strategy Position.**
- Only one strategy with this setting can be started at a time for an individual account and instrument.
- The account and instrument the strategy is started on must not have any working orders which were submitted outside of the strategy, or by another instance of the same strategy. If an order is detected, the strategy can not be started until these orders have been manually managed.

Note: **Adopt account position** will only be available if the developer of a strategy has programmed the strategy to be aware of the real-world account position. If this setting is not available when starting your strategy, the strategy was not programmed in a manner capable of handling account positions. If you are a developer and would like your strategy to handle a real world position, please see the following article here on these properties.

* A previously generated order is considered to match an active strategy order when the order action, order type, quantity, limit price, and stop price are exactly identical.

Synchronize all strategies

The **Synchronize All Strategies** option is found on the Strategies tab of the NinjaTrader **Control Center** and right clicking on the **Strategy Grid.**



Selecting this feature will scan through the **strategy position** of all enabled strategies which are not "*Wait until flat"* on each account and instrument combination (including all instruments under a multi-series strategy) and will then compare the aggregated **strategy position** to the **account position**.

Under the condition that the **account position** does **NOT** match the aggregated **strategy position**, a market order will be submitted to the account to match the aggregated **strategy position**.

Consider the following scenario, where all 4 strategies are running on a live account which is currently **flat**:

Strategy Name	Strategy Position	Start Behavior

Strategy A	1L	Immediately submit
Strategy B	2S	Wait until flat
Strategy C	Flat	Immediately submit
Strategy D	1L	Immediately submit

- Strategy A and D are both showing a 1 Long position and are both "*Immediately submit*"
- Although Strategy B shows 2 Short, the strategy is currently "*Wait until flat*" so it is **NOT** considered in this process
- Strategy C is Flat and does not contain a position
- Therefore, the calculated aggregated strategy position will be 2 long

Selecting **Synchronize All Strategies** with the above combination would then issue a market order to buy 2 contracts on the live account.

> **Note**: The reconciliatory market order is submitted outside of the strategy so your strategy will not be able to manage it from methods like OnOrderUpdate(), OnExecution(), etc.

10.4.2.4 Running a NinjaScript Strategy from a Chart

You can run a NinjaScript strategy in real-time in a live or simulation account within a NinjaTrader chart.

▼ How to run a NinjaScript strategy in a chart

Running a NinjaScript Strategy

To run a NinjaScript strategy within a chart:

1. Select either the **Strategies** menu from within the right click menu, or the **Strategies** icon from the chart tool bar, or press the default CTRL + S Hot Key to access the **Strategies** window.

2. Select a strategy in the "Available" section, then click the **add** button. Alternatively, you can double-click any strategy listed in the "Available" section.
3. Once the strategy is added to the "Configured" section, set any strategy properties to your desired settings.
4. Press the **OK** button to run the strategy.

Note: You must set the "Enabled" property to **True** to turn on the strategy. When this property is disabled, the strategy will be applied to the chart, but will be inactive.

Tips:

- NinjaTrader must be connected to a live brokerage or market data vendor for a strategy to run. You can also use the Replay or Simulated Data Feed connections.

- Strategy menu options will **NOT** appear if you are not connected live
- On terminating a strategy, all strategy generated trade markers or draw objects will be removed from the chart
- A NinjaScript strategy is a self contained automated trading system, and orders generated are live. Canceling strategy-generated orders manually can cause your strategy to stop executing as expected. If you wish to manually cancel an order, terminate the strategy itself.
- Clicking the **"Close"** button to close a position on an account/instrument that has a strategy running will disable the strategy.
- Running and disabled strategies are displayed in the **Control Center** Strategies tab

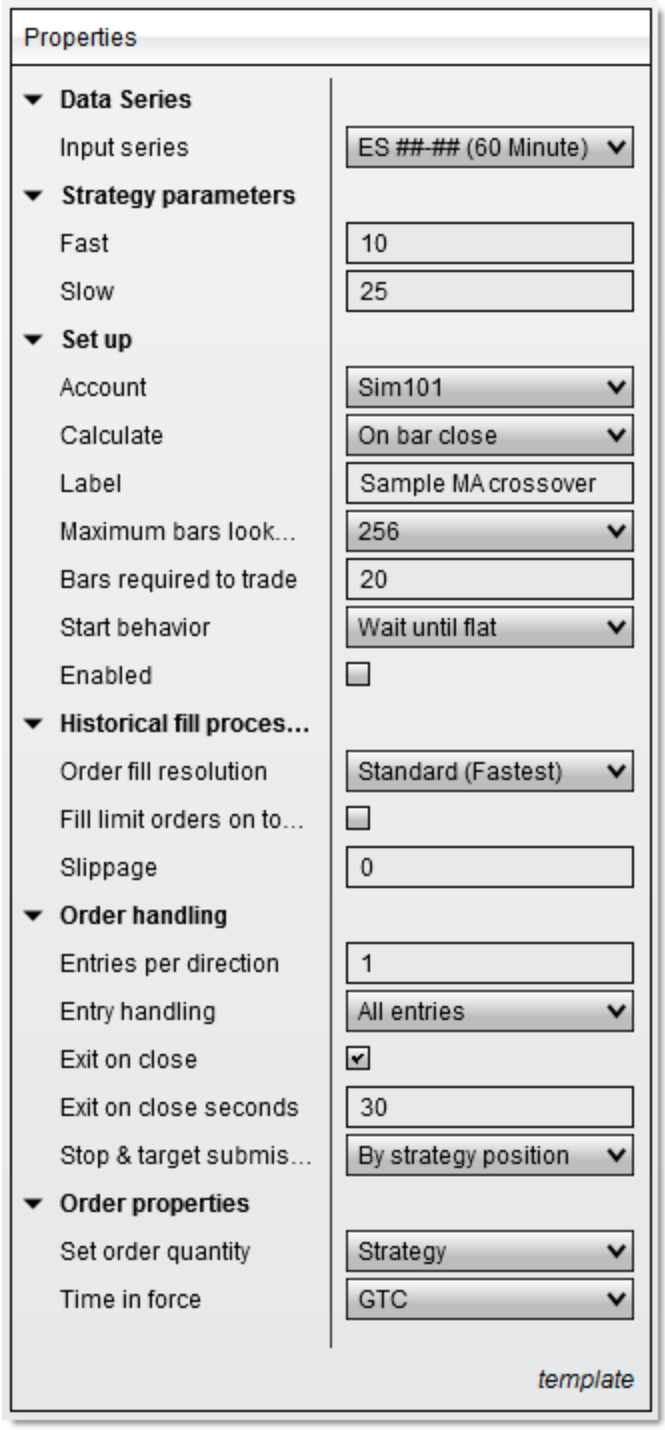
Terminating a NinjaScript Strategy

To terminate a strategy, first select a running strategy in the "Configured" section of the **Strategies** window, then click the **Remove** button. This will completely remove the strategy from the chart and the Control Center's **Strategies** tab. Alternatively, you can set the "Enabled" property to **False** to simply disable the strategy, allowing you to re-enable it at a later point without the need to reset it's properties.

Understanding strategy properties

Strategy Properties

The image below shows the adjustable properties for a strategy available in the Strategies window:



Data Series	Sets the data series on which the strategy will run

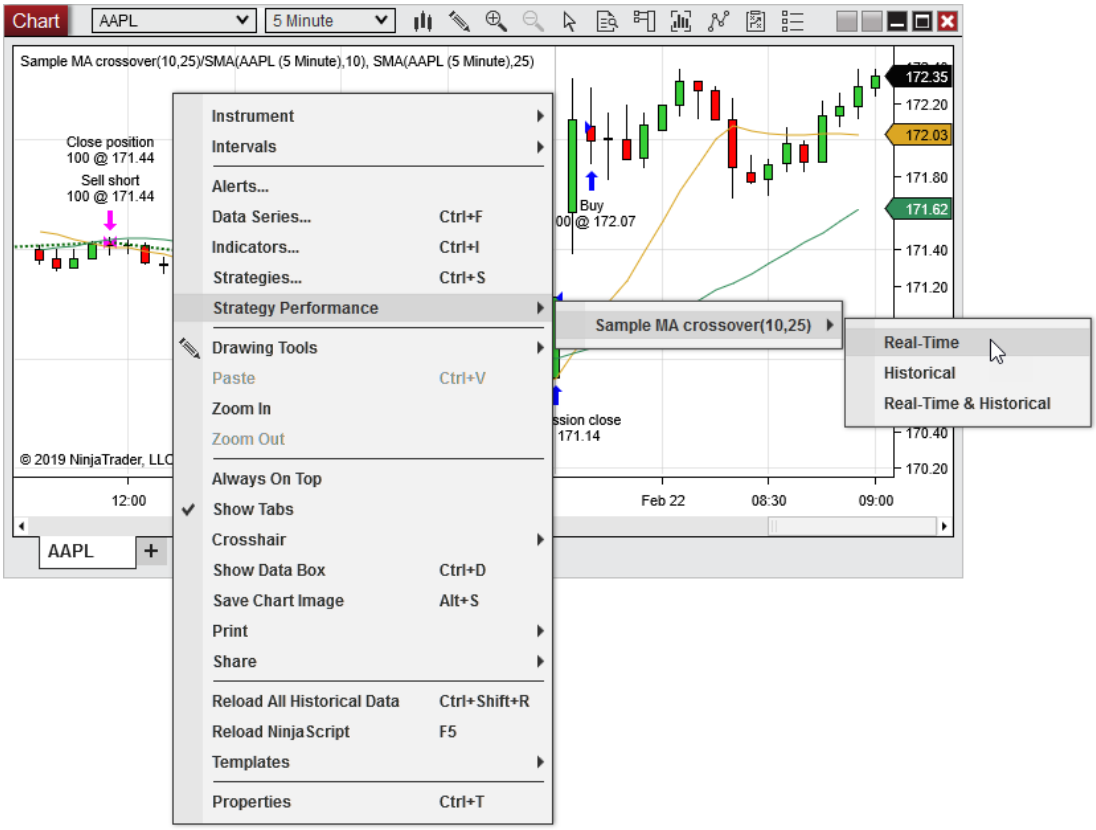
Strategy Parameters	Sets any strategy-specific user-defined inputs
Account	Sets the account to which the strategy will execute orders
Calculate	Sets the **Calculation Mode** for the strategy. Possible values are "On Each Tick," "On Price Change," or "On Bar Close"
Label	Sets a text label that will be displayed on the chart to represent the strategy
Maximum Bars Look Back	Sets the maximum number of historical bars to use for strategy calculations. The TwoHundredFiftySix setting is the most memory friendly
Bars Required to Trade	Sets the minimum number of historical bars required to start taking live trades
Start Behavior	Sets the starting behavior of the strategy, based upon the account position. See the Syncing Account Positions page for more information.
Enabled	Enables or disables the selected strategy
Order Fill Resolution	Sets the way that simulated historical orders will be processed by the strategy. See the Understanding Historical Fill Processing page for more information.
Fill Limit Orders on Touch	Enables the filling of limit orders when touched for the historical portion of the chart
Slippage	Sets the slippage amount in ticks for the historical portion of the chart

Entries per direction	Sets the maximum number of entries allowed per direction while a position is active based on the "Entry handling" property
Entry handling	Sets the manner in which entry orders are handled. If set to "AllEntries", the strategy will process all entry orders until the maximum allowable entries set by the "Entries per direction" property have been reached while in an open position. If set to "UniqueEntries", the strategy will process entry orders until the maximum allowable entries set by the "Entries per direction" property per each uniquely named entry have been reached.
Exit on close	When enabled, open positions will be closed on the last bar of a session
Exit on close seconds	Sets the number of seconds prior to the end of a session at to close any open positions held by the strategy
Stop & target submission	Sets how stop and target orders are submitted
Set order quantity	Sets how the order size is determined, options are: • "Default Quantity" - User defined order size • "Strategy" - Uses the order size specified programmatically within the strategy
Time in force	Sets the order's time in force. Possible values are DAY and GTC

Viewing strategy performance of a NinjaScript Strategy applied to a chart

Strategy Performance

Real-time, Historical, or Historical & Real-time executions for the automated strategy can be accessed within the open chart by right mouse clicking in the chart and selecting the menu item **Strategy Performance**, then hovering the mouse over the desired automated strategy and selecting the type of executions you wish to view from the menu that appears. A Performance window will appear where you can view and analyze the trade data.



The following categories of performance data can be selected:

Real-Time	Displays performance statistics for trades the strategy has taken in real-time **ONLY**
Historical	Displays performance statistics for historical trades **ONLY**, calculated before any real-time trades are taken
Real-Time and	Combines historical and real-time performance statistics in a single report

Historical	

10.4.2.5 Running a NinjaScript Strategy from the Strategies Tab

You can run a NinjaScript strategy in real-time in a live or simulation account via the Strategies tab of the Control Center.

▼ How to run a NinjaScript strategy from the Strategies tab

Setup Tips

Following are some key points and instructions on on how to run a NinjaScript strategy from the Strategies tab of the Control Center window:

- NinjaTrader MUST be connected to a live brokerage or market data vendor
- A NinjaScript strategy is a self contained automated trading system and orders generated are live and not virtual. Cancelling strategy generated orders manually can cause your strategy to stop executing as you designed it. If you want to manually cancel an order, terminate the strategy first.
- Strategies initiated from the Strategies tab will NOT appear in a chart

Running a NinjaScript Strategy

To run a NinjaScript strategy from the Strategies tab:

1. Left mouse click on the Strategies tab found in the NinjaTrader Control Center
2. Right mouse click within the Strategies tab. The right click menu will appear.
3. Select the menu item **New Strategy...** The New Strategy window will appear.
4. Choose the strategy you wish to run from the list of **Available** strategies on the left
5. Set the instrument, interval, and other optional strategy properties (see the "*Understanding strategy properties section below*") and press the OK button
6. Check the box in the Enable column of the Strategies tab next to the strategy you wish to enable.

Note: You must set the "Enabled" property in Step 6 above to **True** to turn on the strategy. When this property is disabled, the strategy will be applied, but will be inactive.

▼ Understanding strategy properties

Strategy Properties (see image below)

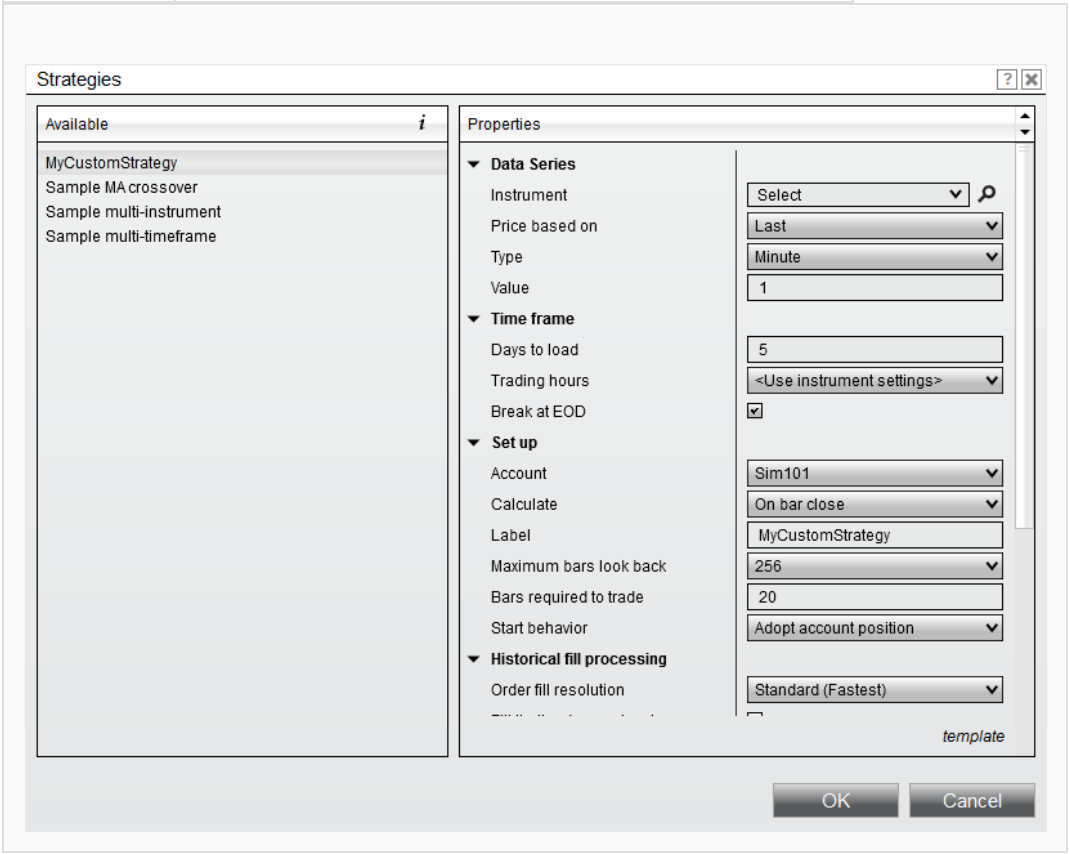
The image below shows the adjustable properties for a strategy available in the Strategies tab of the Control Center (see the "*How to run a NinjaScript strategy in from the Strategies tab*" section above):

Data Series	
Instrument	Sets the instrument(s) the strategy will run against
Price based on	Sets the type of market data used to drive the Data Series.
Type	Sets the bar type of the Data Series.
Value	Sets the Data Series value.
Strategy Parameters	
(...)	Sets any strategy specific user defined inputs
Time Frame	
Days to load	Sets the number of days to load data
Trading hours	Sets the Trading Hours template for the Data Series. (See the "*Trading Hours*" section of the Help Guide for more information)
Break at EOD	Enables or disables the bars being reset at EOD (End Of Day). (See the *"Break at EOD"* section of the Help Guide for more information)
Set up	

Account	Sets the account the strategy will execute orders in
Calculate	Sets the frequency that the indicator calculates: • **On bar close** - will slow down the calculation until the close of a bar • **On price change** - will calculate on when there has been a change in price • **On each tick** - calculate the indicator's value which each incoming tick.
Label	Sets a text label that will be displayed on the chart to represent the strategy
Maximum bars look back	Sets the maximum number of historical bars to use for strategy calculations. The TwoHundredFiftySix setting is the most memory friendly.
Bars required to trade	Sets the minimum number of bars required before the strategy will start processing trades
Start behavior	Sets the starting behavior of the strategy, based upon the account position. See the Syncing Account Positions page for more information.
Historical fill processing	
Order fill resolution	Sets the way that simulated historical orders will be processed by the strategy. See the Understanding Historical Fill Processing page for more information.
Fill limit orders on touch	Enables or disables the filling of limit orders on a single touch of price action.

Slippage	Sets the slippage amount in ticks for the historical portion of the chart
Order handling	
Entries per direction	Sets the maximum number of entries allowed per direction while a position is active based on the "Entry handling" property
Entry handling	Sets the manner in how entry orders are handled. If set to "**AllEntries**", the strategy will process all entry orders until the maximum allowable entries set by the "**Entries per direction**" property has been reached while in an open position. If set to "**UniqueEntries**", strategy will process entry orders until the maximum allowable entries set by the "**Entries per direction**" property per each uniquely named entry.
Exit on close	When enabled, open positions are closed on the last bar of a session
Exit on close seconds	Sets the number of seconds prior to the end of a session when open positions of a strategy will be closed
Stop & target submission	Sets how stop and target orders are submitted
Order properties	
Set order quantity	Sets how the order size is determined, options are: • **Default quantity** - User defined order size

	• **Strategy** - Takes the order size specified programmatically within the strategy
Time in force	Sets the order's time in force



How to view strategy performance

Strategy Performance

While the Account Performance tab will generate performance report against your account's trade history, the Strategy Performance menu allows you to generate a performance report against the trades generated by the selected strategy.

- **Real-time** - Generates performance data for your real-time trades only (since the strategy started running) and will exclude historical trades. If your strategy held a virtual position (calculated against historical data) upon starting, a virtual execution representing the average price of this position will be injected into the real-time results to ensure that a trade pair can be created with the executions resulting from the closing of this position.
- **Historical & Real-time** - Generates performance data for both historical and real-time trade data.
- **Historical** - Generates performance data for historical data only.

10.4.2.6 Working with Strategy Templates

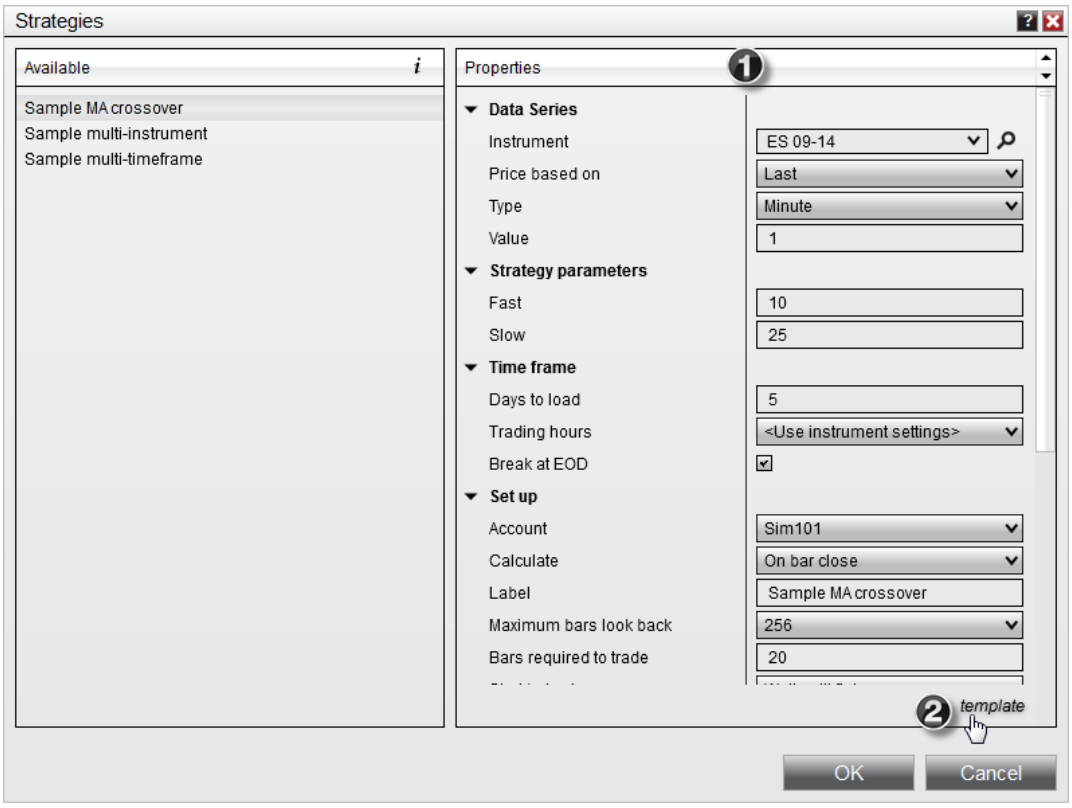
NinjaTrader allows you to save your **Strategy properties** as a template that can be loaded or set as the default for new instance of a **Strategy** when starting the strategy to be used in real-time or for backtesting purposes. There is no limit to the number of templates you can save.

▼ How to save a strategy template

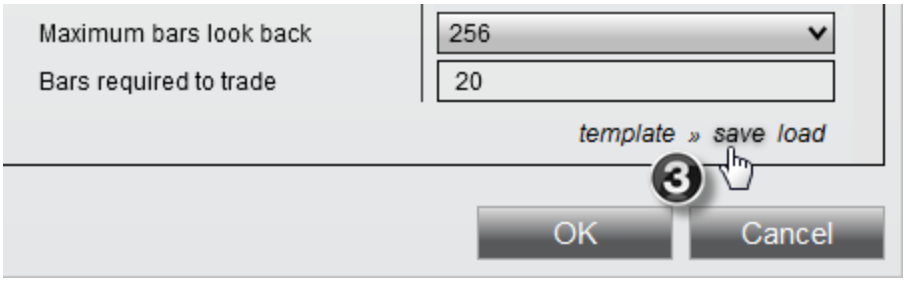
Saving Strategy Parameters in a Template

To save your strategies's various properties in a **template** to be recalled for later:

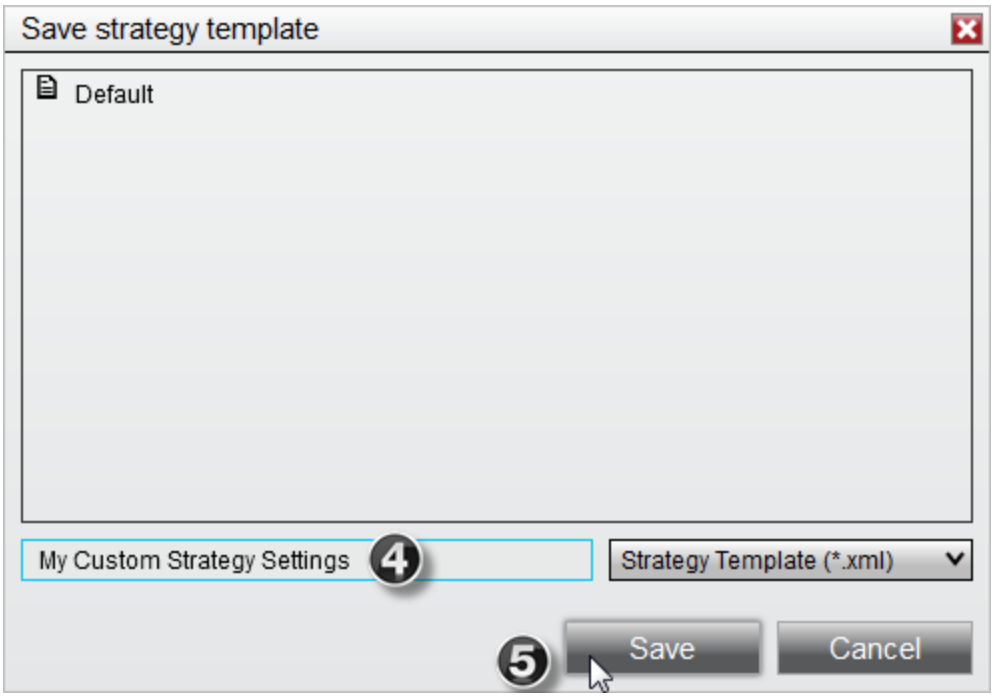
1. Configured your desired **Strategy Properties**
2. Left mouse click on the **template** text located at the bottom right of the **properties** dialog



3. Select the option **save** which will open a **Save Strategy Template** dialog window



4. Enter a custom *name to identify the strategy template
5. Click the **Save** button



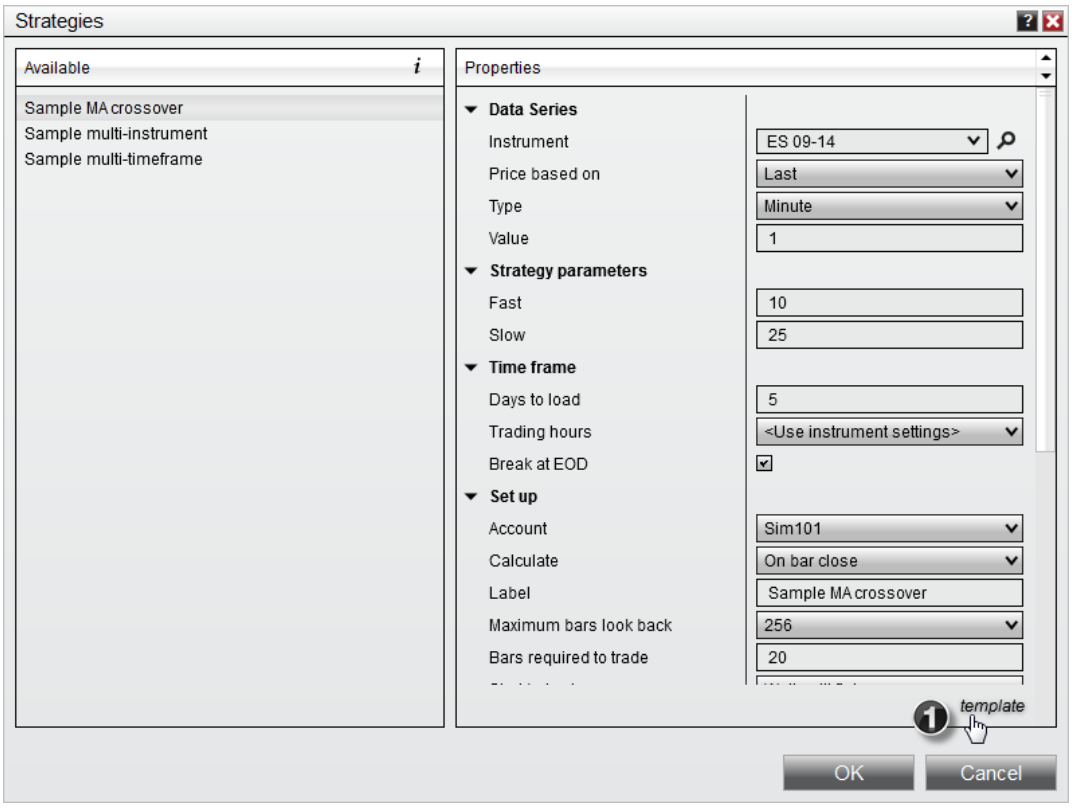
Tip: If you wish to save your **strategy properties** as the **default** values used when recalling these settings, you can call the strategy template name "**Default**" which will automatically load when a new instance of the strategy has been initiated.

How to load a strategy template

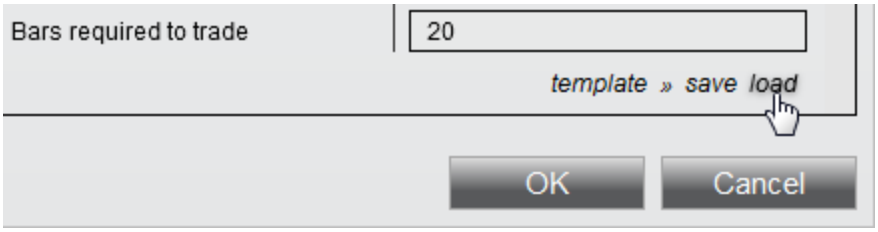
Loading Strategy Parameters from a Template

To recall your previous saved settings:

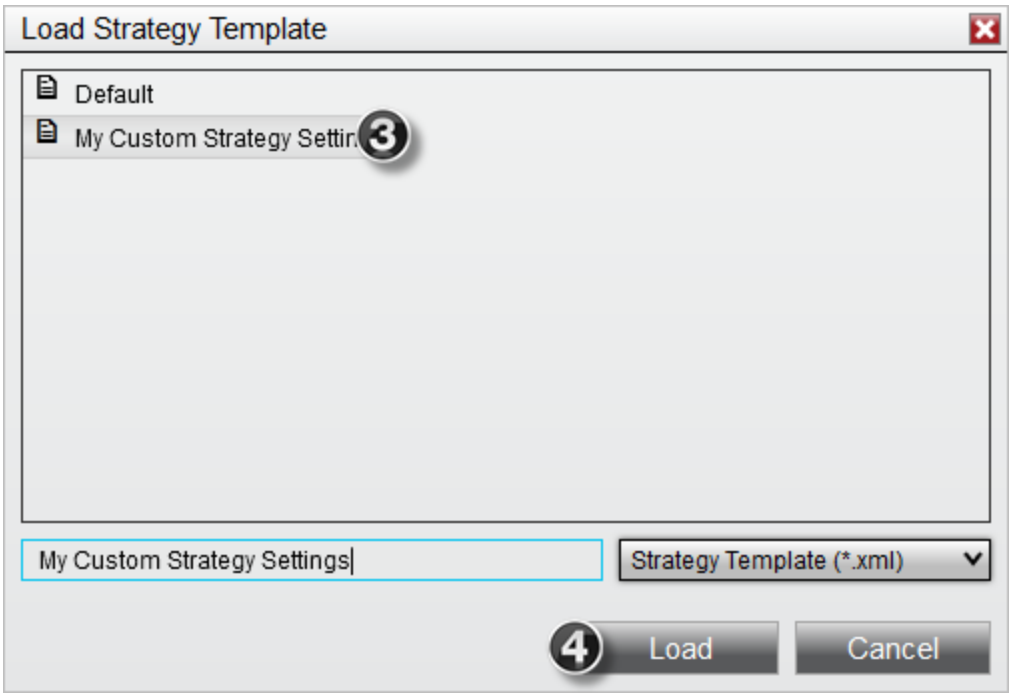
1. Left mouse click on the **template** text located at the bottom right of the **properties** dialog



2. Select the option **load** which will open a **Load Strategy Template** dialog window



3. Select the desired template name from the list of templates
4. Click the **Load** button

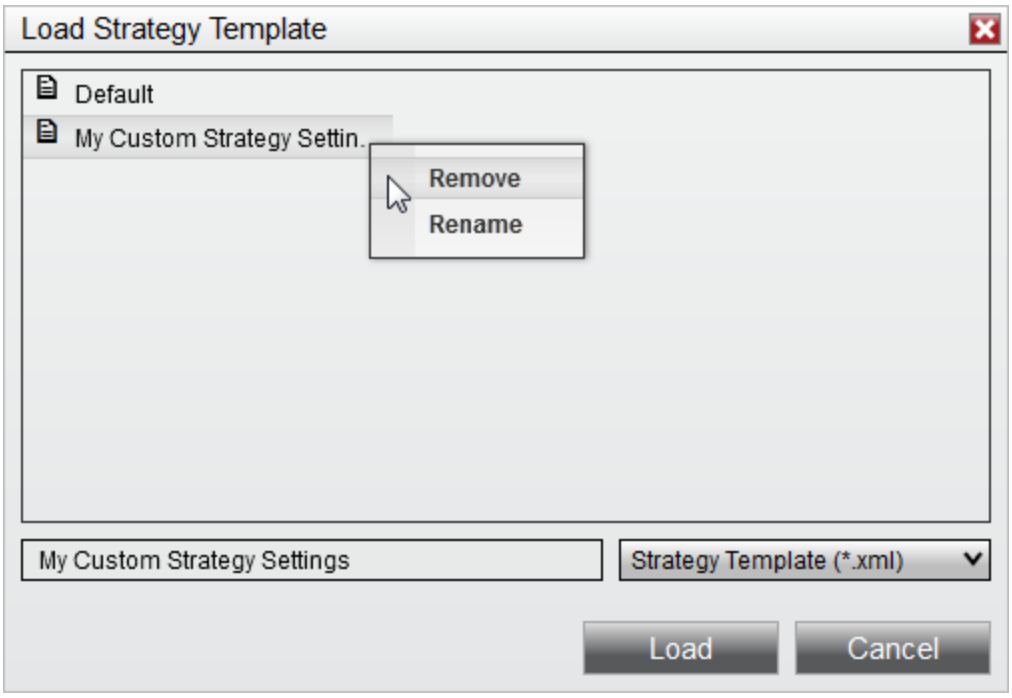


How to remove a strategy template

Removing a Strategy Template

To remove a saved **Strategy Template**:

1. Open the **Load Strategy Template** dialog window (see *"How to load a strategy template"* in the section above)
2. Right click on the template you wish to remove from the Load dialog menu and select the **Remove** menu item



Tip: If you wish to rename an existing template, you can select **Rename** from the same menu

10.5 Backup & Restore

Backup & Restore Overview

Backup & Restore utilities can be located via the **Tools** menu and then clicking on either **Import** or **Export**

The **Backup & Restore** utility provides an easy way to save and recover critical user generated data files such as but not limited to, user preferences, custom NinjaScript files, historical trade data and historical chart data. Backing up your data ensures that you are protected in case of software or hardware failure.

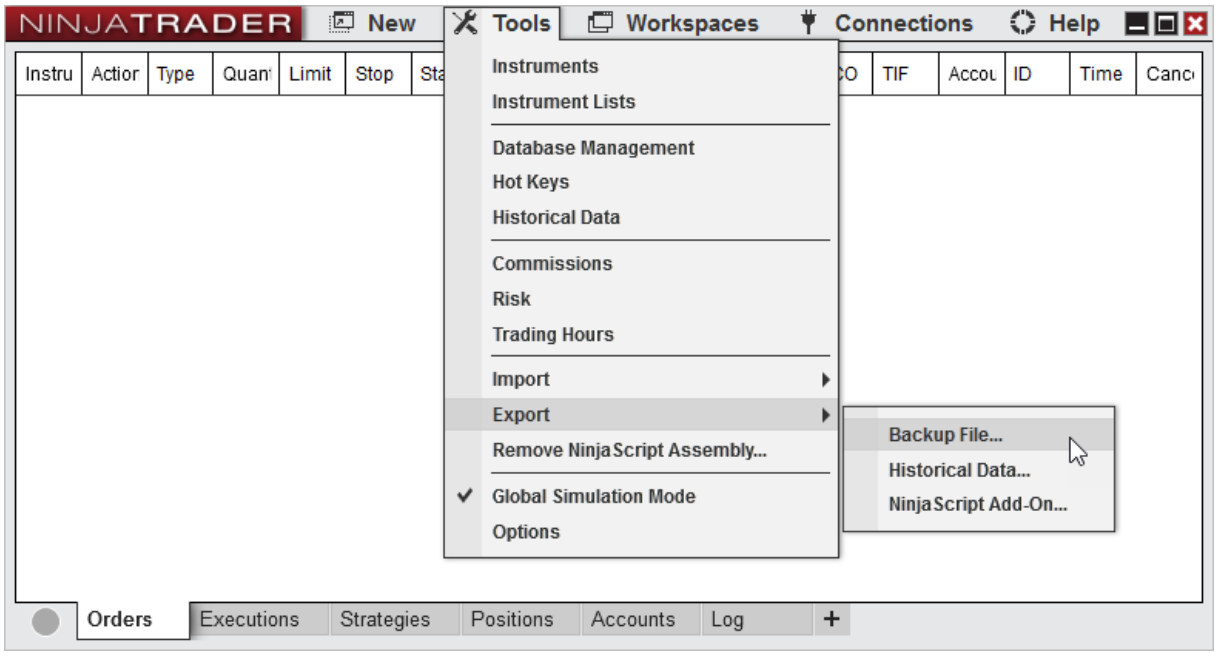
- Creating a Backup Archive
- Restoring a Backup Archive

10.5.1 Creating a Backup Archive

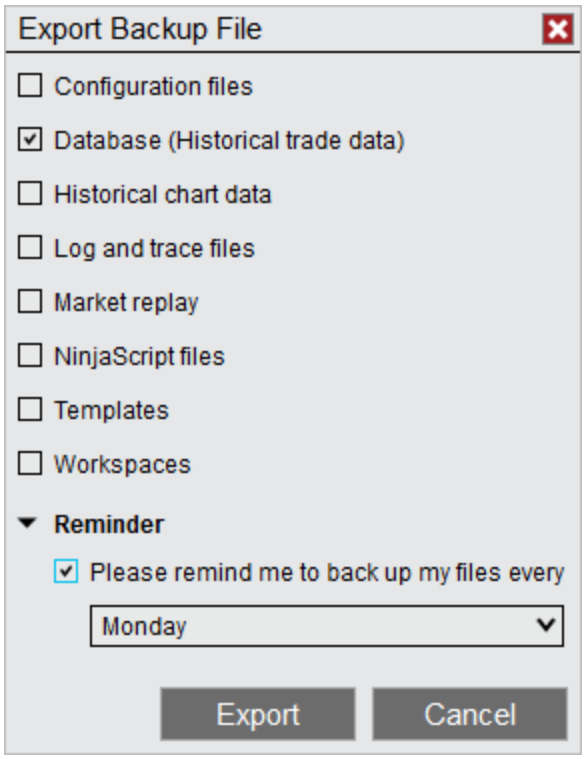
Running your first backup

Complete the following steps to create a Backup Archive.

1. Disconnect from all connectivity providers (if connected) and from within the Control Center window select the **Tools** menu. Then select the menu **Export** and the menu item **Backup File...**



2. The "Backup NinjaTrader" dialog window will appear
3. Select the items you wish to back up (see the table below for definitions)
4. Press the "**Export**" button

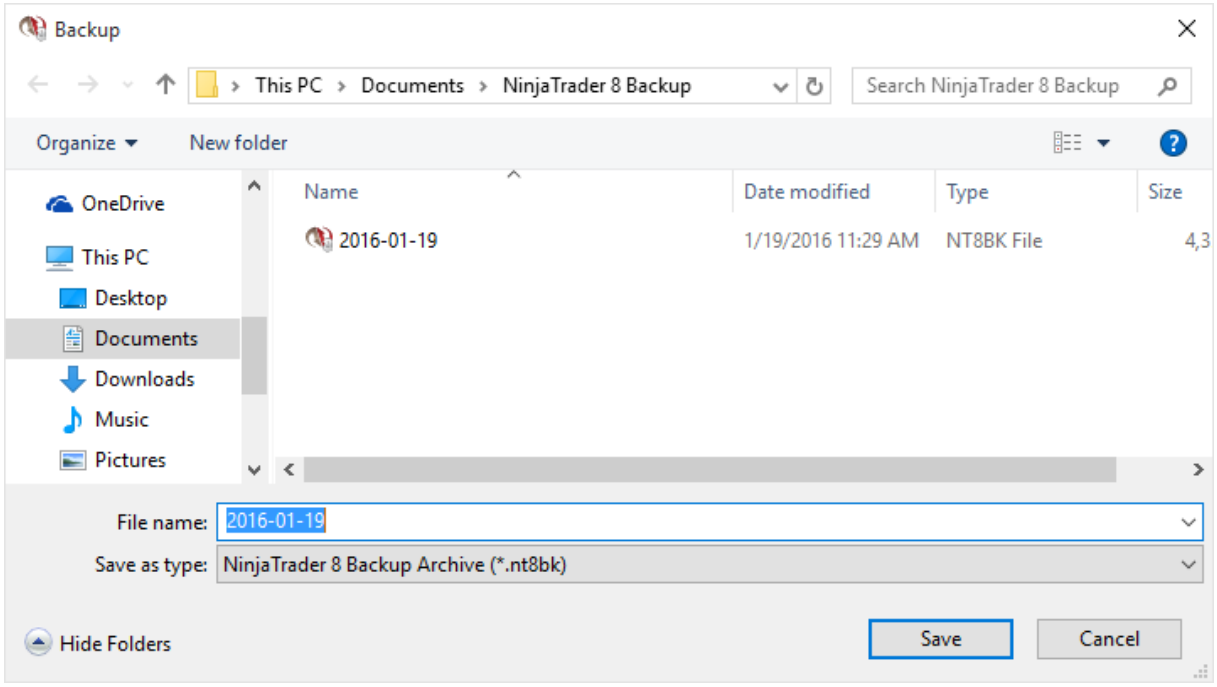


Configuration files	Contains user specific information such as license keys, account settings, and other user defined options
Database (Historical trade data)	Contains your historical trade execution data which is used to build reports in the Account Performance window
Historical chart data*	Chart data which has been recorded from a live connection, downloaded from a data provider, or manually imported
Log and Trace files	Diagnostic files written by the NinjaTrader application to record activity which can be analyzed by our customer service team during support inquiries
Market replay*	Data files used to drive the Level 1 and Level 2 price updates when using the Playback connection
NinjaScript files	Custom developed indicators, strategy, other add-ons. This option includes both user developed and 3rd party vendor files

Templates	Custom user defined configuration and display settings for features such as Charts, Strategies, Market Analyzer, ATM Strategies, SuperDOM
Workspaces	Files which are used to persist the over-all layout of a users working area.

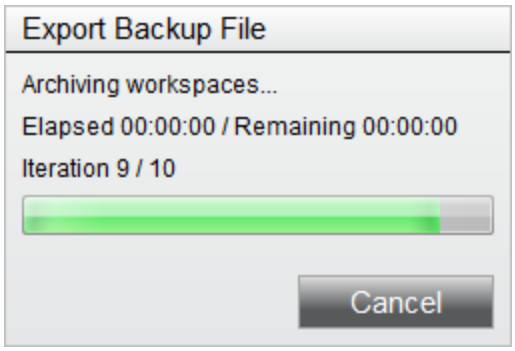
Tip: Market data files such as Historical chart data can often times be re-downloaded from your data provider. Market Replay data can be downloaded from the NinjaTrader servers for the most popular Futures and Forex instruments. If the data files you have stored on your computer are available from your data provider, you can save time and storage space by excluding these items from your backup and simply re-downloaded the data when needed. Please make sure to check with your data provider to ensure they still carry the type of data for the time period you may require.

5. Specify the location the backup will be saved and give the file a name to help you identify your backup file. By default, NinjaTrader will store the backup files in *\Documents\NinjaTrader 8 Backup* folder and will provide your computer's date as the file name.



6. Select the **Backup** button.

You will now be presented with a status bar indicating the estimated time and progress of the backup.



> **Note:** Depending on the amount of information you are backing up, the backup process may go very quickly, and you may not even notice the backup progress window. You can always verify the backup was completed by navigating to the location you specified for the backup and looking for the file name you provided the backup utility. Also keep in mind that if your database is very large (i.e., years of historical chart data), it can take some time for the backup to complete.

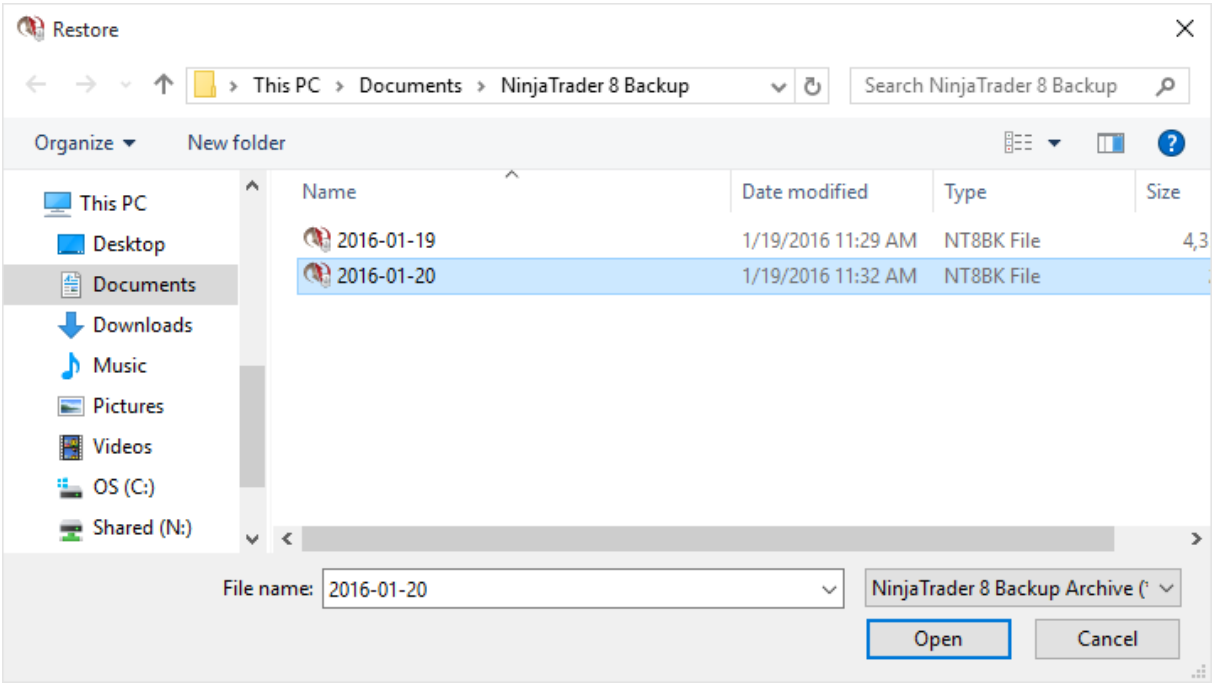
Scheduling a backup

When running a backup, there is an option labeled "Please remind me to back up my files every..." which if selected, will allow you to specify a day of the week or 1st day of the month to receive a reminder via a pop-up notification to trigger a backup.

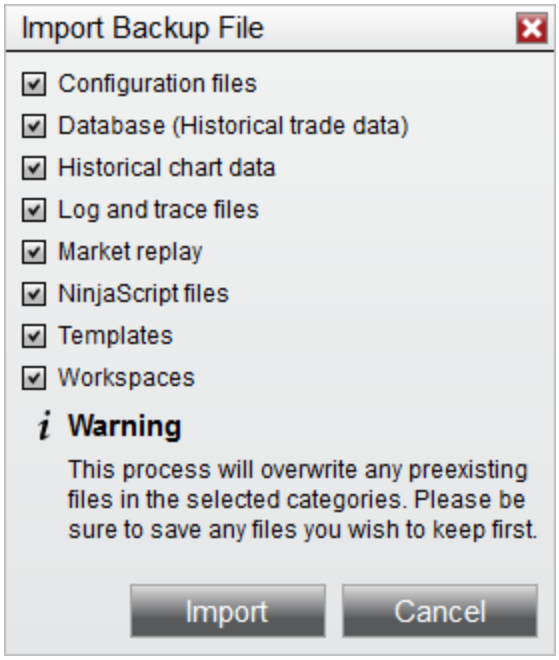
10.5.2 Restoring a Backup Archive

Complete the following steps to restore a Backup Archive.

1. From within the Control Center window select the **Tools** menu. Then select the menu **Import** and the **Backup File...** menu item
2. Select the backup archive to restore from the "Restore" file dialog window

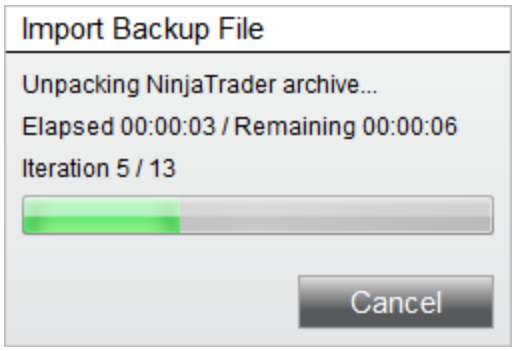


3. Press the "**Open**" button
4. Select the items you wish to restore



5. Press the "**Import**" button

You will now be presented with a status bar indicating the estimated time and progress of the import.



Note: Restoring backups made from previous releases may have issues importing if there have been changes made to NinjaTrader's resource structure. Please contact platformsupport@ninjatrader.com if you are seeing issues importing.

10.6 Charts

Charts Overview

NinjaTrader charts support a multitude of intervals, indicators and drawing tools, as well as discretionary trading using Chart Trader, and automated trading using NinjaScript strategies. The chart window itself is highly customizable and supports a wide range of user definable options.

Management

- Creating a Chart
- Navigating a Chart
- Chart Panels
- Chart Objects
- Working with Price Data
- Working with Multiple Data Series
- Bar Types
- Working with Indicators
- Working with Drawing Tools and Objects
- Saving Chart Defaults and Templates
- Data Box
- Cross Hair
- Chart Properties

Trading Features

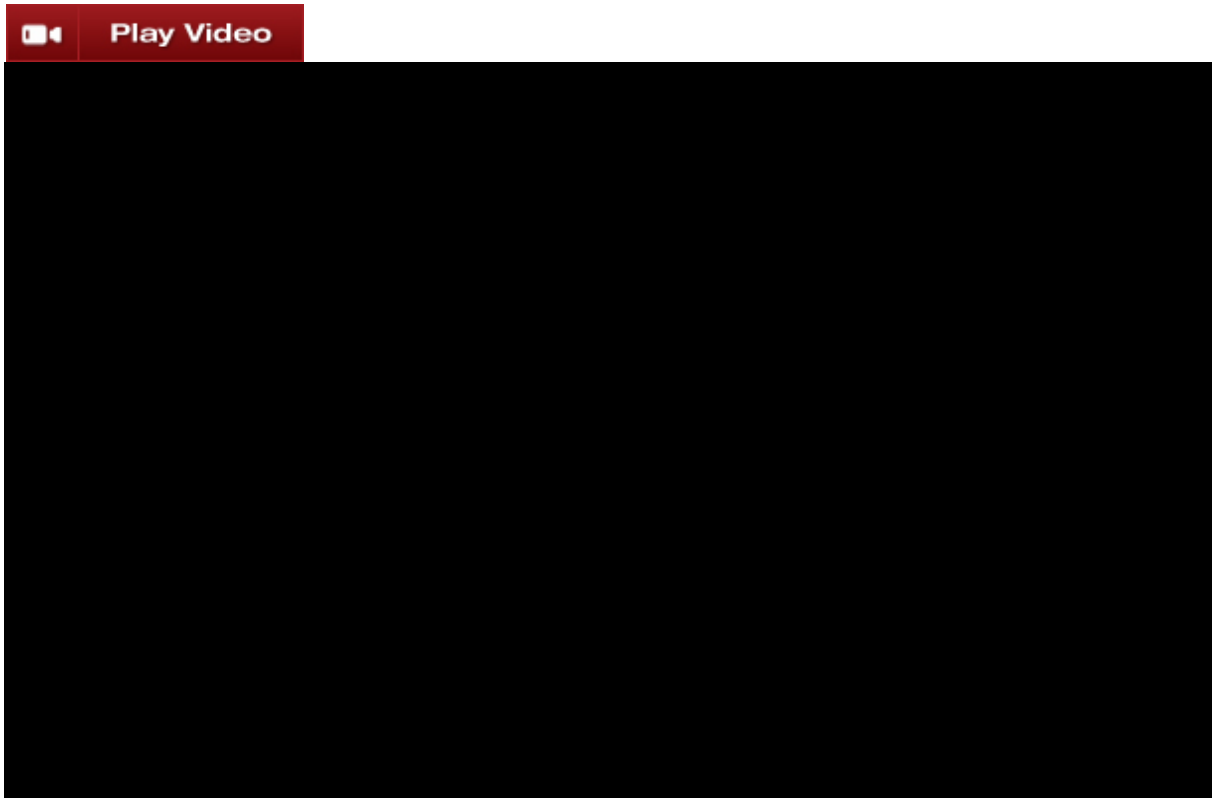
- Trading From a Chart
- Working with Automated Strategies
- Order Flow +
- COT
- Wiseman

Misc

- Break at EOD
- Reload Historical Data
- How Bars are Built
- How Trade Executions are Plotted

10.6.1 Creating a Chart

The following section covers how to open a NinjaTrader chart.



How to open a new chart

Opening a New Chart

To create a new chart, select the **New** menu from the NinjaTrader Control Center, then select the menu item **Chart**. The Data Series window will open where you can choose an instrument and an optional Template to apply to the chart. Please see the "*Working with Price Data*" page of the Help Guide for more information.

NinjaTrader does not limit the number of chart windows that can be opened, however more open windows will require more PC resources. Please see the Performance Tips page for more information on improving PC performance.

Selecting an Instrument

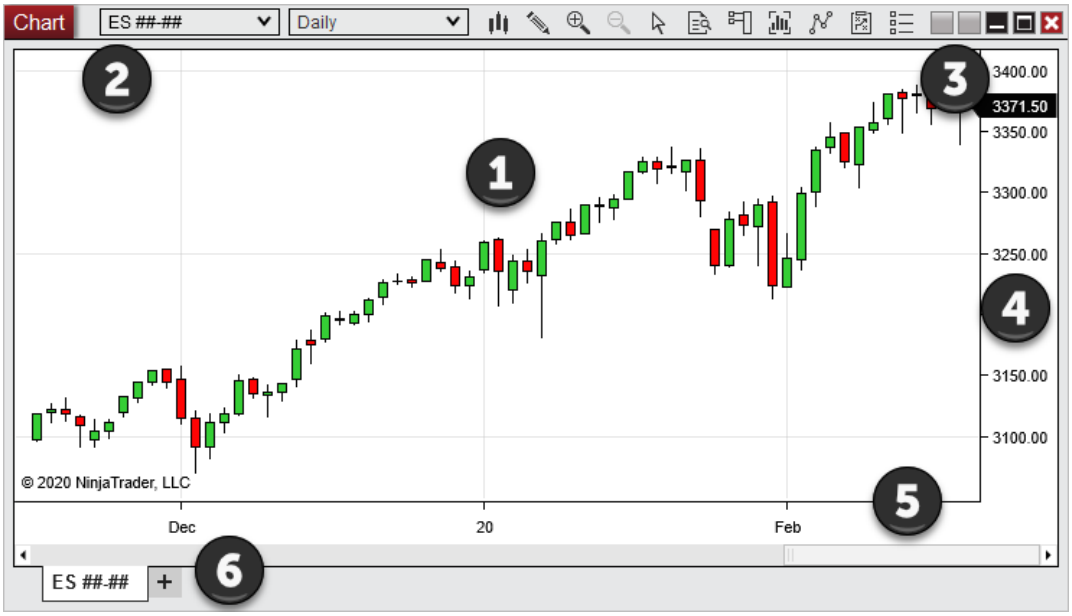
Once inside the Data Series window, there are multiple ways to choose an instrument. You can select an instrument from the available instrument lists, type the instrument symbol into the empty instrument field and press the enter key, or use the instrument lookup window by pressing the magnifying glass button next to

the instrument field. Please see the "Working with Price Data section of the Help Guide for more information on selecting instruments.

Understanding the chart display

Chart Display Overview

Each NinjaTrader chart is a free floating window that can be manually resized by dragging the edges of the window for arrangement within the open Workspace.



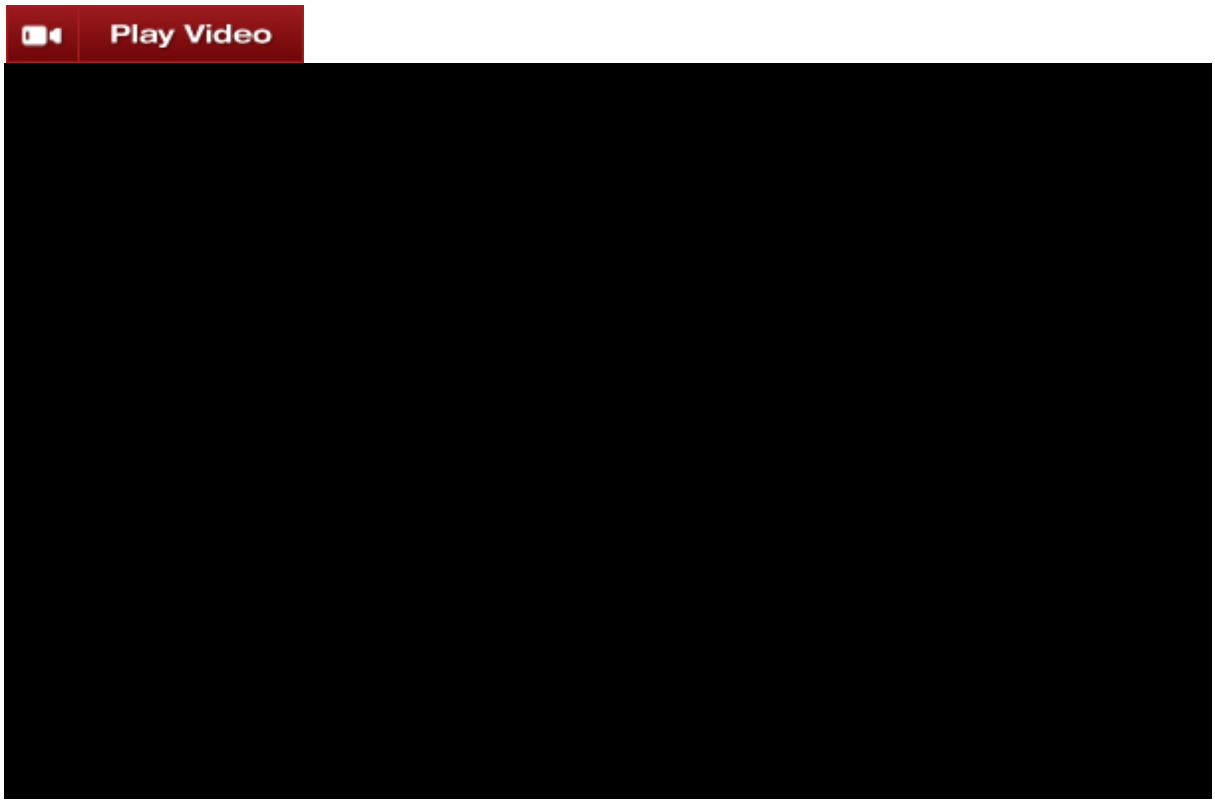
The chart image displays some of the common features you will see inside a NinjaTrader chart window:

1. Chart display area	Main display area of a chart where all chart objects (Data Series, Indicators and Drawing Objects) are plotted.
2. Chart tool bar	Access to chart features. Can be enabled or disabled via chart properties.
3. Link buttons	Window linking links windows to use the same instrument and can be applied to many NinjaTrader windows.

4. Price markers	Displays current price and indicator values in the left or right scale. Can be enabled or disabled on a per chart object basis through the Data Series or Indicators window. Drawing tool objects do not have price markers.
5. Horizontal scroll bar	Scrolls the horizontal axis left and right. (See the "*Navigating a Chart*" section of the Help Guide for more information.) Can be enabled or disabled via chart properties.
6. Chart Tabs	Displays the tabs enabled in the chart window. Tabs can be switched by clicking any configured tab with the left mouse button.

10.6.2 Navigating a Chart

The following section covers navigation and display of NinjaTrader charts.



How to change the horizontal scale and time range of a chart

Horizontal Scaling

To compress or decompress the horizontal axis, left mouse click in the x-axis margin and move the mouse cursor to the left or right. Alternatively, use the Hot Keys CTRL + Up and CTRL + Down.

1. Click and drag to the right will compress the chart's time scale

2. Click and drag to the left will decompress the chart's time scale

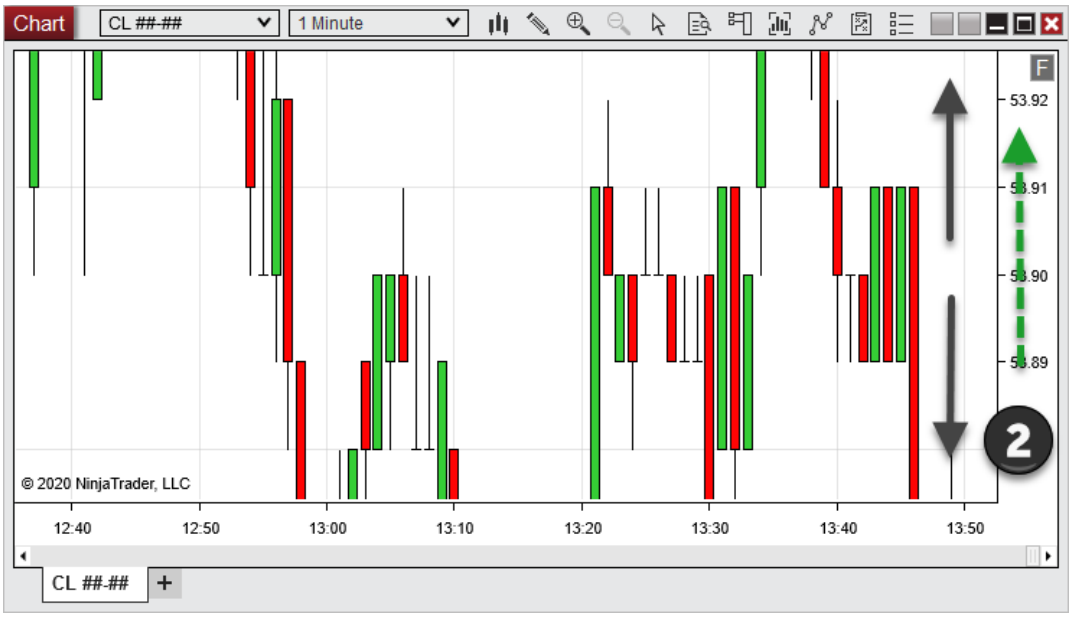
How to change the vertical scale and price range of a chart

Vertical Scaling

To compress or decompress the chart's vertical axis, left mouse click in the y-axis margin and move the mouse cursor up or down as shown in the images below:

1. Click and drag down will shrink the chart's price scale

2. Click and drag up will stretch the chart's price scale



Tip: You can also manually set the chart's price scale to a specific fixed price range from the Chart Panel Properties window.

Fixed vs Automatic Scaling

A box with an "F" (Fixed) will appear in the upper right corner of the chart margin any time the vertical chart axis is manually adjusted. This signifies the chart axis is set to a "fixed" scale. Left mouse click this button to return to automatic scale.

How to scroll a chart (panning)

Horizontal Scrolling (panning chart left or right)

You can pan the chart left or right via the following controls:

Mouse controls	Scrolls the chart
Horizontal chart scroll bar at bottom of chart	1 bar at a time
Left mouse click and hold on chart canvas and drag left or right	1 bar at a time
CTRL key + Left mouse click and hold in the x-axis (time axis) and drag left or right	1 bar at a time
Mouse scroll wheel	3 bars at a time
CTRL key + mouse scroll wheel	9 bars at a time

Keyboard controls	Scrolls the chart
Left arrow key	Backward 1 bar at a time
Right arrow key	Forward 1 bar at a time
Page Up (or CTRL key + left arrow key)	Backward one page at a time
Page Down (or CTRL key + right	Forward one page at a time

arrow key)	
Home key	To the very beginning (first bar)
End key	To the very end (current bar)

Range Icon

If the horizontal axis is scrolled to the left or right from its starting location, a "return" icon will appear in the top right hand corner of the chart. Left mouse click on the icon to return the horizontal axis to view the last "live" data on the chart.

Vertical Scrolling (panning chart up or down)

To pan the chart up or down:

CTRL + Left mouse click and hold on chart margin and drag up or down as depicted in the images below.

1. CTRL + Click and drag down will shift the chart's price scale up

2. CTRL + Click and drag up will shift the chart's price scale down

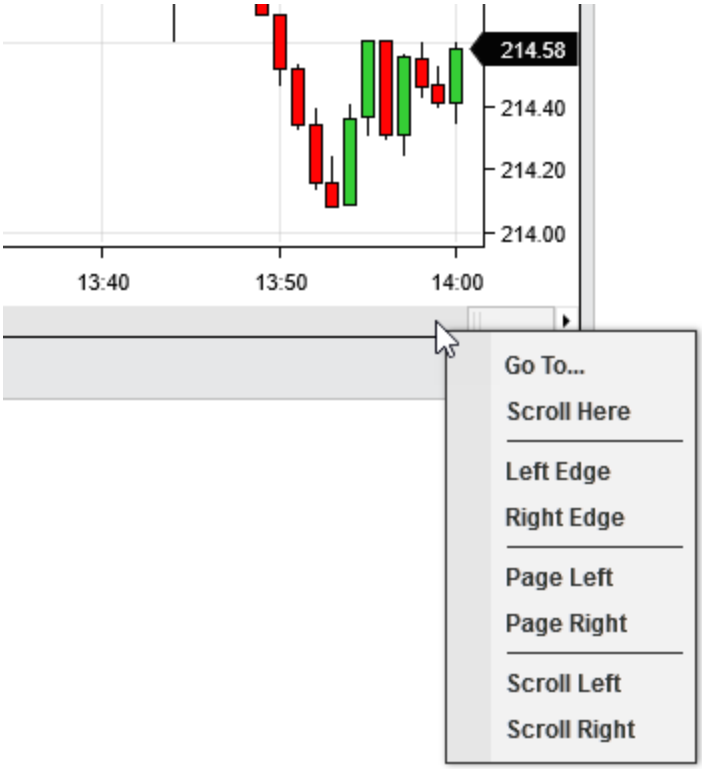
Free Mode Scrolling

You can also navigate the chart by changing both the price axis and time axis at the same time by holding down the CTRL key + Left mouse clicking and dragging in the chart area. This will allow you to move both the price and time axis in whichever direction the mouse is dragged.

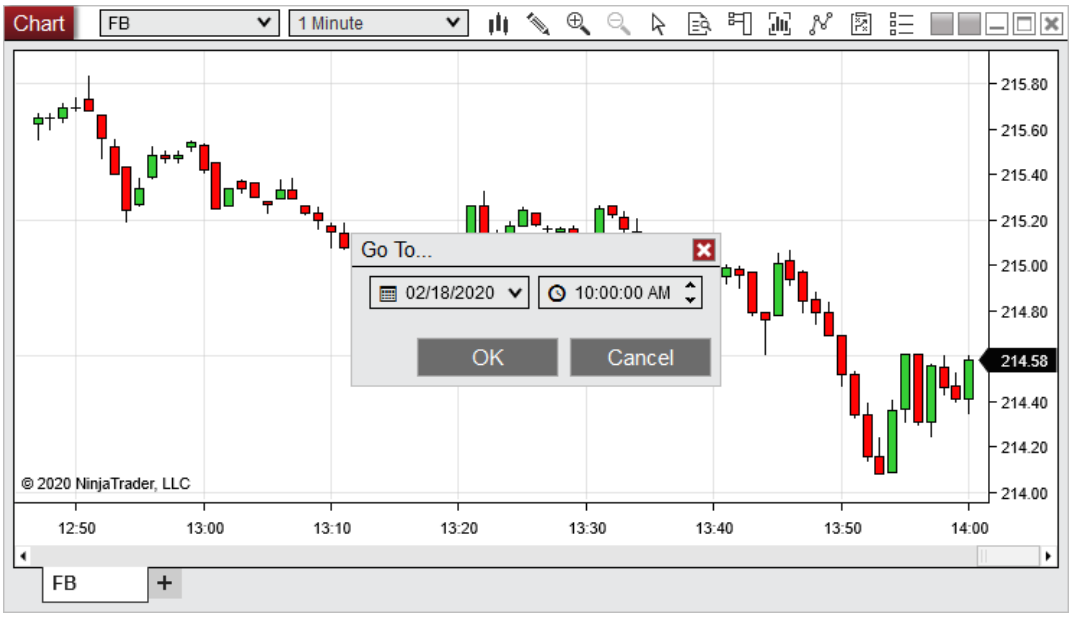
How to go to a specific time on the chart

Go To...

You can go to a specific date and time on a chart by right clicking on the scroll bar and selecting **Go To...**



A Go To... window will appear in which you can enter the desired date and time. Once complete, select **OK** to go to that time.



How to zoom in and out in a chart

Zoom In

To create a zoom frame around a chart area you want to focus in on:

1. Left mouse click on the Zoom In icon in the tool bar, select the **Zoom In** menu item within the right mouse button click context menu, or use the zoom in Hot Key CTRL+ ALT + Z
2. Left mouse click and while holding down the left mouse button, draw a zoom frame region and release the button.

The chart display area will zoom in to the selected frame area.

Zoom Out

Each zoom in can be undone to the prior zoom level with a zoom out. To zoom out, left mouse click on the Zoom Out icon in the chart tool bar, select the **Zoom Out** menu item within the right mouse button click context menu, or use the zoom out Hot Key CTRL+ ALT + O.

How to change the bar spacing and width

Bar Spacing

To change the spacing between bars:

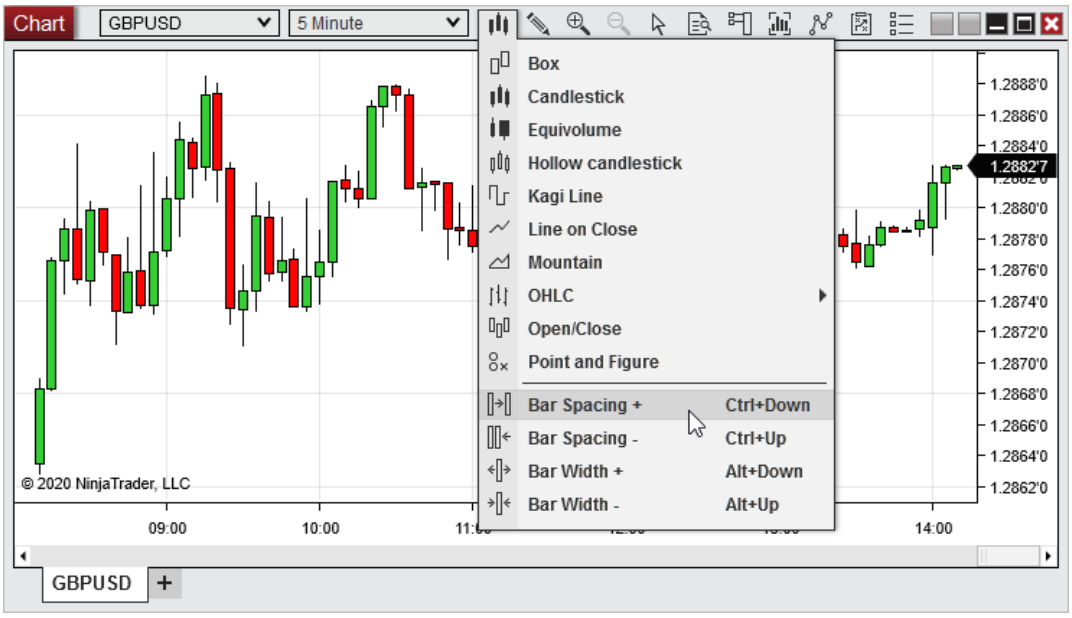
- CTRL + Up arrow key decreases bar spacing
- CTRL + Down arrow key increases bar spacing

Bar Width

To change the width of bars:

- ALT + Up arrow key decreases bar width
- ALT + Down arrow key increases bar width

Alternatively, left mouse click on the "Chart style" chart toolbar icon to access bar spacing and width functions



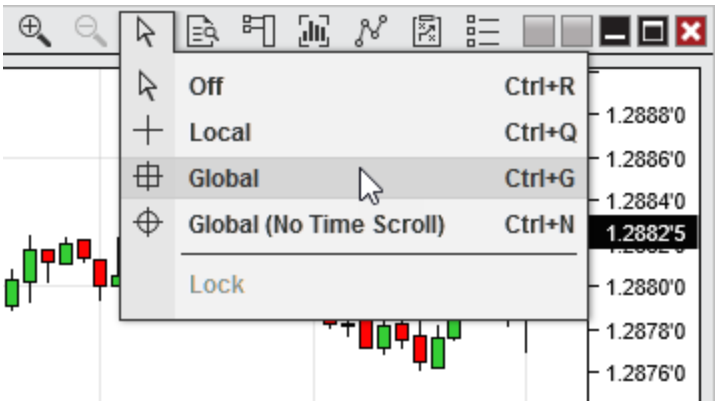
Note: On a multi-series chart, before changing bar spacing or width, you must select the Data Series you want to adjust by left mouse clicking on it. If none is selected, the primary Data Series of the chart will be adjusted.

How to change the cursor type

Cursor Type

You can have either the standard windows pointer, cross hair or global cross hair for chart navigation. You can toggle between cursor modes via the right mouse click context menu cursor sub menu, the "Cursor" chart toolbar icon or via the following shortcut keys:

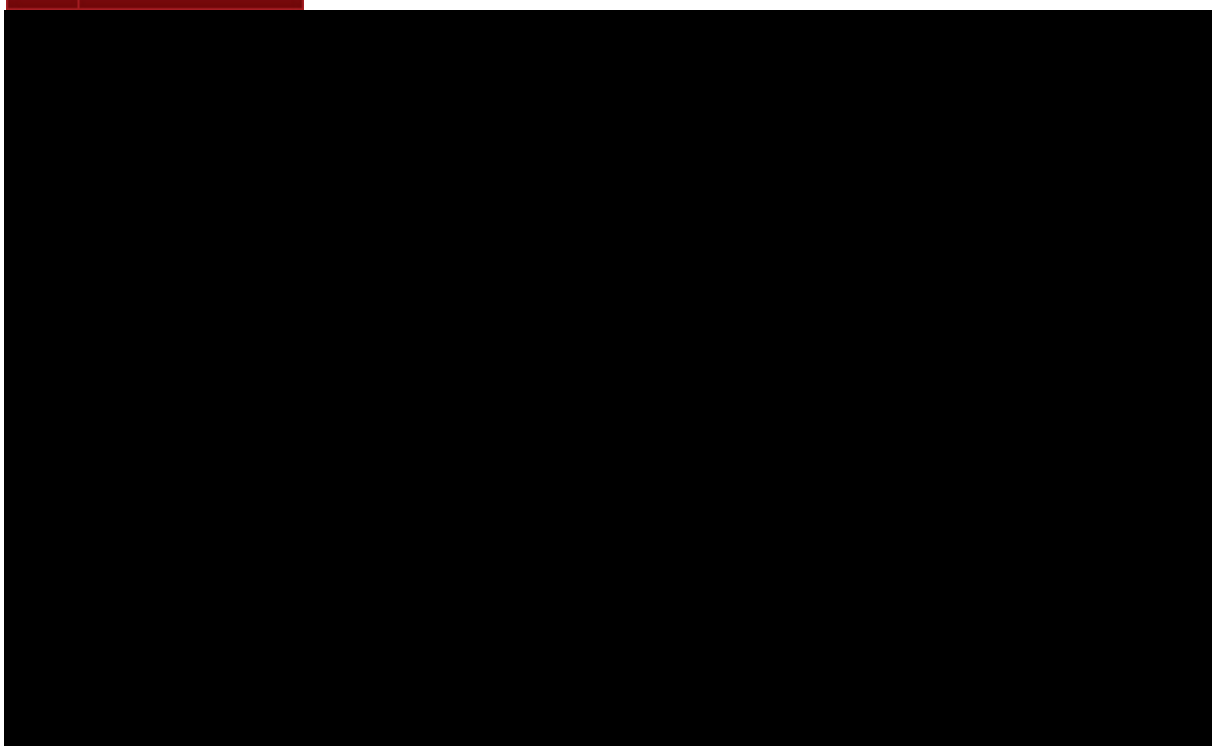
CTRL + R	Pointer
CTRL + Q	Cross Hair
CTRL + G	Global Cross Hair (links crosshairs when enabled on two or more charts)



10.6.3 Chart Panels

A chart is comprised of **Panels** that contain chart objects such as Data Series, Indicators and Drawing Tools. **Panels** are added to a chart during the process of adding/editing a **Data Series** or **Indicator**. Every **Panel** has three independent scales to which you can associate a chart object to. Each scale can be uniquely customized via the panel properties (see "*Understanding panel properties*" sub-section below for more information).

Play Video



Understanding chart panels

Panel Scales

When adding a **Data Series** or **Indicator** to a chart, you can set the **Scale justification** property to align the chart object to any of the following scales within the **Panel**:

- Left
- Right
- Overlay

With the exception of the **Overlay** scale, a price scale will only be displayed in a **Panel** if there is one or more chart objects justified to it. The **Overlay** scale does not have a visible price scale however, any chart objects justified to this scale will display their price markers first on the **Right** scale if one exists otherwise they are displayed on the **Left** scale. All scales can be shared by multiple chart objects.

Tip: In addition to changing a chart objects scale justification property via the **Data Series window** or **Indicators window**, you can drag and drop a chart object onto different scales. Please see the section "*How to drag and drop chart objects"* section located on the Working with Chart Objects topic page.

The image above depicts the continuous ES futures contract justified to the **Right** scale and a Stochastics indicator justified to the **Left** scale within the same **Panel**.

Panel Context Menu

Right mouse click within the price scale to access the **panel context** menu.

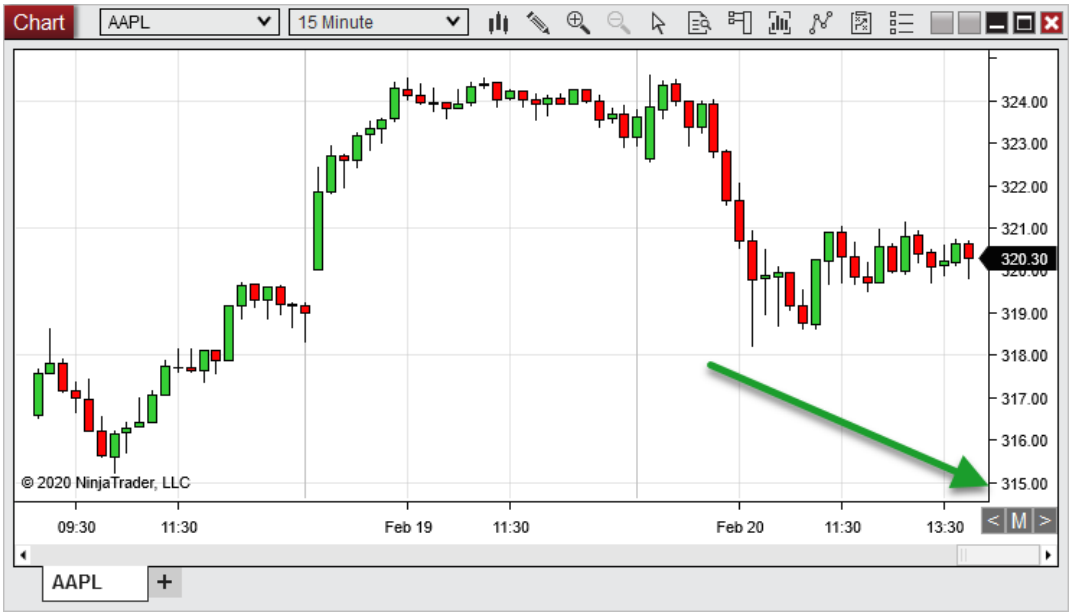
The following actions are available:

Move Up	Moves the **panel** up by one
Move Down	Moves the **panel** down by one
Maximize	Maximizes the **panel**
Restore	Restores the maximized window to the original size
Arrange All	Arranges all **panel**s to default proportions
Remove	Deletes the **panel** from the chart

Properties	Opens the **panel** properties window

Maximized panel display

Selecting **Maximize** from the **panel context** will change selected panel to be the only displayed panel on the chart tab. Using the left facing arrow "**<**" or right facing arrow "**>**" will navigate through each panel on the chart in a maximized display. Selecting the "**M**" button will restore the panels to their original default display.



The image above depicts the continuous ES futures 1-minute **Data Series** panel which has been **Maximized** and displays the controls available to navigate through a maximized panel display.

Understanding panel properties

Panel Properties

The **Panel Properties** menu can be opened by double left mouse clicking within the price scale or selecting the **Properties** menu via the **Panel Context** menu discussed above. The **Panel Properties** window will list the properties below grouped by each scale that is currently active on the **Panel**.

The following properties can be adjusted:

Range	Sets the range to "**Automatic**" or "**Fixed**." A fixed range allows the manual setting of the upper and lower boundary of the chart. The range can also be manually defined via the mouse. Please see the Navigating a Chart topic's section on "*How to change the vertical scale and range of a chart*".
Based on	Sets a value indicating how the "**Automatic**" scale range is calculated. When set to "**Entire Date Range Series Only**", Data Series and Indicator values for the entire date range of the chart (draw objects are ignored) are used to calculate the vertical scale range. When set to "**Screen Date Range**", all visible objects on the screen are used.
Horizontal grid lines	Sets the **Horizontal grid lines** displayed on the chart's price scale to "**Automatic**" or "**Fixed**." A fixed setting allows the manual definition of the intervals displayed include the **Horizontal grid lines interval type** and **Horizontal grid lines interval value**
Horizontal grid lines interval type	Sets a value of either "**Points**", "**Ticks**", or "**Pips**" which is used to calculate the interval between grid lines and labels.
Horizontal grid lines interval value	Sets the vertical interval of the horizontal axis. A value of 0 (zero) will enable the automatic generation of grid line intervals. The Right scale will always take precedence over the left scale if both are set to user defined custom grid line intervals.
Margin type	Sets the calculation mode for determining the upper and lower **panel** margins by "**Price**" or

	"**Percent**". (Percent values are whole percents. For example, entering a value of "1" equals 1%.)
Margin lower	Sets the lower margin value
Margin upper	Sets the upper margin value
Maximum	Sets the scale's upper boundary when using "**Fixed**" range
Minimum	Sets the scale's lower boundary when using "**Fixed**" range
Type	Sets the scaling type to "**Linear**" or "**Logarithmic**"

10.6.4 Working with Objects on Charts

Charts in NinjaTrader can contain and display multiple objects, including **Data Series**, **Drawing Objects**, and indicator plots. Objects on charts can be managed in a number ways, such as dragging and dropping them to new panels, changing the axis of their price scale (if applicable), or changing the order in which they are painted on a chart.

▼ How to drag and drop chart objects

Drag and Drop

A **Data Series** or Indicator can be dragged and dropped to various areas of the chart to quickly change which panel it is displayed in.

Left mouse click on a chart object within a chart, then drag it to any of the following areas of the chart and release the mouse button:

1. Upper limit - Creates a new panel at the top of the chart
2. In between panels - Creates a new panel in between two existing panels
3. Lower limit - Creates a new panel at the bottom of the chart
4. Center area of a panel - Relocates the selected chart object to this panel and automatically determines the most suitable scale justification

5. Left or right margin of a panel - Relocates the selected chart object to this panel (unless already in the selected panel) and changes the scale justification to the selected side of the panel.

When you drag a selected object to the upper or lower edge of a chart, or between two panels, a blue band will appear. This indicates that a new panel will be created when you drop the object in that location.

Tabs and Windows

In addition to moving around within a single chart tab, A **Data Series** or indicator can be dragged and dropped into any other chart window or tabs in your workspace. The following drag and drop actions can be performed:

1. Drag an indicator to an existing tab in any chart window - Duplicates the indicator in the tab or window into which it is dropped, leaving the original instance of the object intact
2. Drag a **Data Series** to an existing tab in any chart window - Replaces the primary **Data Series** in that tab with the one dropped into it
3. Drag a **Data Series** to the upper/lower limit, or between two panels, of a separate chart window - Creates a new panel, creating a multi-series chart if only one **Data Series** had previously been applied
2. Drag an indicator or **Data Series** to a New tab (+) - Creates a new tab and duplicates the object within it

How to copy and paste chart objects

Copy and Paste

A **Data Series**, indicator, or **Drawing Object** can be copied and pasted to various areas of a chart to quickly duplicate an object and its properties. Chart objects can be copied in one of two ways:

- Left mouse click the chart object to select it. Next, right mouse click the object, then click the **copy** menu item.
- Left mouse click the chart object to select it, then use the Windows default CTRL + C Hot Key

After copying, chart objects can be pasted into the following areas:

- Current chart window or tab - **Data Series** and indicators will be duplicated in a new panel. **Drawing Objects** will be pasted with a slight offset from the copied object's location.
- Separate chart window or tab - **Data Series** will be placed in a new panel within the chart window or tab in which it is dropped. Indicators will either be plotted in an existing panel or in a new panel, depending on the indicator's "Overlay" property. **Drawing Objects** will be placed in the same panel number as the one from which they are copied, if it is available.

Note: When an indicator is pasted from one chart to another, the indicator will use the same input series if it is applied to the chart into which the indicator is pasted. Otherwise, it will use the second chart's primary **Data Series**.

How to change the z-order (paint order) of a chart object

Z-Order

Objects within a panel can be adjusted to appear behind or in front of another chart object. The specific layer on which an object sits is referred to as the "**z-order**."

You can change the **z-order** (paint order) of all chart objects within each individual panel. Each chart object is assigned a **z-order** value, which informs you where in the paint order that particular object resides. As a rule of thumb, there are as many **z-order** levels in a panel as there are chart objects in that panel. For example, if you had a **Data Series** and an SMA indicator in the same panel, there would be two painting levels. Level 1 is the top most level, which means that any chart object on Level 1 will be painted above all others. Continuing our example, if the **Data Series** was on Level 1 of 2 and the SMA indicator was on Level 2 of 2, that would mean the **Data Series** would be painted on top of the indicator.

The image below depicts a "Rectangle" drawing object set at z-order Level 3 of 3, which is behind both the Stochastics indicator (Level 2 of 3) and the ES ##-## Data Series (Level 1 of 3).

To adjust the **z-order** of an object:

1. Select the chart object by left mouse clicking on it
2. Hold down the "Shift" key on your keyboard and roll the mouse scroll wheel up or down to change the z-order of the object. As you scroll, the object's **z-order** will be displayed near your mouse cursor.

> **Note**: **Drawing Objects** originating from a NinjaScript indicator or strategy will all generally share the same z-order as the script. In this case, the **z-order** of objects must be changed within the code of the indicator or strategy.

10.6.5 Working with Price Data

A **Data Series** represents a series of price data, which can be displayed on a chart using one of several Bar Types and **Chart Styles**. One or more **Data Series** will be applied to a new chart when it is created, and additional **Data Series** can be added, edited, or removed via the **Data Series** window.

Play Video

Understanding the Data Series Window

The **Data Series** window is used to configure the **Data Series** within a chart, edit **Data Series** parameters, and save default values for different **Period Types**.

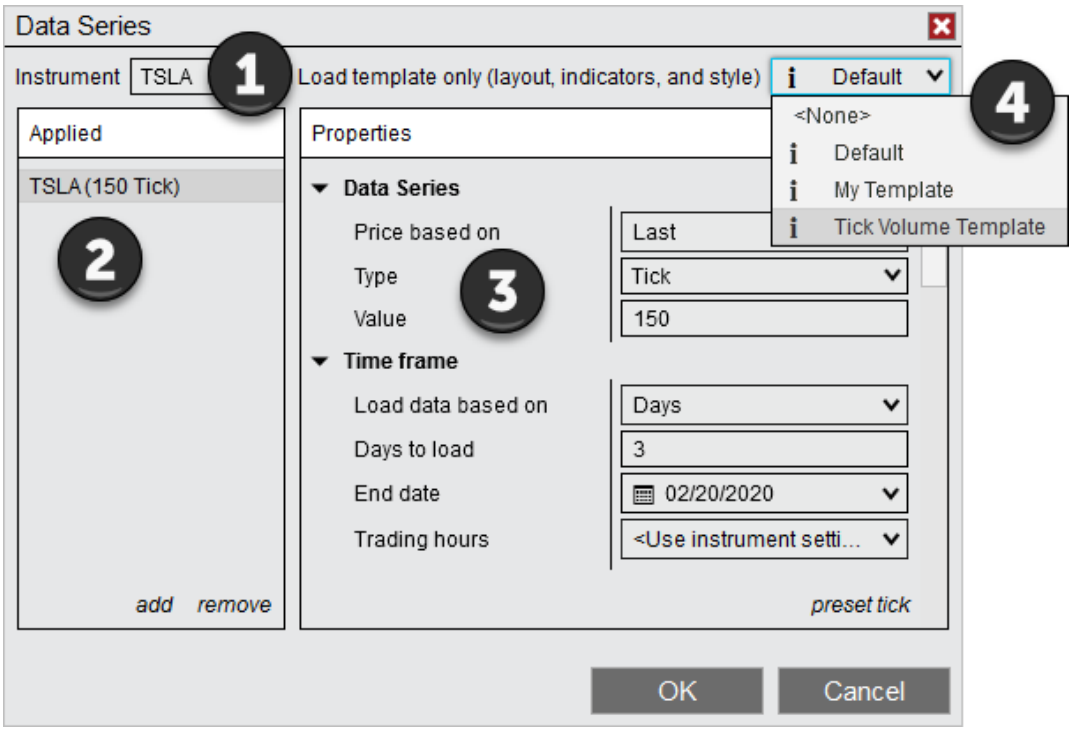
Accessing the Data Series Window

There are multiple ways to access the **Data Series** window:

- Select the **New** menu from the NinjaTrader **Control Center**, then select the **Chart** menu item.
- Right mouse click in the chart background and select the **Data Series** menu item.
- Use the default CTRL+F Hot Key from an open chart.
- Double left mouse click on a **Data Series** within the chart.
- Right mouse click on a selected **Data Series** within a chart, then select the **Properties** menu item.

Sections of the Data Series Window

The image below displays the four sections of the **Data Series** window.



1. **Instrument Selector**
2. **Data Series** currently applied to the chart
3. Selected **Data Series'** parameters
4. Saved Chart Templates that can be applied to the new chart. See the Saving Chart Defaults and Templates page for more information.

Note: If a Chart Template is selected, settings from that template will take precedence over any settings manually configured on the **Data Series**. For example, **Trading Hours** currently configured will be ignored, and the chart will use the **Trading Hours** which were saved in the **Chart Template.**

How to add a Data Series

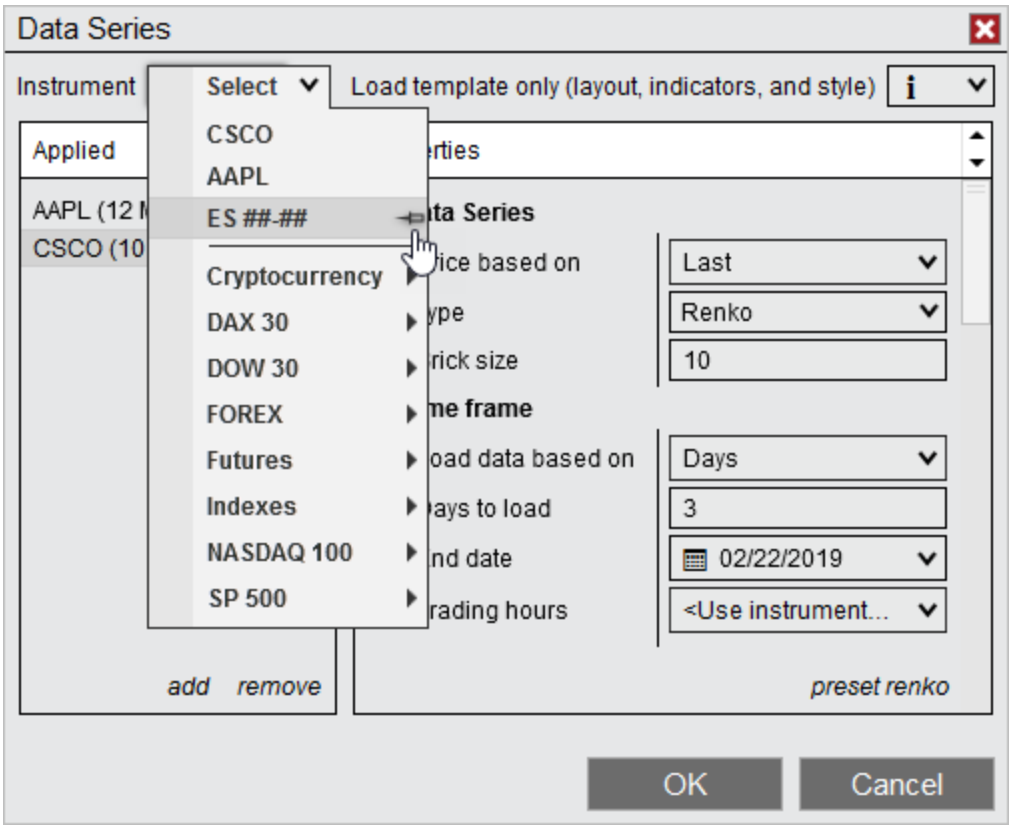
Adding a Data Series

Multiple **Data Series** objects can be applied within a single chart. A new panel is automatically created for each **Data Series** added, unless the "Panel" property is manually changed to an existing panel. There are multiple ways to add a **Data Series** to a chart using the **Data Series** window:

1. Use the **Instrument Selector** dropdown menu to select a recently used or pinned instrument, or any instrument in an **Instrument List**.
2. Type the instrument symbol (including the contract month for futures instruments) directly into the **Instrument Selector**, then press the "Enter" key.
3. Left mouse click on the magnifying glass icon next to the **Instrument Selector**. In the window that appears, use the search field to search available instruments by symbol or description, then double left mouse click on an instrument in the search results to add it to the list of applied **Data Series**.

The added **Data Series** will now be visible in the list in the "Applied" section, allowing you to change any parameters to desired values (see the "*How to edit Data Series parameters*" section below).

Tip: A **Data Series** can also be added by typing directly into an open chart. Type the plus symbol (+) followed by the instrument symbol, contract month for Futures, and appropriate interval value. For example, typing "+ES ##-## 5M" will add a 5 minute ES continuous contract **Data Series** to the selected chart (See the "*How to change a Data Series*" section below for more information).



In the image above, we can use the Instrument Selector to add a recently viewed or pinned instrument, as well as any instruments in an **Instrument List**.

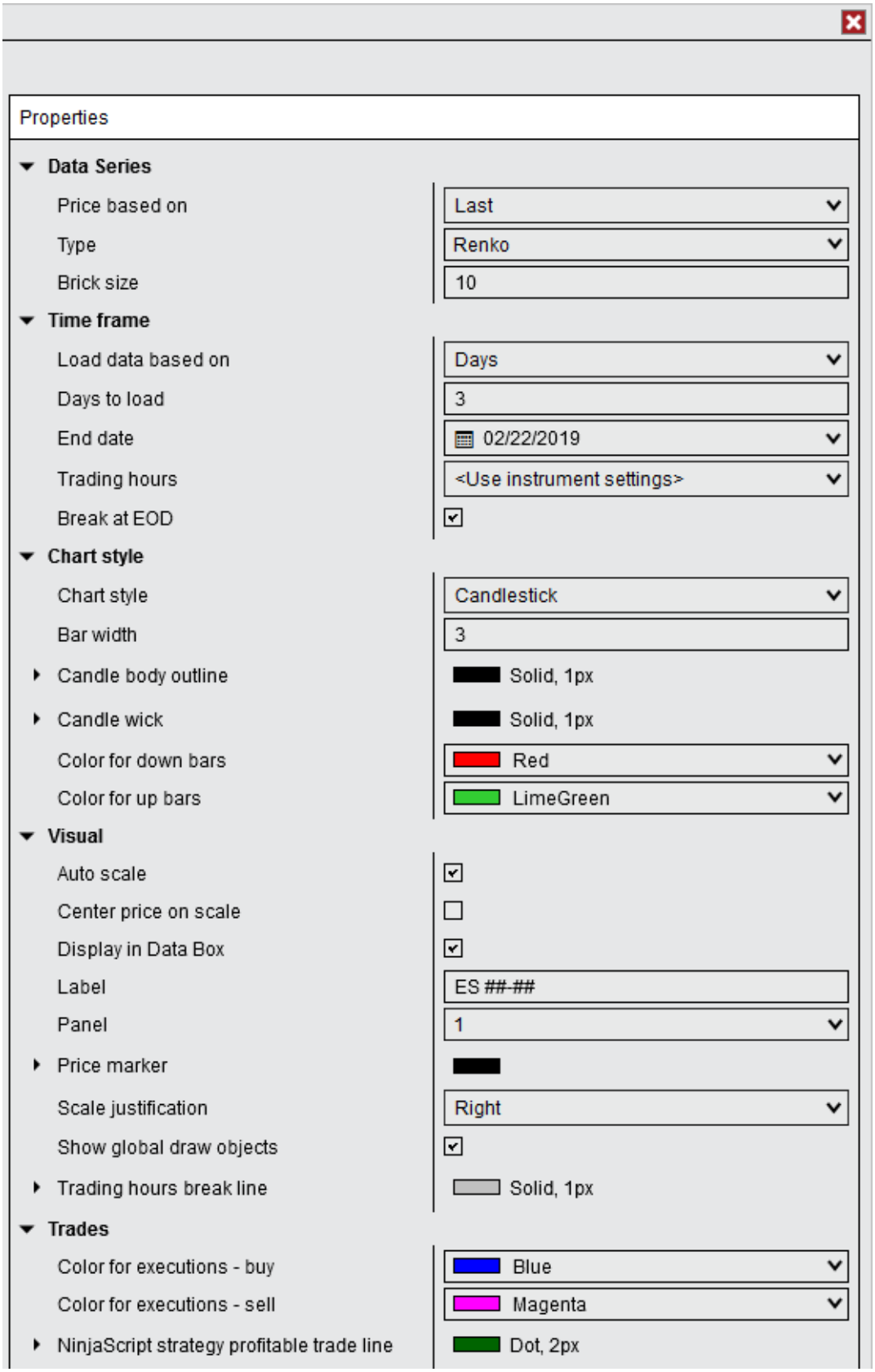
How to edit Data Series parameters

Editing a Data Series

A **Data Series** object's parameters are available to configure within the **Data Series** window once it has been added to a chart (see the "*How to add a Data Series*" section above).

To edit **Data Series** parameters:

1. Open the **Data Series** window (see the "*Understanding the Data Series window*" section above).
2. Select the **Data Series** you would like to edit in the "Applied" section.
3. Once selected, the **Data Series** parameters will be available to edit on the right hand side.



Available **Data Series** parameters can be found in the list below:

Data Series Parameters

Price based on	Sets the type of market data used to drive the **Data Series**
Type	Sets the bar type of the **Data Series**. See the Bar Types page for more information.
Value	Sets the **Data Series** value, based on the selected **Bar Type**
Tick Replay	Enables Tick Replay on the selected **Data Series**. This option will only display when "Show Tick Replay" is enabled in the Options window
Load data based on	Determines how much data is loaded based on number of bars, number of days, or a custom date range.
Days Back / Bars Back / Start Date	Sets the value for the amount of historical data to load, based on the "Load Data Based On" setting. The label on this property will change based upon what you have selected for the "Load Data Based On" property.
End date	Sets the end date of the chart. If the specified end date is within the range of an applied **Trading Hours** template whose end time falls on a future date, then the Chart will end on that future date.
Trading Hours	Sets the **Trading Hours** template to be used for the **Data Series**. See the Trading Hours page for more information.

Break at EOD	When enabled bars will be cut off at the end of the **Trading Hours** session regardless of whether it is fully complete. So a 10 range bar may close with a range of 3 or a 4 hour bar may close after 2 hours. When disabled, such a bar will continue to develop until it is complete, potentially causing it to post outside of the **Trading Hours** session. For more information, see the Break at EOD page.
Chart Style	Sets the style of the bars. Custom Chart Styles can be created via NinjaScript to extend the pre-built list.
Bar Width	Sets the width of the bars drawn on the chart
Additional Chart Style Options	Additional options for configuring bar colors and related properties will be displayed beneath the Bar Width property, depending on which **Chart Style** you have selected.
Auto Scale	When enabled, the **Data Series** will be part of the chart's auto scaling. • In case the chart is set to a fixed scale, this property has no effect. • In case there are no objects on the chart which have this property set to true, the first chart object will be used for the chart's auto-scaling.
Center on Price Scale	When enabled, the current price will be centered on the price axis, and all visible historical bars will be scaled accordingly
Display in Data Box	Enables or Disables the display of the selected **Data Series** in the Data Box

Label	Sets the label text to be displayed in a chart panel when more than one **Data Series** has been applied to a chart. This can be left blank to remove the label entirely.
Panel	Sets the panel in which the selected **Data Series** will be plotted. When more than one **Data Series** has been added to a chart, all but the first **Data Series** in the list will provide the option to plot in a **New Panel** in the "Panel" field.
Price Marker	Expanding this property will allow you to change the color for the price markers on the chart, as well as enable or disable the price markers' visibility.
Scale justification	Sets the scale on which the **Data Series** will be plotted. Possible values are "Right," "Left," and "Overlay"
Show Global Draw Objects	Sets whether **Global Drawing Objects** will be displayed for this **Data Series**. See the "*Understanding local vs. global drawing objects*" section of the Working with Drawing Tools & Objects page for more information.
Trading Hours Break Line	Sets the color, dash style, and width of the **Trading Hours** break line plotted on the chart for the selected **Data Series**
Color for Executions - Buy	Sets the color for Buy-side execution markers
Color for Executions - Sell	Sets the color for Sell-side execution markers

NinjaScript Strategy Profitable Trade Line	Sets the color, dash style, and width for the lines connecting entries and exits of profitable trades taken by a NinjaScript strategy
NinjaScript Strategy Unprofitable Trade Line	Sets the color, dash style, and width for the lines connecting entries and exits of unprofitable trades taken by a NinjaScript strategy
Plot Executions	Sets the plotting style of the trade executions. **Note**: Real-time executions are timestamped based on the timezone set in the "General" section of the **Options** window, which can be accessed from the Tools menu in the **Control Center**. Please see the How Trade Executions are Plotted page for more information.

Saving Data Series Parameters as Default

You can optionally save your customized **Data Series** parameters as default. Defaults are saved based on the **Interval Type** selected. Saving defaults will recall your customized settings the next time you add a **Data Series** with that specific **Interval Type** to a chart. Please see the Saving Chart Defaults and Templates page for more information.

How to change a Data Series

Data Series can be edited in several ways after being added to a chart.

Changing an Instrument via the Chart Toolbar

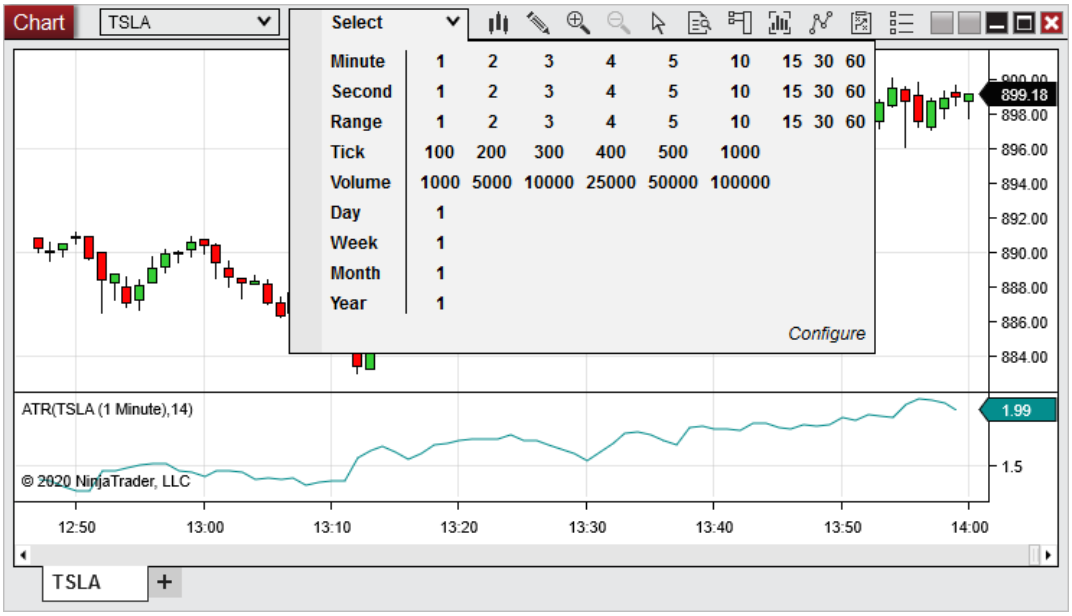
To change an instrument using the chart toolbar:

1. Left mouse click on the instrument drop down menu in the chart toolbar

2. Select a recent or pinned instrument from the top of the list, or expand any of the **Instrument Lists** for additional selections (for more information about editing **Instrument Lists**, see the Instrument Lists page).

Using the Interval Selector

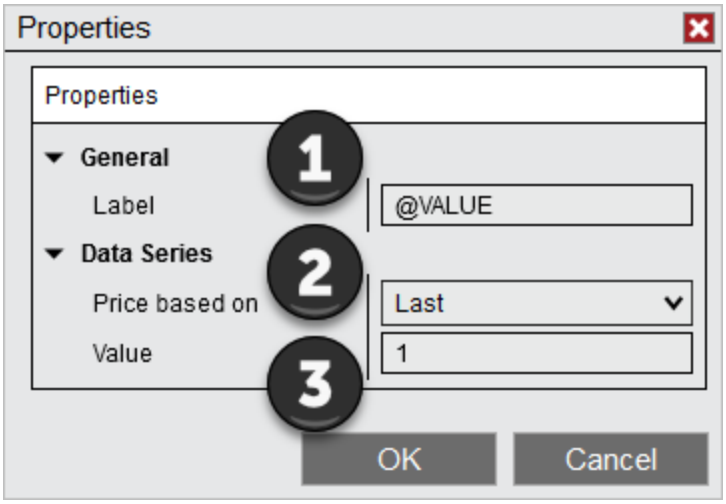
The **Interval Selector** can be used to change a **Data Series** interval directly from the chart toolbar. The **Interval Selector** comes pre-populated with commonly used intervals, but you can add additional intervals of your choice at any time. To access the **Interval Selector**, left mouse click the dropdown menu displaying the currently selected interval, located next to the instrument dropdown menu on the chart toolbar. To change the currently selected interval, select any of the values corresponding to the row labeled with your desired interval type. For example, to switch to a 5,000 **Volume** interval, click the "5000" option in the "Volume" row.



Adding Intervals to the Interval Selector

To add a new interval to the **Interval Selector**, first click the **Configure** option. The **Configure** window that appears is separated into two sections. In the "Intervals" section on the left side, you can select any existing **Interval Type** to view, add, edit, or remove any specific interval value set up for that interval type. In this section, you can add new **Interval Types** to the list via the **add** option, remove an **Interval Type** from the list via the **remove** option, or move **Interval Types** higher or lower in the list via the **up** and **down** options. In addition to the **Interval Types** already available, you can add Heiken Ashi, Kagi, Line Break, Point and Figure, or Renko to the list.

With an **Interval Type** selected in the "Intervals" section, you can manage the specific intervals available for that type in the "Values" section. To add a specific interval to the list for a specific **Interval Type**, select the **add** option. A window will appear, in which you can set the label and **Data Series** options to be used when that interval is selected:



1. The "Label" field sets the label that will be displayed in the **Interval Selector** for this interval. Entering "@VALUE" in this field will display the value entered in the "Value" field in the section below. Alternatively, you can enter any text or numbers in this field to label the interval.

2. The "Price Based On" field determines whether the underlying **Data Series** will be based upon the Ask, Bid, or Last price for the selected instrument.

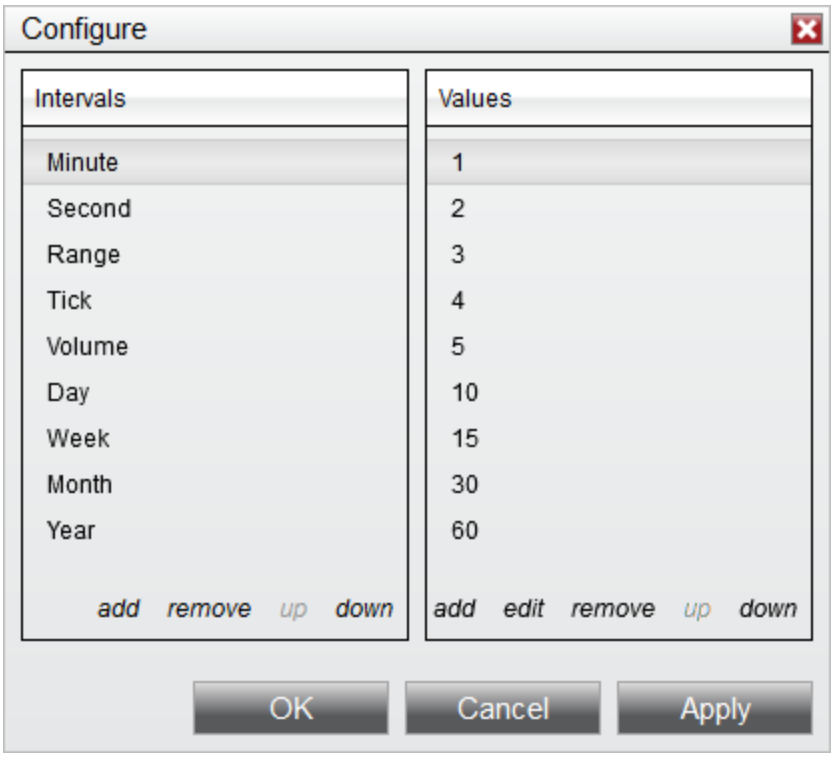
3. The "Value" field sets the value to be used for the interval, based on the **Interval Type**.

Editing, Sorting, and Removing Intervals

To remove an interval from the list for a specific interval type, first select the interval, then select the **remove** option.

To edit the parameters of an existing interval, select the **edit** option instead.

To change the placement of an interval in the list, first select the interval you wish to move, then select the **up** or **down** options to move it higher or lower in the list. Moving an interval higher in the list will cause it to be displayed further to the left in the **Interval Selector**, and moving it lower in the list will cause it to be displayed further to the right.



The **Configure** window pictured above allows the addition, removal, or editing of interval types and specific intervals in the **Interval Selector**.

Changing and Adding Instruments and Intervals with the Keyboard

You can change instruments or intervals by pressing a letter or number key in a selected chart. When a letter or number key is pressed, the **Instrument Overlay** appears. Within the **Instrument Overlay**, you can change the instrument, interval, or chart type by using the formats in the table below and pressing the "Enter" key when finished. If multiple instruments are displayed in the chart, you can change a specific instrument by left mouse clicking to select it before typing. If no instrument is selected, the primary instrument is changed.

To change an instrument:	Type the instrument symbol (Add the contract month for futures instruments). Examples: "ES ##-##" for E-mini S&P 500, "AAPL" for Apple stock, or "EURUSD" for Euro/USD forex pair.
To change an interval:	Type interval value plus the interval suffix (Value +suffix). Examples: "5M" for 5 minute bars, "100T" for 100 tick bars, "1D" for 1 Day bars,10 etc.
Available suffixes:	Suffix interval:
M	Minute
T	Tick
V	Volume
R	Range
S	Second

D	Day
W	Week
M O	Month
Y	Year
R E	Renko
To change instrument and interval	Type the symbol and interval together. For example, typing "AAPL 5M" will change to a 5 minute chart of Apple stock.
To add additional series of primary instrument	Type a plus sign (+) plus the interval. For example, typing "+5M" will add a 5 minute **Data Series** of the primary instrument.
To add additional series of any instrument	Type a plus sign (+) plus the instrument and interval. For example, typing "+AAPL 5M" will add a 5 minute series of Apple stock. If no interval is provided, then the same interval as the primary series will be added.

Removing a Data Series

Removing a Data Series

There are three ways to remove a **Data Series** from your NinjaTrader chart:

- Open the **Data Series** window *(see the "Understanding the Data Series window" section above).* Select a **Data Series** from the "Applied" section, then select the **Remove** option, then press the **OK** button to close the **Data Series** window.
- Left mouse click a **Data Series** on your chart to select it, then press the "Delete" button on your keyboard.

- Left mouse click a **Data Series** on your chart to select it, then right mouse click the **Data Series** and select the **Remove** menu item.

If only one **Data Series** is applied to a chart, it cannot be removed. However, the original **Data Series** added to a chart can be removed *if* there is at least one other **Data Series** is still applied.

10.6.6 Working with Multiple Data Series

Multiple Data Series

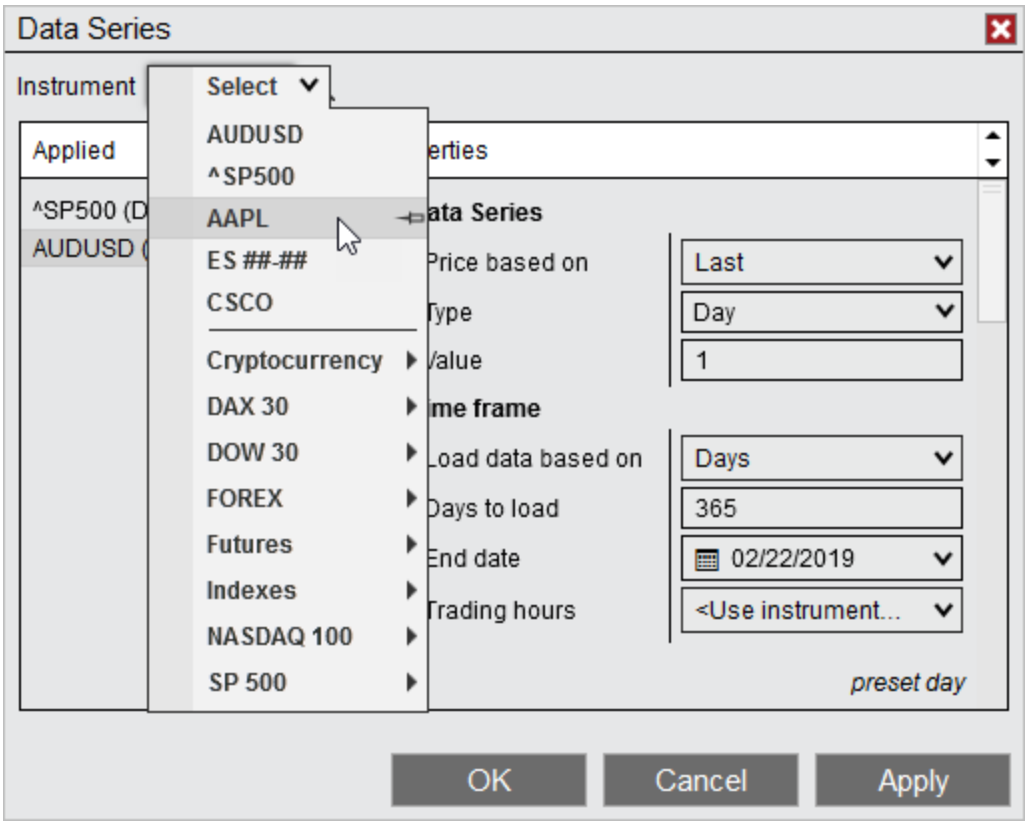
Multiple **Data Series** objects can be be viewed within a single chart window, and there are several ways to add **Data Series** to a chart.

▼ How to Add a Data Series

Adding Data Series

When you open a new chart, one or more **Data Series** will be applied, based on the instruments that you selected when creating the chart. You can add more **Data Series** to the chart (or remove existing **Data Series**) at any time via the following process:

1. Open the **Data Series** dialogue by either clicking the **Data Series** menu item on the chart toolbar, right-clicking in the chart and selecting the **Data Series** menu item from the Right Click menu, or double-clicking any selected **Data Series** on the chart.
2. Use the **Instrument Selector** or the **Search Tool** above the "Applied" section in the **Data Series** window to select a new **Data Series**.
3. Configure the **Data Series'** parameters as desired in the "Properties" section, then click the **OK** button



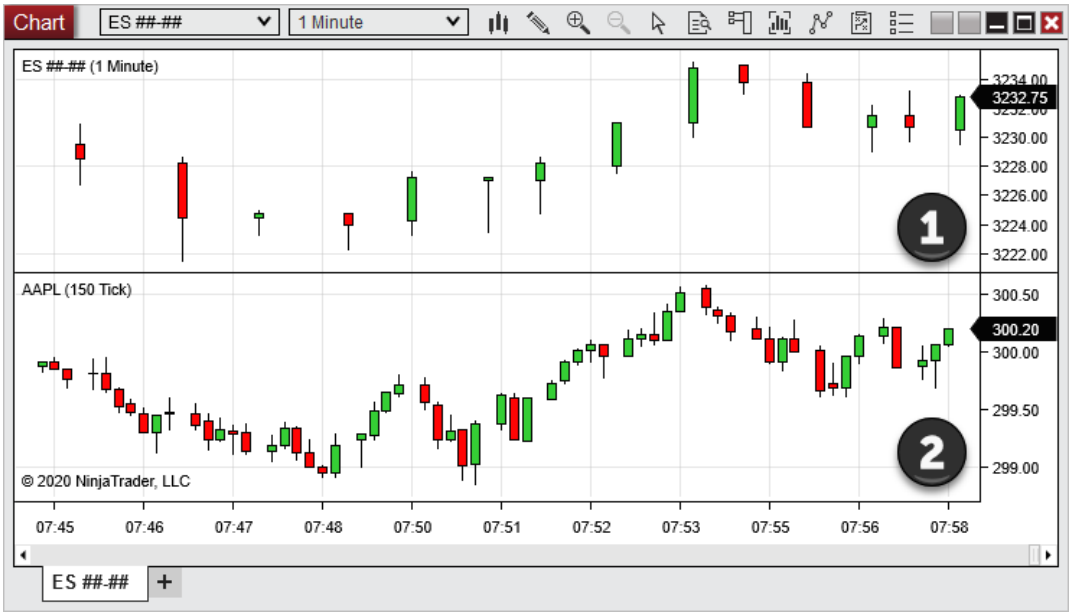
In the image above, we can use the **Instrument Selector** to add a **Data Series** to a chart which is already open.

Managing Multiple Data Series

Multiple Data Series

The image below shows two Data Series plotted within one chart window:

1. ES ##-## (1 Min)
2. AAPL (150 Tick)



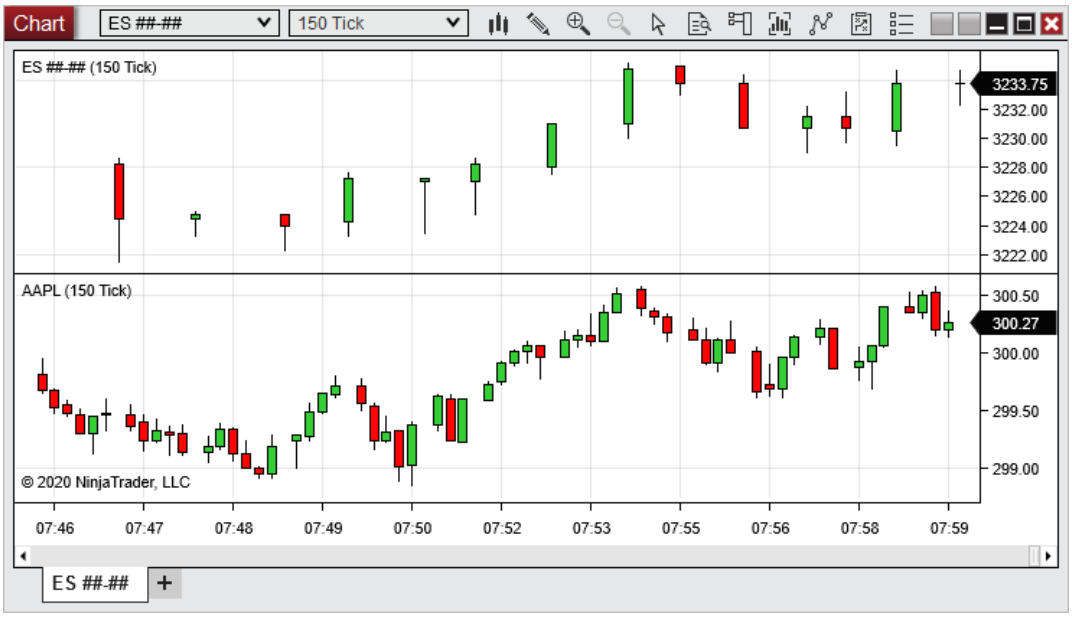
Each instrument is placed in its own panel by default, with the scale shown in the right margin of the chart. Many separate panels can be displayed within a single chart window. Instruments and indicators can alternatively be plotted within a single panel, as well. The scale of each **Data Series** can be justified to the right, to the left, or overlayed on the panel. Please see the "*Understanding panels*" section of the Navigating a Chart page for more information.

Tip: When more than one panel is displayed in a chart, you can temporarily maximize a panel to fill the entire chart window by right mouse clicking in the price axis of that specific panel, then selecting the **Maximize** option. To restore the panel to its original size and placement, you can then right mouse click in the price axis of the maximized panel, then select the **Restore** option.

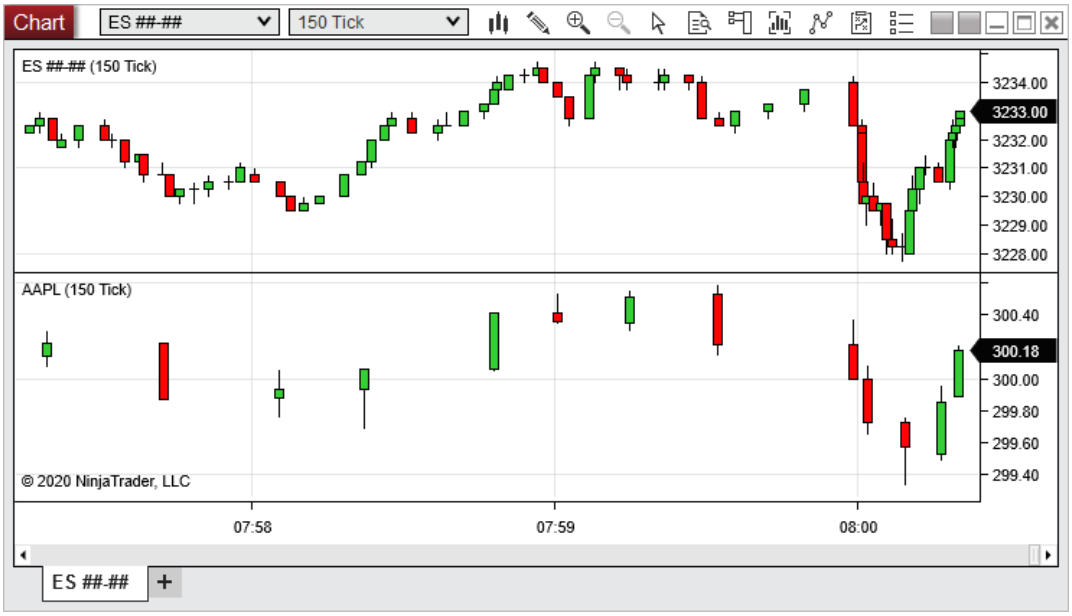
Equidistant Bar Spacing

Equidistant Bar Spacing is a chart property that determines whether bars are plotted with an equal distance from each other or plotted on a horizontal axis with even time spacing. The two images below display the same chart with this property set to True and False. When set to True, the distance between bars is equal throughout the chart. When set to false, the distance between bars is not necessarily the same. Bars are instead plotted on a fixed x-axis timeline on which every inch along the axis represents an equal amount of time. This provides the benefit of being able to gauge momentum on non-time based charts, such as tick or volume, by visualizing how long it takes to finish building the next bar. Gaps may occur if no bar formed during the time interval, and overlapping bars may occur if

bars are formed near the same time period. Both gaps and overlapping can be seen in the second image below. Equidistant Bar Spacing can be enabled or disabled within the Chart Properties window.



The image above shows two 150 **Tick Data Series** with "Equidistant Bar Spacing" set to True.



The image above shows the same two **Data Series** with "Equidistant Bar Spacing" set to False.

Equidistant Bar Spacing with Multiple Data Series

When adding two or more **Data Series** to a chart, the bar spacing will be determine by the "Primary" data series, which is typically the first series added to the chart.

Configuring Which Data Series is Primary

You can optionally re-configure another series to be "Primary" by right clicking on the chart bars and selecting "**Set as Primary**".

Aggregated X-Axis Time Line

When using multiple **Data Series** with different **Trading Hours** templates, NinjaTrader will set the time axis scale using the earliest begin time and latest end time of all **Trading Hours** templates applied to the **Data Series** on the chart. For example, if one instrument has a session begin time of 7:00 AM and an end time of 2:00 PM, and another has a session begin time of 8:00 AM and an end time of 4:00 PM, the chart will have a session begin time of 7:00 AM (from the first instrument) and an end time of 4:00 PM (from the second instrument).

10.6.7 Bar Types

NinjaTrader supports a large variety of chart **Bar Types**. This page explains how each **Bar Type** is created in a chart. Please see the Working with Price Data page for information on how to change **Bar Types**.

Notes:

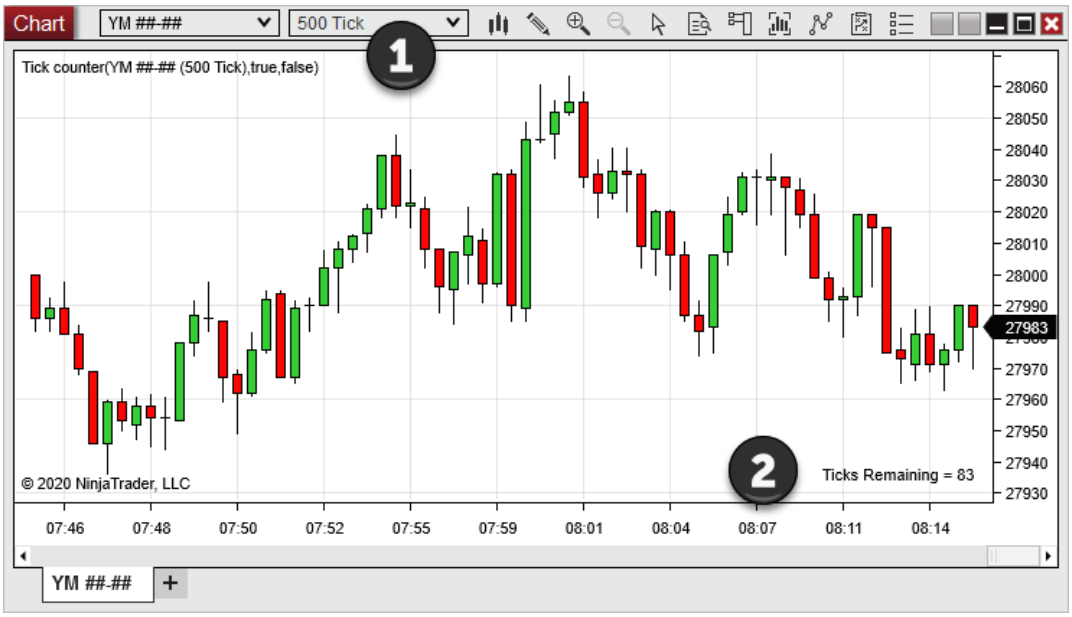
- For some **Bar Types**, the last bar of a session may be built as an incomplete bar due to the session ending before the bar could be completed. Each new session will have bars freshly built beginning from the first tick of the session. For example, the last bar of a session in a 10,000 Volume chart may contain a volume less than 10,000, while the next bar which builds on a new session would contain 10,000 volume. This behavior can be changed via the "Break at EOD" **Data Series** property. For more information, see the Break at EOD page.
- When backtesting different **Bar Types** (most notably Point and Figure and Renko) the backtest can yield different results than what you would experience in real-time, due to the nature of how the bars are constructed and the possibility of not having enough granular information to simulate what would have happened in real-time. Please see the Discrepancies: Real-time vs Backtest page for more information.

- When working with TickReplay, the bars will be built from tick data available through the provider or local repository. For developing NinjaScript objects taking advantage of this option, please see this link.

Understanding Tick bars

Tick Bars

A **Tick** bar is based on a specific number of ticks. A bar will continue to develop until the specified number of ticks is reached. The next tick will then result in a new bar being created.

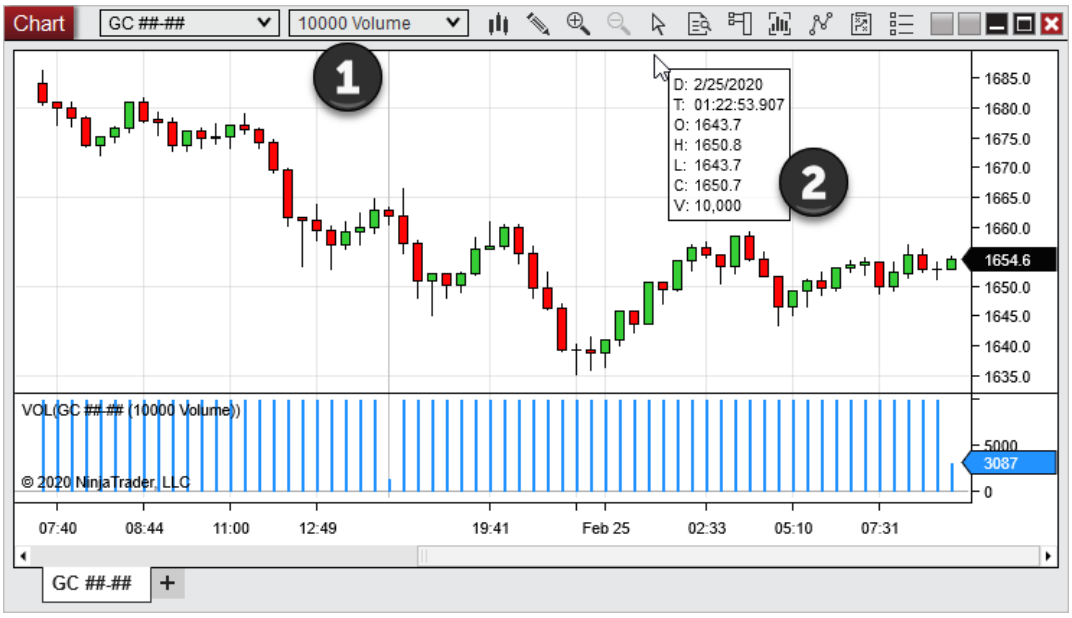


1. Each historical bar in the the 500 Tick chart shown above plots a total of 500 ticks.
2. The "Tick Counter" indicator has been applied to the chart to show the number of ticks remaining in the current bar.

Understanding Volume bars

Volume Bars

A **Volume** bar is based on a specific number of units traded (volume). A bar will continue to develop until the specified volume is reached, and once that level is surpassed, a new bar will be created.



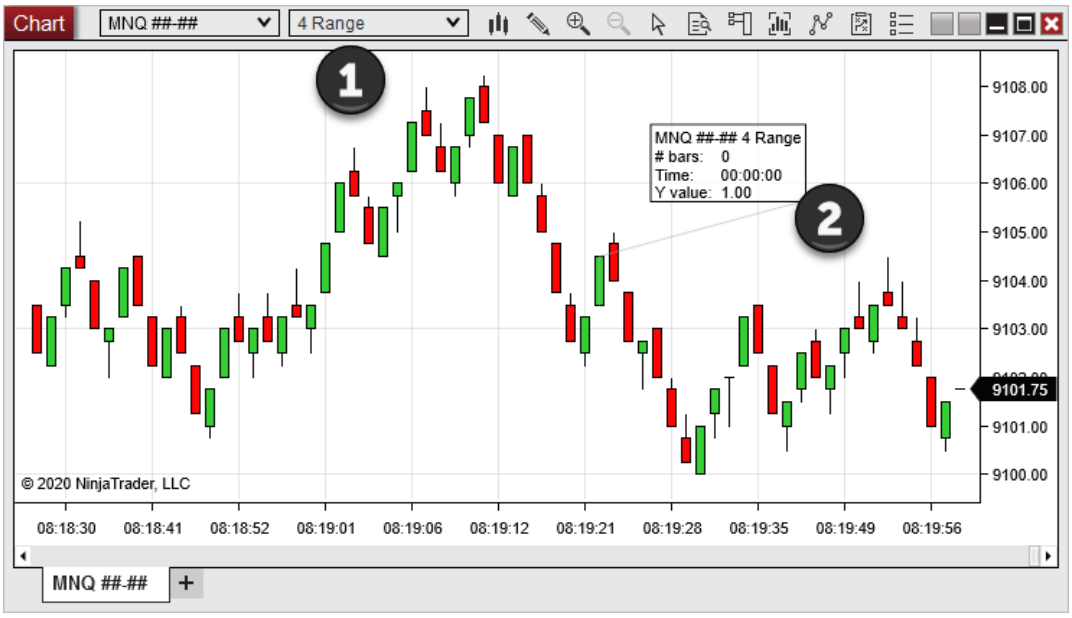
1. Each historical bar in the 10,000 **Volume** chart shown above contains a volume of 10,000 contracts.
2. This is verified by the "VOL" indicator plotted below the price bars and the Volume displayed in the Mini Data Box.

Understanding Range bars

Range Bars

A **Range** bar is based on a specified tick price range. The bar will continue to develop until the price range is broken, at which point a new bar will be created.

> **Note**: A tick in this instance is different from a tick in a **Tick** bar described in the sub-section above. A tick in a **Tick** bar represents the point at which an actual trade occurred, whereas a tick in a **Range** bar represents a price increment, or a movement on the price axis of the chart. This increment is the smallest price movement the instrument can make, and may differ by instrument. For example, a tick on the e-mini S&P 500 (ES) equates to a movement of 0.25, while a tick on AAPL stock equates to a movement of 0.01. More information on setting an instrument's **Tick Size** can be found on the Editing Instruments page.

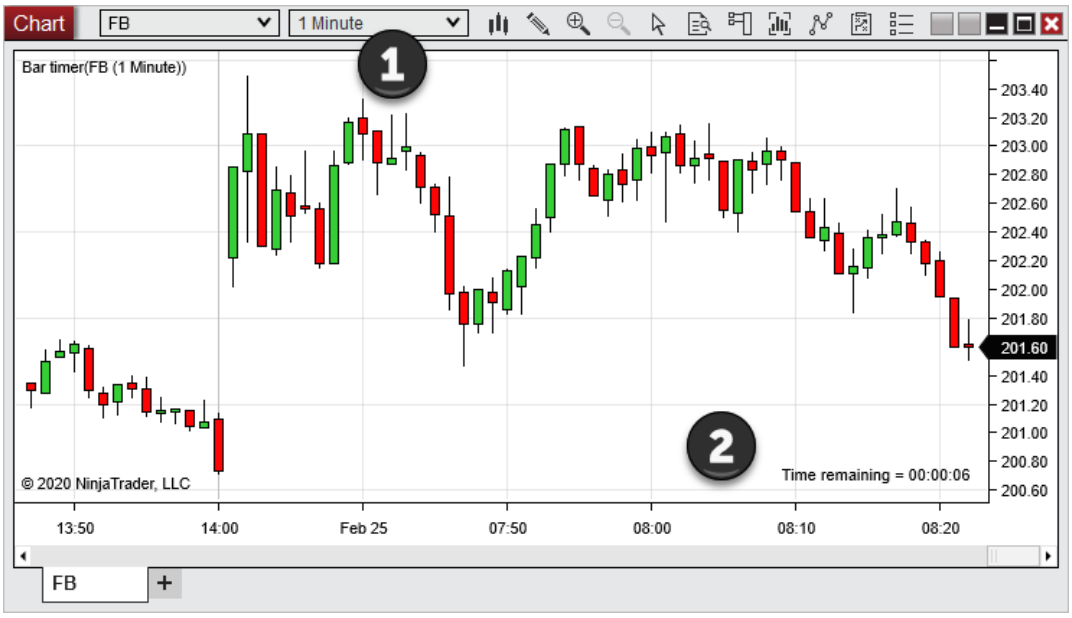


1. Each historical bar in the 4 **Range** chart shown below represents exactly 4 ticks of price movement.
2. The **Ruler** Drawing Tool verifies that each bar consists of 4 ticks. (The "Y value" of 1.00 shown in the **Ruler's** display flag is equivalent to 4 ticks for the e-mini S&P 500 continuous contract instrument on the chart.)

Understanding Time based bars

Time Bars

Second, **Minute**, **Day**, **Week**, **Month**, and **Year** bars are all built based on the passage of time. A bar will develop for a specified amount of time, and once this time is exceeded, a new bar will begin.



1. Each historical bar in the 1 **Minute** chart shown below represents price movement during one minute in time.
2. The "Bar Timer" indicator has been applied to the chart to show the time remaining for the current bar.

Note: Intraday time based charts are built off the Trading Hours definitions set for the individual chart's DataSeries. For daily charts and higher this is not the case though, here the Trading Hours are governed by the provider recording the data. For the NinjaTrader Historical Data Servers daily bars will be recorded using the ETH (Electronic Trading Hours) definitions for the respective instrument. Further for providers which support accessing the official settlement value, NinjaTrader will use this value as the daily bar close - for more information regarding your specific provider please consult this link.

Understanding Heiken Ashi bars

Heiken Ashi Bars

Heiken Ashi in Japanese translates to "Average Bar" in English. These bars are intended as a way to isolate ongoing trends. **Heiken Ashi** bars may appear to plot the Open, High, Low, and Close of price within a specified time period, similar to Candlestick bars. However, these bars use unique formulas to calculate OHLC values based on mathematical averages. Like Candlesticks, **Heiken Ashi** bars

are based on the passage of time, and can be set to any **Second**, **Minute**, **Day**, **Week**, **Month**, or **Year** interval.

> **Note**: Calculated value will be rounded to the instrument's nearest tick size. This is done to ensure accuracy in order submission and execution during backtesting.

The chart below displays **Heiken Ashi** bars based on a 2-minute interval:

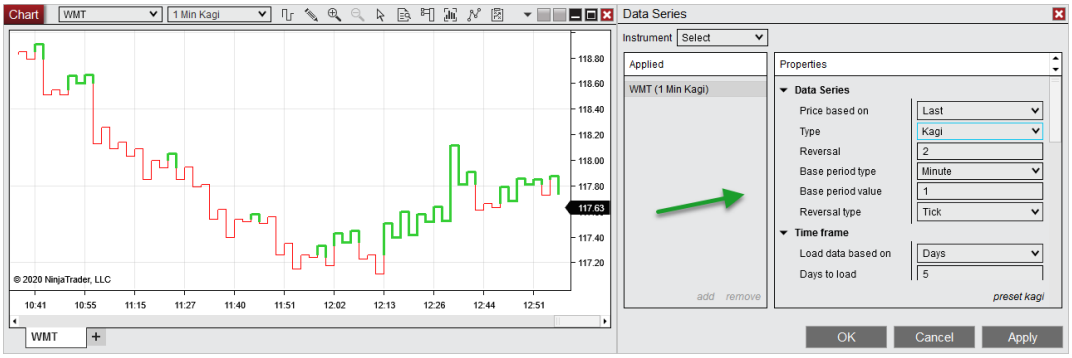
Understanding Kagi bars

Kagi Bars

Kagi bars are based on price movement. A **Kagi** bar will plot in the direction of price until price reverses a specified amount, known as the **Reversal**. The bar will then change direction, but stay the same color until the last bar's High or Low is surpassed. The length of time the bar will develop depends upon the **Base period**.

For example, suppose the price of an instrument is heading down and the **Reversal** is set to 2 ticks. The line will continue to plot downward until price reverses more than 2 ticks. At this point, the line will change direction, but stay red by default. Once the last **Kagi** bar High is exceeded, the line will change to green

by default, and the same rules will apply in the opposite direction. The chart below displays a 1 Minute **Kagi** chart with a **Reversal** set to 2 ticks.



Understanding Renko bars

Renko Bars

Renko bars are based on price movement. Each bar is known as a "brick," and is plotted as green by default when price is moving up and red by default when price is moving down. A new brick is plotted when price exceeds the High or Low of the previous brick by a specified amount, known as the **Brick size**. The chart below displays **Renko** bars with a **Brick size** of 7:

Understanding Point and Figure bars

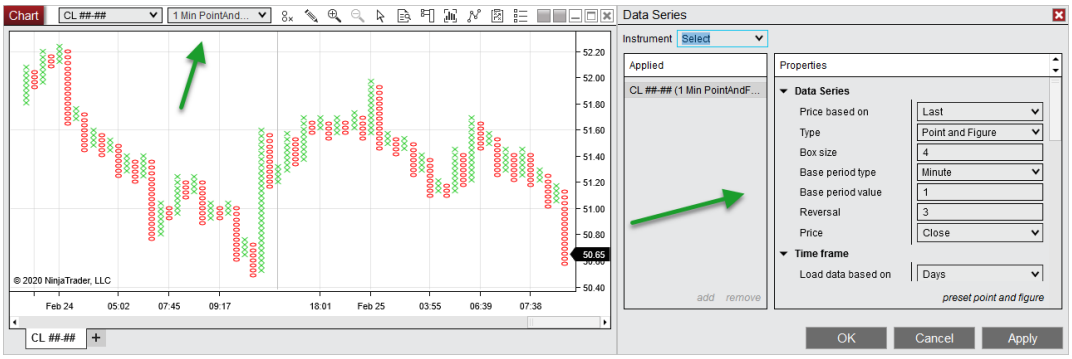
PointAndFigure Bars

PointAndFigure bars are built based on price movement. Each bar plots a column made up of either X's representing a rising price or O's representing a decreasing price. Each X or O is referred to as a "box" and represents the price distance defined by the **Box size** (set in terms of ticks). A new X or O box will be added to the bar when price moves more than the **Box size,** warranting the addition of another box.

Another parameter, called the **Reversal**, sets the amount of price movement needed from the High or Low to change from X's to O's, or from O's to X's. A column will continue indefinitely until a price reversal equal to the **Reversal** amount (set in number of boxes) occurs. There can never be two columns of X's or O's next to each other for a given session, as any additional X's or O's would be added to the current column instead. When a reversal occurs, the next column begins one box size above the last Low for X's, or one box size below the last High for O's.

For example, the chart below shows **PointAndFigure** bars based on a 1 Minute **Data Series**. The **Box size** is set to 4 and the **Reversal** is set to 3.

> **Note**: The prices of the X's and O's are represented by the exact middle of the X or O, rather than the top or bottom.



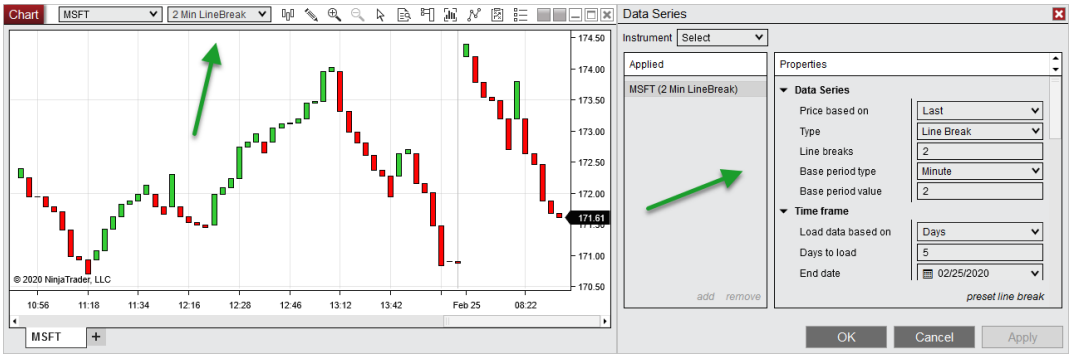
Understanding Line Break bars

Line Break Bars

Line Break bars are built based on price movement. **Line Break** bars must break above or below the High or Low of a specific set of prior bars before a new bar will

be drawn. The "Line Breaks" parameter sets the number of previous bars in the set whose High or Low the current price must break.

For example, if the "Line Breaks" parameter is set to 2, as shown in the chart below, the first bar will be drawn based on whether the Close was above or below the Open. The second bar in the chart is drawn with a green color by default if price exceeds the first bar's High and red by default if price drops below the Low of the first bar. No new bar is drawn if price does not exceed the High or Low of the previous bar. The third bar is only plotted once price breaks the High/Low of the last 2 bars, since the **LineBreaks** parameter is set to 2. If the last break occurred on the upside, a color change will occur when price breaks the last Low. If the last break occurred on the downside, a color change will occur when price breaks the last High.



Understanding Order Flow Volumetric Bars

For information on how to work with the Order Flow Volumetric Bars and Bar Statistics, please see the Order Flow Volumetric Bars page in the OrderFlow + section of the Help Guide.

10.6.8 Chart Styles

NinjaTrader supports a large variety of **Chart Styles**. This page explains how each **Chart Style** is created in a chart, and provides tips for reading charts of different styles. Please see the Working with Price Data page for information on how to change **Chart Styles**.

> **Note**: Some chart styles are intended to be used with a specific Bar Type, and will be most effective when paired with that **Bar Type**. For example, **Point and Figure** can be found in both the **Bar Types** and **Chart Styles** menus, and these two will be most useful in tandem on a chart. As another example, the **Renko Bar Type** can be most effective

when paired with the **Open/Close Chart Style**. When you select a **Bar Type**, the recommended **Chart Style** will be selected automatically, but it can still be changed afterward.

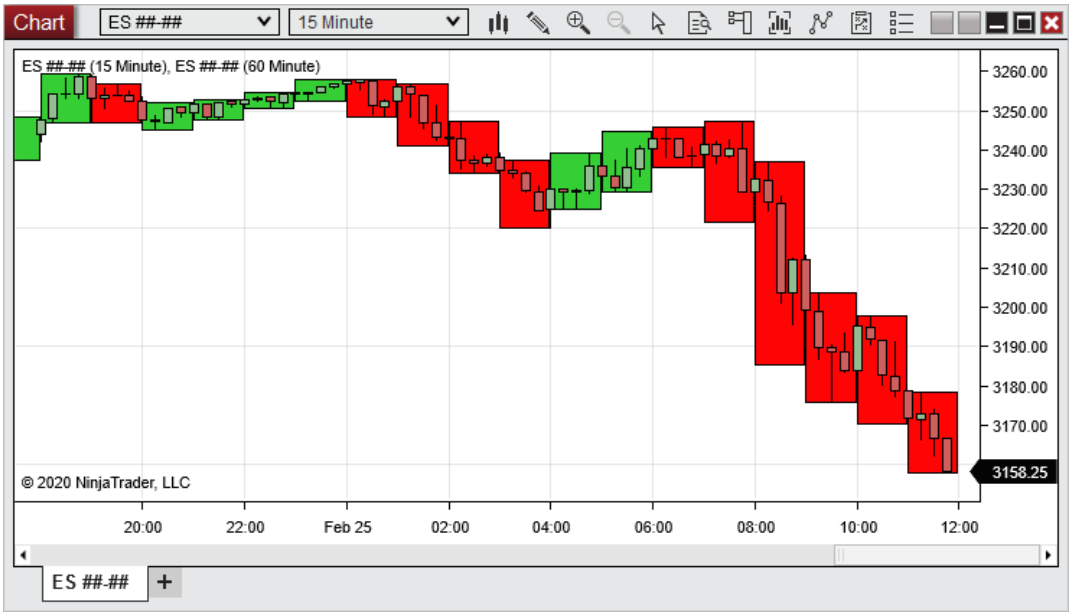
Understanding the Box Chart Style

Box Chart Style

The **Box Chart Style** was specifically designed to simplify multi-timeframe analysis on charts. The **Box** style draws a rectangular shape for each bar, colored green by default for up bars, and red by default for down bars. Rather than differentiating between the Open, High, Low, and Close of a specific time interval, the **Box** style displays only the High and Low. This is done to allow for a second **Data Series** which will show greater price-action granularity to be painted on top of **Box** bars. For example, in the image below, the **Box** style is used to show the High and Low of a higher timeframe, while **Candlesticks** are used to show more precise intra-bar price action on a lower timeframe for the same instrument.

Reading a Box Chart

Box bars can be used to essentially define a range of trading that occurred within a specified timeframe, but they will not reveal anything about intra-bar price action. The **Box** style is most effective when paired with a lower-timeframe **Data Series** in the same **Chart Panel**, as in the example below.



In the image above, a 60-minute **Data Series** of the E-Mini S&P 500 futures contract is using the **Box** style, and is painted behind a 15-minute **Data Series** of the same instrument, clearly showing the shorter timeframe price movement within the longer timeframe. For more information on painting one **Data Series** on top of another, see the "*How to change the z-order (paint order) of a chart object*" section of the Working with Objects on Charts page.

Understanding the Candlestick Chart Style

Candlestick Chart Style

The **Candlestick Chart Style** plots four data points per bar: Open, High, Low, and Close. **Candlesticks** are generally painted one color for up bars (green by default), and another color for down bars (red by default).

Reading a Candlestick Chart

Candlesticks are broken into two main sections, a candle body and a wick. In an up bar, the top of the candle body represents the Close price, and the bottom represents the Open price. In a down bar, the top of the candle body represents the Open price, and bottom represents the Close price. In either up or down bars, the high point of the wick represents the High price, and the low point of the wick represents the Low price.

Tip: NinjaTrader's pre-loaded "Candlestick Pattern" indicator is designed to identify common candlestick patterns when using this **Chart Style**.

Understanding the Equivolume Chart Style

Equivolume Chart Style

The **Equivolume Chart Style** plots four data points per bar: Open, High, Low, and Close. Additionally, the width will vary per bar. **Equivolume** bars are generally painted one color for up bars (green by default), and another color for down bars (red by default).

Reading a Candlestick Chart

Equivolume bars are broken into two main sections, a candle body and a wick. In an up bar, the top of the candle body represents the Close price, and the bottom represents the Open price. In a down bar, the top of the candle body represents the Open price, and bottom represents the Close price. In either up or down bars, the high point of the wick represents the High price, and the low point of the wick represents the Low price. The width of the bar indicates how much volume was received within that bar in comparison to the other bars in view. So a wide bar indicates high volume and a thin bar indicates low volume.

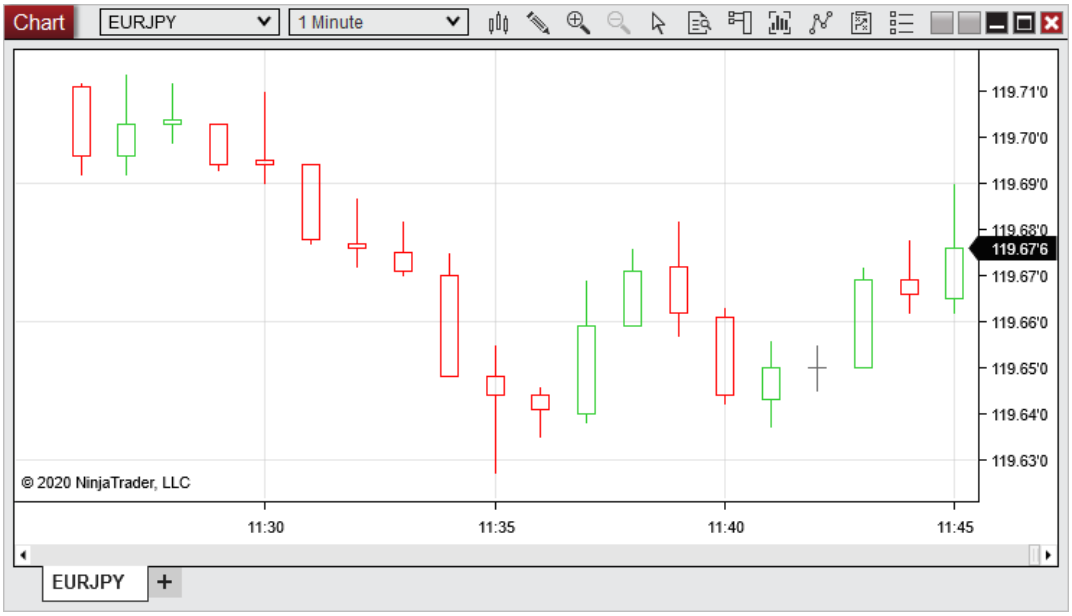
Understanding the Hollow Candlestick Chart Style

Hollow Candlestick Chart Style

The **Hollow Candlestick Chart Style** plots four data points per bar: Open, High, Low, and Close. **Hollow Candlesticks** are generally painted one color for up bars (green by default), another color for down bars (red by default), and another color for doji base (dim gray by defaut).

Reading a Hollow Candlestick Chart

Hollow Candlesticks are broken into two main sections, a candle body and a wick. In an up bar, the top of the candle body represents the Close price, and the bottom represents the Open price. In a down bar, the top of the candle body represents the Open price, and bottom represents the Close price. In either up or down bars, the high point of the wick represents the High price, and the low point of the wick represents the Low price. In a doji bar, the Open price and Close price are the same, represented by a line on the wick.



Tip: NinjaTrader's pre-loaded "Candlestick Pattern" indicator is designed to identify common candlestick patterns when using this **Chart Style**.

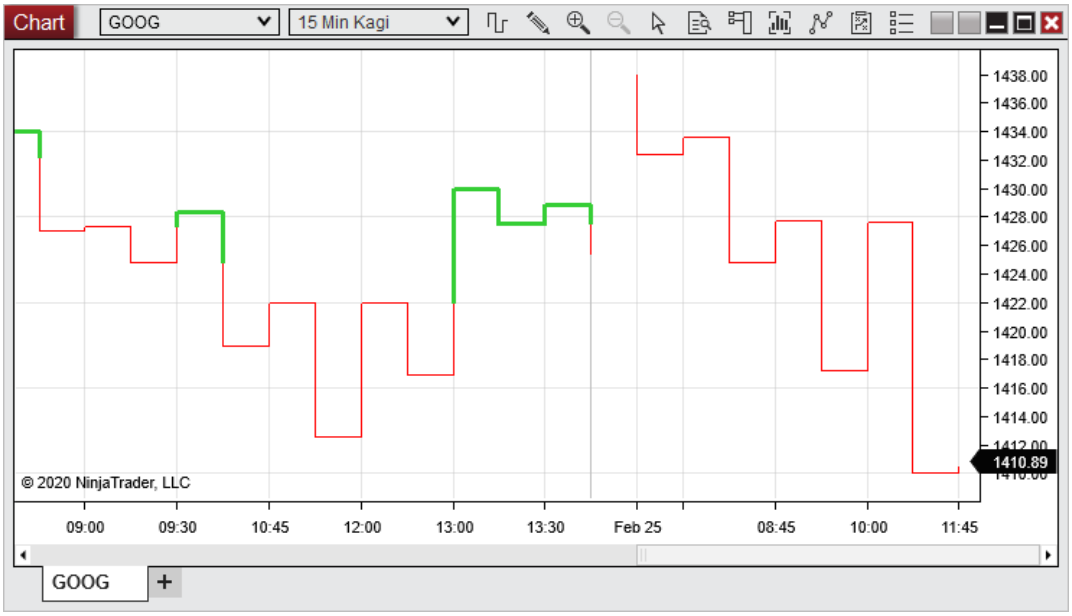
Understanding the Kagi Line Chart Style

Kagi Line Chart Style

The **Kagi Line Chart Style** was specifically designed to function with the **Kagi Bar Type**, which offers an alternative way of analyzing price action with a different perspective than traditional time-based bars.

Reading a Kagi Line Chart

For more information on reading and setting up **Kagi Line** charts, see the "*Understanding Kagi Bars*" section of the Bar Types page.



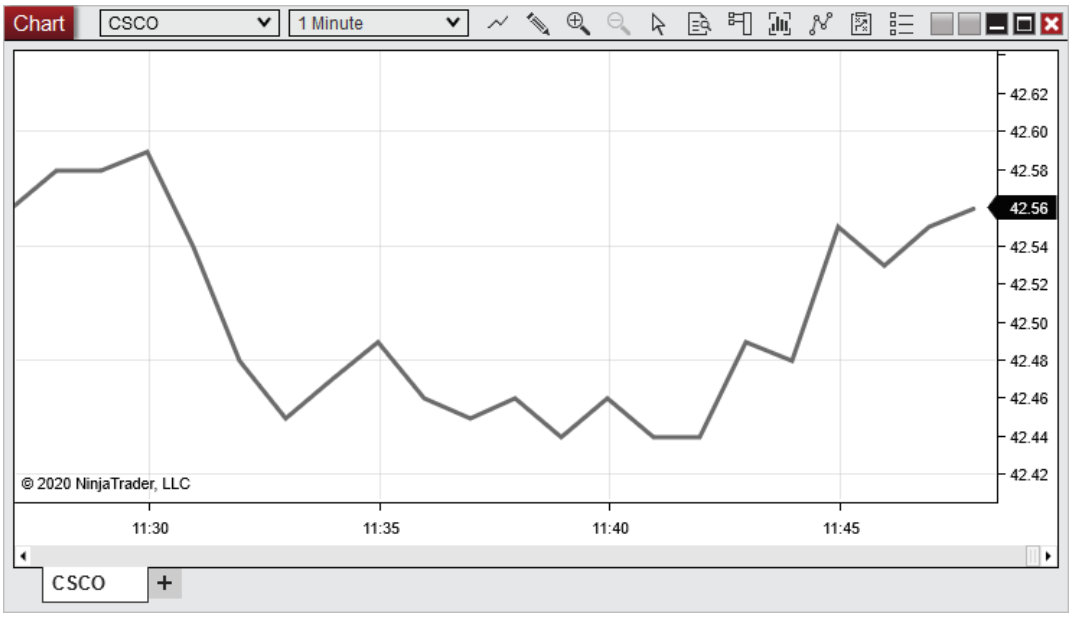
Understanding the Line on Close Chart Style

Line On Close Chart Style

The **Line on Close Chart Style** reduces price-action noise by focusing solely on the Close price of an instrument at a specific time interval. This style connects the Close price at the end of each interval with a straight line.

Reading a Line on Close Chart

When looking at a **Line on Close** chart, it is important to differentiate between the line itself and the pivots between the line's many segments. Each point at which the line pivots represents a Close price for the instrument, while the lines between those points do not necessarily represent true historical prices. Instead, they are drawn as a way to smooth the transition from one Close price to another.



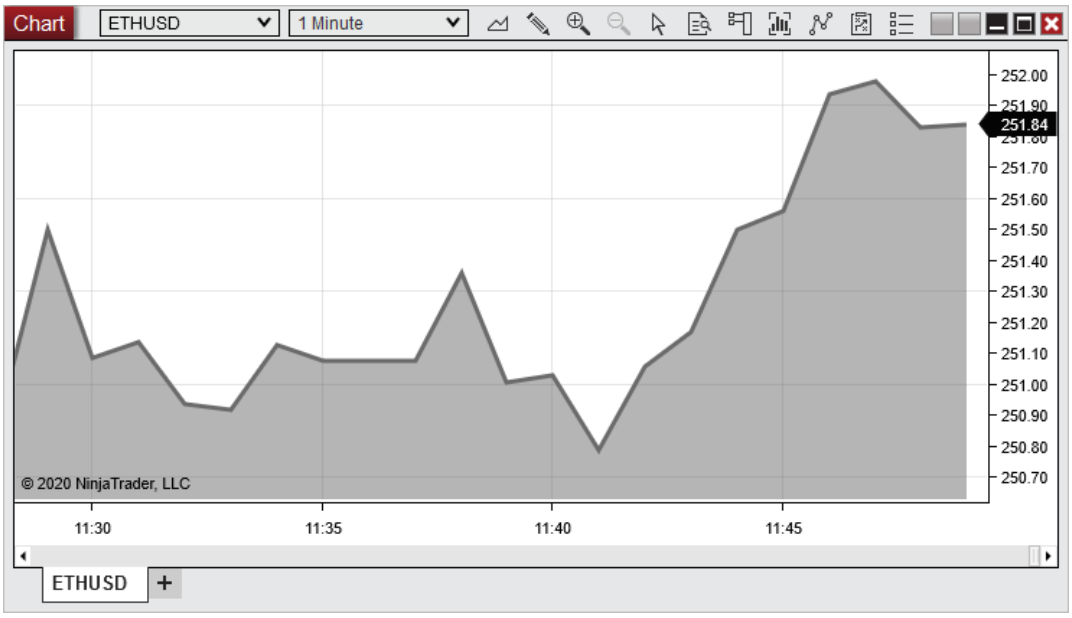
Understanding the Mountain Chart Style

Mountain Chart Style

The **Mountain Chart Style** functions similarly to the **Line on Close** style covered in the previous section. The **Mountain** style connects Close prices of a chosen interval with straight line segments, and also colors the region below the connected line segments with a solid color.

Reading a Line on Close Chart

When looking at a **Mountain** chart, it is important to differentiate between the line itself and the pivots between the line's many segments. Each point at which the line pivots represents a Close price for the instrument, while the lines between those points do not necessarily represent true historical prices. Instead, they are drawn as a way to smooth the transition from one Close price to another. It is also important to understand that the shaded area does not necessarily represent historical price points, but is intended simply as a visual aid.



Note: The outline color, fill color, and opacity of this **Chart Style** can be changed via the Data Series window.

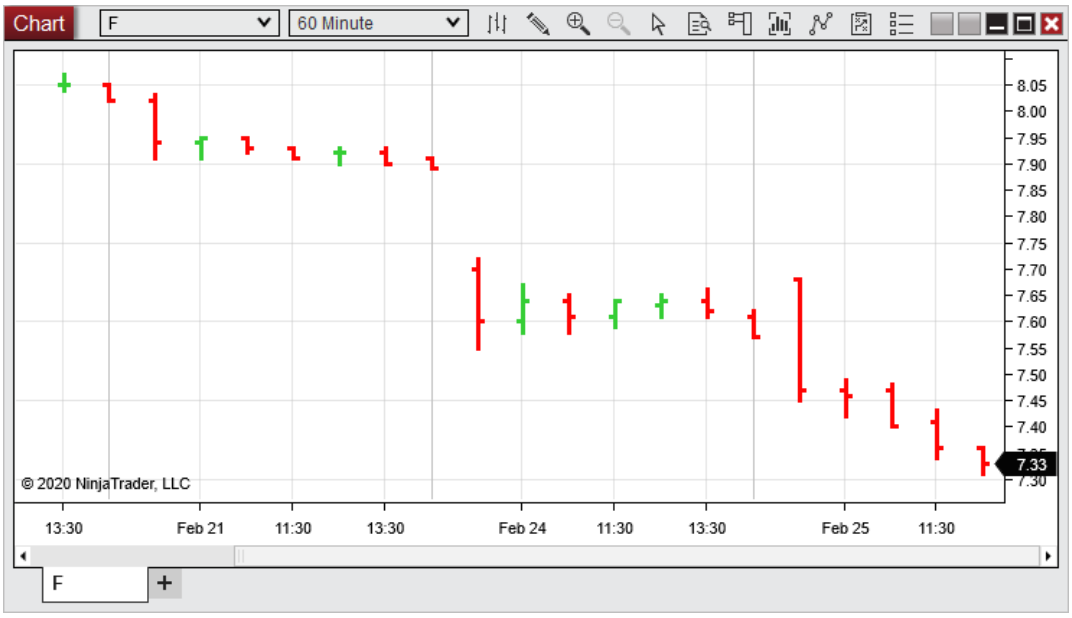
Understanding the OHLC Chart Style

OHLC Chart Style

The **OHLC Chart Style** plots four data points per bar: Open, High, Low, and Close. Like **Candlesticks**, **OHLC** bars are generally painted one color for up bars (green by default), and another color for down bars (red by default).

Reading an OHLC Chart

The small left- and right-facing flags on each bar hold the key to interpreting **OHLC** charts. When the right-facing flag is higher on the bar than the left-facing flag, this indicates an up bar, and when the left-facing flag is higher, this represents a down bar. This should correspond to the colors of the bars, as well. The space between the flags represents the Open-to-Close price action, while the parts of the bar extending beyond the flags represent the High and Low (regardless of bar direction).



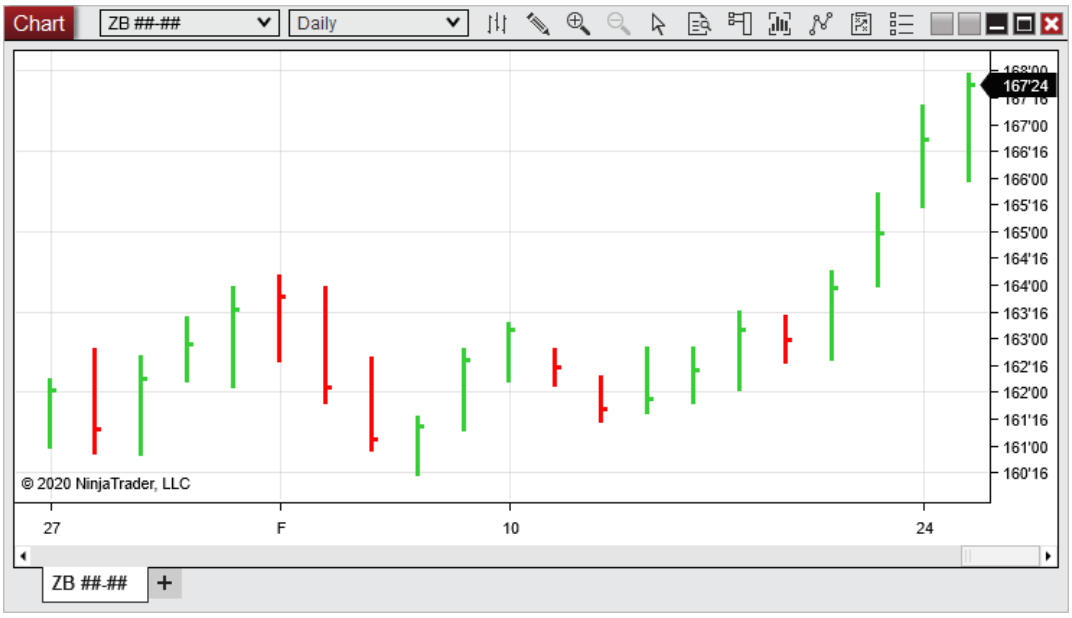
Understanding the HLC Chart Style

HLC Chart Style

The **HLC Chart Style** plots three data points per bar: High, Low, and Close. Like **OHLC** bars, **HLC** bars are generally painted one color for up bars (green by default), and another color for down bars (red by default).

Reading an HLC Chart

HLC bars include only one flag extending to the right of each bar, as opposed to **OHLC** bars, which include both left- and right-facing flags. The right-facing flag represents the Close price of a bar, while the extreme upper and lower points of the bar represent the High and Low, respectively.



Note: Some day traders prefer **HLC** bars to **OHLC** bars because they assume that the Open price of any bar should always be one tick away from the Close price of the prior bar. Note that this will not necessarily be the case with Daily or higher time intervals.

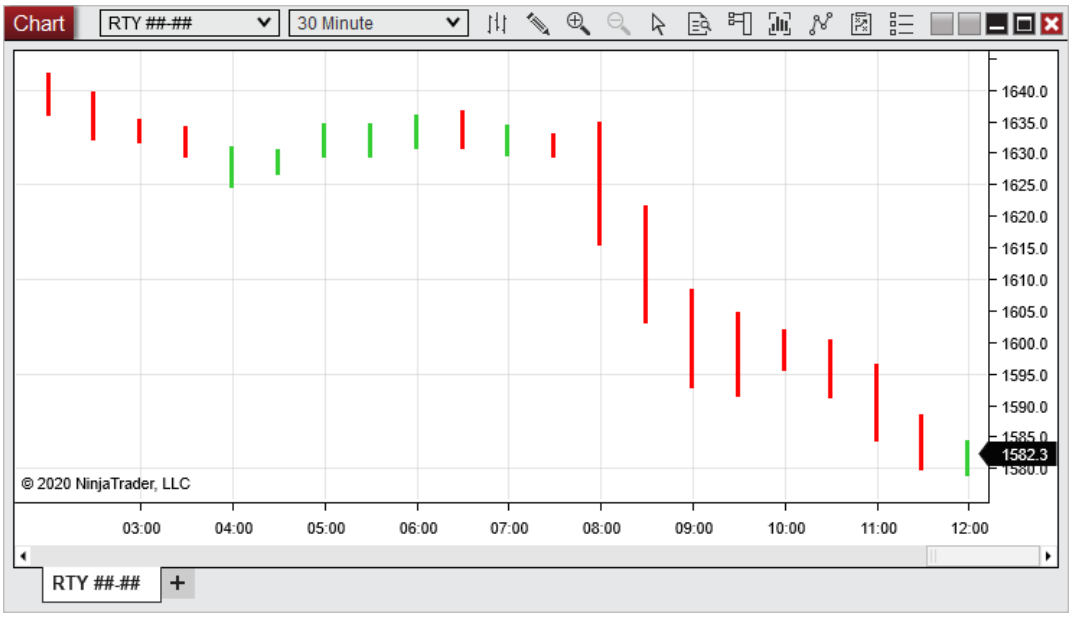
Understanding the HiLo Chart Style

HiLo Chart Style

The **HiLo Chart Style** plots two data points per bar: High and Low. Like **OHLC** bars, **HiLo** bars are generally painted one color for up bars (green by default), and another color for down bars (red by default).

Reading a HiLo Chart

HiLo bars remove the left- and right-facing flags found on **OHLC** and **HLC** bars. The upper and lower points of each bar represent the High and Low, respectively.



Tip: This **Bar Type** can be useful to quickly determine the trading range of a higher-timeframe interval, such as one week or one month, while eliminating intra-bar price-action noise that is not useful in defining a range.

Understanding the Open/Close Chart Style

Open/Close Chart Style

The **Open/Close Chart Style** simplifies intra-bar noise by taking High and Low prices out of the equation. This **Chart Style** paints up bars in a green color by default, and down bars in a red color by default, and simply plots the difference between the Open and Close during a chosen interval.

Reading Open/Close Charts

In an up bar, the bottom of an **Open/Close** bar represents the Open price, while the top of the bar represents the Close price. In a down bar, the top represents the Open, while the bottom represents the Close.

Tip: When drawing support/resistance or trend lines, some traders prefer to anchor these lines to candle bodies while ignoring wicks. If this is your chosen method, then the **Open/Close Chart Style** can be a good alternative to traditional **Candlesticks**.

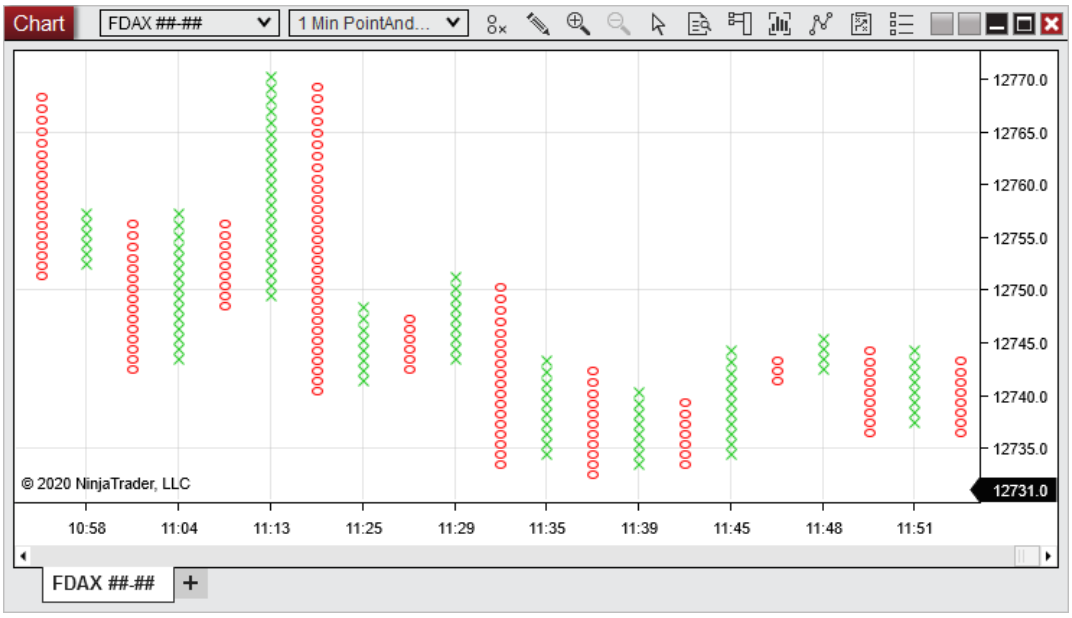
Understanding the Point and Figure Chart Style

Point and Figure Chart Style

The **Point and Figure Chart Style** was specifically designed to function with the **Point and Figure Bar Type**, which is an alternative way of analyzing price action from a different perspective than traditional time-based bars.

Reading a Point and Figure Chart

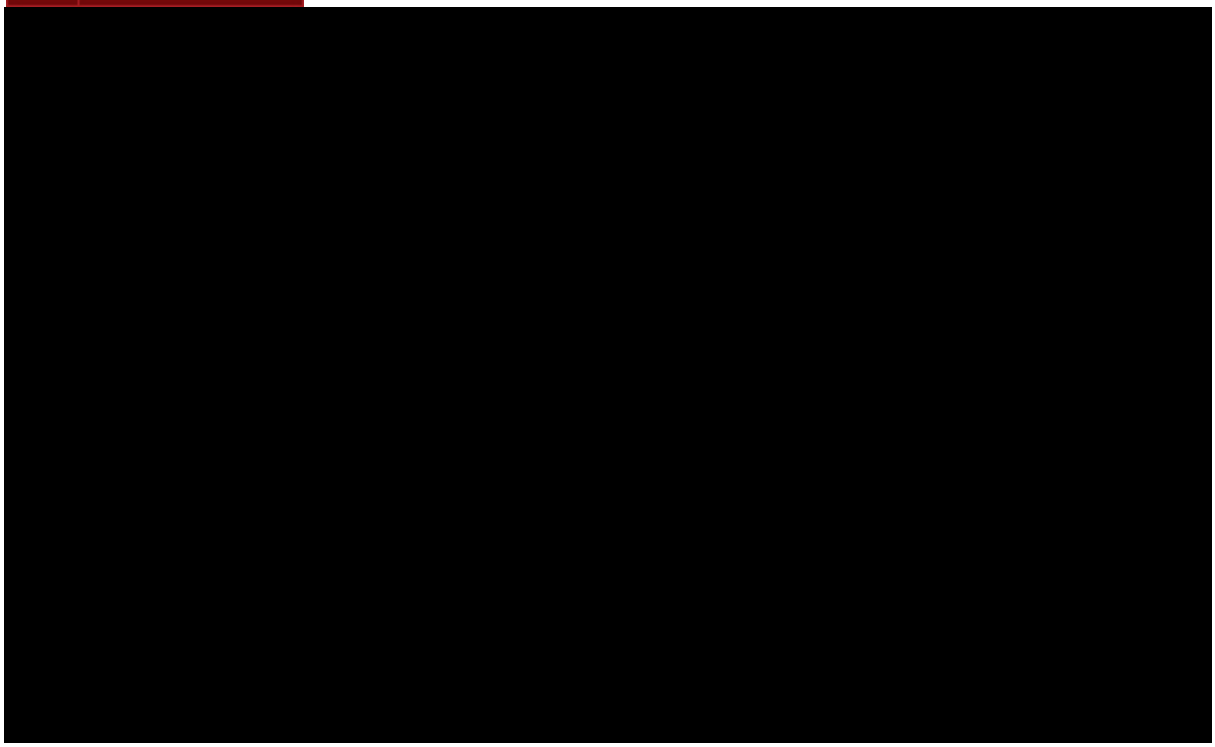
For more information on reading and setting up **Point and Figure** charts, see the "*Understanding Point and Figure Bars*" section of the Bar Types page.



10.6.9 Working with Indicators

NinjaTrader comes with over 100 pre-built technical indicators, which can be added, removed and edited via the Indicators window. Indicators can be applied to charts, the SuperDOM, or Market Analyzer columns, and custom technical indicators can be created via the NinjaScript Editor.

Play Video



Understanding the Indicators window

The **Indicators** window is used to add, remove and edit all indicators within a chart.

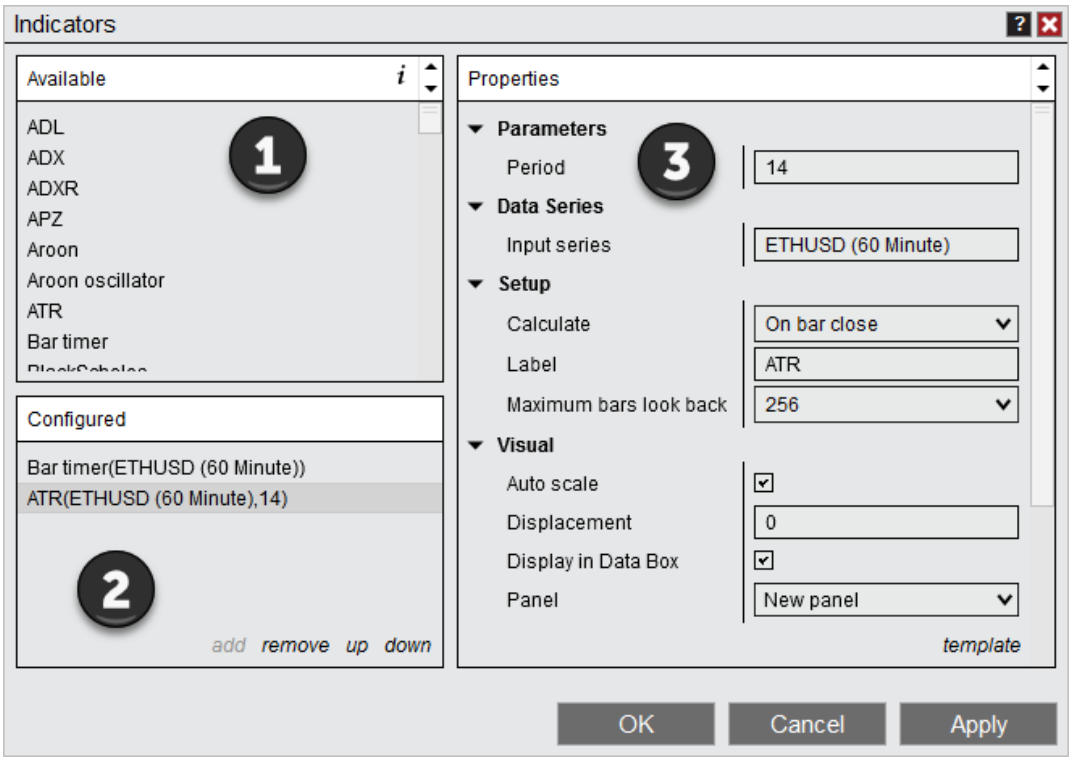
Accessing the Indicators Window from a Chart

There are multiple ways to access the **Indicators** window from a chart:

- Left click on the **Indicators** icon in the chart toolbar
- Right mouse click in the chart background when no chart object is selected, and select the **Indicators** menu item
- Double click on an indicator within a chart
- Right click on a highlighted indicator within a chart and select the **Properties** menu item
- Use the default Ctrl + I Hot Key when the chart has focus.

Sections of the Indicators Window

The image below displays the three sections of the **Indicators** window.



1. The "Available" section displays a list of available indicators
2. The "Configured" section displays indicators currently applied to the chart or SuperDOM
3. The "Properties" section displays the selected indicator's parameters

How to add an Indicator

Adding an Indicator

To add an indicator to a chart:

1. Open the **Indicators** window *(see the "Understanding the Indicators window" section above)*
2. Left mouse click on the indicator you want to add in the "Available" section, then press the **add** option in the "Configured" section. Alternatively, you can simply double click on the indicator in the "Available" section to add it to the "Configured" section.
3. The indicator will now be visible in the "Configured" section
4. The indicator's parameters will now be editable on the right side of the **Indicators** window *(see the "How to edit an indicator" section below)*

How to edit an Indicator's parameters

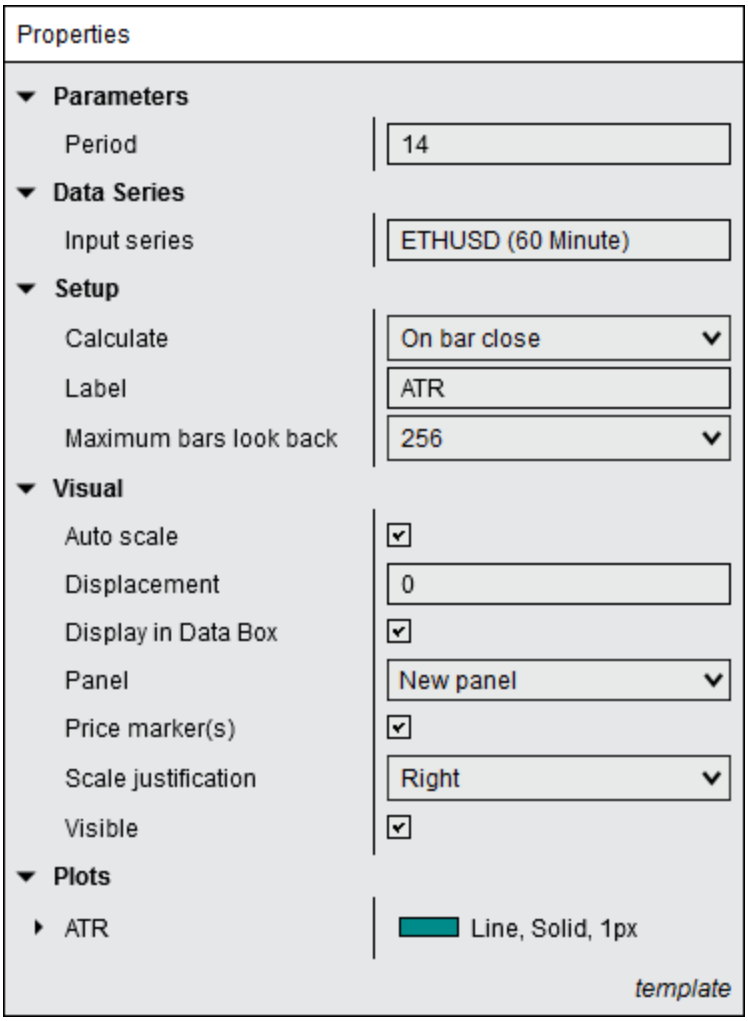
Editing an Indicator

You can customize any indicator from the **Indicators** window.

1. Open the **Indicators** window (see the *"Understanding the Indicators window"* section above)
2. Highlight the indicator you would like to edit from the list of applied indicators
3. Once highlighted, this indicator's parameters will be available to edit in the "Properties" section.

Chart Indicator Parameters

The following parameters are common to all indicators applied on a chart:



Input Series	Please see the "*Indicator Input Series*" section on this page for further information.
Calculate	Sets the frequency at which the indicator performs its calculations. See the note below for information on each possible setting for this property
Label	The label displayed on the chart. Leaving the field blank will remove the label from being displayed on the chart. Enclosing a label in quotations ("MyEMA" for example) will display the text within the quotations and exclude the system added trailing series information.
Maximum Bars Look Back	Determines the maximum number of bars the indicator can look back to perform calculations on historical data. This is set to 256 by default (the most memory-friendly setting), but it can be changed to "infinite" to allow for a greater look back period.
Auto Scale	When enabled, the indicator will be included in the chart panel's vertical automatic scaling
Displacement	Sets the number of bars by which to displace the indicator plots
Display in Data Box	Enables or disables the inclusion of the indicator's plot values in the **Data Box**
Panel	Sets the panel in which the indicator is plotted. If you select "Same as input series," the indicator will be linked to the **Input Series** and automatically move if the **Input Series** is modified to a different panel.
Price marker(s)	When enabled, the indicator value is plotted in the axis selected under the "Scale Justification" property.

Scale justification	Sets the scale on which the indicator will be plotted. Possible values are "Right," "Left," and "Overlay"
Visible	Enables or disables visibility and function of the indicator on the chart
Plots	Sets a variety of parameters, such as color, for the plots drawn by the indicator

Note: The "Calculate" property offers three possible settings to control how often an indicator performs its calculations:

- On Bar Close - Run calculations once on the close of each bar of the **Input Series**
- On Each Tick - Run calculations on each incoming tick of price data (CPU intensive)
- On Price Change - Runs calculations on each change in price

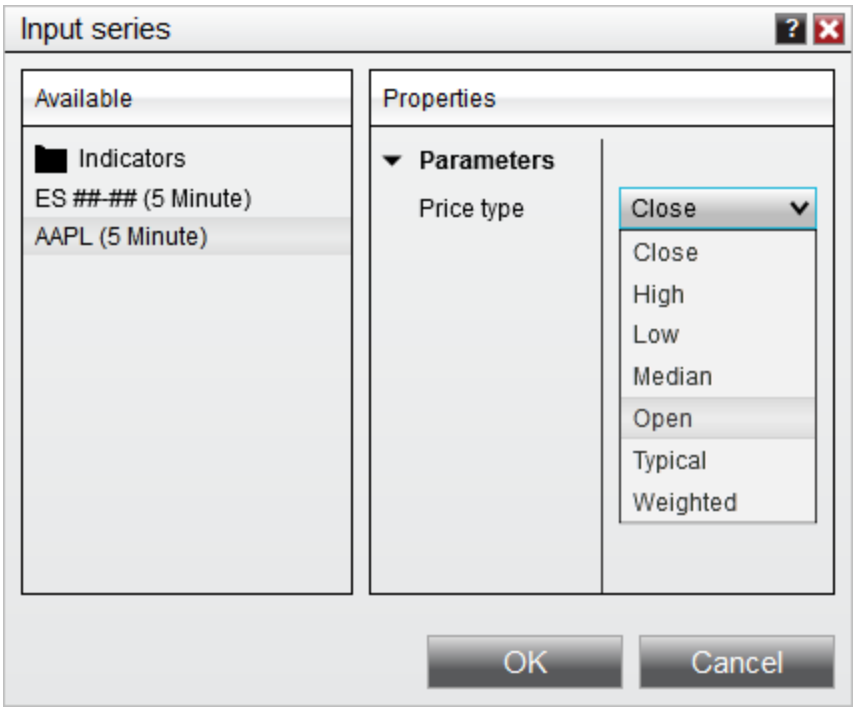
Saving an Indicator's Parameters

You can optionally save your customized indicator's parameters as templates. Saving it as Default will recall your customized settings the next time you add that specific indicator to a chart.

Please see the Saving Chart Defaults and Templates page for more information.

Indicator Input Series

The indicator **Input Series** dialogue allows you to select the **Input Series** for your indicator's calculations. To access this window, left mouse click within the "Input Series" field. You can then select the Close, High, Low, Median, Open, Typical, or Weighted price of any **Data Series** applied to the chart. Alternatively, you can choose another indicator as the input series. When you select another indicator as the input series, The "Properties" section of the **Input Series** dialogue will display properties related to the indicator being used as the **Input Series**, allowing you to configure it to your desired settings. This allows you to nest multiple indicators. Once you have selected the **Input Series** of your choice, left mouse click the **OK** button to exit the **Input Series** window.



In the image above, we can select one of the **Data Series** applied to the chart, or another indicator, for use as an indicator's **Input Series**.

Note: To take advantage of this feature NinjaScript indicators will need to implement the Input ISeries as their main data input.

How to remove an Indicator

Removing an Indicator From a Chart

There are three ways to remove an indicator from a NinjaTrader chart:

- Open the Indicators window (see the *"Understanding the Indicators window"* section above). Next, select an indicator from the "Configured" section, then select the **Remove** option, and finally press the **OK** button to exit the **Indicators** window.
- Left mouse click to select the indicator on your chart, then press the Delete key on your keyboard.
- Left mouse click to select the indicator on your chart, then right mouse click the indicator and select the **Remove** menu item.

Custom Indicator development

In addition to the indicators that come pre-built with the NinjaTrader application, you also have the ability to create custom indicators of your own. For example, you could create your own custom multi-series indicators using price and volume data to apply to your charts or share with fellow traders.

For more information on using NinjaScript to build custom indicators please see the NinjaScript section of the user help guide, or click here to view NinjaScript indicator-development tutorials.

Working with Indicators in Market Analyzer columns

Please see the Working With Columns page for information on working with indicators in **Market Analyzer** columns.

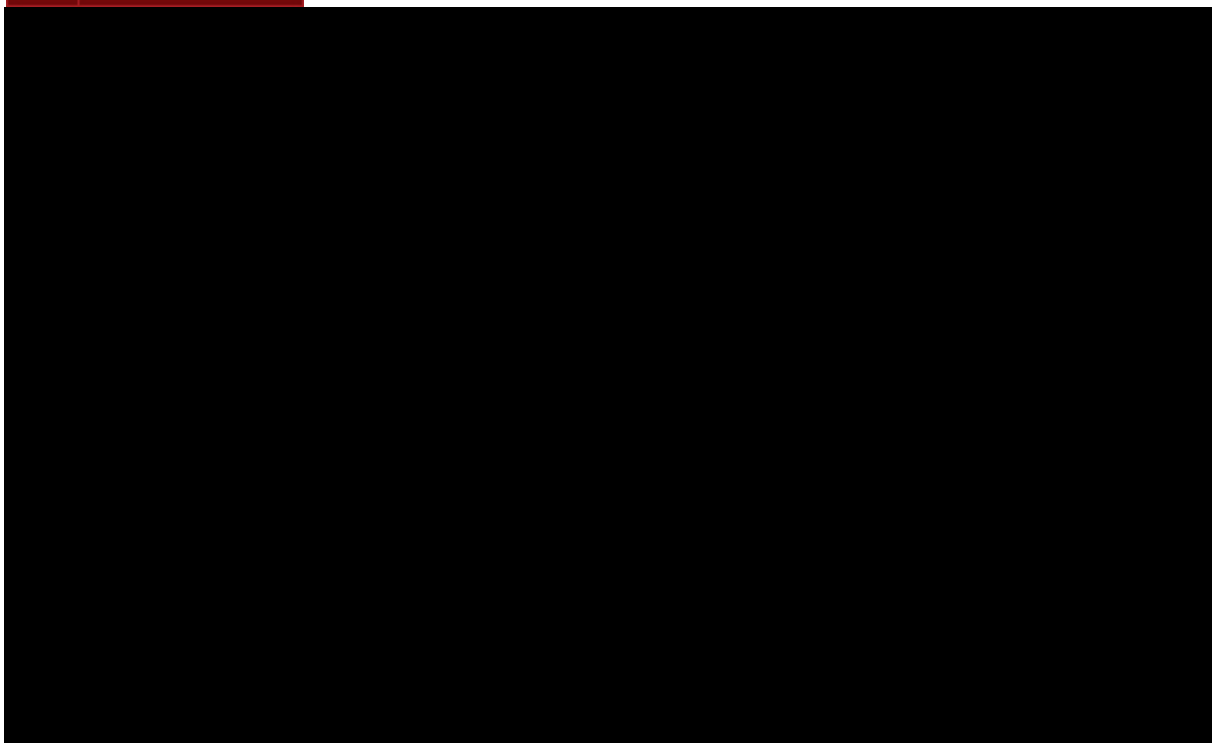
Working with Indicators in the SuperDOM

Please see the **SuperDOM** Working with Indicators page for information on working with indicators in the **SuperDOM**.

10.6.10 Working with Drawing Tools & Objects

There are many customizable **Drawing Tools** and objects available to use in NinjaTrader charts. **Drawing Tools** can be applied to individual charts or all open charts displaying the same instrument, and templates for each **Drawing Tool** can be saved to apply commonly used properties in the future.

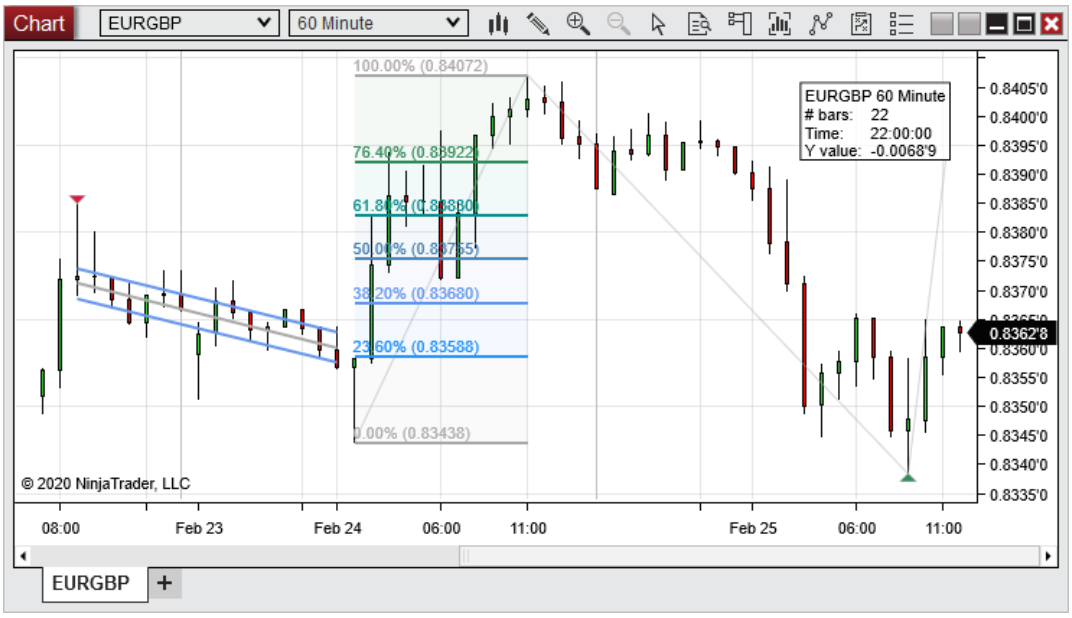
Play Video



How to draw on a chart

Drawing on a Chart

Various **Drawing Tools** are available and customizable within a chart. The image below shows an example of several **Drawing Tools** applied to a chart.



Accessing Drawing Tools

Drawing Tools can be accessed in three ways:

- Right mouse click within the chart and select the **Drawing Tools** menu, then select an individual **Drawing Tool** from the list that appears
- Left mouse click on the **Drawing Tools** button in the toolbar at the top of the chart, then select the **Drawing Tool** you wish to use
- Press the default or custom Hot Key for a specific drawing tool (see the list of default hot keys under the "Available Drawing Tools" heading below)

Stay in Draw Mode

When "Stay in Draw Mode" is enabled from the **Drawing Tools** menu, any drawing tool you select will remain selected after creating a drawing object with that tool. The **Drawing Tool** can then be used to draw multiple objects without having to access the **Drawing Tools** menu each time.

Line Tools

There are multiple **Line** tools which can be utilized. They can be placed as desired or after placing the first anchor, you can hold SHIFT on the keyboard and move the mouse around to adjust the line in 45 degree increments. This is based on chart scaling at the time the line is placed.

Ruler Tool

The **Ruler** measures the number of bars, length of time, and y-axis distance between two anchor points for a **Data Series**. The measurement data is attached in a flag at a third, independent anchor point.

To use the **Ruler**:

1. Select the **Ruler Drawing Tool** from the **Drawing Tools** menu
2. If you have more than one **Data Series** or indicator applied to your chart, first select the **Data Series** or indicator you wish to measure by left mouse clicking on it
3. Left mouse click on the chart where you wish to place the first anchor point
4. Left mouse click a second time on the chart where you wish to place the second anchor point
5. Left mouse click a third time to set the anchor point for the **Ruler** display flag.

The anchor points can be relocated by left mouse clicking on an anchor point and dragging it to a new location.

In the image above, we see the ruler tool used to measure a distance of 42 bars over 7 hours, with a y-axis movement of 10.75 points.

Risk-Reward Tool

The **Risk-Reward** tool can help you to determine the placement of your profit targets to achieve a specific risk/reward ratio on any trade.

To use the **Risk-Reward** tool:

1. Select the **Risk-Reward Drawing Tool** from the **Drawing Tools** menu
2. If you have more than one **Data Series** applied to your chart, first select the **Data Series** you wish to work with by left mouse clicking on it
3. Left mouse click on the chart at the entry price of an active, pending, or hypothetical trade
4. Left mouse click a second time on the chart at the point representing the maximum risk you are willing to take on the trade
5. Open the **Properties** window for the **Drawing Object** you have just placed (See the "*Understanding Drawing Object Properties*" section below)
6. Enter your desired Risk/Reward ratio in the "Ratio" field, then select the **OK** button

Once the object has been drawn and the Risk/Reward ratio set, two lines will extend outward from the first anchor point. The first line, culminating in a number colored red by default, represents the maximum risk you are willing to take, as specified by the second anchor point. The second line, culminating in a number colored green by default, represents the price point determined by multiplying the risk by the chosen Risk/Reward ratio.

1. In the image above, the first anchor point is set at 1.08900, with a risk/reward ratio of 1.5
2. The second anchor point (the maximum risk) is set at 1.08430
3. Based on the 0.00470 distance between the first and second anchor points (1.08900 - 1.08430), the third anchor point is automatically placed at 1.09605 to achieve a 1:1.5 risk/reward ratio (1.08900 + (0.00470 * 1.5))

Region Highlight X / Region Highlight Y

The **Region Highlight X** and **Region Highlight Y** tools allow you to highlight or shade an entire horizontal or vertical region on a chart. The **Highlight Region X**

tool will highlight a horizontal region, and the highlighting will extend indefinitely upwards and downwards, keeping the highlight in place if you choose to re-scale the chart on the price axis. The **Highlight Region Y** tool will highlight a vertical region, and in the same way, will extend the highlighting indefinitely to the right and left, allowing you to draw a region which will continue to cover the entire width of the chart as new bars come in, or as you scroll backwards on the time axis.

To use the **Region Highlight X** or **Region Highlight Y** tools:

1. Select one of the two tools from the **Drawing Tools** menu
2. When using **Region Highlight X**, click on the chart where you would like to place the first anchor point, then click once more to the left or right of that point to place the second anchor point
3. When using **Region Highlight Y**, begin the same way, but place the second anchor point above or below the first anchor point

The image above shows the **Highlight Region X** tool in use, highlighting a 7-leg uptrend.

The image above shows the **Highlight Region Y** tool in use, highlighting a recent consolidation.

Available Drawing Tools

Following are the available **Drawing Objects** and their associated default hot keys found within the **Drawing Tools** menu:

Ruler	Ctrl + F3
Risk/Reward	Ctrl + F4
Region Highlight X	Ctrl + F1
Region Highlight Y	Ctrl + F2
Line	F2
Ray	F3

Extended Line	F4
Arrow Line	Ctrl+F2
Horizontal Line	F6
Vertical Line	F7
Path	Ctrl + 4
Fibonacci Retracements	F8
Fibonacci Extensions	F9
Fibonacci Time Extensions	F10
Fibonacci Circle	F11
Andrew's Pitchfork	Ctrl + F8
Gann Fan	Ctrl + F9
Regression Channel	Ctrl + F10

Trend Channel	Ctrl + 2
Time Cycles	Alt + F11
Ellipse	Ctrl + F11
Rectangle	Ctrl+ F12
Triangle	Ctrl + F6
Polygon	Alt + F10
Order Flow Volume Profile	Ctrl + 3
Arc	Ctrl + F7
Text	F12 (Tip : pressing Alt + Enter while editing the draw text content lets you create line breaks)
Chart Marker: Arrow Up Arrow Down Diamond Dot Square Triangle Up Triangle Down	 Alt+F2 Alt +F3 Alt +F5 Alt +F6 Alt +F7 Alt +F8 Alt +F9

Applying a Drawing Object to a Chart

To apply a **Drawing Object** to a chart, using a **Drawing Tool**:

1. Select a drawing tool from the **Drawing Tools** menu. The cursor will change to resemble a pen (Right clicking or pressing the "Esc" key will cancel the operation).
2. Left mouse click on the chart where you wish to set the first anchor point.
3. Left mouse click again on the chart for any other necessary anchor points. Once all anchor points are set, the cursor will change back to the cursor type you had previously selected.

Once the **Drawing Object** is applied to the chart, it can be selected by left mouse clicking on it. Once selected, the object can be moved throughout the chart, and the anchor points can be moved by left mouse clicking and dragging to a new location.

Understanding snap mode

Snap Mode

Drawing Objects can be attached to price and/or time data within the chart by using any of the **Snap Mode** options available in the **Drawing Tools** menu:

Disabled	Disables **Snap Mode** and allows the **Drawing Object** anchor point(s) to be placed anywhere on the chart
Bar	Sets the x-axis value of **Drawing Object** anchor point(s) to the bar interval values only
Price	Sets the y-axis value of **Drawing Object** anchor point(s) to the price, rounded to the nearest tick
Bar and Price	Sets the x- and y-axis of **Drawing Object** anchor point(s) to be aligned with bar interval values, **Data Series** OHLC, and indicator price values only

Understanding drawing object properties

Each **Drawing Tool** can be customized using the **Drawing Objects** window.

Accessing the Drawing Object Properties

To access the **Drawing Objects** dialogue:

1. Left mouse click on a drawing object to select it (once selected, the anchor points will be visible).
2. Either double left mouse click on the drawing object, or right mouse click and select the **Properties** menu item.

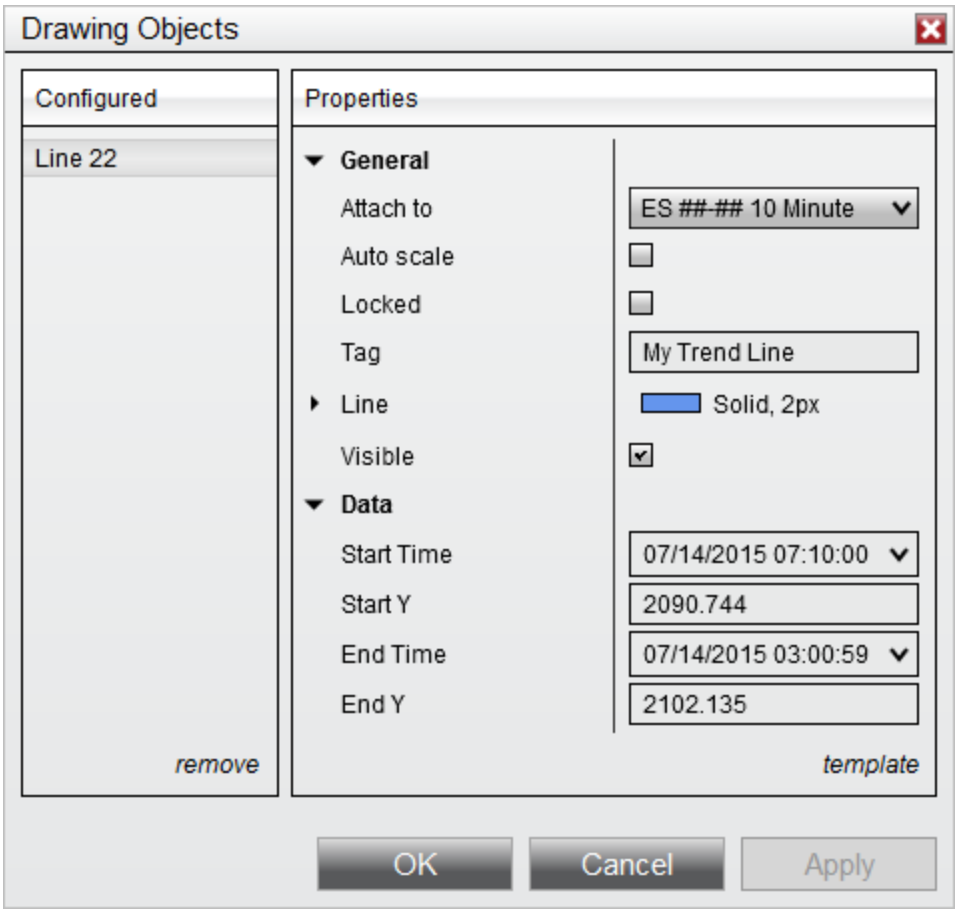
The **Drawing Objects** dialogue is also accessible directly from the **Drawing Tools** menu on the chart toolbar, or by right mouse clicking in a chart, then selecting the **Drawing Tools** menu item.

> **Note**: Regardless of the method used to open the **Drawing Objects** dialogue, all **Drawing Objects** on the chart will be accessible in the dialogue. At any time, you can select a different object from the list in the "Configured" section to edit its properties.

Drawing Object Properties Menu

Properties vary between drawing objects. There are common properties, as shown in the image below, and there are also specific properties depending on the type of **Drawing Object**.

The general properties of the drawing object are located in the General section. The image below shows the General section properties for all **Drawing Objects**, as well as addition properties unique to the **Line Drawing Object**.



The properties listed below are included for all **Drawing Tools**, in addition to each tool's unique properties:

Attach to	Applies the **Drawing Object** to the selected instrument on a single chart or all charts with the same instrument (see the "*Understanding local vs. global drawing objects*" section below). Additionally, a drawing object can be applied to an indicator so that it will be associated to the indicator and it's scale
Auto Scale	Adds the **Drawing Object** to the auto-scaling of the chart.
Locked	Locks the **Drawing Object** in position on the chart, making it immovable. You can also lock a

	Drawing Object by left clicking the object to select it, then right clicking the object to view the right click menu, then selecting **Lock**.
Tag	The Tag property is a naming convention used to access the drawing object via NinjaScript. Any Tag values generated via NinjaScript are grayed out and cannot be changed. Each **Drawing Object** must have a unique Tag value.
Visible	Enables or disables the visibility of the **Drawing Object** on the chart

The Data section displays the data locations of the **Drawing Object** anchor points in the chart. These fields can be modified to change the location of the **Drawing Object** within the chart. Some drawing tools will include additional properties in the Data section, and some may include variations of the following properties:

Start Time	Sets the x-axis start value of the drawing object
Start Y	Sets the y-axis start value of the drawing object
End Time	Sets the x-axis end value of the drawing object
End Y	Sets the y-axis end value of the drawing object

Understanding Drawing Object templates

Drawing Object properties can be saved as a template, allowing you to quickly apply those settings to a new **Drawing Object** of the same type in the future.

What is Saved

The following properties are saved in the General section:

- Auto scale

- Color
- Dash Style
- Width

Attach to will default to the **Data Series** on which the object is drawn. Tag will be automatically updated for each new drawing object. Locked will default to False. Visible will default to True. Properties within the Data section will **NOT** be saved.

Saving Drawing Object Templates

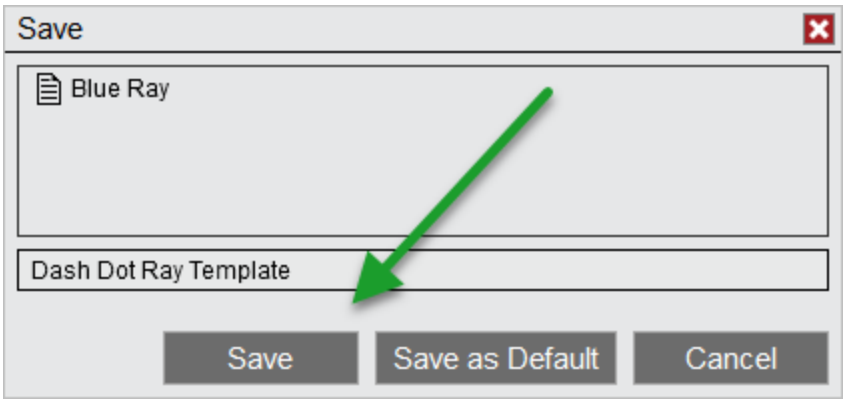
To save **Drawing Object** settings:

1. Open the **Drawing Object Properties** window by either double left mouse clicking on the drawing object or right mouse clicking and selecting **Properties**.
2. Set desired parameters
3. Left mouse click on the **template** text located in the bottom right of the Properties dialog. Selecting **save** will open the **Save** window, in which you can enter a name for a new template, select an existing template to overwrite it or could click **Save as default** to save your settings as the new default applied settings for that drawing object.
4. Click the **Save** / **Save as Default** button when finished

If you wish to load a previously saved template, you can select the **load** option after left mouse clicking on the **template** text.
If you change your settings and later wish to go back to the original settings, you can left mouse click on the **template** text and select the **reset** option.

In the image below, a template will be saved for the **Ray** drawing tool.

In the image below, we can enter a name for the new **Ray** template and save it for future use.



Loading a Drawing Object Template

A **Drawing Object** template that was previously saved can be applied to any **Drawing Object** of the same type. For example, a template for the **Fibonacci Extensions** tool can be applied to a **Fibonacci Extensions Drawing Object**, but not to a **Line Drawing Object**.

To load a **Drawing Object** template:

1. Left mouse click on the **template** text, then select the load option.
2. The Load window will appear. Select the template to load from the list of templates, then press the **Load** button.

Removing a Drawing Object Template

To remove a **Drawing Object** template from the list of saved templates:

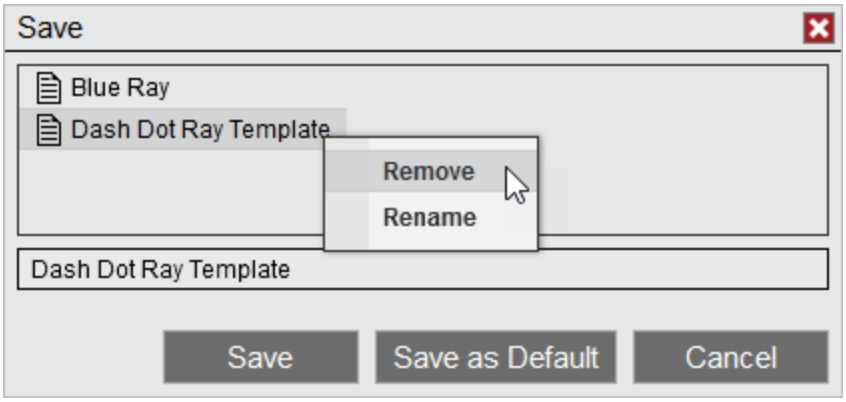
3. Left mouse click on the **template** text, then select either the **Save** or **Load** menu items
1. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the template for removal from the list of templates, then select the **Remove** menu item.

Renaming a Drawing Object Template

To rename a **Drawing Object** template from the list of saved templates:

4. Left mouse click on the **template** text, then select either the **Save** or **Load** menu items
2. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the existing template in the list, then select the **Rename** menu item.

In the image below, we can either remove or rename the selected **Drawing Object** template.



How to remove drawing objects

Removing Drawing Objects

To remove a single **Drawing Object**:

1. Left mouse click on the **Drawing Object** to select it (when selected, the anchor points will appear)
2. Press the Delete key on the keyboard or right mouse click on the drawing object and select the **Remove** menu item

To remove multiple **Drawing Objects** at the same time:

1. Select the **Drawing Tools** menu via right mouse clicking in chart or via left mouse clicking the **Drawing Tools** icon in the chart toolbar
2. Left mouse click on the **Remove All Drawing Objects** menu item, and dialogue box will appear to confirm that you wish to remove all drawing object.
3. Click the **Yes** button to confirm

Notes:

- Removing a **Global Drawing Object** will remove the object from all charts.
- Using the **Remove All Drawing Objects** menu item will **NOT** remove any locked drawing objects from the chart. **Drawing Objects** placed via NinjaScript will not be removed by this method, either.

Understanding local vs. global drawing objects

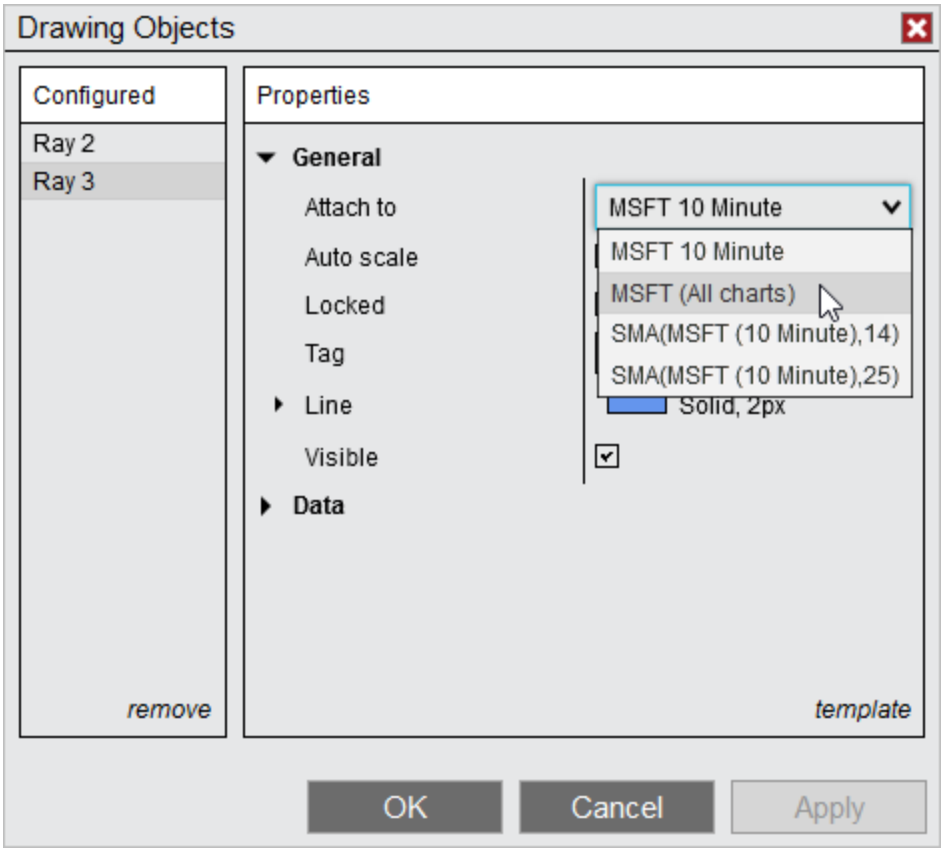
Drawing Objects can be applied to a specific chart (local), or to all charts of the same instrument (global).

How to Enable a Global Drawing Object

To enable a **Global Drawing Object**:

1. Apply a **Drawing Object** to the chart (see the "*How to draw on chart*" section above)
2. Access the **Drawing Object's** properties from the **Drawing Objects** dialogue (see the "*Understanding drawing object properties*" section above)
3. Locate the "Attach to" drop down menu and select "Instrument name" (All charts)

The **Drawing Object** will now be applied to all charts for that specific instrument as well as any new charts opened for that instrument. **Global Drawing Objects** are stored even when a chart of the instrument is not open.



Tips:

- You can set **Global Drawing Objects** to be drawn in all currently open workspaces in the General section of the **Options** window. To access the **Options** window, select the **Tools** menu from the **Control Center**, then select the **Options** menu item. In the General section of the **Options** window, enable or disable the "Global drawing objects across workspaces" property.
- If you wish to exclude a **Global Drawing Object** from one or more charts, you can do so by setting the property "Show global draw objects" to false in the Format Data Series window on the chart(s) you wish to exclude.
- Making a draw object global means that the object would be redrawn and thus the z-order reset to its default.
- A drawing object marked as global will only display on other charts that match the scale justification of the plot it is attached to.

Understanding drawing object levels

Drawing Object Levels

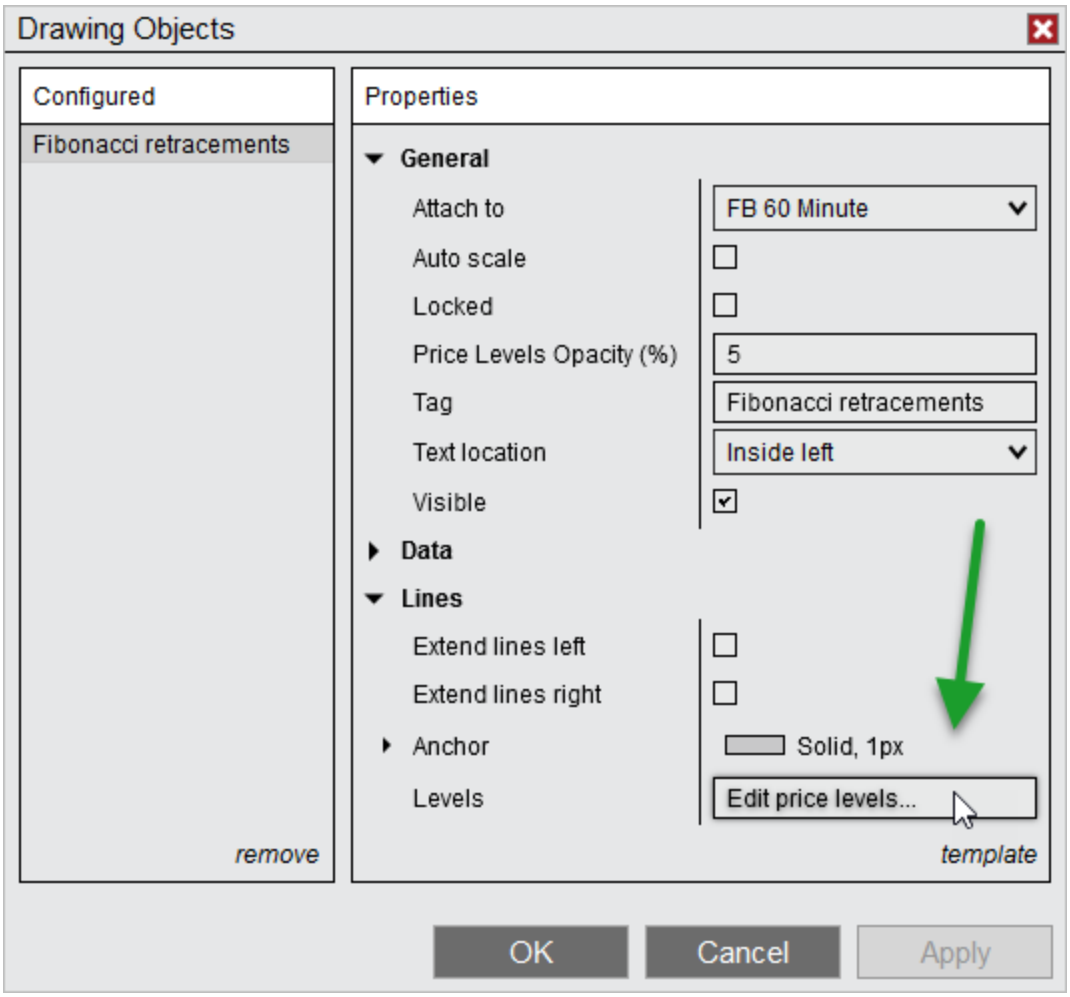
Drawing Tools that include lines drawn at multiple, customizable price levels, such as **Fibonacci Retracements**, include a "Levels" property which can be used to add, remove, or edit levels displayed in objects drawn with that **Drawing Tool**.

The following **Drawing Tools** include a "Levels" property in the **Drawing Objects** dialogue:

- Fibonacci Retracements
- Fibonacci Extensions
- Fibonacci Time Extensions
- Fibonacci Circle
- Andrew's Pitchfork
- Trend Channel

Managing Drawing Object Levels

To add, remove, or edit levels, first left mouse click on a **Drawing Object** to select it, then either double-left mouse click the **Drawing Object**, or right mouse click it and select the **Properties** menu item to open the **Drawing Objects** dialogue. The Levels field will display the number of levels currently applied. Left mouse click within this field to open the **Levels** dialogue, in which you can manage the levels applied to that object.



Adding Drawing Object Levels

In the **Levels** dialogue, click the **add** option to add a new price level. A new level will be added to the bottom of the list in the "Configured" section, and will be automatically selected for editing. You can then customize the new level's line color, dash style, width and value (in percent) the "Properties" section. You can also enable or disable visibility of the level in this section.

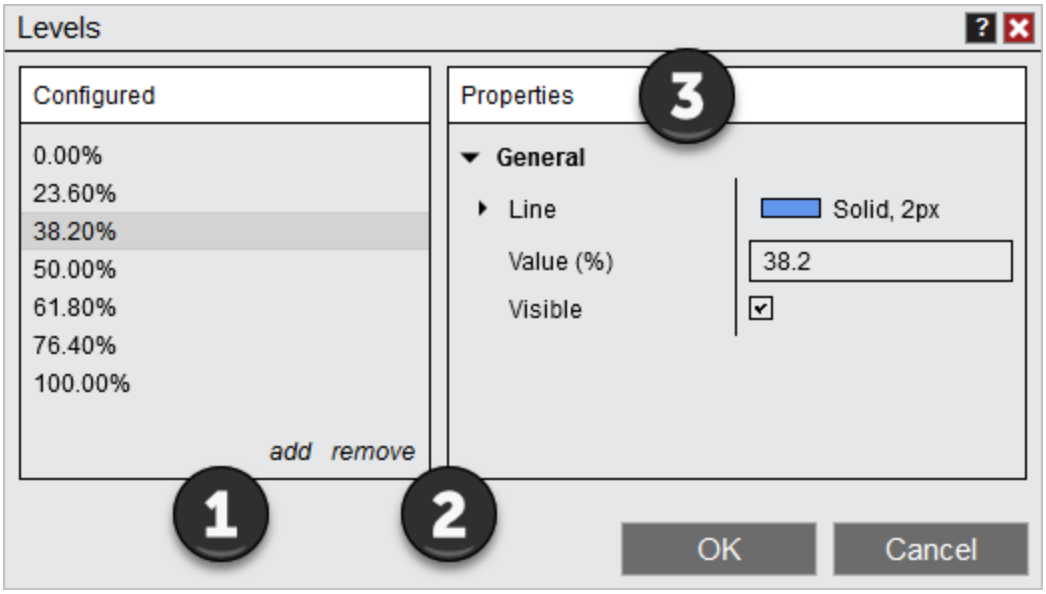
> **Note**: The value property of a level is always expressed in percentage terms, and the placement of the line corresponding to that level will be based upon the anchor points you set for that particular **Drawing Object**.

Removing Drawing Object Levels

To remove a level from within the **Levels** dialogue, first select the level you wish to remove from the list, then select the remove **option**.

Editing Drawing Object Levels

To edit an existing level from within the **Levels** dialogue, first select the level you wish to edit, then change any of the properties for that level in the Properties section. When all properties are set to your desired values, click the **OK** button to save the changes and close the window.



1. Add new price levels with the **add** option
2. Remove existing price levels with the **remove** option
3. Edit properties for new or existing levels in the Properties section

10.6.11 Working with Automated Strategies

Automated NinjaScript strategies can be enabled within an open chart. Both real-time and historical strategy trades will be displayed on the chart. For more information on creating and managing NinjaScript strategies, see the NinjaScript Overview page.

Running a NinjaScript Strategy from a chart

Please see the Running a NinjaScript Strategy From a Chart page for more information on applying and enabling strategies from charts, as well as more information on managing strategy properties.

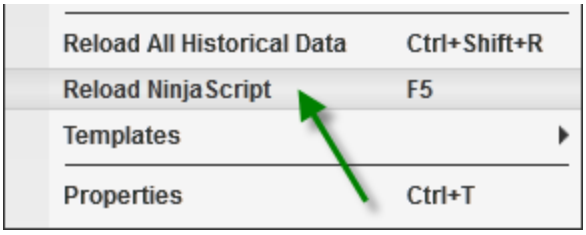
Working with automated strategies in a chart

Strategy Persistence

Automated strategies always persist on a chart whenever it is open, even if Enabled is set to false inside the chart's **Strategies** window. For example, if you shut down NinjaTrader with an enabled strategy in a chart, then reopen NinjaTrader, the strategy will still be applied to the chart with the property Enabled being set to false. This allows you to enable the strategy without having to reconfigure the parameters. However, when the chart containing the automated strategy is closed, the strategy will not persist; it will be disabled and removed.

Reloading NinjaScript

An automated strategy can be reloaded by right mouse clicking in the chart and selecting the menu item **Reload NinjaScript**. Reloading an automated strategy will remove the existing instance of the strategy and add a new one in the chart.



Viewing automated strategy executions in a chart

Executions

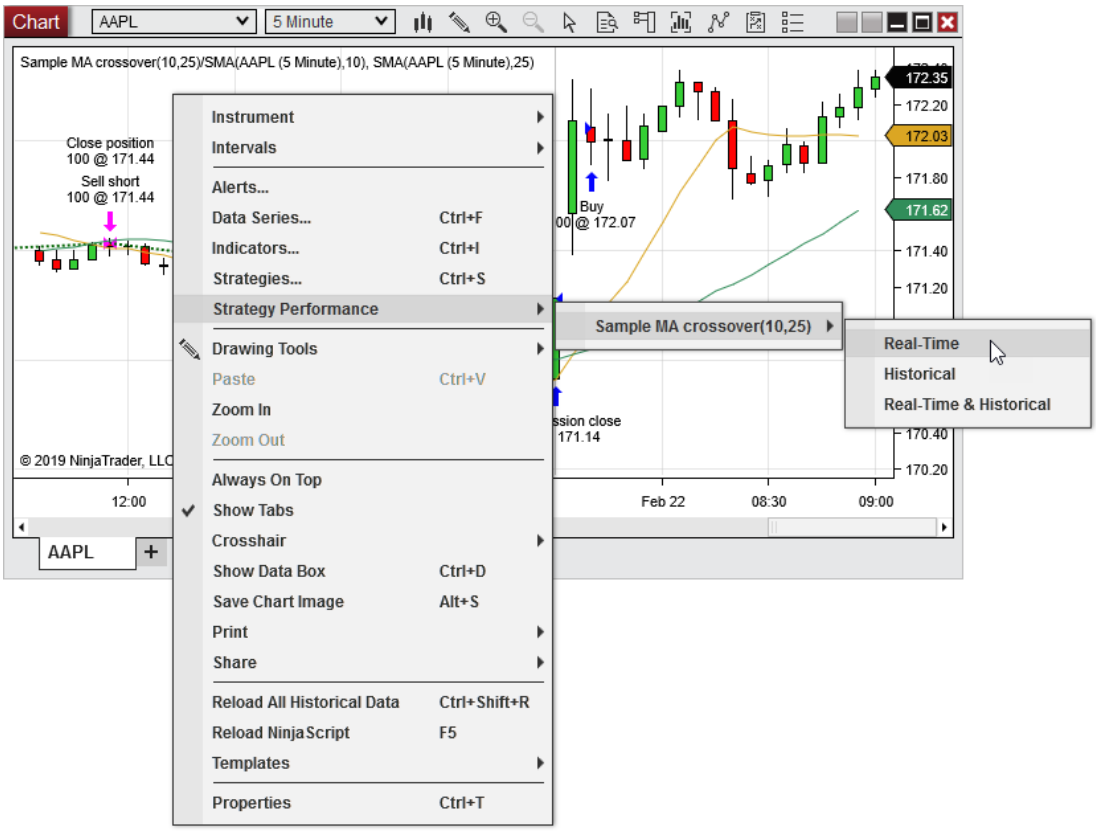
Automated strategy trade executions will be displayed in the chart, depending on the Plot Executions parameter of the **Data Series**. The chart below shows several executions from orders placed by an automated strategy, and each execution is labeled with an appropriate name. Only executions pertaining to the strategy on the chart will be visible when the strategy is enabled. Any manual executions, or executions from strategies not applied to the chart, will **NOT** be shown. Execution markers are configured for each Data Series by selecting the Plot Executions parameter from the Data Series window.

Note: You can view historical trades when a strategy is applied to a chart because the IncludeTradeHistoryInBacktest property is set to **true** by default when a strategy is applied to a chart. You can set this property to false in your code for leaner memory management, at the cost of not being able to access this information. For more information, see the Working with Historical Trade Data page.

Viewing strategy performance

Strategy Performance

Real-time, Historical, or Historical & Real-time executions for the automated strategy can be accessed within the open chart by right mouse clicking in the chart and selecting the menu item **Strategy Performance**, then hovering the mouse over the desired automated strategy and selecting the type of executions you wish to view from the menu that appears. A Performance window will appear where you can view and analyze the trade data.



The following categories of performance data can be selected:

Real-Time	Displays performance statistics for trades the strategy has taken in real-time **ONLY**
Historical	Displays performance statistics for historical trades **ONLY**, calculated before any real-time trades are taken
Real-Time and Historical	Combines historical and real-time performance statistics in a single report

Understanding strategy templates

Each NinjaScript strategy's parameters can be saved as a template for later use,

and multiple templates can be saved for each strategy. Once saved in a template, the customized parameters can be loaded quickly whenever the specified template is applied to an instance of the strategy for which it was created.

What is Saved

All parameter settings are saved, with the following exceptions:

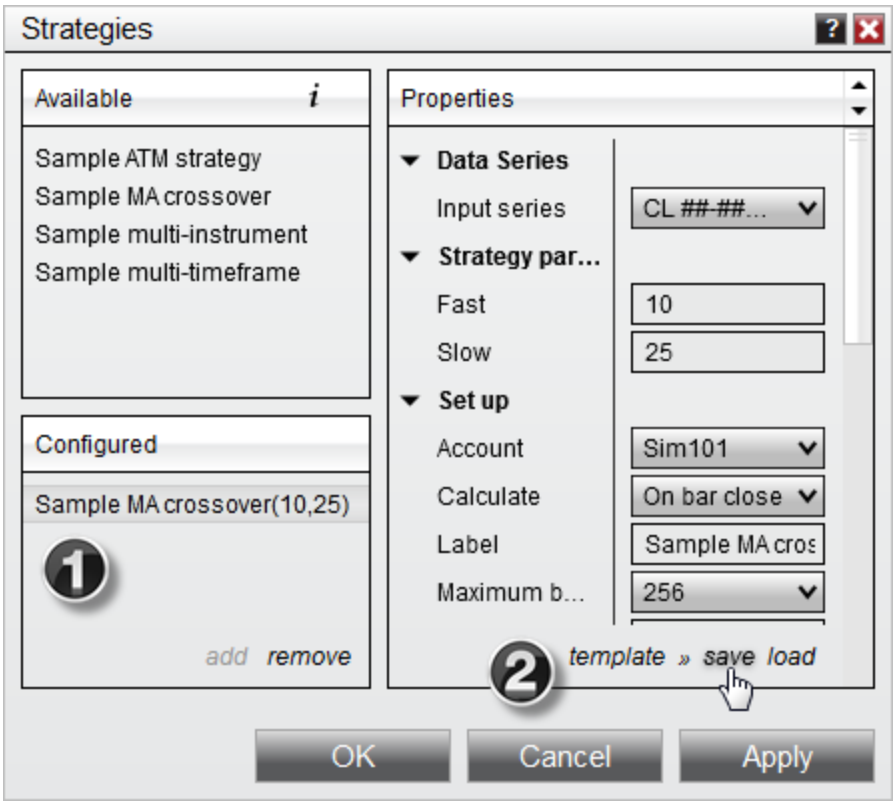
- Account defaults to the **Sim101** account
- Enabled defaults to False

Saving Custom Strategy Settings

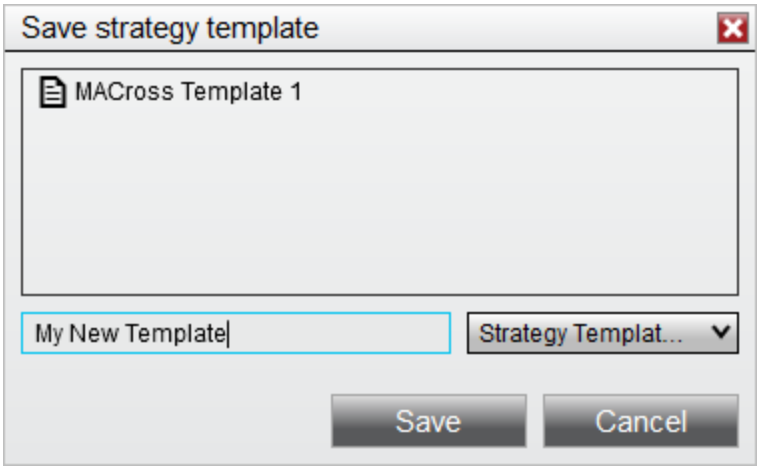
To save custom NinjaScript strategy parameters as default:

1. Set parameters to desired values
2. Left mouse click on the **template** text located in the bottom right of the properties dialog. Selecting **save** will open the **Save Strategy Template** window, in which you can enter a name for a new template or overwrite an existing template.

If you wish to load a previously saved template, you can select the **load** option after left mouse clicking on the **template** text.
If you change your settings and later wish to go back to the original settings, you can left mouse click on the **template** text and select the **restore** option.



1. In the image above, the "Sample MA Crossover" strategy is applied, as seen in the "Configured" section.
2. A new template can be saved for the selected by clicking the **template** text, then selecting **save**.



The **Save Strategy Template** window will allow you to name and save a new template for the configured strategy.

10.6.12 Saving Chart Defaults and Templates

NinjaTrader allows you to save many of your customized chart settings as default, or to save templates for later use. This can save time by automatically setting up your **Data Series**, indicators, **NinjaScript** strategies, chart properties, and drawing objects the way you prefer. Saved default settings apply to any new instances of these items that you create, while templates can be applied to either new or existing items.

Understanding Data Series default settings

Each **Period Type** and **Chart Style** can have different default settings saved for the customizable **Data Series** parameters. Once saved as default, the customized parameters will load when the **Period Type** is selected.

What is Saved

For **Period Types**, all parameter settings are saved, with the following exceptions:

- Bar width
- End date will default to the current day's date
- Label will default to the instrument name
- Panel will stay on its current panel
- Scale justification will default to "Right"
- Session template will default to "<Use instrument settings>"

For **Chart Styles**, all parameters within the **Chart Style** section of the **Data Series** window will be saved.

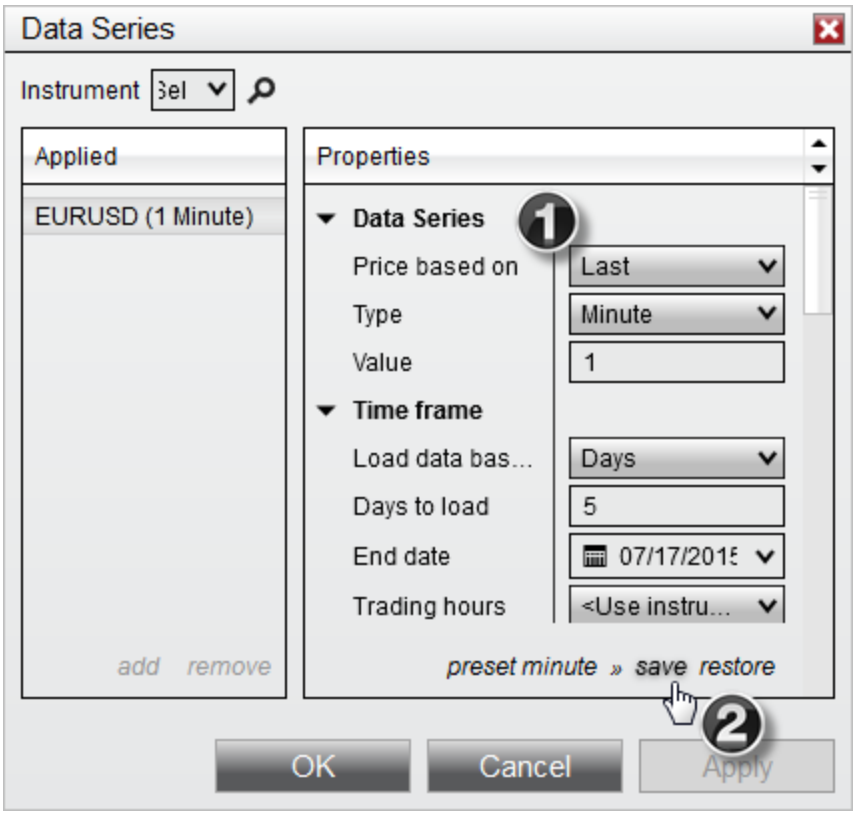
> **Note**: **Period Type** default settings overrule **Chart Style** default settings. When you change the **Period Type**, the parameters saved in that **Period Type**'s default settings will change any parameters which may have already been loaded. However, when a **Chart Style** is changed, any saved defaults for that **Chart Style** will be used.

Saving Custom Data Series Settings by Period Type

To save **Data Series** parameters as default for a particular **Period Type**:

1. Set the **Data Series** parameters to desired values
2. Left mouse click on the **preset** text located in the bottom right of the properties dialog. Selecting the option **save** will save these settings as the default used every time you select that period type for a **Data Series**.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the **preset** text and select the **restore** option to return to the original settings.



1. In the image above, we have selected the Minute **Period Type**.

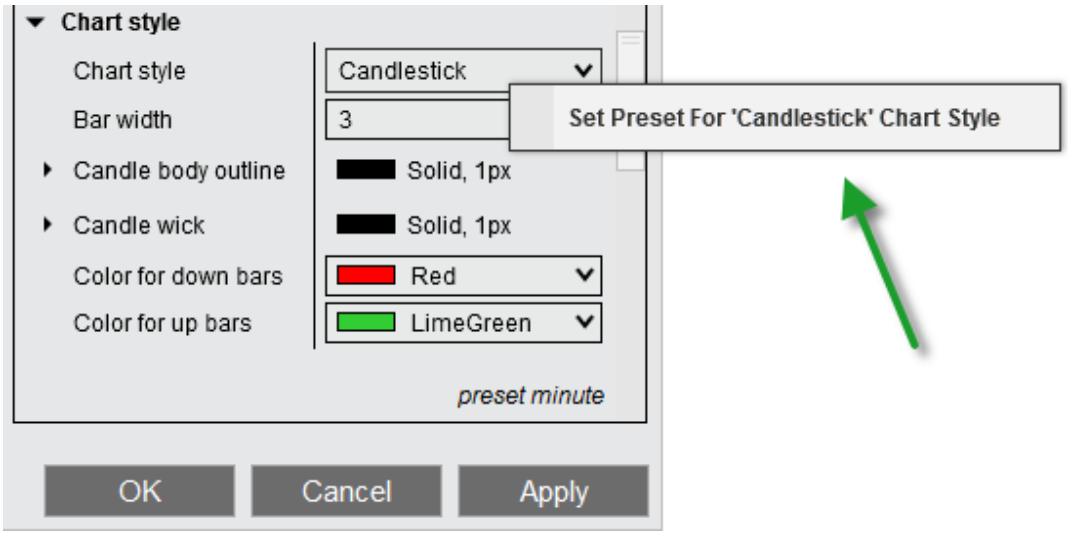
2. Notice the **preset** text changes to **preset minute**. By clicking **save**, data series parameters will be saved for the Minute **Period Type** specifically.

Saving Custom Data Series Settings by Chart Style

To save **Data Series** parameters as default for a particular **Chart Style**:

1. Set the **Chart Style** parameters to desired values
2. Right mouse click on the **Chart Style** dropdown menu, then select **Set Preset For "X" Chart Style**, where X represents the currently selected **Chart Style**.

In the image below, we can set the defaults for the Candlestick **Chart Style** specifically.



Understanding indicator default settings

Each individual indicator's parameters can be saved as default or as a custom template. Once saved as default, the customized parameters will load whenever the specified indicator is added to a chart.

What is Saved

All parameter settings are saved, with the following exceptions:

- Input series will default to the first **Data Series** applied to the chart
- Panel will use the default NinjaTrader settings

Saving Custom Indicator Settings

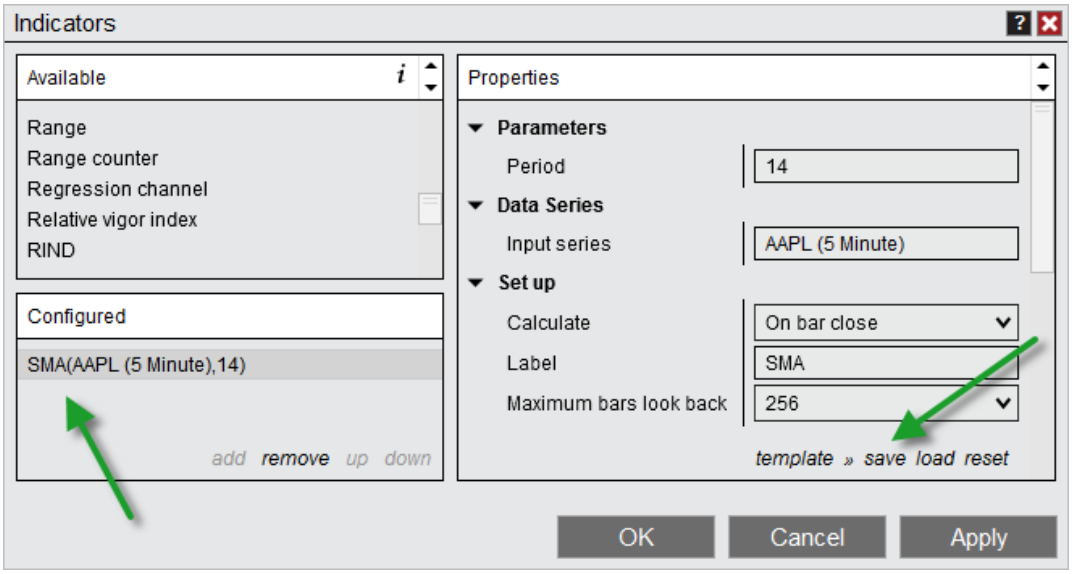
To save Indicator parameters:

1. Set the **Data Series** parameters to desired values
2. Left mouse click on the **template** text located in the bottom right of the properties dialog. Selecting **save** will open the **Save** window, in which you can enter a name for a new template or select an existing template to overwrite it. Naming it Default will save these settings as the default used every time you apply that indicator to a chart.
3. Click the **Save** button when finished

If you wish to load a previously saved template, you can select the **load** option after left mouse clicking on the **template** text.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the **template** text and select the **restore** option.

In the image below, the parameters will be saved for the selected SMA indicator. Any time an SMA indicator is applied to a chart, the saved parameters will be loaded.



Understanding strategy templates

Please see the Working with Automated Strategies page for more information on saving and managing templates for **NinjaScript** strategies.

Understanding chart property default settings

Customized chart properties can be saved as default. Once saved as default, the customized properties will be loaded whenever a new chart is opened. The **Chart Properties** window can be opened by left mouse clicking on the **Properties** icon in the chart toolbar, by selecting the menu item **Properties** from the right click menu in the chart, or via the default CTRL + T Hot Key.

What is Saved

All property settings are saved.

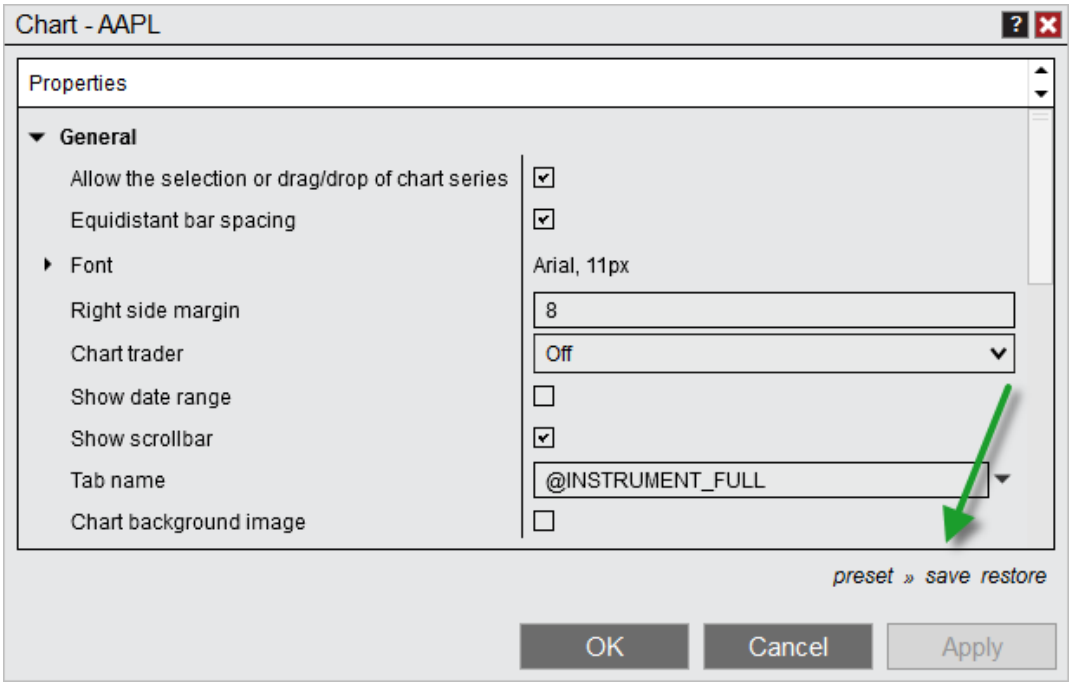
Saving Custom Chart Property Settings

To save custom chart parameters as default:

1. Set parameters to the desired values
2. Right mouse click within the chart window, then select the **Templates** menu item, then select **Save as Default**.

All chart properties can be restored to NinjaTrader default settings by left mouse clicking on the **preset** text within the **Chart Properties** window, then selecting **restore**.

In the image below, all chart properties will be saved as the default for new charts.



Tip: **Chart Templates** (including the default chart template) will overrule chart property default settings. If you wish to use your preset chart property defaults, select <None> as the chart template when opening a new chart.

Understanding Drawing Object templates

Please see the Working with Drawing Tools & Objects page for more information on saving and managing drawing object templates.

How to save a Chart Template

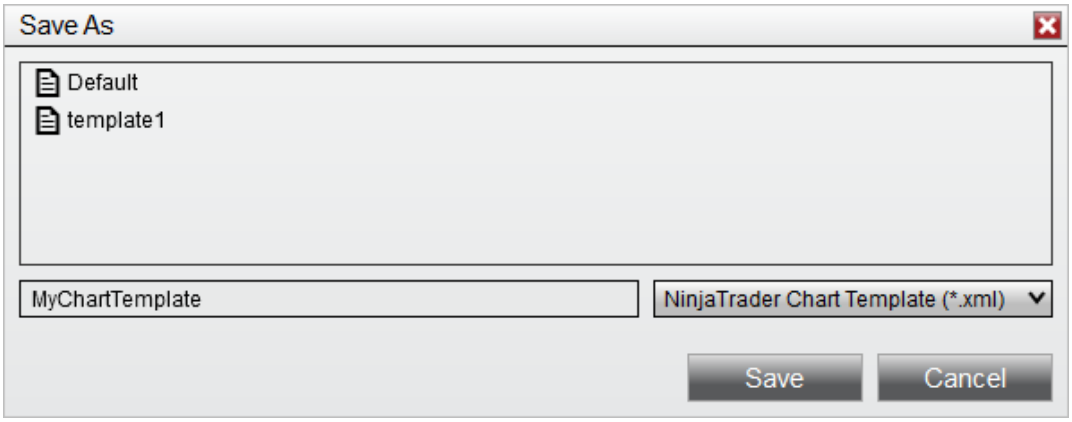
Chart properties, chart panel properties, and indicator settings can be saved as a **Chart Template.** A **Chart Template** can be applied to a new chart or an open chart to load customized chart settings, provided the template and chart share the same number of Data Series objects.
Since templates are intended to be able to be applied to any data series, they do not include items that would be unique to the data series, such as drawing objects. A Trend Channel drawn on AAPL would not be relevant on a COKE chart.

Saving a Chart Template

To save a **Chart Template:**

1. Once you have a chart set up to your liking, right mouse click within the chart and select the menu item **Templates**, followed by **Save As**
2. The **Save As** window will appear. Enter a name for your template and press the **save** button.

In the image below, we are saving a new chart template named "MyChartTemplate."



Changing the Default Chart Template

A **Chart Template** can be saved as the default template used for all new charts. Once saved, the default template will determine the properties of each new chart opened, unless you specify a different template.

To save a **Chart Template** as default:

1. Right mouse click within an open chart and select the **Templates** menu
2. Select the menu item **Save as Default**

How to load, remove, or rename a Chart Template

Loading a Chart Template

A **Chart Template** that was previously saved can be loaded on any chart that has the same number of **Data Series** as the chart which was used to save it.

To load a **Chart Template**:

1. Right mouse click and select the menu item **Templates** followed by the **Load** menu item
2. The Load window will appear. Select the template to load from the list of templates, then press the **Load** button.

> **Note**: If a **Chart Template** is loaded, settings from that template will take precedence over any settings manually configured on the Data Series. For example, **Trading Hours** currently configured will be ignored, and the chart will use the **Trading Hours** which were saved in the **Chart Template.**

Removing a Chart Template

To remove a **Chart Template** from the list of saved templates:

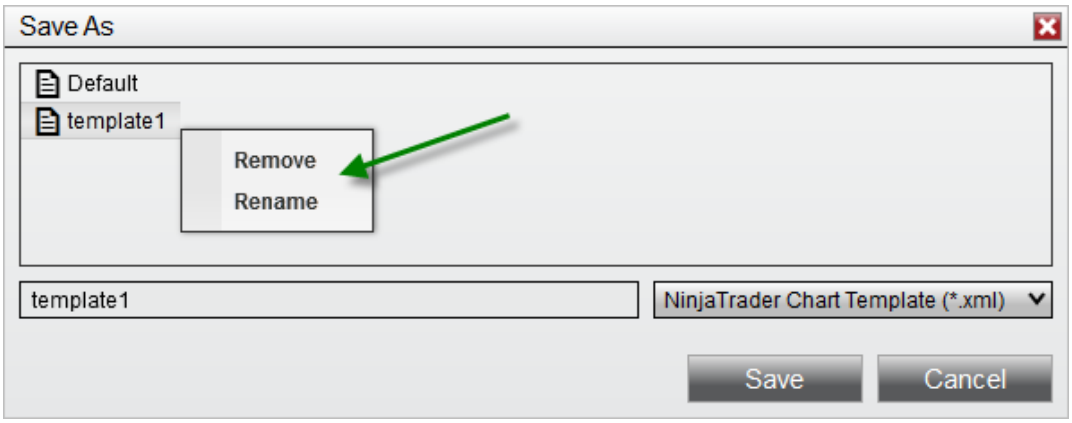
1. Right mouse click within a chart and select the menu item **Templates** followed by either the **Save As** or **Load** menu items
2. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the template for removal from the list of templates, then select the **Remove** menu item.

Renaming a Chart Template

To rename an existing **Chart Template** from the list of saved templates:

3. Right mouse click within a chart and select the menu item **Templates** followed by either the **Save As** or **Load** menu items
4. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the template from the list of templates, then select the **Rename** menu item.

In the image below, we can either remove or rename the selected **Chart Template**.



10.6.13 Data Box

The **Data Box** and **Mini Data Box** allow you to access both bar and indicator values on your chart at a glance. The **Mini Data Box** provides a compressed view of your chart data, while the **Data Box** provides a more comprehensive view of the data.

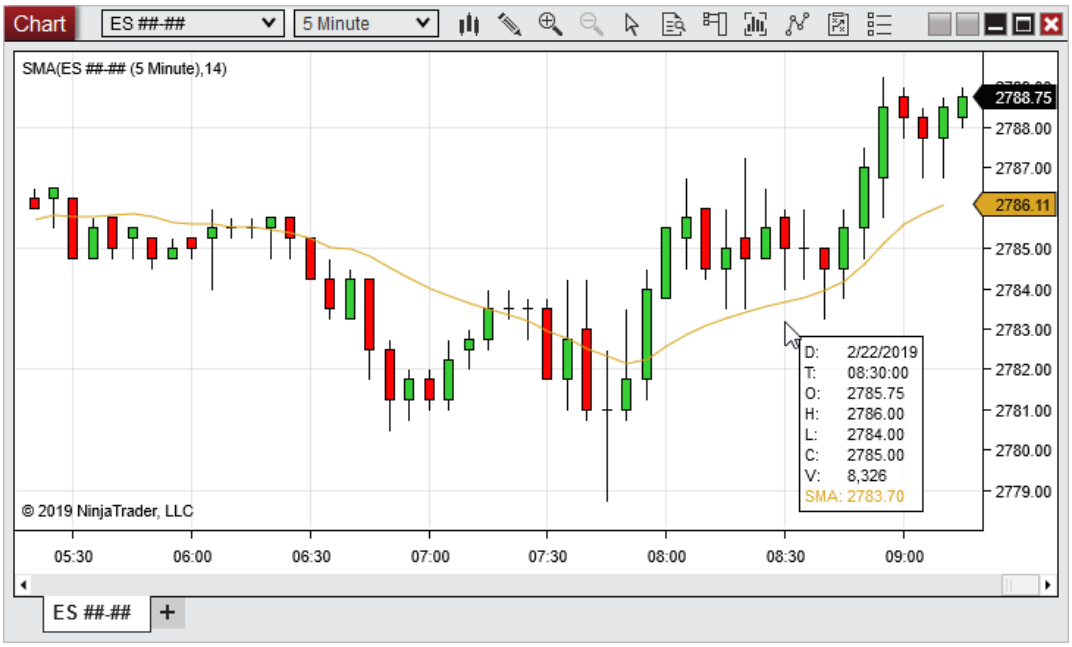
Understanding the Mini Data Box

Opening the Mini Data Box

To access the **Mini Data Box**, hover your mouse cursor over the chart panel from which you would like to see values, then press down on your middle mouse button. After pressing and holding down your middle mouse button, the **Mini Data Box** will appear with a range of information related to the data series and indicators in the chart panel in which you click. You can then continue holding down your middle mouse button as you move around the chart to view values for other bars, or release your middle mouse button to hide the **Mini Data Box** once more.

Mini Data Box Display

The **Mini Data Box** displays the Date/Time, Open, High, Low, Close and Volume information of the selected bar on the chart, as well as the values of any indicators plotted in that chart panel. This view is ideal for quick access to information on a specific bar.



The display order of data in the **Mini Data Box** is as follows:

Date	Date of the bar corresponding to the location of your cursor
Time	End-of-bar time stamp corresponding to the location of your cursor
Price Data	Open, High, Low, and Close values for the bar
Volume	Volume for the bar
Indicator Values	Values for indicators contained in the specific chart panel in which you view the **Mini Data Box**. Indicators will be displayed in the same order in which they were added in the Indicators window.

Tip: If more than one data series is applied in the same chart panel, they will be displayed in the **Mini Data Box** in the order in which they were added to the chart, and all indicators using a specific data series as input will be displayed beneath the data for that specific data series.

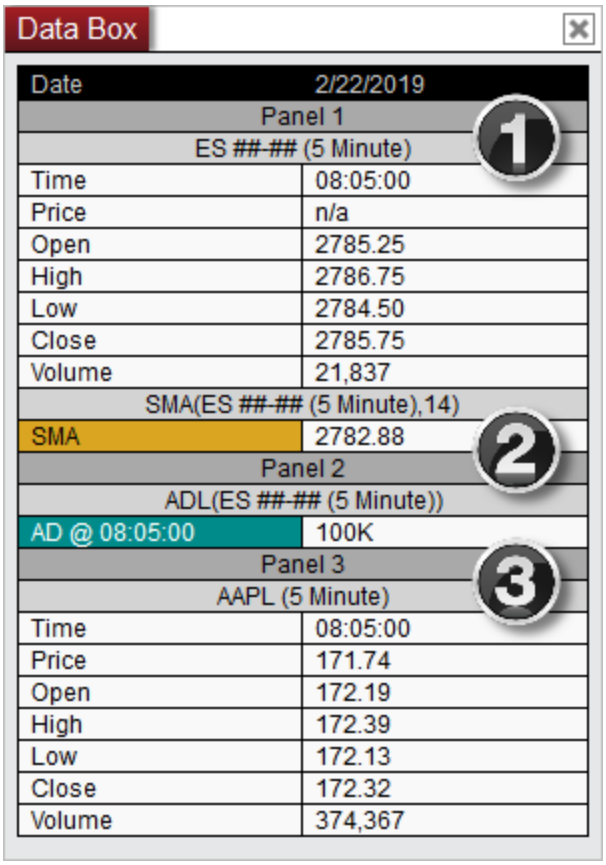
Understanding the Data Box

Opening a Data Box

The **Data Box** displays all bar data and indicator values on your chart, based on your mouse cursor position. You can enable or disable this window via the right mouse click context menu, the **Show Data Box** chart toolbar icon, or by using the default shortcut CTRL+D. If you have multiple charts open, the **Data Box** will display the values of the chart over which your mouse cursor is currently hovering. Being able to use one **Data Box** for multiple charts eliminates the need to open multiple **Data Boxes**, which conserves monitor space.

Data Box Display

The **Data Box** displays the date at the top of the window, followed by additional data organized by panel. Under each panel heading, any data series displayed in that panel will be listed first, followed by any indicators displayed in that panel.



In the image above, we can see:

1. Panel 1 includes an ES ##-## data series and a 14-period SMA indicator using the ES ##-## as its input series.

2. Panel 2 includes an ADL indicator only, using the ES ##-## as its input series.

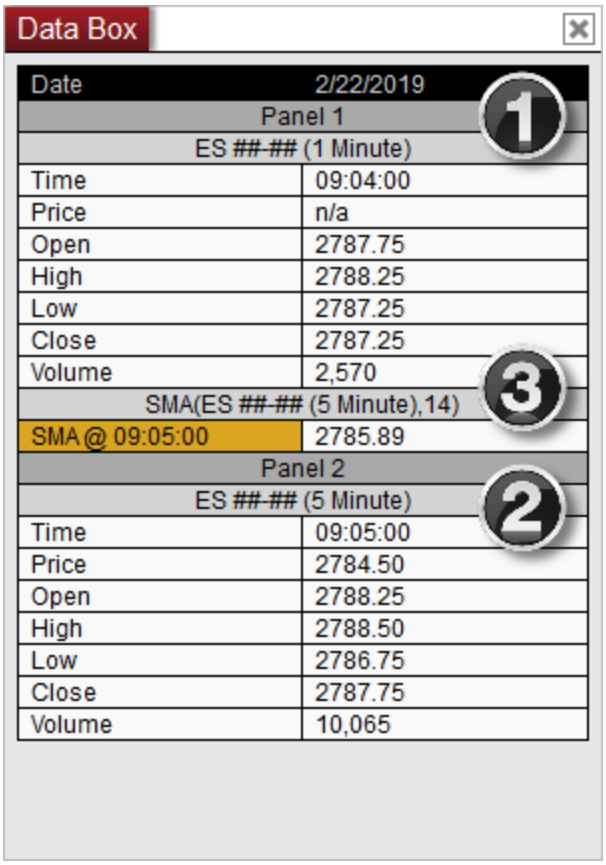
3. Panel 3 includes a second data series, AAPL, with no indicators.

The column splitter can be re-sized by hovering your cursor until the sizing arrows appear. Once the sizing arrows are showing you can press down on your left mouse button and drag the column splitter to the desired location, then release the left mouse button.

> **Note**: Indicators or Data Series with the **Display in Data Box** parameter set to false will **NOT** be displayed in the **Data Box**.

Indicator Time Stamps on Multi-Series Charts

Indicator plot names listed in the **Data Box** are followed by a time stamp indicating which bar time the corresponding indicator uses as its input series. This will allow you to quickly see which data series is being used by each indicator on a multi-series chart.



As an example, notice the following about the image above:

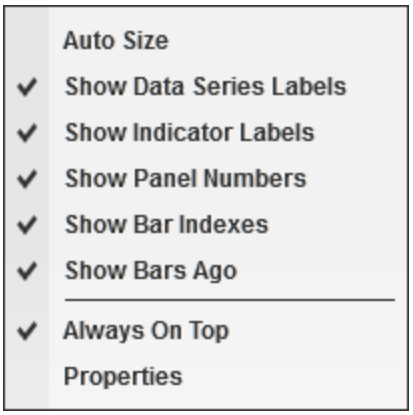
1. Panel 1 contains a 1 minute Data Series.

2. Panel 2 contains a 5 minute Data Series.

3. In addition to the Data Series, Panel 1 also contains an SMA indicator which uses the 5-minute Data Series (contained in Panel 2) as its input series. Although this indicator is plotted in Panel 1, the time stamp reveals that its values are based upon the Data Series in Panel 2.

Right Click Menu

Right click anywhere in the **Data Box** to access the right click menu.



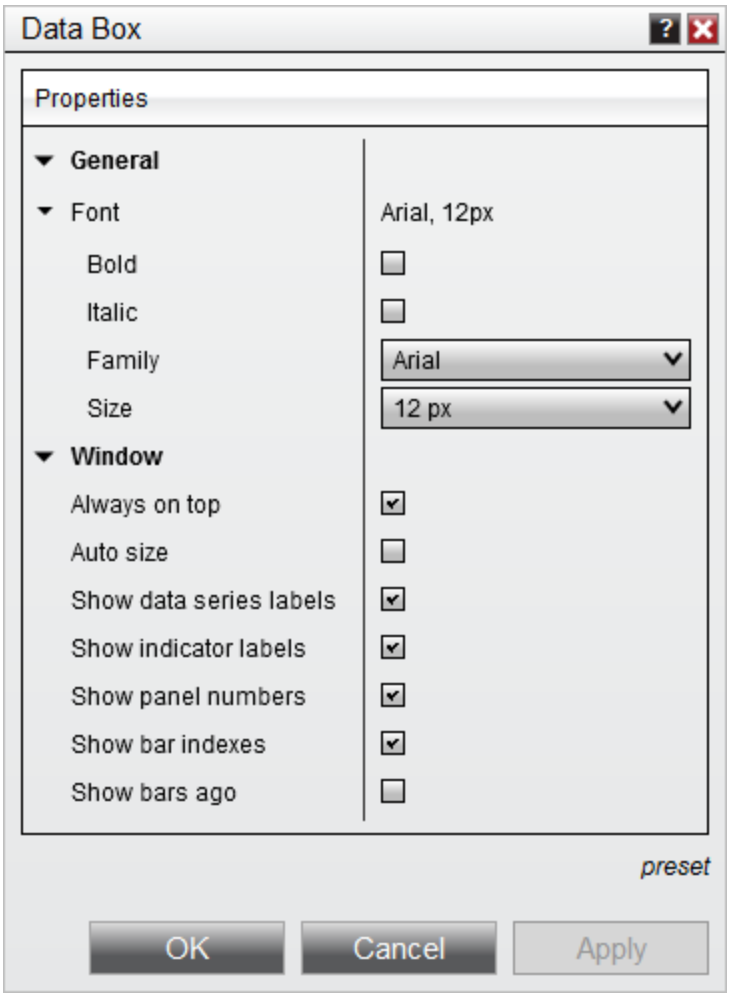
The following options are available:

Auto Size	When enabled, your data box will re-size as you move your cursor between charts to meet each chart's display requirements.
Show Data Series Labels	Enables or disables the display of the Data Series labels
Show Indicator Labels	Enables or disables the display of the Indicator labels
Show Panel Numbers	Enables or disables the display of the Panel numbers
Show Bar Indexes	Enables or disables the display of the bar number of the bar being viewed. The first bar on the chart has an index of 0, and each bar thereafter increments its bar index by 1.
Show Bars Ago	Enables or disables the display of the Bars Ago value of the bar being viewed. The latest bar on the chart is considered 0 bars ago, and each preceding bar

	increments its Bars Ago value by 1.
Always On Top	When enabled, this will keep the **Data Box** above all other windows in your workspace, so the values are always visible
Properties	Opens the **Data Box Properties** window

Data Box Properties

Many options in the Data box can be changed within the Data Box Properties window. To access this window, first right click within the **Data Box**, then click **Properties**. The following properties are available:



Font	Set the font family and size, and enable or disable bold or italics
Always On Top	When enabled, this will keep the **Data Box** above all other windows in your workspace, so the values are always visible
Auto Size	When enabled, your data box will re-size as you move your cursor between charts to meet each chart's display requirements.

Show Data Series Labels	Enables or disables the display of the Data Series labels
Show Indicator Labels	Enables or disables the display of the Indicator labels
Show Panel Numbers	Enables or disables the display of the Panel numbers
Show Bar Indexes	Enables or disables the display of the bar number of the bar being viewed. The first bar on the chart has an index of 0, and each bar thereafter increments its bar index by 1.
Show Bars Ago	Enables or disables the display of the Bars Ago value of the bar being viewed. The latest bar on the chart is considered 0 bars ago, and each preceding bar increments its Bars Ago value by 1.

Once you have your properties set to your preference, you can left mouse click on the **preset** text located in the bottom right of the properties dialog. Selecting the option **save** will save these settings as the default settings used every time you open a new Data Box.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

10.6.14 Cross Hair

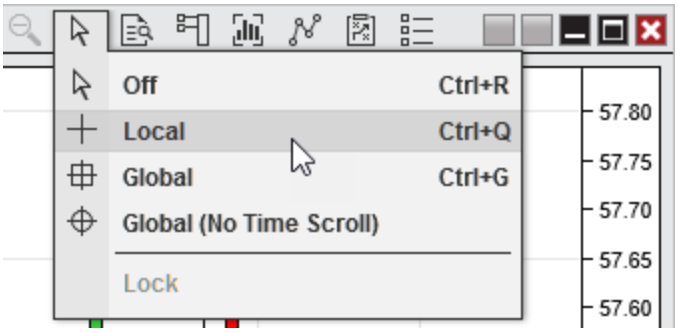
The **Cross Hair** changes the cursor to a pair of intersecting vertical and horizontal lines, allowing you to use your cursor to pinpoint specific coordinates on your chart. The lines displayed by the **Cross Hair** extend to the X (time) axis and Y (price) axis of the chart, and include markers in both axes to display the precise position of the cursor. The **Global Cross Hair** allows you to link **Cross Hairs** from multiple chart windows. This means that as you

move the **Global Cross Hair** in one chart, all other **Global Cross Hairs** will move together by automatically staying at the same time and price.

How to enable the Cross Hair

Enabling the Cross Hair

There are multiple ways to enable the **Cross Hair** within a chart window:



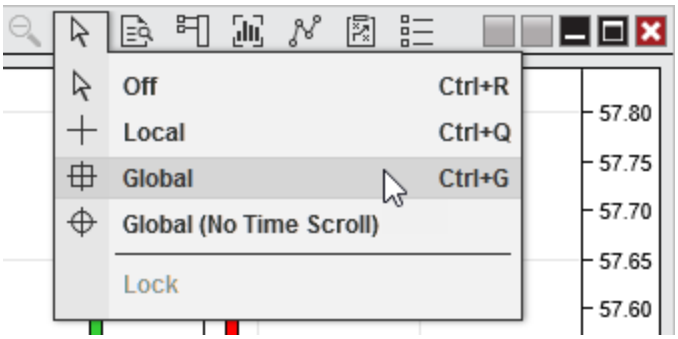
- Left mouse click on the **Cursor** icon in the chart toolbar and select the **Local** menu item.
- Right mouse click within the chart and select the **Crosshair** menu, then select the **Local** menu item.
- Use the default CTRL +Q Hot Key

The cursor icon within the chart toolbar will change to a cross hair icon, letting you know that Cross Hair is enabled on the chart.

How to enable the Global Cross Hair

Enabling the Global Cross Hair

Just like the **Cross Hair**, there are multiple ways to enable the **Global Cross Hair** within a chart window:



- Left mouse click on the **Cursor** icon in the chart toolbar and select the **Global** menu item.
- Right mouse click within the chart and select the **Crosshair** menu, then select the **Global** menu item.
- Use the default CTRL +G Hot Key

The cursor icon within the chart toolbar will change to a cross hair icon with a square border, letting you know that **Global Cross Hair** is enabled on the chart.

Tips:
- If the active **Global Cross Hair** moves outside the viewable horizontal range of any other chart with **Global Cross Hair** enabled, the horizontal axis in the inactive charts will automatically scroll to keep aligned with the active cursor. If you wish to use the **Global Cross Hair** with time-axis scrolling disabled, you can select **Global (No Time Scroll)** from either the chart toolbar or the **Crosshair** menu. With **Global (No Time Scroll)** selected, the cursor icon within the chart toolbar will display a cross hair with a round border.

How to use the Global Cross Hair

Using the Global Cross Hair

The **Global Cross Hair** must be enabled on more than one chart in order to take full advantage of its functionality.

The images below shows two CL charts, a 1 Minute and 5 Minute, both with **Global Cross Hair** enabled. Notice the time and price cross hair values in each chart are the same. When the cursor is moved in any chart with **Global Cross Hair** enabled, cross hairs in all other charts with **Global Cross Hair** enabled will move as well, to stay at the same time and price coordinates.

1. The cursor is active on the 1-minute chart, and the time and price axis values corresponding to the position of the **Global Cross Hair** are displayed.

2. The position of the Global Cross Hair on the 5-minute chart automatically updates based on the position of the cursor in the 1-minute chart.

Cross Hair Options

Cross Hair Options

- You can optionally lock the crosshair in a specific point in time. To lock the time (vertical) line of the **Cross Hair** or **Global Cross Hair**, while retaining the ability to move the horizontal line, you can enable the **Locked** property within the Crosshair section of the Chart Properties window, or use the default CTRL +L Hot Key. When using the **Global Cross Hair**, locking will apply to all charts with **Global Cross Hair** enabled.

- Additional options related to the **Cross Hair** or **Global Cross Hair** can be set in the Chart Properties window. The following properties can be set:

Color	Sets the color for the **Cross Hair** lines
Cross hair Type	Sets the type of **Cross Hair** to be enabled, including **Local**, **Global**, **Global (No Time Scroll)**, and **Off**
Dash Style	Sets the style to be used for the **Cross Hair** lines, including **Solid**, **Dash**, **Dash Dot**, **Dash Dot Dot**, and **Dot**
Draw cursor only	Enables or disables drawing only a mini **Cross Hair** without the full lines but including the price / time axis labels. This mode can improve performance for setups operating with a lesser powerful GPU.
Locked	Enables or disables **Cross Hair** locking
Width	Sets the width of the **Cross Hair** lines

10.6.15 Trading from a Chart

Please see the Chart Trader section under Order Entry section for more information on trading in the chart.

10.6.16 Chart Properties

Many of the visual display settings of NinjaTrader charts can be customized using the Chart Properties window.

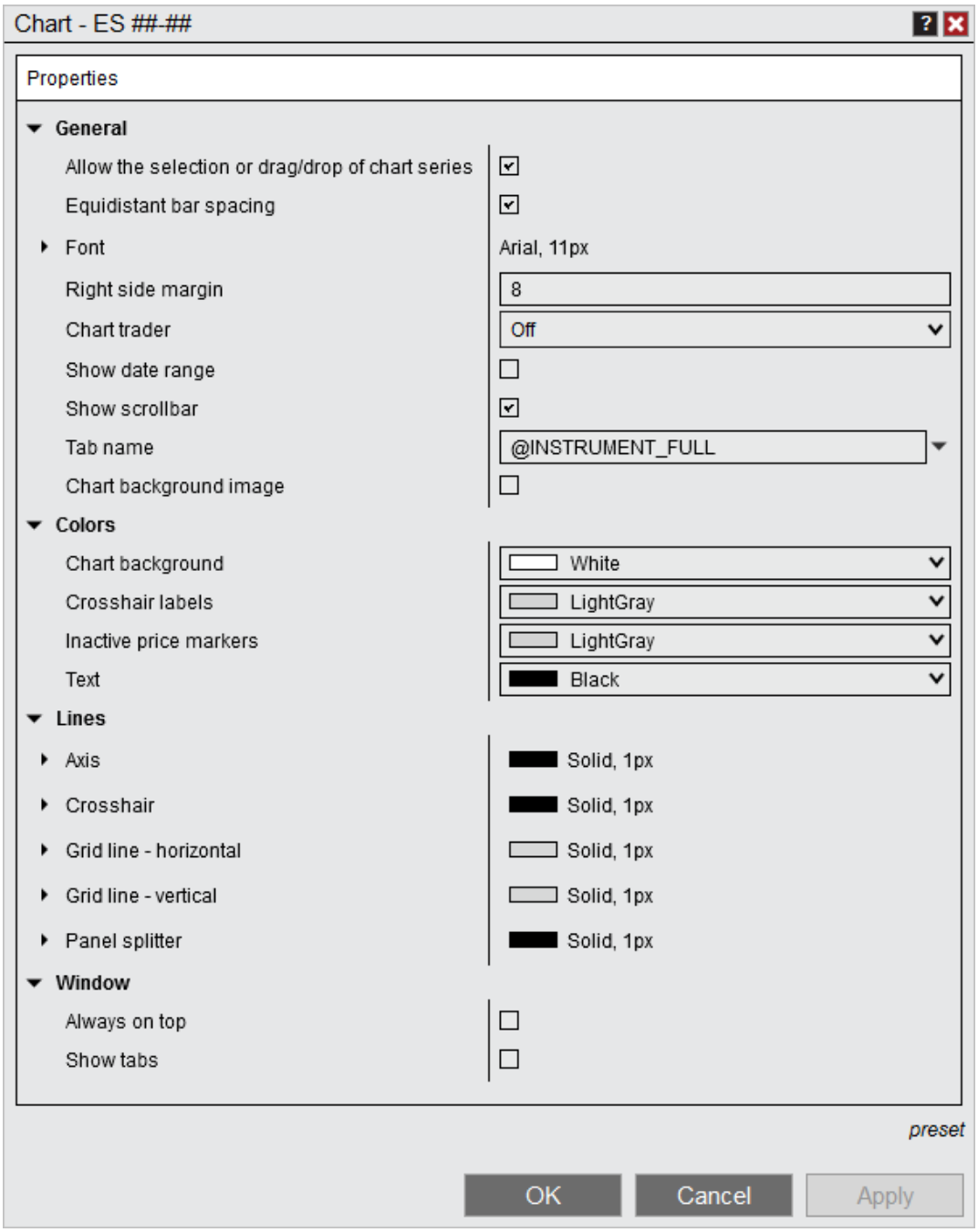
How to access the Chart Properties window

The Chart Properties window can be accessed in the following ways:

1. Left mouse click the Properties button in the chart toolbar
2. Right mouse click within the chart and select the menu item **Properties**
3. Use the default CTRL + T Hot Key

Available properties and definitions

The following chart properties are available for configuration within the Chart Properties window:



General

Allow the selection or drag/drop of chart series	Enables or disables the selection of **Data Series** and indicators for drag and drop
Equidistant bar spacing	Enables or disables plotting bars an equal distance from each other. Please see the "Working with Multiple Data Series" section of the Help Guide for more information.
Font	Sets the font display properties for the chart
Right side margin	Sets the spacing between the Y-axis and the current bar in pixels
Chart trader	Sets the chart trader display mode.
Show date range	Enables or disables showing the date range label in to the top left of chart. The date range reported are the dates that are currently visible in the chart.
Show scrollbar	Enables or disables showing the horizontal chart scroll bar
Tab name	Sets the name displayed in the tab. By default the instrument name is displayed.
Chart background image	Enables or disables the option to use an image as the background for the chart
Color	
Chart background	Sets the chart background color

Crosshair label	Sets the color for the cross hair label
Inactive price markers	Price markers display the current price of bars and indicators on the Y-axis. When looking at the current bar, the price markers will take the color of the data series. When scrolling back through historical bar data, the markers are inactive (not real-time) and will be displayed by the color set on this property
Text	Sets the font display properties for the chart
Lines	
Axis	Sets the drawing properties for both the vertical and horizontal chart axis
Crosshair	Sets the drawing properties of the crosshair
Grid line - horizontal	Sets the drawing properties of the horizontal grid lines
Grid line - vertical	Sets the drawing properties of vertical grid lines
Panel splitter	Sets the drawing properties of the splitter drawn between panels.
Window	
Always on top	Sets if the window will be always on top of other windows.

Defaults for the Chart Properties window can be saved by left mouse click on the "Set Default" button. Please see the "*Saving Chart Defaults*" section of the Help Guide for more information.

Using Tab Name Variables

Tab Name Variables

A number of pre-defined variables can be used in the "Tab Name" field of the **Chart Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.6.17 Reload Historical Data

While you are connected to a market data provider that supports historical data, right mouse click within a chart to bring up the context menu and select the **Reload All Historical Data** menu item. Historical data for the base interval unit (minute bars for a 5 minute chart for example) will be reloaded for all charts of the same instrument. Reloading all historical data will overwrite all locally stored cache and repository data.

10.6.18 How Bars are Built

NinjaTrader builds chart bars from the data provided by your data provider. There are multiple elements in the bar building process that can influence how bars are built.

Understanding the variables involved in building chart bars

Bar Time Stamp

NinjaTrader stamps a bar with the closing time of the bar. For example, a minute bar with a time of 9:31:00 AM has data from 9:30:00 AM through 9:30:59 AM. Using end of bar time stamps is required in order to be able to plot multiple series of differing time frames within a single chart all accurately synchronized to time.

Discrepancies Between Different Data Feeds

Different data feeds produce different charts, especially when using tick based intervals vs time based intervals. Market data vendors each employ various methods for tick filtering, throttling and time stamping. As a result, no data stream is 100% identical and thus can cause subtle differences in charts. Since NinjaTrader supports many of the leading brokerage and data feed technologies, it is highly likely that two traders using NinjaTrader on different data feeds will have minor differences when plotting the same market and time interval.

Time Settings

Different session templates as well as the date range of data being plotted can affect the chart display and indicator values.

Real-Time Tick Filter

If you have the real-time tick filter enabled, it is possible that your offset percent (the percent away a tick is in value from the last traded price to be considered a bad tick) may be too tight and thus a good tick (gap up/down on session open for example) could be excluded from the bar.

Understanding the underlying base data type required for constructing various chart bars

Base Data Types Used to Build Bars

A chart bar (period type) requires a base data type as its source for bar construction. Following are NinjaTrader supported period types and their required base data type values. A check mark represents the data base value that is needed to build the period type.

Period type	Base data type values		
	Tick	Minute	Daily
Tick	✓		
Volume	✓		
Range	✓		
Second	✓		

Renko	✓		
Minute		✓	
Day			✓
Week			✓
Month			✓
Year			✓

The base data is important to understand. If you are connected to a market data vendor that does NOT support "tick data," you will NOT be able to build chart bars that use "tick data" as its base data such as tick, volume, range or second charts. A matrix of supported data vendors and their varying levels of service is located here.

Understanding why a chart can look different after reloading historical data from the server

As ticks come into NinjaTrader in real-time, they are time stamped based on your local PC time if they do not already have an associated time stamp that is provided from the real-time data source. NinjaTrader then builds bars based on the time stamp of the incoming tick and displays these bars in your chart in real-time.

Let's say you have a tick (tick "A") with a time stamp of 10:31:00 AM which gets packaged into the 10:32:00 AM bar and happens to be the high of that bar. An hour later, you reload historical data from your historical data provider into NinjaTrader. This process will overwrite the existing data. The 10:32:00 AM bar now looks different since the high made by TICK "A" is now part of the prior bar, 10:31:00 AM. How is this possible?

- Your PC clock could have been off so the time stamp is delayed
- Your internet may have been lagging so the tick came in slightly delayed and therefore the time stamp is delayed
- Due to standard latency, even 50ms delay (which is normal) could be the difference between a 10:30:59 and 10:31:00 time stamp

- There is no way of knowing how the historical data provider packages their bars

The only way to ensure that data always looks the same is if every connectivity provider sent ticks with time stamps AND that all vendors synchronized on time stamps. Unfortunately, this is just not a reality nor plausible scenario.

Loading Historical Data

Please see the "*Historical & Real-Time Data*" section of the Help Guide for more information.

10.6.19 How Trade Executions are Plotted

Trade executions in NinjaTrader are tied to specific timestamps based on when the execution actually occurs as opposed to specific bars on the chart. NinjaTrader does it this way to allow you the flexibility of using multiple charts of differing period types and still being able to visualize where the **trade executions** occurred. The following article outlines some scenarios with time based execution plotting.

Understanding trade executions and the local PC clock

When a **trade execution** occurs in NinjaTrader it is timestamped natively by your provider if they support that or locally by NinjaTrader. One situation that can arise is that your PC clock is not in sync with your data feed. When this happens the trade execution may be shown on the chart on a bar where it seems like the fill is not feasible.

Example: Data feed bar is currently timestamped as 4:26PM. Local PC clock is 4:21PM.

When a market order is placed under the above situation, the **trade execution** will occur at 4:26PM prices, but be shown on the chart at 4:21PM.

To prevent these types of issues please ensure your local PC clock is in sync with your data feed. Please reference the Historical & Real-Time Data chart to see if your data provider timestamps their data or if the data is timestamped locally by your PC clock. It is important to maintain a sync between your PC clock and the data feed's timestamping.

Understanding trade executions on charts with tick based intervals

When using a chart with tick based intervals in NinjaTrader it is possible to have several bars with the same timestamp. This usually happens during high volatility times when heavy trading is happening within a very short amount of time. Since there are many bars with the same timestamp, NinjaTrader can only plot the **trade execution** on the first bar with the same timestamp of the execution since the executions are not tied to specific bars, but tied to specific timestamps. This can appear as if the **trade execution** occurred with an invalid fill price, but in reality the execution did occur on a valid price, just on a later bar with the same timestamp.

Example: Many ticks occurred on the 16:35:54 timestamp seen in the x-axis below the chart. **Trade execution** was at price 1058.75 on 16:35:54.

Since the execution occurred on 16:35:54 it is plotted on the first bar with the same timestamp. In this particular case, the first bar was not at the same price as the execution price so it would appear to be filled outside of the bar. Checking bars being plotted later on we find that 1058.75 was a valid price for timestamp 16:35:54 and that this execution was in fact on a valid price.

10.6.20 Break at EOD

Break at EOD (End Of Day)

You can optionally set NinjaTrader to break its bars on each new end of day session, or continue building until completed. This property can impact the way price is analyzed during the end of a trading session, and can therefore affect the way an indicator or strategy is calculated.

With **Break at EOD** enabled, your bars are ensured to have a known starting point of which to give you a consistent, repeatable chart. This is especially important for bar types that are not based on time such as Volume and Tick based charts. However, for other bar types such as Range, Point and Figure, or others which are built around price action, enabling this property may cause bars to complete before their criteria has been satisfied. If you prefer the bar to satisfy to completion before creating a new bar, regardless of the time of day, you will want to ensure you have *disabled* the **Break EOD** property.

Break EOD vs Non-Break EOD

Below you will find a two examples which will compare how Range type bars will be handled at the end of a trading session depending on the Chart's **Break EOD** property.

1. **Break EOD** *enabled* - a new bar was formed during the new trading session before the 6 range bar had completed

2. **Break EOD** *disabled* - a new bar was not formed until the criteria for the 6 point range was satisfied

10.6.21 Order Flow +

Order Flow + Overview

NinjaTrader Order Flow + studies are powerful tools to assist in order flow, volume and market depth analysis. Order Flow + studies are included for all NinjaTrader lifetime license users.

Order Flow + contains the following bar type and indicators

- Order Flow Volumetric Bars
- Order Flow Cumulative Delta
- Order Flow VWAP
- Order Flow Volume Profile
- Order Flow Trade Detector
- Order Flow Market Depth Map

10.6.21.1 Order Flow Volumetric Bars

NinjaTrader Order Flow Volumetric bars provide a detailed 'x-ray' view into each price bar's aggressive buying and selling activity. This technique primarily attempts to answer the question which side was the most aggressive at each price level. This is done by calculating the delta (greek for difference) between buying and selling volume (please see the Delta type property explanation below).

With the delta value known for each price level in the bar, it is then classified per each session for analysis and emphasizes the buying / selling strength unfolding. This is done by a gradient coloring approach shading the value cells in the bar, where the level of sensitivity for the gradient can be set via the Shading sensitivity property. The higher this value is set, the finer the gradient can be applied to various levels of strength - the NinjaTrader default is 20 levels.

This can be thought of as a way of not only saying who 'won / lost' the price level's auction, but also by what margin or strength. This is not a signal in itself per se, but rather a mechanical means to classify the buying vs selling activity at each individual price level and thus offer the trader a more detailed look what happens inside the price bars.

A second comparison of buy sell volumes is the Imbalance detection. Here the price level buying and selling volumes are compared diagonally to understand which side of the market was stronger by exceeding the set Imbalance ratio. For example if the buying volume was 1000 contracts and the selling volume diagonal below was 300 then buying Imbalance was

detected (assuming a default Imbalance ratio of 1.5). This can be helpful especially if multiple Imbalances 'cluster' close together to form support / resistance areas.

NinjaTrader Order Flow Volumetric bars can provide a large degree of details and facilitate displaying the information in a dynamically sized way, as the text is re-sized as your horizontally or vertically adjust the chart's scale range.

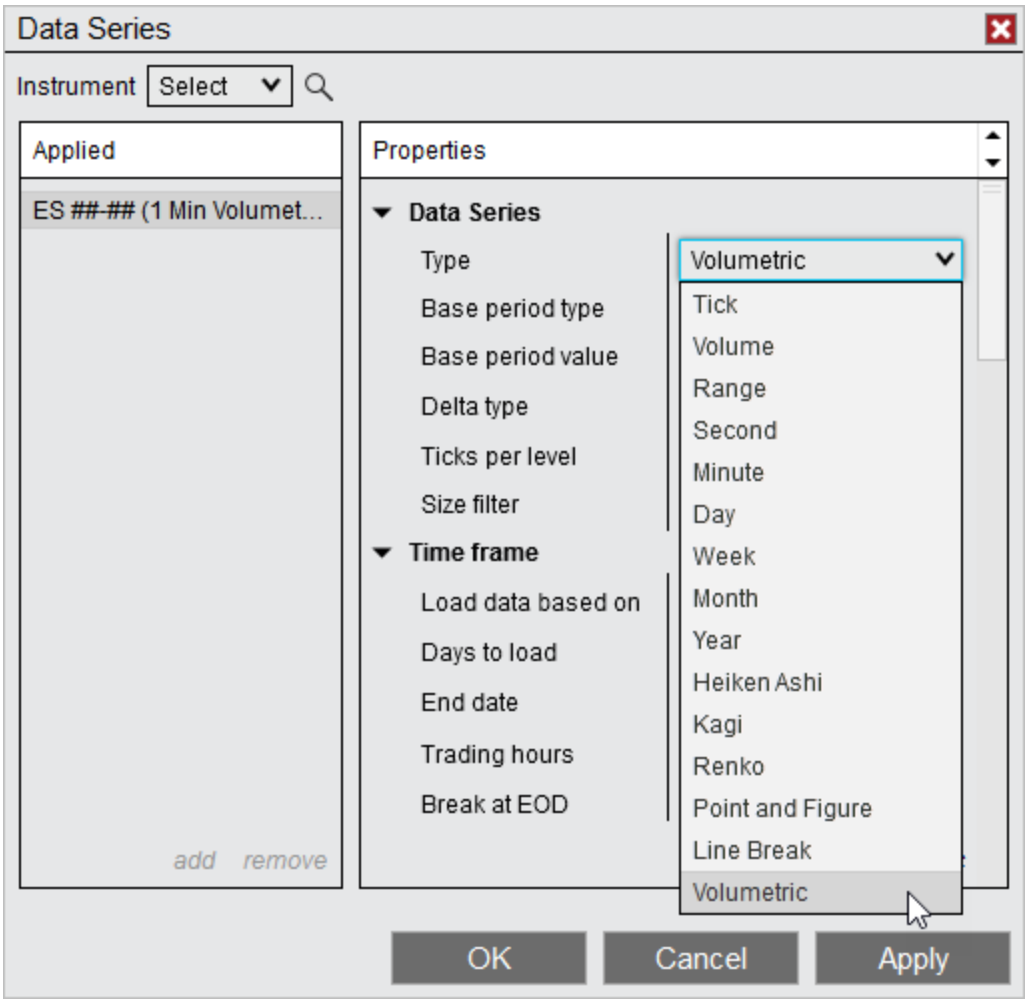
Critical: To perform the delta buy / sell aggressor classification (DeltaType BidAsk), historical bid / ask tick data access by your provider is needed. To see which data providers can offer which type of data in NinjaTrader, please review this table. If your provider could only support 'last' historical tick data, then the classification could still be made using DeltaType UpDownTick mode.

Forex spot data charting is traditionally driven from the bid side only, since no 'true' last exists. Provided values would not represent a centralized auction based market as with stocks or futures - as such Order Flow Volumetric Bars would not be supported on Forex spot data.

Order Flow Volumetric Overview

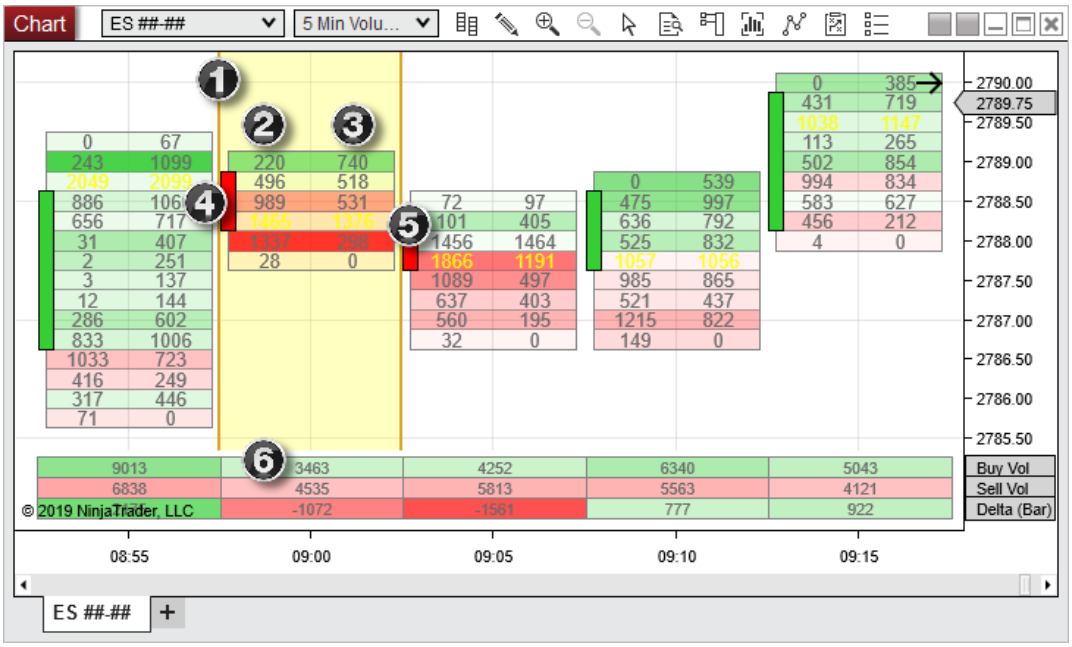
Order Flow Volumetric Overview:

You can apply the Order Flow Volumetric bars within a Chart Data Series window under **Type**.

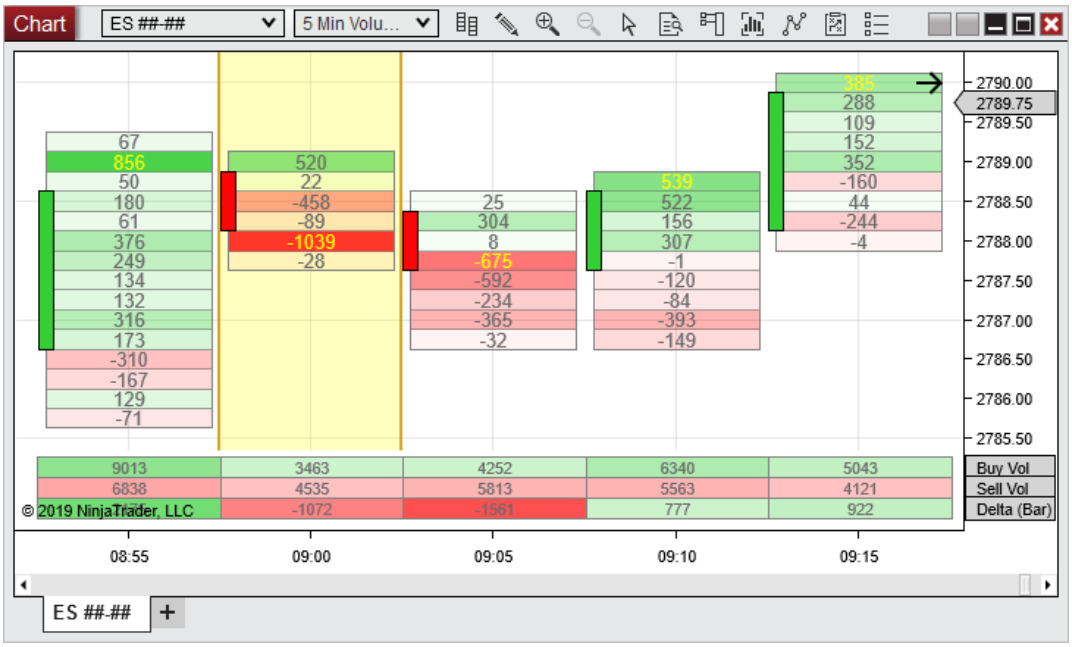


Below we show a 5 minute Order Flow Volumetric BidAsk style chart of the popular E-Mini S&P 500 contract. An exemplary bar in yellow is annotated to show the different components you will work with on an Order Flow Volumetric chart.

1. Order Flow Volumetric bar
2. Sell Volume per each price level seen in the bar
3. Buy Volume per each price level seen in the bar
4. Open / Close bar
5. Maximum highlight in the bar - this shows the price(s) with the highest volume or delta in the bar
6. Bar Statistics panel (only 3 out of possible 10 values activated here)



On the next image, we see an excerpt of the same chart, however now the chart style type is changed to Delta. The bar highlighted yellow corresponds to the annotated bar above - this style, instead of the individual buy / sell volumes, show the combined delta value for each price level.
If positive the level was seeing buying strength, negative if selling strength.



Now let's take a look how the actual values we see in the bar are calculated from the market data, as an example take the first level of our Volumetric BidAsk bar in the first screen-shot (numbers right below the 2/3 annotated numbers) :

We see a bid or selling volume of **220** and a ask or buying volume of **740**.

Taking the difference (Buy volume - Sell volume) so **740 - 220** we get the delta value of **520** - which we see as first cell value in the 2nd screen-shot showing the Delta type.

The shading taking place in the bars will always be based on this delta value calculated, the chart style just defines what kind of textual data detail will be displayed (the actual Bid Ask Volumes or the Delta value).

Moving on to the lower portion Bar Statistics (6), we see the Buy Vol and Sell Vol for the bar and a delta as well exposed - however, please note that these are summed values for the entire bar.

If we sum all the buy volumes we would get **3463**; summing sell volumes for the bar is **4535**. All the price level deltas summed would equal the delta of the entire bar **-1072**.

Order Flow Volumetric Imbalance charting:

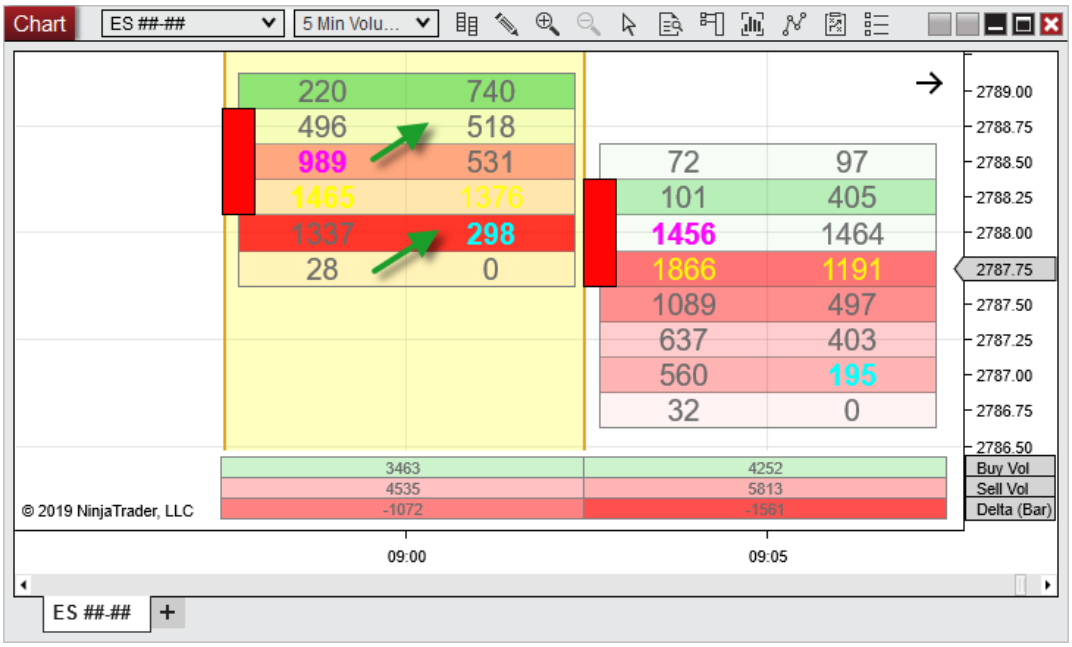
The screen-shot below explains the workings of the Imbalance detection in more detail. You can see the Buy / Sell volumes are compared diagonally here to arrive at the classification if buy or sell imbalance is present.

Let's run through the first calculation for the example bar in yellow:

1. Buy volume of **518** is compared to the diagonal below sell volume of **989**
2. Dividing the buy volume into the sell volume we get a ratio of **989 / 518 = 1.9092...**
3. NinjaTrader by default sets the ratio for Imbalance at 1.5, so this level gets marked with Sell Imbalance (magenta text color per default).
4. As a further condition, a minimum difference between the compared values must be present. This value is defaulted to 10 - which is valid in our example as well.

In the case that both Imbalance and Maximum would trigger for the same cell, the Maximum would override and be displayed (example shown below at annotation 1).

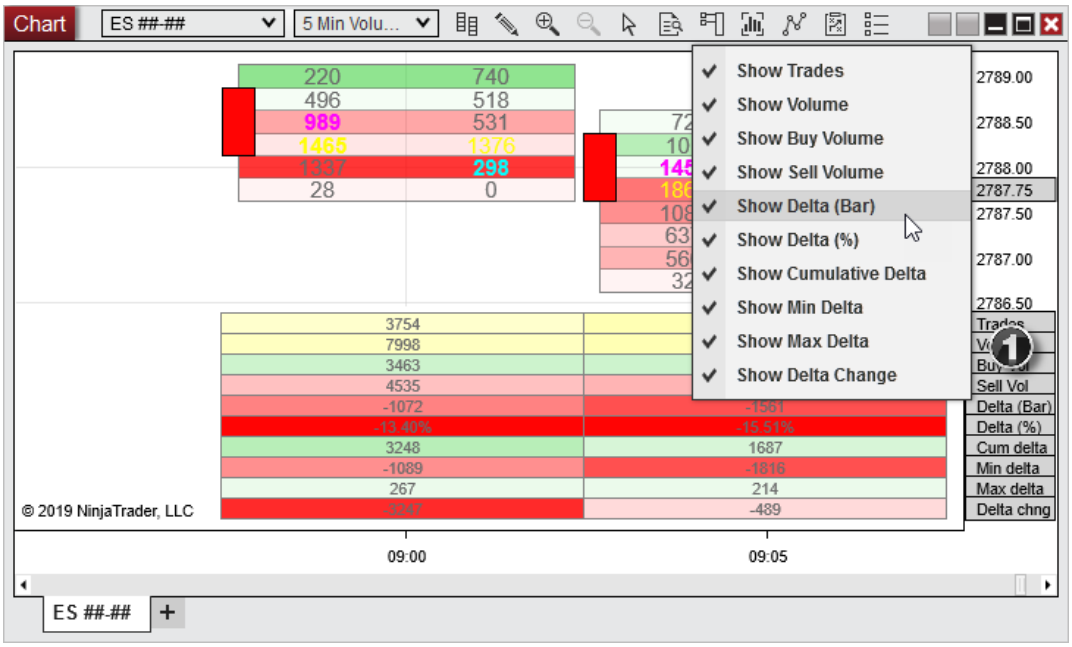
NinjaTrader 8.0.19.0 or newer also offers the option to compare horizontal for Imbalance (Imbalance mode setting).



Order Flow Volumetric Bar Statistics:

The Volumetric Bar statistics show important values for each Volumetric bar in a static grid-like fashion. The same gradient strength shading as for the main Volumetric bars is applied here.

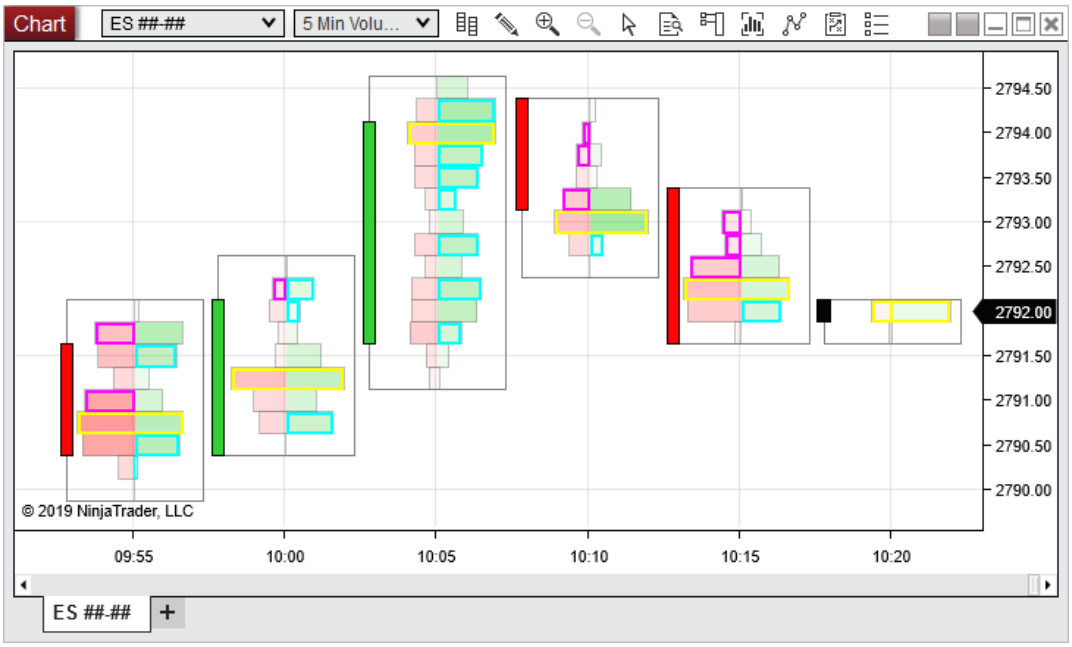
1. Via a right click in the price scale section the individual statistic values could be enabled / disabled 'on the fly'.



Order Flow Volumetric Bar data shown as profile:

Below chart is an example of showing the volumetric bid ask volume bar data as distribution profile, additionally 'hide text' is checked - which means we see the maximum (yellow) as well as imbalance (cyan / magenta) marked via the cell borders coloring.

Showing the data in this fashion can give traders an easier read, as differences between light and high volume price areas becomes visually more striking.

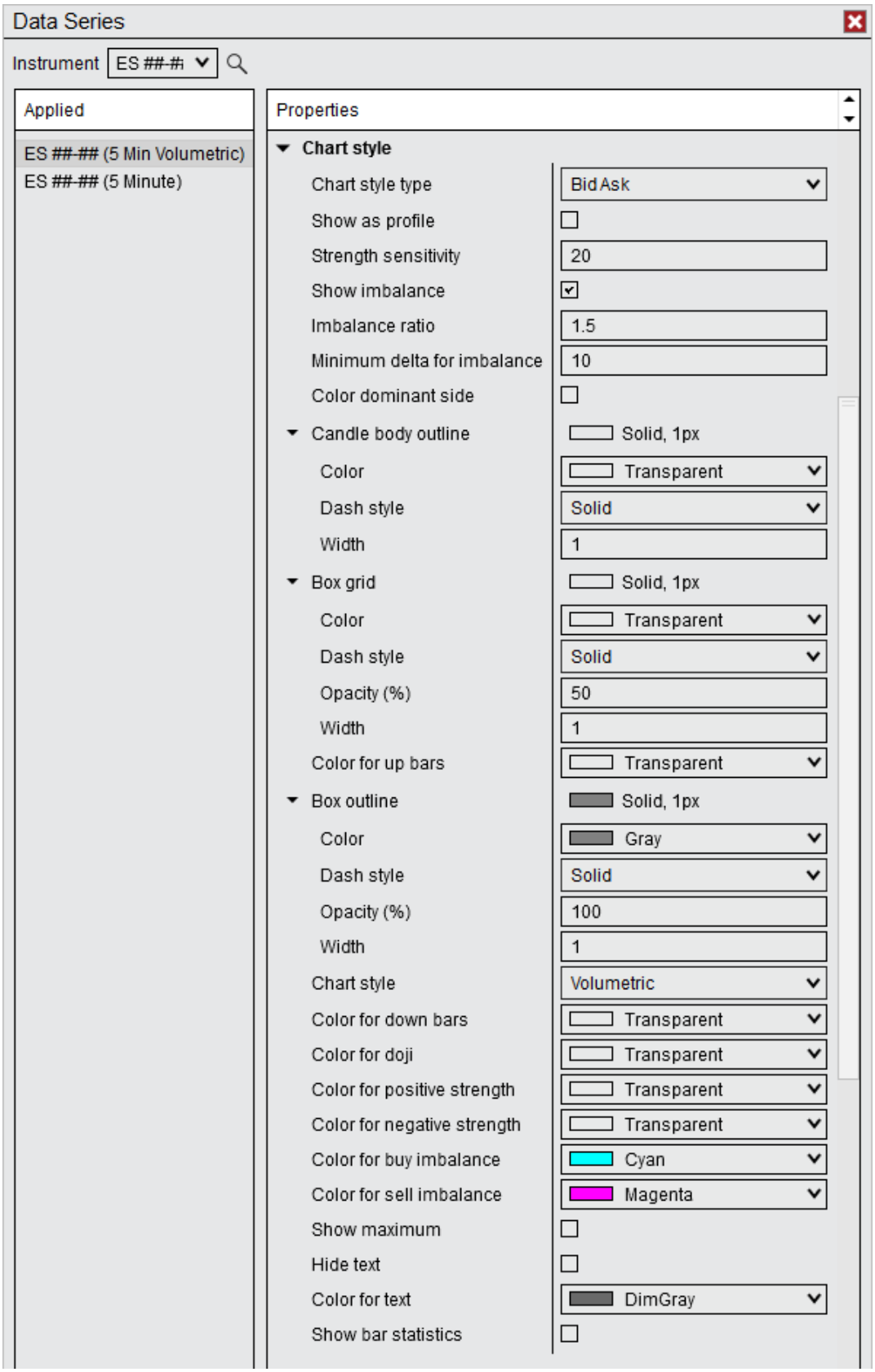


Order Flow Volumetric Imbalance Customization Example

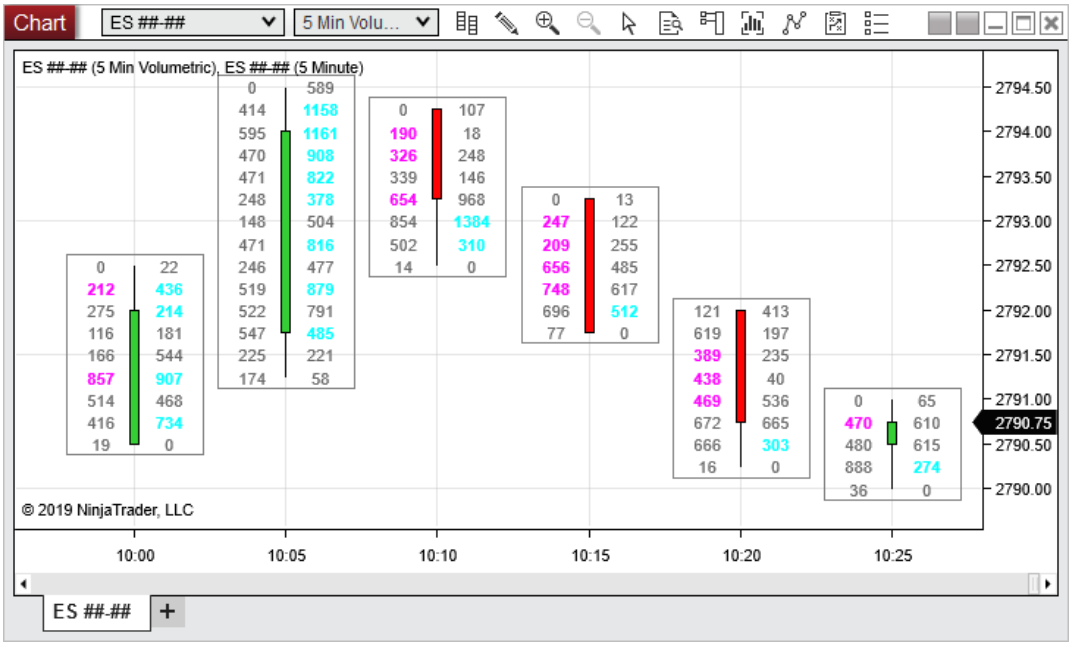
Order Flow Volumetric Imbalance Customization example:

This section presents an example of how NinjaTrader Order Flow Volumetric bars can be highly customized to your trading style. Traders focused on Order Flow Volumetric Imbalances may consider working these charting ideas into their NinjaTrader setup.

The settings we present below could be used as a starting point -



The regular 5 Minute CandleStick chart is brought in to this chart via a second Data Series, so forming a MultiSeries chart with our main 5min Order Flow Volumetric bars. This can advantageous if you prefer to plot the regular bar / candlestick portion in the middle of the bar between the Buy/Sell volume columns. In NinjaTrader 8.0.19.0 or newer, this can now be also accomplished without a second DataSeries by enabling the Center Open/Close bar plotting option.



Order Flow Volumetric Bars parameters

Data series:

Base period type	Sets the base period type the Volumetric bars should be calculated on, possible values include: Tick, Volume, Second, Minute, Day,

	Week, Month, Year, Range
Base period value	Period of your chosen Base period type for Volumetric bars (i.e. **3** Minute, **450** Tick)
Delta type	Sets how the delta is calculated for buy / sell aggressor classification. Possible values are: BidAsk or UpDownTick. BidAsk - Accumulates the volume of orders filled at the bid or less vs ask or more. Orders filled at the ask or more price are considered buying pressure. Orders filled at the bid or less price are considered selling pressure. If the current tick price is between ask and bid, the volume will be recorded to the same pressure as the previous tick. UpDownTick - Accumulates the volume of up ticks vs down ticks. Up ticks are considered buying pressure. Down ticks are considered selling pressure. If the current tick price is the same as the previous tick price, the volume will be recorded to the same pressure as the previous tick.
Ticks per level	Sets the level of aggregation for individual price levels, i.e. if price levels should be merged together, default 1 – so each price level delta result is seen individually inside the price bars (Please note that with a higher Ticks per level set, there could be Volumetric remainder cells as a result that are actually smaller than your set Ticks per level, as not all bar ranges could be evenly divisible by the Ticks per level value)

Size filter	Default 0, could be set higher to limit seeing the delta only for trades higher than the size filter setting, for example tracking larger trades only - keep in mind this could be potentially set too high for your chart, so Volumetric bars value cells could show 0 volume, so it never met this criteria then to be considered in the analysis.
Tick replay	Check to allow indicators or strategies to access Tick replay data, the main Volumetric bars though will always be built off a 1 tick series.

Chart style:

Chart style type	Sets the chart style type used, possible values are : BidAsk or Delta BidAsk - allows you seeing the individual buy / sell volume cells inside the Volumetric bar, buy (ask) volume is shown on the right side, sell (bid) volume shown on the right side. Additionally this style can highlight the maximum and imbalance conditions. Delta - allows to see the single value total net delta per each price cell, maximum can be shown as well (no imbalance option for this style)
Center open close bar	When checked, allows centering of the Open / Close bar on the BidAsk Chart style type
Show as profile	Enables to show the Volumetric bar data as profile / distribution
Show volume	Enables to show the volume distribution to the right of the Volumetric bar.

Strength sensitivity	Sets how many gradient levels should be calculated to provide the buy / sell strength shading for the Volumetric bars and Bar Statistics, the default setting is 20. This is reset at every session break to ensure the gradient strength classifications easily comparable across various days market action.
Size display filter	Allows only displaying Bid / Ask or delta numbers higher than the threshold. Default 0 and a visual only setting.
Show imbalance	Enables the display of Imbalance coloring Only applicable for the Volumetric BidAsk chart style type With 'Hide text' checked will display imbalances by coloring the respective imbalance cell borders
Imbalance ratio	Sets the ratio used for comparing the buy / sell volumes for accessing if Imbalance is present, default 1.5 Only applicable for the Volumetric BidAsk chart style type
Imbalance mode	Sets the comparison mode for Imbalance : Diagonal or Horizontal
Minimum delta for imbalance	Sets the minimum delta to be seen diagonally across compared buy / sell volume columns for displaying Imbalance, default value is 10. Only applicable for the Volumetric BidAsk chart style type
Color dominant	Enables to display the dominant ('winning') strength side only in the Volumetric strength

side	display Only applicable for the Volumetric BidAsk chart style type
Box outline	Sets options for the display of the outline of the Volumetric boxes
Box grid	Sets options for the display of the inner grid of the Volumetric bars
Candle body outline	Sets options for the display of the body outline of the Open/Close bar
Color for up bars	Sets the up brush color used for Open/Close bar
Color for down bars	Sets the down brush color used for Open/Close bar
Color for doji	Sets the doji brush color used for Open/Close bar
Color for positive strength	Sets the positive brush color used for positive strength gradients
Color for negative strength	Sets the negative brush color used for negative strength gradients
Color for buy imbalance	Sets the brush color used for buy imbalance Only applicable for the Volumetric BidAsk chart style type
Color for sell imbalance	Sets the brush color used for sell imbalance Only applicable for the Volumetric BidAsk chart style type

<table>
<tr><td>Show maximum</td><td>Enables the display of the maximum value for the Volumetric bars, if identical values would be seen across cells, then all cells sharing the maximum would be highlighted.

BidAsk style: If Color dominant side is checked, it will highlight both the Buy and Sell price levels with the highest volume, else it will highlight the price level with the highest combined Buy / Sell volume (Total volume for the price level)

Delta stye: highest absolute delta level would be highlighted

With 'Hide text' checked will display the maximum by coloring the respective maximum cell borders</td></tr>
<tr><td>Color for maximum</td><td>Sets the color used for maximum display</td></tr>
<tr><td>Hide text</td><td>Enables to hide the Volumetric cell values text display</td></tr>
<tr><td>Color for text</td><td>Sets the text color used for the displaying the Volumetric cell values</td></tr>
<tr><td>Show bar statistics</td><td>Enables the display of the Volumetric bar statistics in the lower portion (static location) of the chart. You can checkmark each desired statistic, possible statistics are -

<table><tr><td>Trades</td><td>The total number of trades in the bar</td></tr><tr><td>Volume</td><td>The volume for the bar</td></tr></table></td></tr>
</table>

Buy vol	The Buy vol for the bar
Sell vol	The Sell vol for the bar
Delta (bar)	The total delta for the bar (all individual price level delta summed)
Delta (%)	The bar delta expressed as percentage of volume for the bar
Cumulative delta	The bar delta value cumulated throughout the session
Min delta	The minimum delta seen in the bar (intrabar). This could be positive as well, i.e. a strong up bar with no selling pressure
Max delta	The maximum delta seen in the bar (intrabar). This could be negative as well, i.e. a strong down bar with no buying pressure
Delta change	The change in delta from the previous bar's delta value
Delta SH	The delta since last time price touched the high of the bar, usually negative
Delta SL	The delta since last time price touched the low of the bar,

	usually positive. The statistics can be toggled as well 'on the fly' from the chart's right scale by clicking on the statistic labels.
Color for base	Sets the brush color used for base bar statistics strength gradients (Trades, Volume)
Statistics grid	Sets options for the display of the bar statistics grid of the Volumetric bars

Order Flow Volumetric Values NinjaScript access

For information on how to access the Order Flow Volumetric Bars and Bar Statistic values in NinjaScript, please see the Order Flow Volumetric Bars page in the NinjaScript section of the Help Guide.

10.6.21.2 Order Flow Cumulative Delta

Description

An indicator that accumulates the volume of orders filled at bid and ask prices or up and down ticks throughout the session or bar and compares them to determine buy/sell pressure.

Order Flow Cumulative Delta Overview

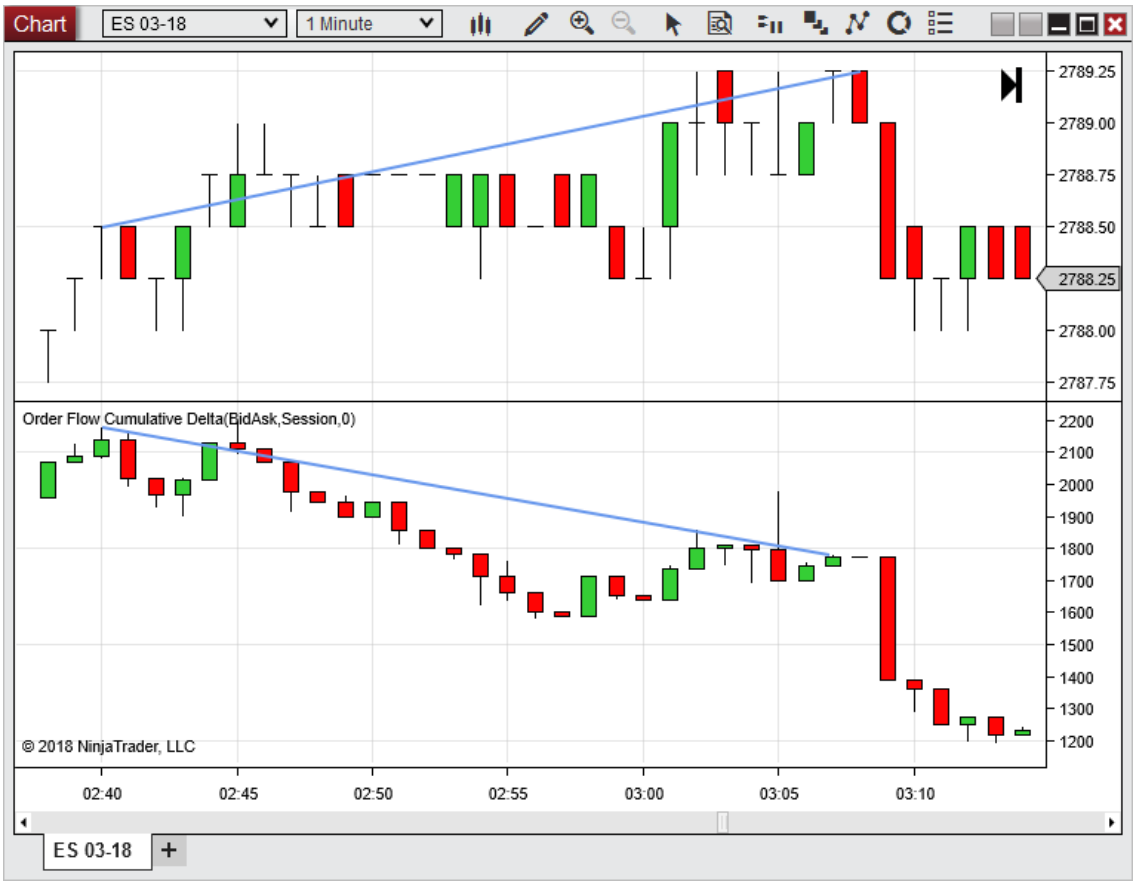
Display

There are 2 view of the Order Flow Cumulative Delta, both of which can be calculated on a **Delta type** of **Bid Ask** or **Up Down Tick**.

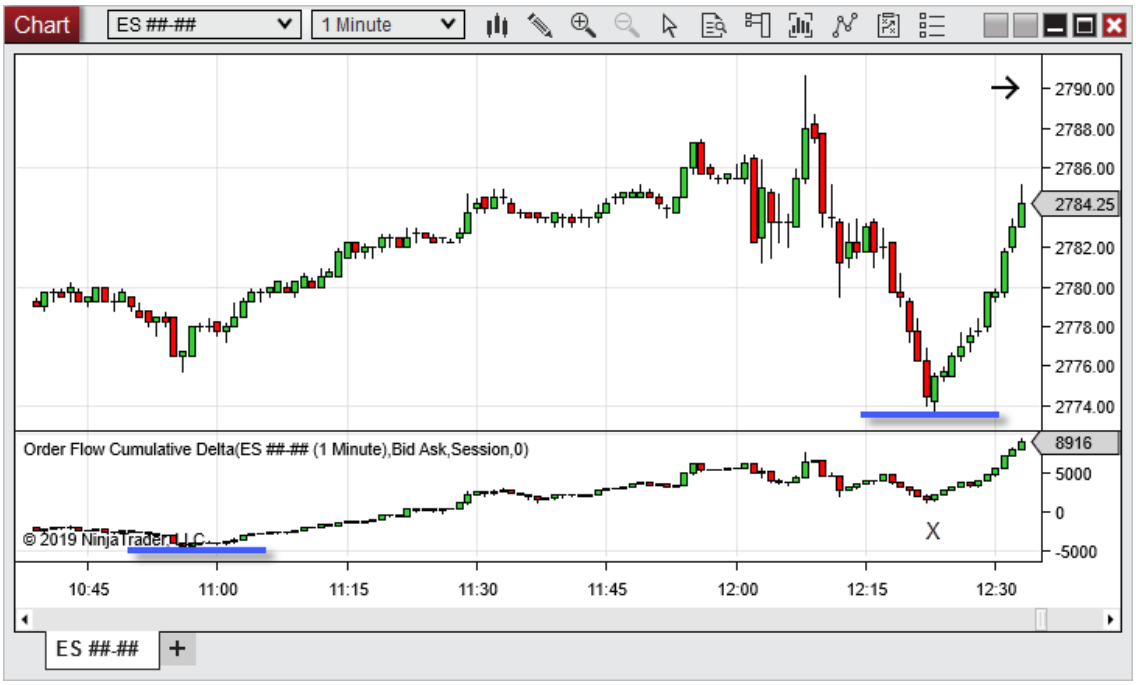
1) **Session**, in which the delta will accumulate over the session. Each close price will be carried over to the open of the next bar.
2) **Bar**, in which the delta will accumulate over a bar. Each bars open will start over and have an open of zero.

Using the Order Flow Cumulative Delta

A common function of the Order Flow Cumulative Delta is to confirm trends. In the image below we can see that the price is bullish. However, this was not confirmed by the Order Flow Cumulative Delta which was bearish. The market then became bearish.



Another common function is to confirm new daily highs or lows. In this image we can see that the price reached a new low for the day, but this was not confirmed by the Order Flow Cumulative Delta. The bears could not hold momentum and then the market became bullish.



Notes:
1. To plot historically with **Delta type** **Bid Ask** requires historical bid ask stamped tick data. See the Data by Provider section for information on what providers offer historical bid/ask stamped tick data.
2. Volume bars could split a single tick into multiple bars for both historical and real-time data. As such Order Flow Cumulative Delta could run into tracking limitations on the internally added 1 tick series, as all the volume would be processed on the first 1 tick bar / delta calculation.

Critical: Forex instruments are not supported with the **Delta type** **Bid Ask**. Since forex has no last price, this mode will result in the cumulative delta bars moving in one direction. If using the **Delta Type** **Up Down Tick** mode, this will be calculated on data series **Price based on** property. **Last** data is bid data for forex instruments.

Order Flow Cumulative Delta Parameters

Delta type	

	<table><tr><td>Bid Ask</td><td>Accumulates the volume of orders filled at the bid or less vs ask or more. Orders filled at the ask or more price are considered buying pressure. Orders filled at the bid or less price are considered selling pressure. If the current tick price is between ask and bid, the volume will be recorded to the same pressure as the previous tick.</td></tr><tr><td>Up Down Tick</td><td>Accumulates the volume of up ticks vs down ticks. Up ticks are considered buying pressure. Down ticks are considered selling pressure. If the current tick price is the same as the previous tick price, the volume will be recorded to the same pressure as the previous tick.</td></tr></table>
Period	<table><tr><td>Session</td><td>Accumulates volume per session</td></tr><tr><td>Bar</td><td>Accumulates volume per bar</td></tr></table>
Size filter	Input to exclude volume less than the selected value
Candle body outline	Color and line settings for the candle body outline

Candle wick	Color and line settings for the candle wick
Color for down bars	Color for bars that have a close less than the open
Color for up bars	Color for bars that have an open greater than the open

Order Flow Cumulative Delta Values NinjaScript access

For information on how to access the Order Flow Cumulative Delta values in NinjaScript, please see the Order Flow Cumulative Delta page in the NinjaScript section of the Help Guide.

10.6.21.3 Order Flow VWAP

Description

Volume Weighted Average Price. A total of the dollars traded for every transaction (price multiplied by number of shares traded) and then divided by the total shares traded for the day. Also included are standard deviation bands.

Order Flow VWAP Overview

Display

The VWAP line is the green and red line. The VWAP by default is LimeGreen when below the price and Red when above the price.
The blue bands are the standard deviations of the VWAP. By default the closest one is a 1X multiplier, followed by a 2X multiplier, and a 3X multiplier.

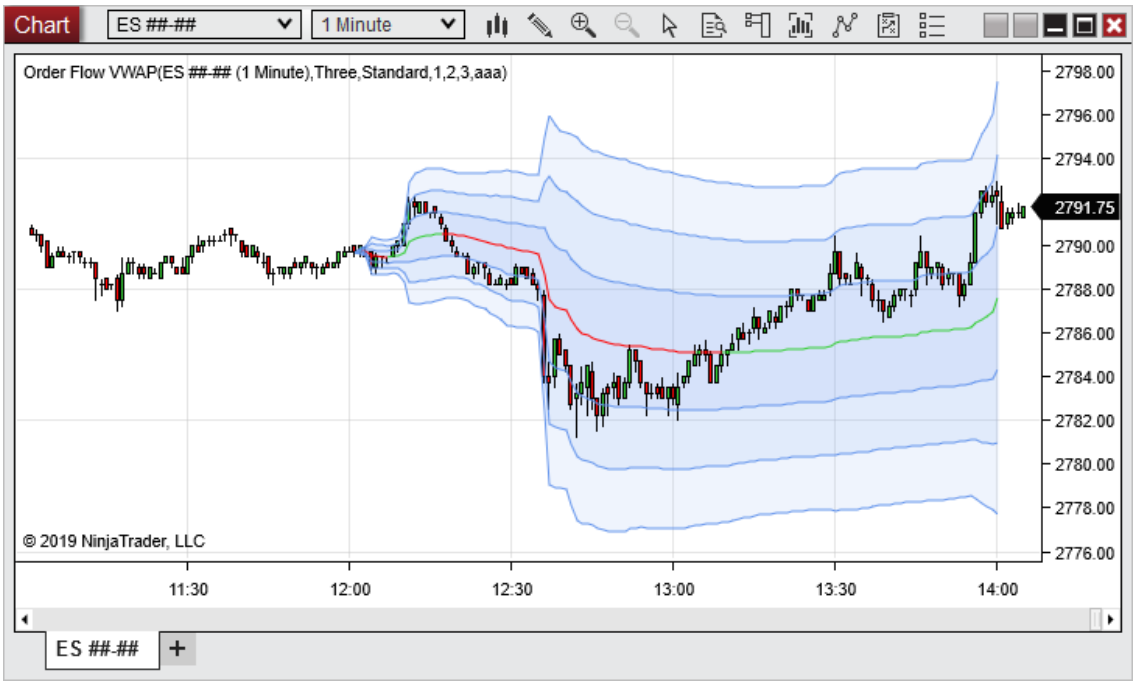
Using the Order Flow VWAP

When using the Order Flow VWAP it is typical that users look for purchase signals when the price is below the VWAP. This would indicate that the purchase would be at a lower price than what is the average purchase per share for traders during the VWAP period. The price being below the VWAP can also result in pressure for the market to move back towards the VWAP. The opposite would be true for when the VWAP is above the price.

The VWAP and standard deviations are often used to indicate potential levels of support and resistance.

Since the Order Flow VWAP is a cumulative indicator, the longer it runs for the more the VWAP can lag. Using the **Trading hours** parameter you can adjust during what

times the indicator will run for. These settings work in combination with the selected Data Series **Trading hours** and will not calculate outside of that.



Order Flow VWAP Parameters

Reset Interval	Session	Resets VWAP calculations every session
	Week	Resets VWAP calculations weekly*
	Month	Resets VWAP calculations monthly*
Resolution	Standard	Runs indicator on selected Input series

	Tick: Runs indicator on a 1 tick time frame of the selected Input series (*most accurate and most resource intensive, therefore not available for Reset Intervals of Week and Month*)
Trading hours	Selection for what session to calculate on if VWAP is in Reset Interval Session mode
Std Dev bands	Selection for how many standard deviation bands to display
Std Dev 1 multiplier	Input for what multiplier to use for standard deviation 1
Std Dev 2 multiplier	Input for what multiplier to use for standard deviation 2
Std Dev 3 multiplier	Input for what multiplier to use for standard deviation 3
Color for above price	Color for when the VWAP line is above the price
Color for below price	Color for when the VWAP line is below the price
Color for band area	Fill in color between the standard deviation lines
Base opacity for band area	Opacity for the most outer standard deviation band. The opacity for each inner band is increased by 10%

Order Flow VWAP Values NinjaScript access

For information on how to access the Order Flow VWAP values in NinjaScript, please see the Order Flow VWAP page in the NinjaScript section of the Help Guide.

10.6.21.4 Order Flow Volume Profile

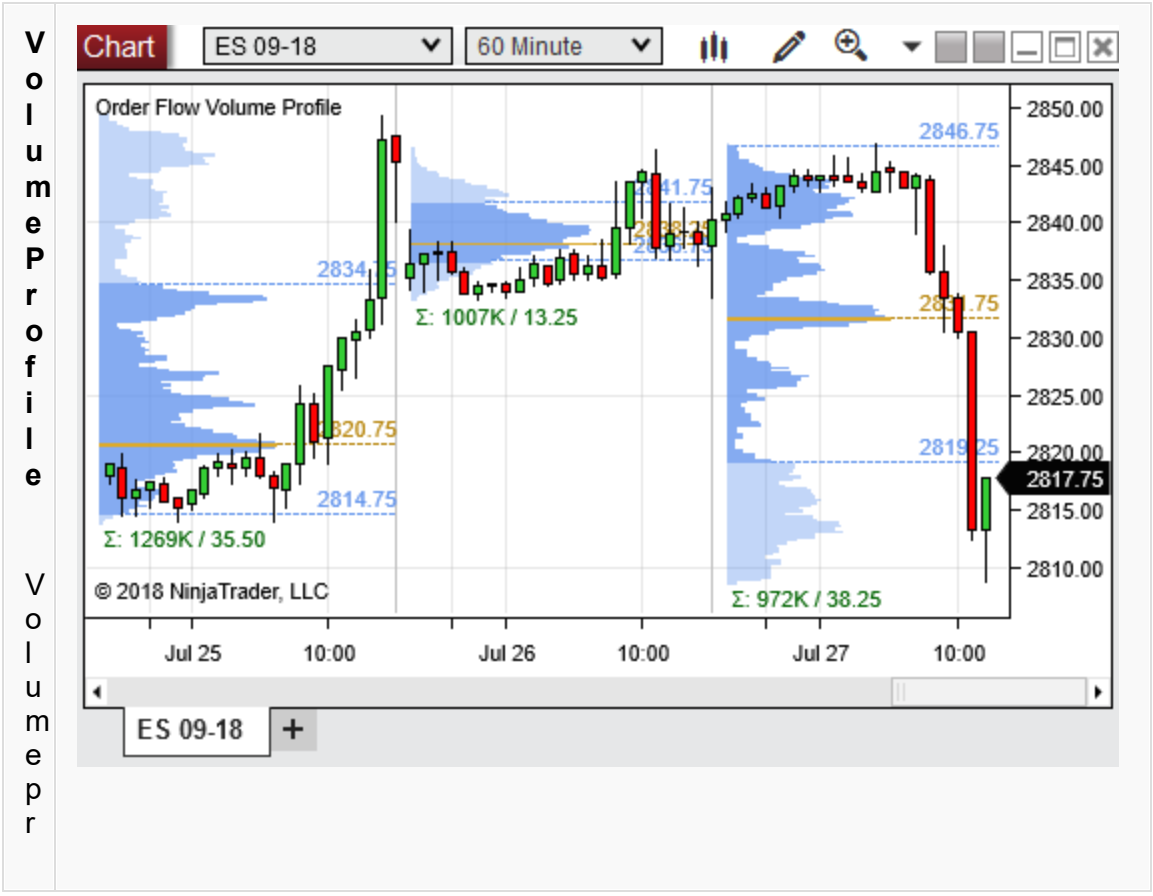
Description

Order Flow Volume profile tools are available as both an Indicator and Drawing Tool and along with the suite of the 'Order Flow +' tools only available to NinjaTrader lifetime license holders. The indicator is used to plot singular static profiles containing a certain defined range of data or repeating profiles on a per bar or per session basis. The drawing tool allows you to easily define both the start and end point to create a custom profile for any bar range on a chart.

Order Flow Volume Profile Concepts

Profile Types

Order Flow Volume Profile has many settings which can be customized to achieve several types of profiles. There are 3 modes for selecting what you want to use to generate the profile.



ofilesdisplaylasttradedvolumedatausingyourselle	

```
ctedsourcedata' resolution' issuedtobuildtheprof
```

ile.

PriceProfile

PriceprofilesdisplaysaI

e t t e r o r b l o c k i f u n d e r l y i n g t r a d e s i n s i d e o f a s p	

ecificti mewindow. Thesetimewindowsare30min

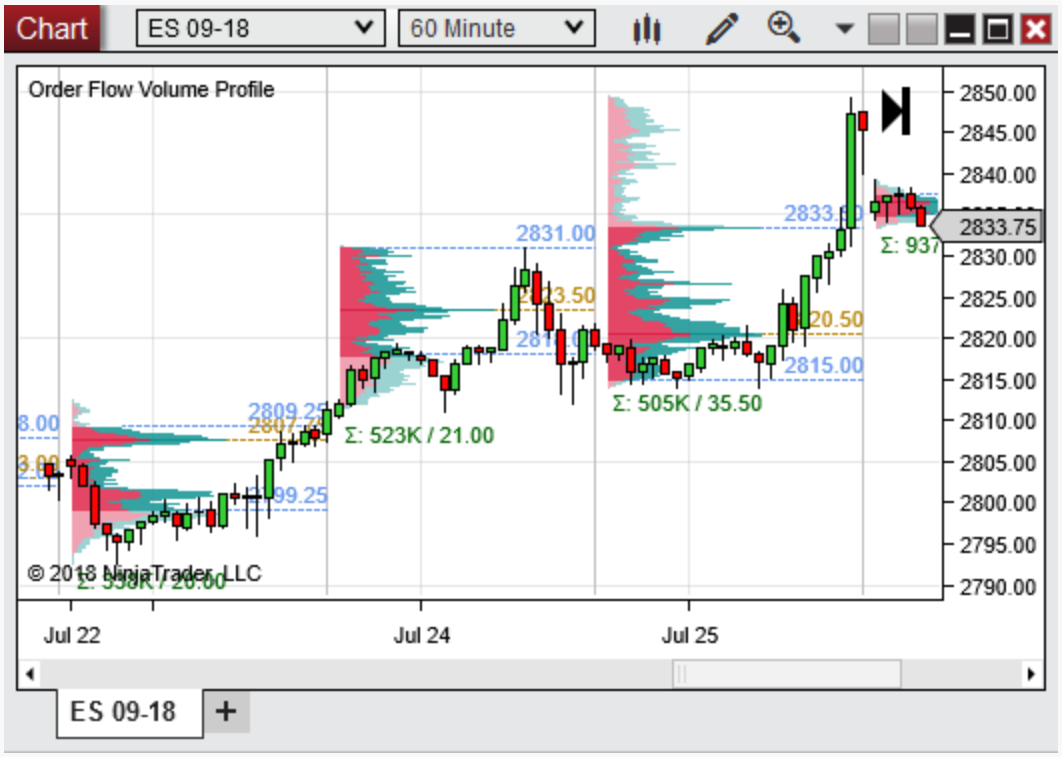
u t e i n l e n g t h . I f a s i n g l e t i c k o c c u r s i n s i d e t h a t	

timewindowatapriceallettterorblockwillbeplo

tted. Inlet terdisplaymode the 'A' letterstart

sat8:00CST.

TickProfile Tickprofilesdisplay



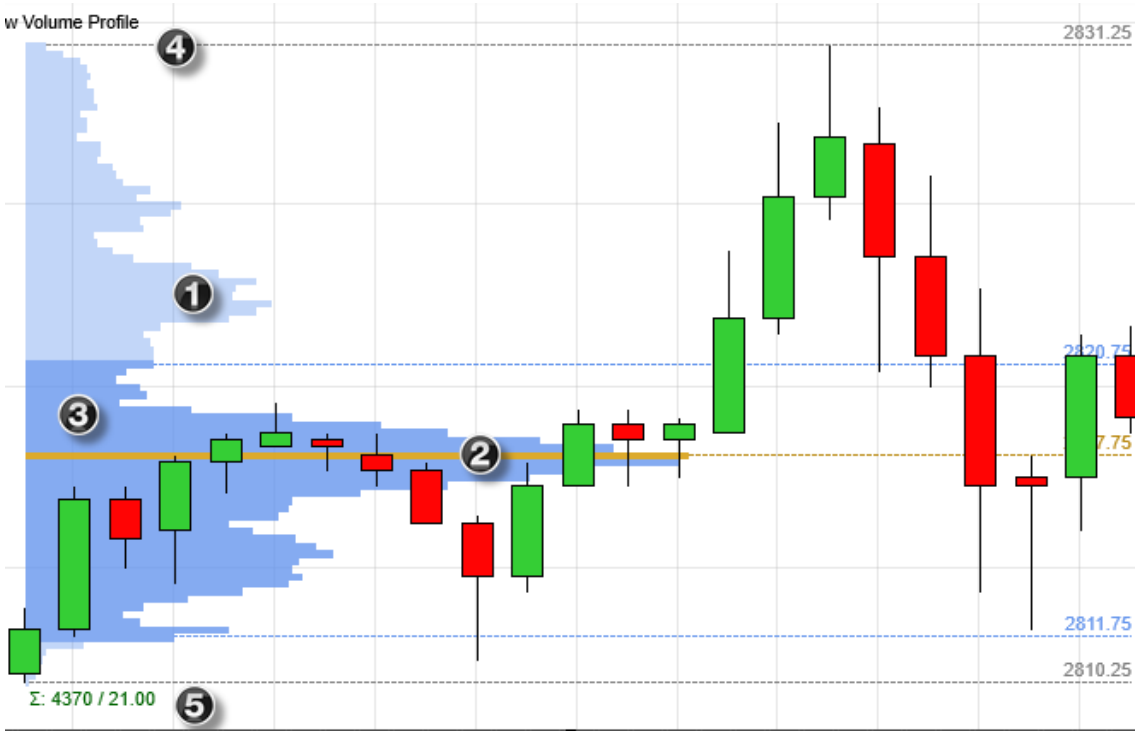
thecountof'ticks'ortradesthatoccuratataspecificp	

rice. This produces a quantity agnostic profile which ch

displaypuretradingactivityirregardlessofvolume

.

Profile Components

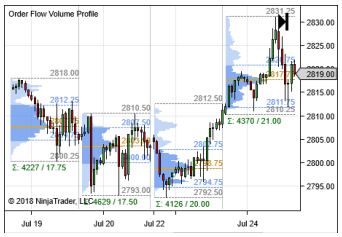
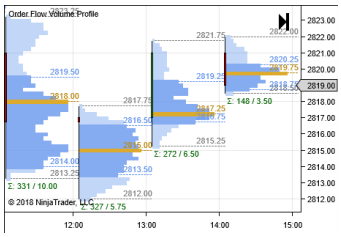
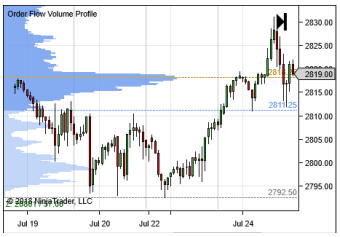


1. **Profile**: Overlaid behind the bars each bar represents the profile per price. Multiple price levels can be aggregated using 'Ticks per level'.
2. **POC**: The point of control (POC) is the single largest data point in the profile.
3. **Value area**: The range where 68% (configurable) of the volume traded, can be referenced as 'VA' and has opacity settings to visually separate this area from the rest of the profile.
4. **Range**: The highest price and lowest price of the profile.
5. **Profile Summary**: The total volume and range of a volume profile. Range metric is configurable.

Profile Periods

There are 3 methods to select the 'period' you which to generate a profile for, each one renders slightly differently. It is not uncommon to use multiple profile types on the same chart as each profile gives you a different view on the underlying data.

Session	Bars	Composite

		
This is a repeating profile created for each session defined. The profile by default is displayed below the bars and across all bars that make up the session. You can group multiple sessions by defining the property 'Sessions' greater then 1 (count is run from left to right, so forward).	When Bars is selected, one profile per bar is rendered. Space is added between the bars so that the profile is readable. This space is defined in the 'Visual' section below 'Profile width (px)' which is a defaulted to 100. You can group multiple bars by defining the property 'Bars' greater then 1.	A composite profile is a single profile drawn as an overlay on the entire chart that is composed of data defined by the 'Compose by' sub property that is exposed when this mode is selected.

Display Modes

Multiple display modes can be used. By default there are 5 render modes available which is changed with the 'Display mode' property on the indicator.

Standard

Singled

efinedcolorprofilerenderingdisplayvolumeb

y p r i c e i n f o r m a t i o n . T h e c o l o r c a n b e s e l e c t e d b y	

modifying the 'Color for profile' property. The

```
rearetwoproopertiestodefinethеopacity.' Pro
```

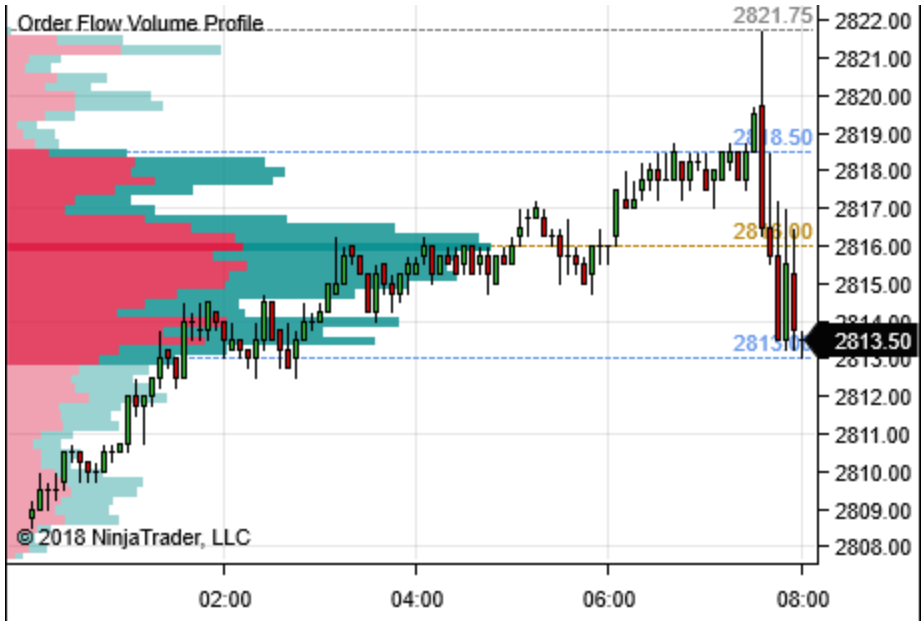
```
fileopacity' sets the opacity above and below this
```

```
evalueareaandd'Valueareaopacity'setstheopa
```

```
city used when drawing the area in the value area
```

.

Buy sell



The volume bar now splits by buy volu

meandsellvolumeasclassifiedpertheDeltatyp	

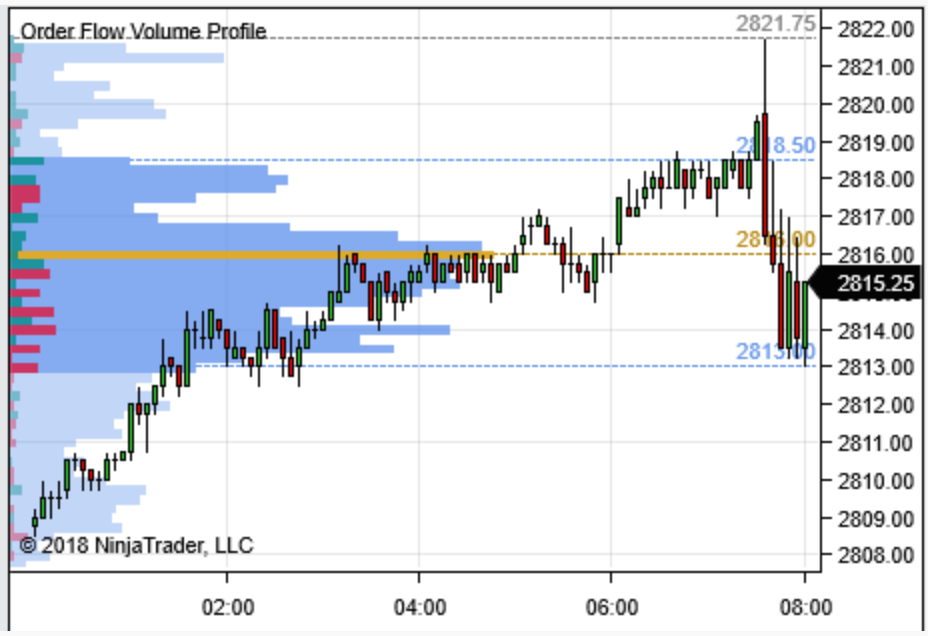
e
p
r
o
p
e
r
t
y
.

N
o
t
e
:
T
h
i
s
m
o
d
e
r
e
q
u
i
r
e
s
'
T
i
c
k
'
r
e

solutionondata.

Delta .

Displaysthestandardv



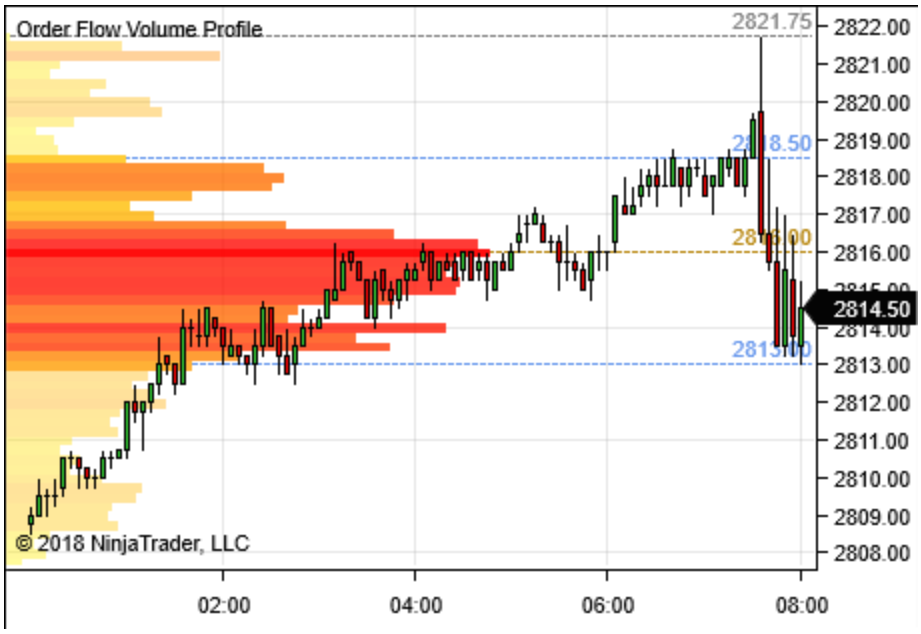
volume profile and overlays buy/sell data delta a a	

s c l a s s i f i e d p e r t h e D e l t a t y p e p r o p e r t y . T h i s d i	

splaymodehighlightsimbalancesinbuy/sellvo	

lume.

Note: This mode requires 'Tick' resolut

iondata	
Heat Displays aggradient in the profil	

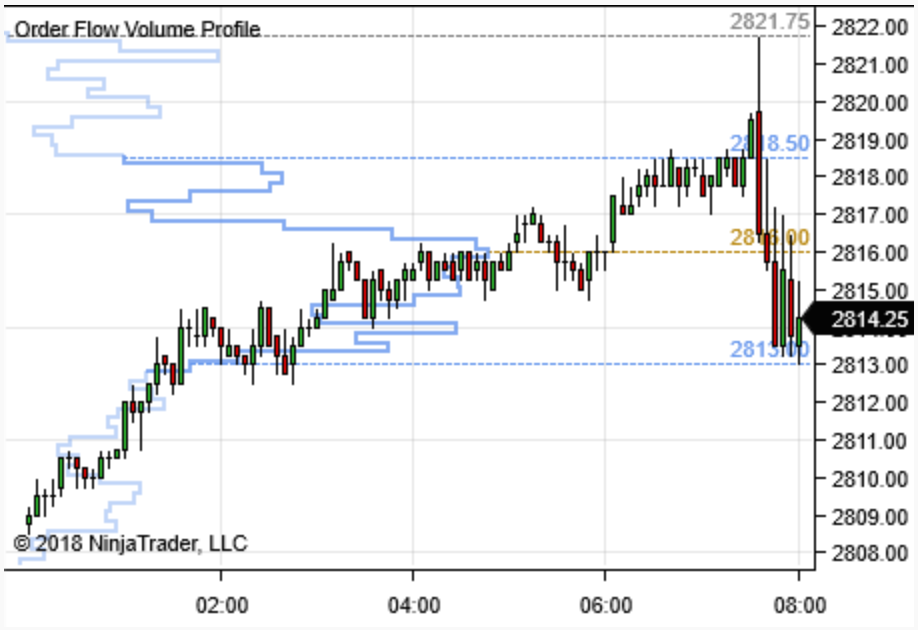
ebasedonthehighestvolumeandthelowestvolum

e
i
n
t
h
e
p
r
o
f
i
l
e
.
H
i
g
h
v
o
l
u
m
e
a
r
e
a
s
a
r
e
e
m
p
h
a
s
i
z
e

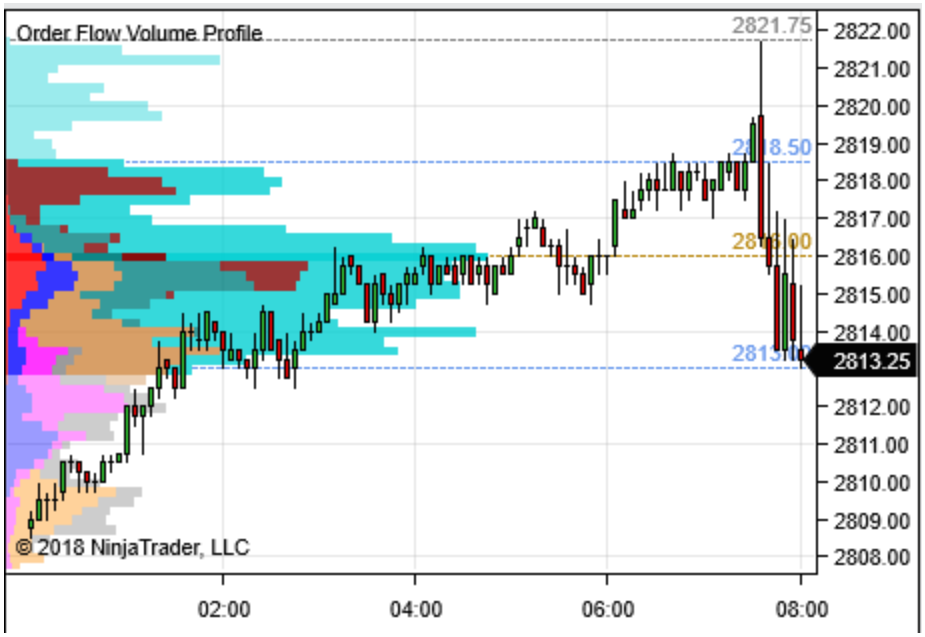
d i n t h i s m o d e .

O u t l i n e

D i s p l a y s t h e v o l u m e p r



o f i l e w i t h o n l y t h e o u t l i n e o f t h e p r o f i l e b e i n g p	

lotted.	
Timecolor Every 30 minutes a new colo	

risplottedanddisplayedinthepro file.The pro

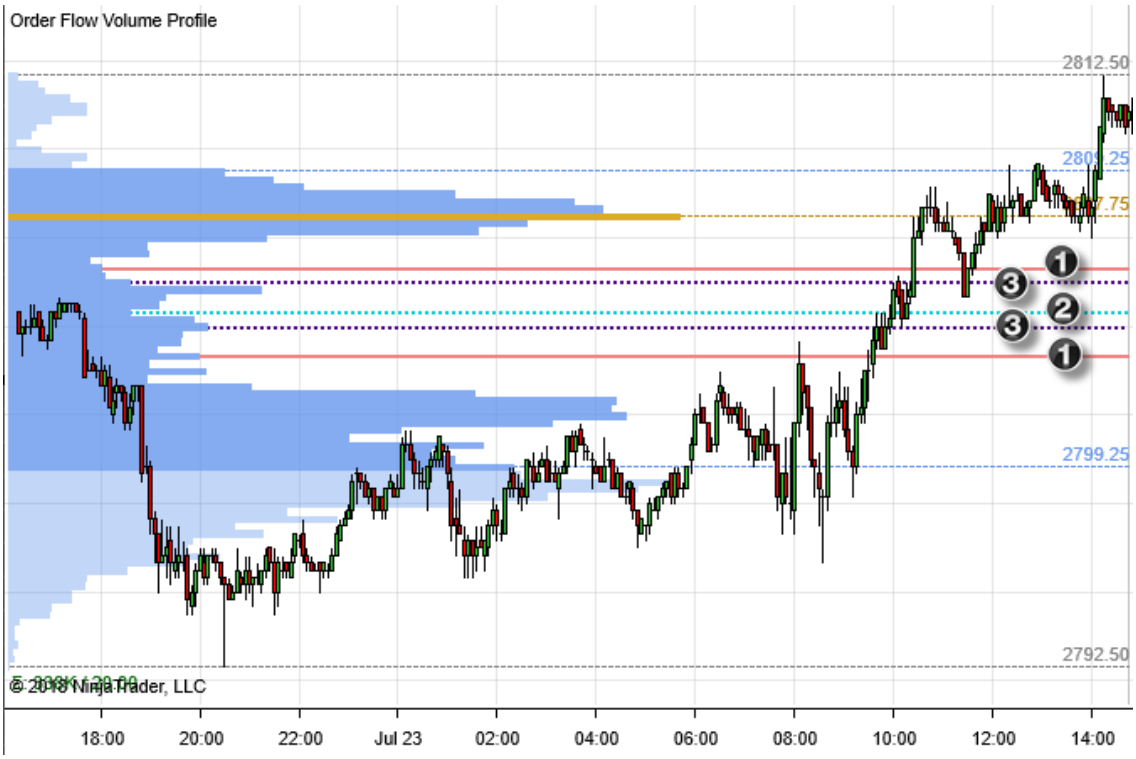
filecolorisdefinedinthepropertiies'Times's	

ection. You can see what time of day the major ity o

f t h e v o l u m e o c c u r r e d i n t h e m o d e .

Initial Balance

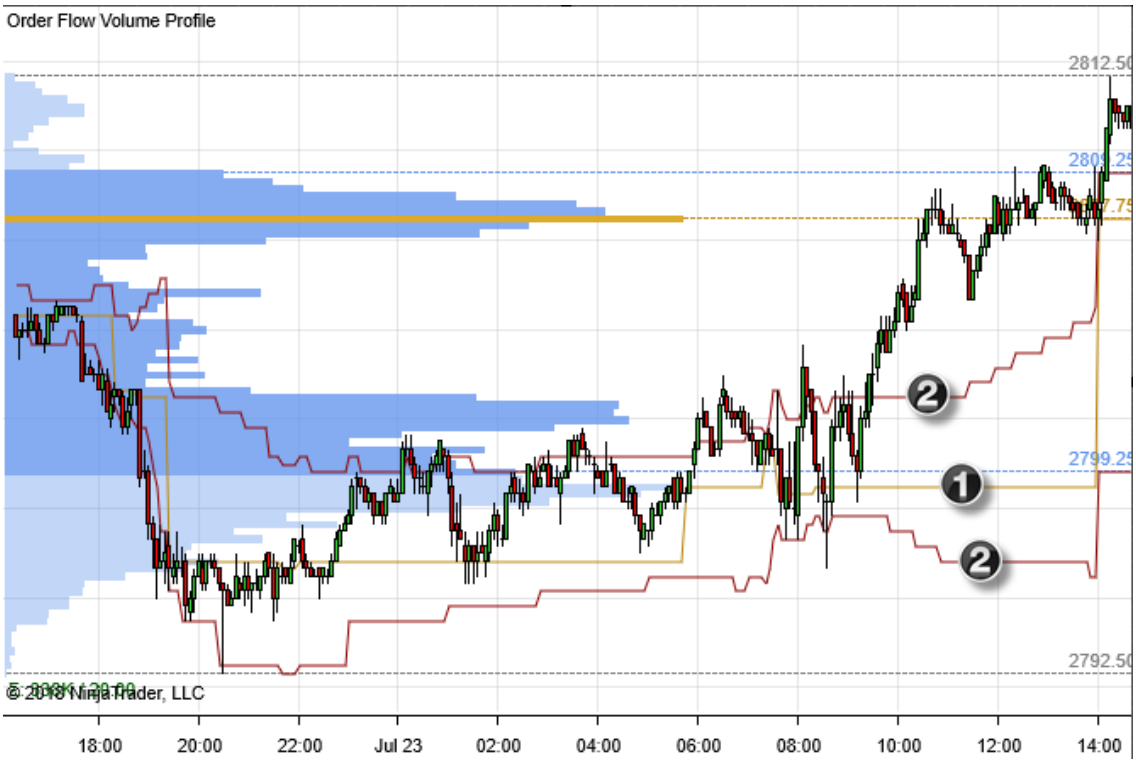
The initial balance is an optional feature which will enable plots to show you what the current range, POC, and Value area a set number of minutes into the session. A typical use case is so that you can compare what the POC and VA was early on in the profile and make a comparison to where it ended up. The initial balance range needs to have the property 'Initial balance time (in minutes)' set to a non-zero value. Once 'Initial balance time' is defined the there are 3 lines which will be enabled and can be configured, you can enable only the lines you wish to see to display the 'Initial Range', 'Initial POC', and 'Initial Value Area'.



1. Initial Range: This is the range of the profile used in the initial profile calculation that was traded during the 'initial balance time (in minutes)'
2. Initial POC: The value of where the point of control (POC) is located during the 'initial balance time (in minutes)'
3. Initial Value Area: The value area during the 'initial balance time (in minutes)'

Developing POC and Value Area

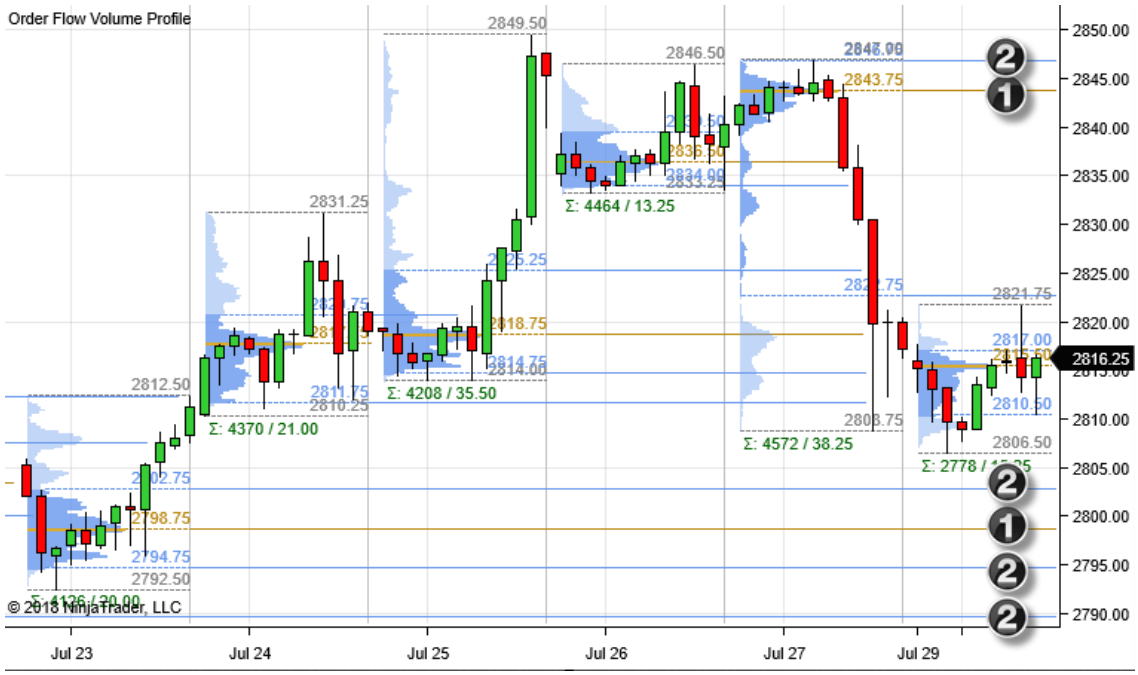
The developing POC and Value Area is an optional feature which when enabled will show you bar by bar how these value change overtime for the profile.



1. Developing POC: For each bar in the chart the value of the POC is plotted so you can see the change over time.
2. Developing Value Area: For each bar in the chart the value of the value area high and value area low so you can see the change over time.

Extended "Naked" Point of Control and Value Area

The extended "Naked" Point of Control and Value Area will take any POC of Value Area of a profile and plot it forward from the end time of the profile until such time as the price is traded again in the future. Uncovered or 'naked' are expected to be 'filled' sometime in the future and may be an area of support or resistance. There visibility and plot properties are set in the 'Lines' section



1. Naked POC
2. Naked Value Area

Note: If a line is enabled, but it is not seen, it may be under another enabled line.

Understanding Price Profiles

Price Profiles

Price profiles displays a letter or block if the underlying instrument trades inside of a specific time window. These time windows are 30 minute in length and if a single trade occurs in the time range a single block or letter is plotted. NinjaTrader supports price charts on intraday charts only. Its recommended to use a minute base chart such as 60 minute, 240 minute, 480 minute base chart types. Typically the 'Hide Bars' property is enabled for price profiles. This allows you to focus on the price profile on its own, the candles are no longer needed once you become proficient reading the price profile since both time and price is displayed once you memorize the time/color chart below.

To easily see where the session has it's Open / (developing) Close NinjaTrader displays separate O / C markers. These letters are then excluded in the time/color letter progression for the session.

Scaling the Price Profile

Profile scaling may be required to get your price profile to render as you expect. When rendering in letter mode if there is not enough space to render a letter which would be large enough to read on the screen the profile will fall back to block rendering mode. In this case we need to adjust chart scaling and ticks per level depending on the instrument you wish to plot the price profile on.

1. To adjust the distance between profiles, use the time scale drag in the time axis to compress and decompress the underling bars. If you've reached the maximum your able to adjust using the time axis, try adjusting the underling bars up or down as needed, start with a 60 minute chart and increase the minute range by double each time to compress the chart in the time axis.
2. To adjust the price scale use left click and drag in the price label area to adjust price scaling and put the chart into 'fixed' scaling mode. You can hold down left mouse and drag and CTRL on the keyboard to move the range being displayed to focus in on the area you wish to focus on. See the section 'Navigating a Chart' for more information on working with price scales.

Price profile times

It is important to look into the 'Order Flow + Volume Profile' properties, specifically the 'Times' section to understand the time slices reported by the tool and what color and letter is associated. The time displayed in converted to your local time zone and is a fixed frame of referenced from starting at 8:00 CST. This time is arbitrarily picked and provides all instruments to share the same time slice, therefore the letter 'A' and associated color will always be the same calendar day and not effected by the individual instrument, this allows comparison between charts based on a fixed time scale. The color for each time slice is customizable, once you have it defined

Note: Only intraday charts are supported. Daily, weekly, or monthly charts are not supported at this time.

Reference Table

A	8:00 - 8:30 AM CST	a	8:00 - 8:30 PM CST
B	8:30 - 9:00 AM CST	b	8:30 - 9:00 PM CST
D	9:00 - 9:30 AM CST	d	9:00 - 9:30 PM CST
E	9:30 - 10:00 AM CST	e	9:30 - 10:00 PM CST

F	10:00 - 10:30 AM CST	f	10:00 - 10:30 PM CST
G	10:30 - 11:00 AM CST	g	10:30 - 11:00 PM CST
H	11:00 - 11:30 AM CST	h	11:00 - 11:30 PM CST
I	11:30 - 12:00 PM CST	i	11:30 - 12:00 PM CST
J	12:00 - 12:30 PM CST	j	12:00 - 12:30 AM CST
K	12:30 - 1:00 PM CST	k	12:30 - 1:00 AM CST
L	1:00 - 1:30 PM CST	l	1:00 - 1:30 AM CST
M	1:30 - 2:00 PM CST	m	1:30 - 2:00 AM CST
N	2:00 - 2:30 PM CST	n	2:00 - 2:30 AM CST
P	2:30 - 3:00 PM CST	p	2:30 - 3:00 AM CST
Q	3:00 - 3:30 PM CST	q	3:00 - 3:30 AM CST
R	3:30 - 4:00 PM CST	r	3:30 - 4:00 AM CST
S	4:00 - 4:30 PM CST	s	4:00 - 4:30 AM CST
T	4:30 - 5:00 PM CST	t	4:30 - 5:00 AM CST
U	5:00 - 5:30 PM CST	u	5:00 - 5:30 AM CST
V	5:30 - 6:00 PM CST	v	5:30 - 6:00 AM CST
W	6:00 - 6:30 PM CST	w	6:00 - 6:30 AM CST
X	6:30 - 7:00 PM CST	x	6:30 - 7:00 AM CST
Y	7:00 - 7:30 PM CST	y	7:00 - 7:30 AM CST

Z	7:30 - 8:00 PM CST
z	7:30 - 8:00 AM CST

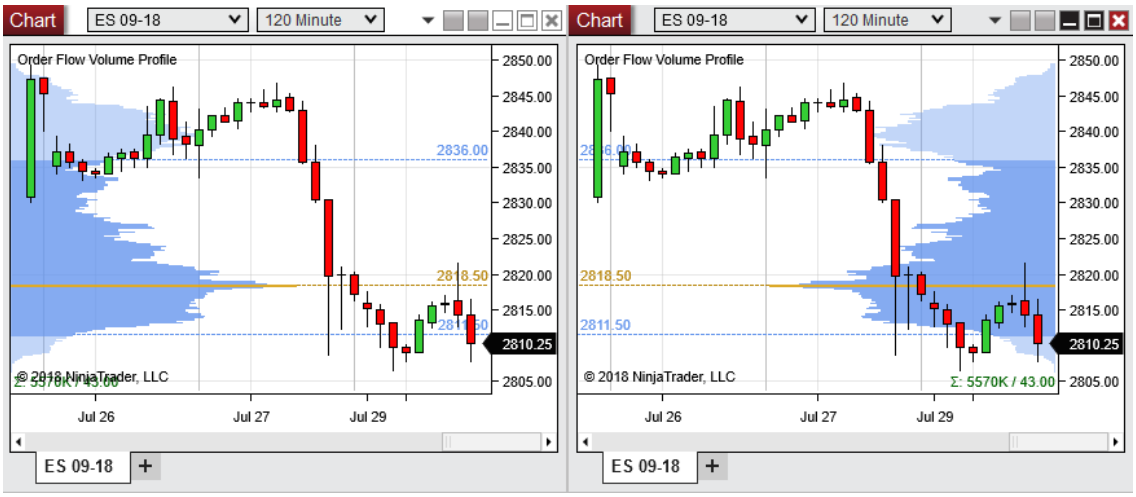
Understanding Composite Profiles

Composite Profiles

A composite profile is a singular large profile rendered on either the left or right side of the chart containing data as defined by the composite properties which become visible once selected. Order Flow Volume Profile has many settings which can be customized to achieve several types of profiles. Its not uncommon to apply a composite profile and at the same time adjust the chart margin to allow for spacing on the right side of the chart. To do this right click on the chart and select 'Properties', adjust the 'Right side margin' property to a higher value. Depending on the 'Profile width (%)' selection of the chart will determine the number of pixels you will need to enter, work in 100 px increments until you find the right size for the configured chart.

You have several options to determine what data to be included in the composite profile:

Weeks Back	The number of weeks back to be factored in this profile.
Days Back	The number of days back from the current day to include in the profile.
Start Date	Manually specify a start date for the profile.
Chart	All the data available in the charts 'Start date' and 'End Date' are displayed in the profile.
Visible Screen Range	The profile will dynamically update based on the data determined from the first bar in view in the chart and the last bar in view on the chart. Moving the chart from left to right will change what data is displayed in the profile.



Note: Composite profiles using 'minute' data being placed on a chart which has bar ranges less then 1 minute is not compatible. In this scenario tick resolution must be used.

Note: You want to make sure that the data you wish to compose a profile of actually exists on your chart to have a complete profile. You may need to right click on the chart > Data Series > and increase the days to load for the chart.

Understanding the Order Flow Volume Profile Drawing Tool

Order Flow Volume Profile Drawing Tool

The Order Flow + Volume Profile Drawing Tool allows you to configure a profile using all the similar settings and configurations which are applicable from the indicator and manually draw out and specify a start and end time for the profile right on the chart. Settings which do not apply are not available on the drawing tool.

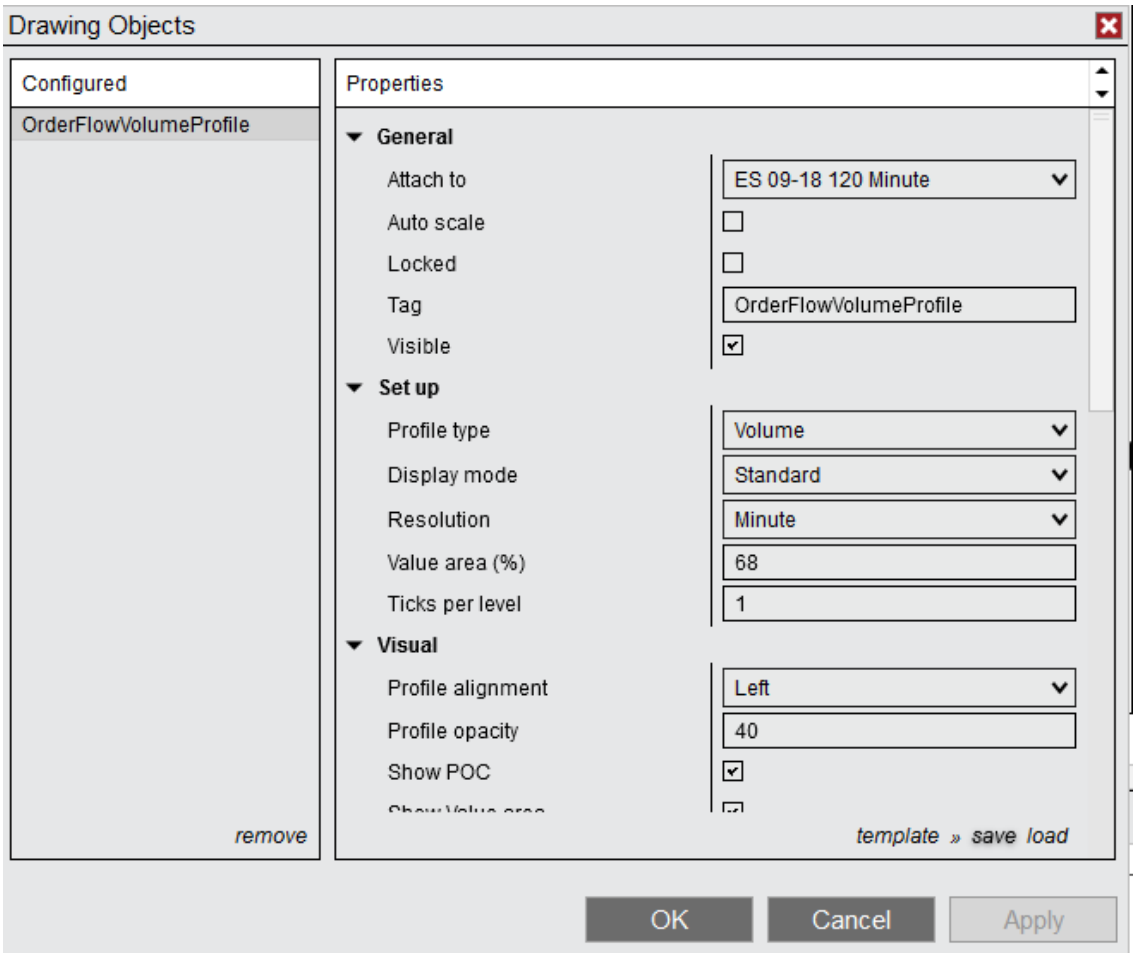
Drawing a Profile

To draw a profile bounds, in the Drawing tool menu select 'Order Flow Volume Profile' from the menu and single click to define the start point of the profile. An outline box will begin to be displayed where you drag your mouse to the ending point for the profile bounds and left click again to complete the draw operation. The bounds box will stay centered and overlaid of the data which will be display the profile. Once the bounds have been selected NinjaTrader will attempt to load the requested data ('Calculating...' text will appear inside the outline box), by default minute resolution data is used however you can change the properties to use 'Tick' resolution data depending on the

type of charts you will be drawing on. Minute works well for daily or any larger time frame chart whereas on more granular intraday charts you will want to change the default to 'Tick' resolution. The section below will detail how to change the default so that future drawing do not require changing the property each time.

Configuring the Drawing Tool

Double clicking the drawing tool or single left click + right click and 'Properties' will allow you to customize the drawing tools various properties. As there are several properties you will want to set a preset for this drawing tool as you have it setup as you prefer. To do this in the 'Drawing Objects' select 'template' and then save, if you save a template with the name 'Default' this will be the template used moving forward on any newly created volume profile. More detailed info on NinjaTrader's drawing tools and operating them, could be reviewed here.



Note: The Order Flow Plus Volume Profile drawing tool will not work on time ranges (historical data) outside the current selected playback range.

Order Flow Volume Profile 'Set up' Parameters

Profile type	This is the top level parameter for selecting the type of profile you would like to generate. Each one of these profiles can be completely different since its constructed with different source data. Volume and tick share rendering options however price is unique in that its rendered using blocks or letters.

	<table><tr><td>Volume</td><td>Volume profiles display last traded volume data using your selected source data 'resolution' is used to build the profile.</td></tr><tr><td>Tick</td><td>Tick profiles display the count of 'ticks' or trades that occur at a specific price. This produces a quantity agnostic profile which display pure trading activity irregardless of volume.</td></tr><tr><td>Price</td><td>Price profiles displays a letter or block if underlying trades inside of a specific time window. These time windows are 30 minute in length. If a single tick occurs inside that time window at a price a letter or block will be plotted. In letter display mode the 'A' letter starts at 8:00 CST.</td></tr></table>
Display mode	Defines how the profile will be rendered and what data is impressed. Volume & Tick Display Modes: <table><tr><td>Standard</td><td>Single defined color profile rendering display volume by price information.</td></tr><tr><td>Buy sell</td><td>The volume bar now split by buy volume and sell volume as classified per the Delta type property.</td></tr></table>

	Note: This mode requires 'Tick' resolution data.
Delta	Displays the standard volume profile and overlays buy/sell data delta as classified per the Delta type property. This display mode highlights imbalances in buy/sell volume. Note: This mode requires 'Tick' resolution data.
Heat	Displays a gradient in the profile based on the highest volume and the lowest volume in the profile. High volume areas are emphasized in this mode.
Outline	Displays the volume profile with only the outline of the profile being plotted.
Time color	Every 30 minutes a new color is plotted and displayed in the profile. The profile color is defined in the properties 'Times' section.

Price Display Modes:

Letters	A letter is displayed if the last price traded anywhere in the price range in a 30 minute window. The letter definitions can be found in

	the 'Times' category in the indicator properties. Capital 'A' letter starts at 8:00 AM CST. When not enough space is available for a 'letter' to be rendered, the profile we default to 'Boxes' display mode below.
Boxes	Instead of displaying a letter as per 'letter' mode above, only the corresponding color to the letter is displayed. This modes is used when you want to display more data on the screen.
Delta type (Only visible when Buy Sell or Delta Display mode is selected)	Sets how the delta is calculated for buy / sell aggressor classification.
BidAsk	Last trade at the ask or higher is considered buying volume, Last trade at the bid or lower selling volume.
UpDownTick	Last trade happens while Ask > Last Ask is considered buying, Last trade happens while Bid < Last Bid considered selling, all volume in between is added to the prior direction - this mode is an important proxy for markets / data providers where best bid / ask information is not available with last price tick data

<table>
<tr><td>Profile period</td><td>
<table>
<tr><td>Sessions</td><td>Each session as defined by the Charts Data Series or the manual Trading hours selected gets a rendered Volume Profile. The profile by default is displayed below the bars and across all bars that make up the session. You can group multiple sessions by defining the property 'Sessions' greater then 1 (count is run from left to right, so forward).</td></tr>
<tr><td>Bars</td><td>When Bars is selected, one profile per bar is rendered. Space is added between the bars so that the profile is readable. This space is defined in the 'Visual' section below 'Profile width (px)' which is a defaulted to 100. You can group multiple bars by defining the property 'Bars' greater then 1.</td></tr>
<tr><td>Composite</td><td>A composite profile is a single profile drawn as an overlay on the entire chart that is composed of data defined by the 'Compose by' sub property that is exposed when this mode is selected.

Compose by:

<table>
<tr><td>Weeks Back</td><td>The number of weeks back to</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

	be factored in this profile.
Days Back	The number of days back from the current day to include in the profile.
Start Date	Manually specify a start date for the profile.
Chart	All the data available in the charts 'Start date' and 'End Date' are displayed in the profile.
Visible Screen Range	The profile will dynamically update based on the data determined from the first bar in view in the chart and the last bar in view on the chart. Moving the chart from left to right will change what data is displayed in the profile.

		Note: You want to make sure that the data you wish to compose a profile of actually exists on your chart to have a complete profile. You may need to right click on the chart > Data Series > and increase the days to load for the chart.
Trading hours	Only data defined in the trading hour template will be used to calculate the volume profile. The default will use all the data as displayed in the chart. **Note**: This setting can only be used as a filter, meaning that all the base data must exist in the chart. E.G. A chart set using an ETH template and using a RTH template volume profile will work correctly, the opposite would not as the indicator will not load the additional data needed for outside RTH chart hours in our example.	
Resolution	Minute	The profile is generated from 1 minute data for the profile period selected. This data is much faster to load and works as a sensible default to generate and approximate profile. Volume data is evenly divided amongst each price

	between the high and the low of the 1 minute bar.
Tick	The profile is generated using 1 tick data. This will be the most accurate and most resource intensive. Only select tick when you know the range of tick data you will be processing is limited.

Ticks per level	Sets the level of aggregation for individual price levels, i.e. if price levels should be merged together, default 1 – so each price level is seen and accounted for individually in the profiles
Value area (%)	The default of 68 percent for Value Area will display a 1 standard deviation range from the point of control and is customizable by the user. Value area is shown using a different opacity in the profile and additionally has a 'line' and 'label' which is drawn at the Value area high and Value area low.

Order Flow Volume Profile 'Visual' Parameters

Profile alignment	Left	The profile is drawn using the left edge.
	Right	The profile is drawn using the right edge.

Display in margin	Only available for Profile period Composite, if selected will display the profile and including lines in the right side margin
Profile width (%) or (px)	This will be in pixels or in percentage, depending on the profile period selection. With percentage it indicates what percentage of space between sessions the bars will take. With pixels it will be how many pixels of space the bars will take.
Profile opacity	This is the base opacity used in to render the profile.
Show POC	This will shade the POC (Point of Control) the color you have selected in the 'Color for POC' property.
Show Value area	Enables the display of the value area.
Value area opacity	Volume bars which are within the value area are plotted with less opacity then the profile, allowing them to standout. To disable this, set the opacity to the same value as the 'Profile opacity'
Color for POC	The color which is used to define the highest volume (Point of Control) seen for the profile.
Color for profile	In 'Standard' draw mode, this property defines the color for the profile.
Color for buy	In 'BuySell' and 'Delta' draw mode, this property defines the color for buy classified orders.
Color for sell	In 'BuySell' and 'Delta' draw mode, this property defines the color for the sell classified orders.

Color for high heat	In 'Heat' draw mode, this property defines the top gradient for the highest volume seen in the profile.
Color for low heat	In 'Heat' draw mode, this property defines the bottom gradient for the lowest volume seen in the profile.
Hide bars	Convenience method to hide the underlying bars enhancing focus on the profiles. This can be done using this option via the indicator or by manually setting the DataSeries bar color to 'Transparent'.

Order Flow Volume Profile 'Lines' Parameters

The 'Lines' section defined all available and configurable line plots. Each line property once extended can have its Color, Dash style, Width, Visible, and Label property defined.

POC	Enable and customize display of the Point of Control line. Please see 'Profile Components' section in the Order Flow Concepts section for more information.
Value area (%)	Enable and customize display of the Value area lines. Please see 'Profile Components' section in the Order Flow Concepts section for more information.
Range	Enable and customize display of the Range lines. Please see 'Profile Components' section in the Order Flow Concepts section for more information.
Initial POC	Enable and customize display of the Initial Point of Control line. Please see 'Initial

	Balance' section in the Order Flow Concepts section for more information.
Initial Value Area	Enable and customize display of the Initial Value Area lines. Please see 'Initial Balance' section in the Order Flow Concepts section for more information.
Initial Balance Range	Enable and customize display of the Initial Balance range lines. Please see 'Initial Balance' section in the Order Flow Concepts section for more information.
Developing POC	Enable and customize display of the Developing POC lines. Please see 'Developing POC and Value Area' section in the Order Flow Concepts section for more information.
Developing Value Area	Enable and customize display of the Developing Value Area lines. Please see 'Developing POC and Value Area' section in the Order Flow Concepts section for more information.
Extended naked POC	Enable and customize display of the Extended POC line. Please see 'Extended 'Naked' POC and Value Area' section in the Order Flow Concepts section for more information.
Extended naked Value Area	Enable and customize display of the Extended Value Area lines. Please see 'Extended 'Naked' POC and Value Area' section in the Order Flow Concepts section for more information.

Order Flow Volume Profile 'Label' Parameters

The 'Lines' section defined all available and configurable line plots. Each line property once extended can have its Color, Dash style, Width, Visible, and Label property defined.

Font	Font settings used for labels and summary data.
Show volume labels	Optionally display the underlying data that makes up the volume profile. To visually see text data you may be required to increase 'Ticks per level' to allow more space for the text to render depending on the scale of the chart.
Show profile summary	This shows total volume, and range information as summary statistics in the bottom left of the profile range.
Summary display unit	Configure the range display unit for summary data (Price, Percent, Ticks, Currency, Pips)
Color for volume labels	Sets the text color used when 'Show volume labels' is enabled.
Color for profile summary	Sets the text color used when 'Show profile summary' is enabled.

10.6.21.5 Order Flow Trade Detector

Description

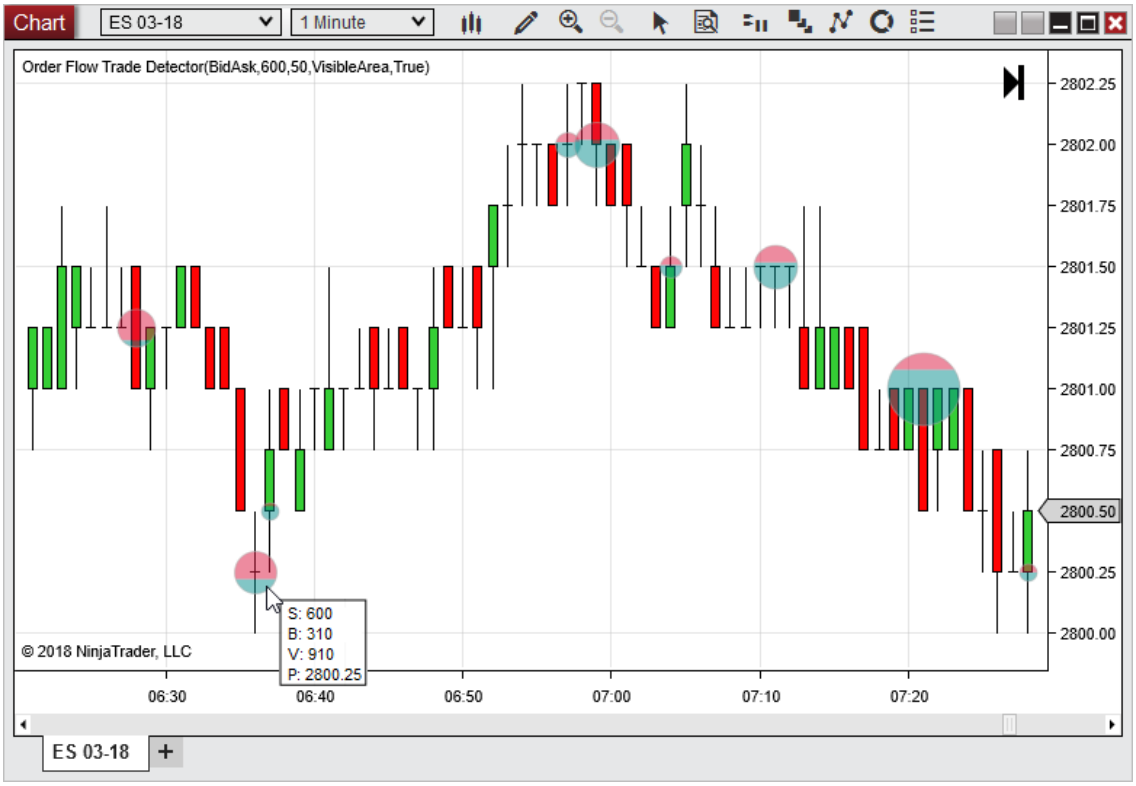
An indicator which displays significant trade volume either at a consistent bid/ask level, at price levels, or individual block trades. The marker is colored based on what percent of orders were considered buy or sell orders. Buy volume is volume that occurred at the ask or higher. Sell volume is volume that occurred at the bid or lower.

Order Flow Trade Detector Overview

Display

In the image below we can see where considerable volume has occurred per the

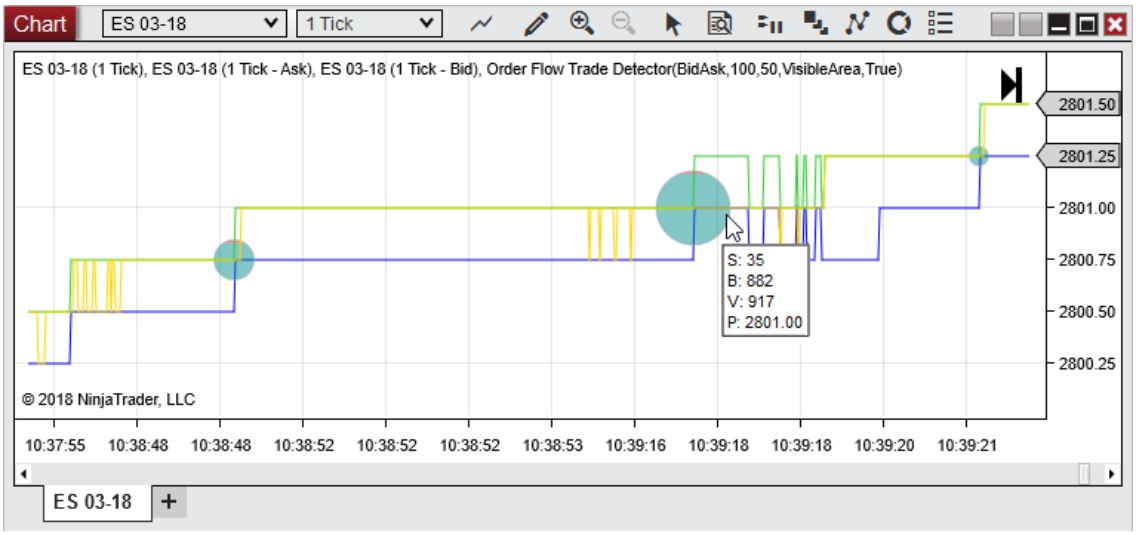
settings that were input. The markers are plotted on the Last price, once the market has finished building.
Hovering the mouse over the trade marker will display information about the marker by default. Here there was a volume of 600 on the sell side (indicated by S: 600), there was a volume of 310 on the buy side (indicated by the B: 310), the total volume was 910 (indicated by the V: 910), and the marker is at the price of 2800.25 (indicated by the P: 2800.25).



Using the Order Flow Trade Detector

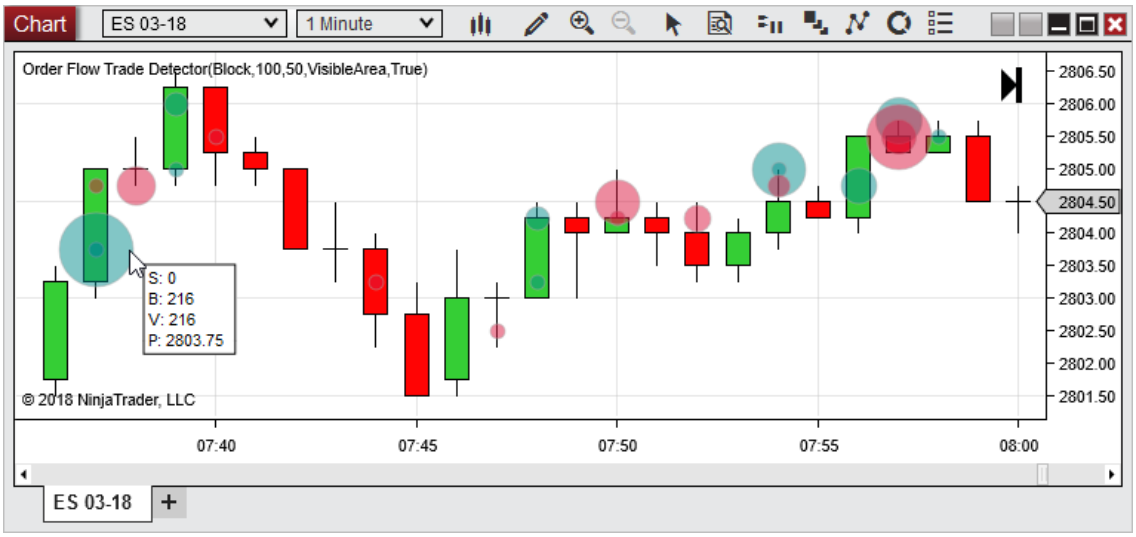
There are 3 main modes to help indicate areas of volume that may be of interest in your order flow analysis.
1) **Consistent bid/ask**
2) **Price**
3) **Block**

1) To detect areas of significant volume that occurred while the last price jumped between a bid and ask price that stayed the same, across multiple bars, you would select **Consistent bid/ask**.



2) To detect large volume that occurred at a price level within the bar, you would select **Price** as seen in the image below.

3) To detect large volume from individual ticks, you would select **Block** as seen in the image below.



Notes:
1. If there is an excessive amount of trade markers on the screen you will need to increase the Minimum volume for a marker to decrease the number of markers.
2. To plot historically requires historical bid/ask stamped tick data. See the Data by Provider section for information on what providers offer historical bid/as stamped tick data.
3. May not display on Forex instruments as expected. Many Forex data providers do not provide volume or they just assign a volume of 10,000 or 100,000 to every tick.
4. The study can appear to draw to an unexpected bar on bars types which call RemoveLastBar (such as Renko, LineBreak etc.) due to the way these bars types update already-closed bars.

Order Flow Trade Detector Parameters

Base large volume on	Consistent bid/ask	With this setting volume will continue to accumulate to form a marker so long as the bid and ask levels stay the same. If the bid and/or ask price changes, a new accumulation of volume will

	occur to create the next marker. The marker will be plotted on the Last price.
Price	With this setting volume will continue to accumulate to form a marker at each price level within a bar. Each new bar will restart accumulation of volume at each price level of the current bar.
Block	With this setting a marker will be plotted for each tick that has a volume equal or greater than the **Minimum volume for a marker**.

Minimum volume for marker	Indicates how much volume must occur before a marker is plotted
Maximum marker size	Indicates the diameter in pixels of the marker which has the largest volume
Base marker size on	

Visible area	The size associated to markers will be distributed based on the volume of the markers in the visible area
Session	The size associated to markers will be distributed based on the volume of the markers in the current session

Show values on hover	When enabled and the mouse is hovered over a marker the buy volume, sell volume,

	total volume, and price will be displayed. Notes: The values displayed will be for the top marker if markers overlap one another. Disabling this setting can improve performance when there is a large amount of markers on the screen.
Color for buys	Color used for volume received at the ask or more
Color for sells	Color used for volume received at the bid or less
Color for outline	Outline color of the marker
Opacity	Opacity of the marker

10.6.21.6 Order Flow Market Depth Map

Description

A real-time data indicator that displays the highest market depth volume received at each price level per bar, then maps them to the chart in comparison to other market depth values received within the visible area. This is done by taking the volume range and applying each depth value 1 of 20 opacity values, set by selected bid/ask depth colors. The larger the depth volume, the more opaque the area will be.

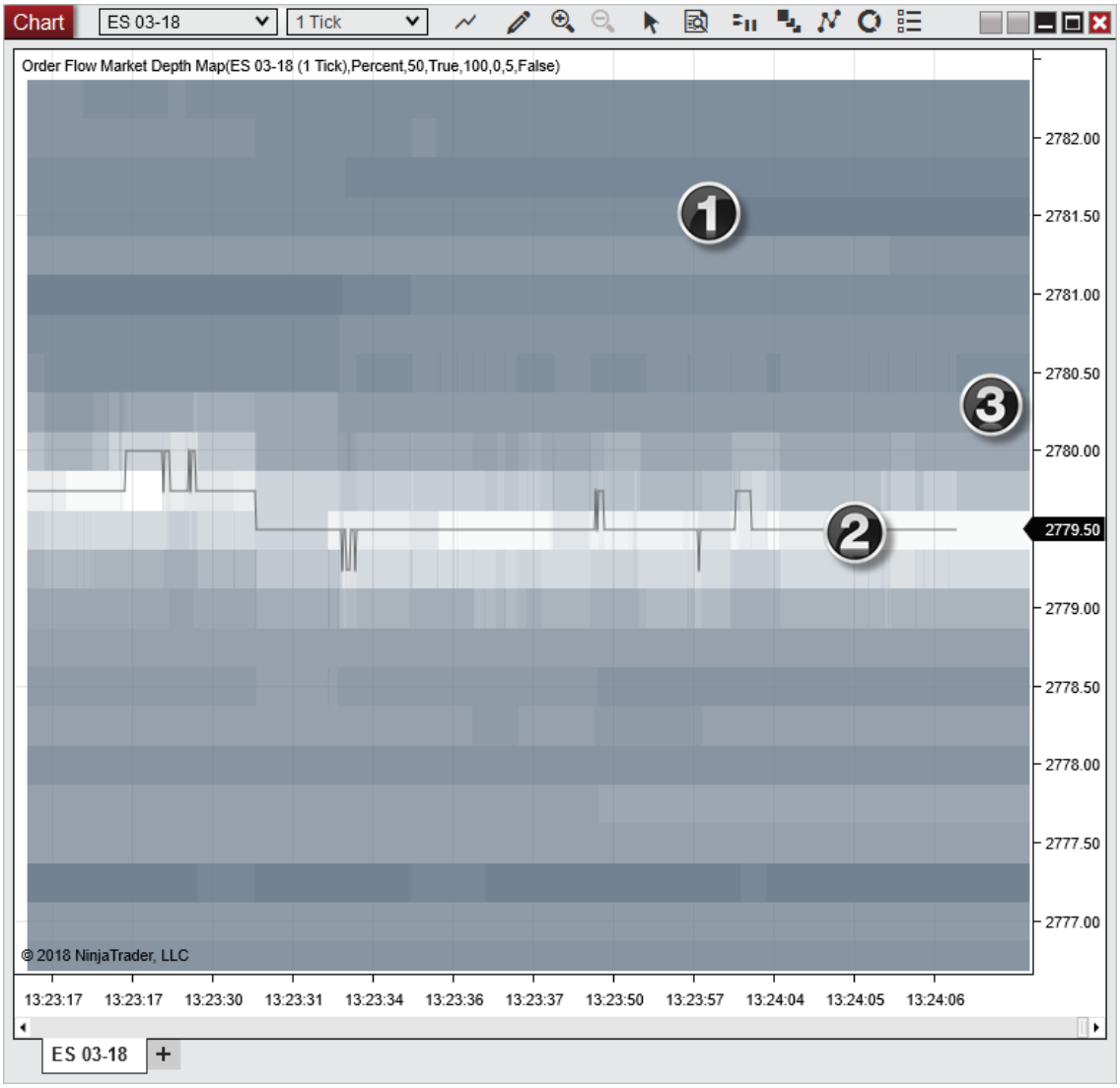
Order Flow Market Depth Map Overview

Display

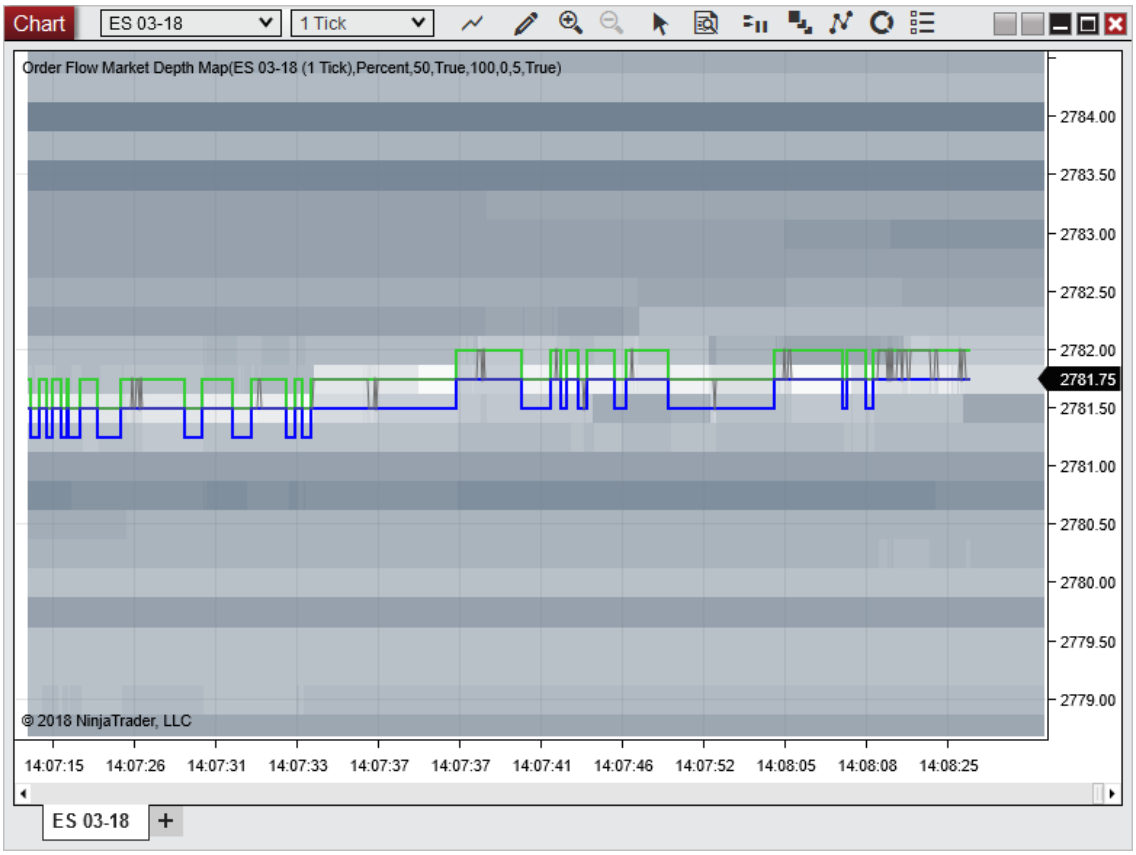
After applying the Order Flow Market Depth Map it will begin to plot off of real time level II data (Display mode Historical Depth shown) with 10 levels per side of the market by default.
With the ES chart below there are 20 real-time market depth values, 10 at bid or below and 10 at ask or above. By default, depth values received at higher or lower values will be extended to the current bars.

1) The areas with the most opaque indicate the areas with the largest market depth volume within the visible area.
2) The areas with the transparency indicate the areas with the smallest market depth volume within the visible area.

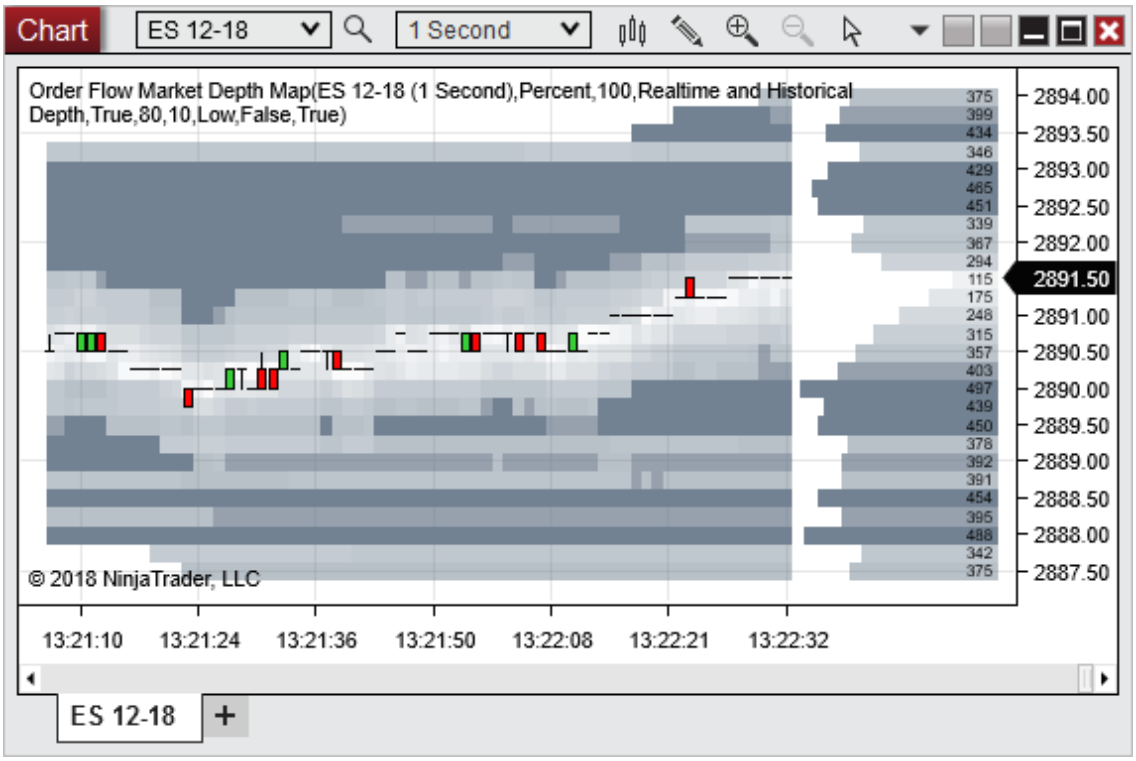
3) The area between the last price and the price axis is the current depth values building for the current bar.



Shown in the image below, Enabling **Show bid/ask line on close** will display a line on close for the bid and ask prices.

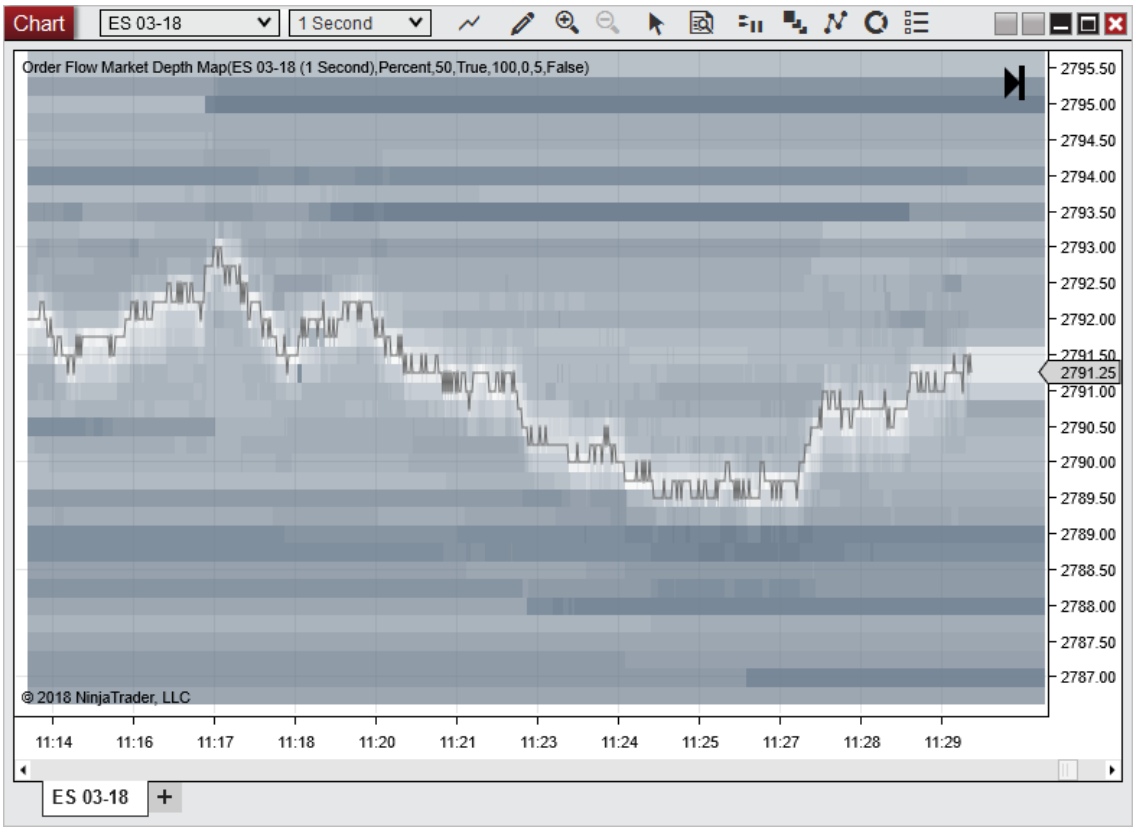


Order Flow Market Depth Map by default also shows Realtime Depth.
Realtime Depth means it displays the realtime Order Flow Market Depth Map values of the currently forming bar. So this will display the largest depth that was received at that price level within the most recent bars time frame. To more closely match the SuperDOM or Level II window you would need to run the Order Flow Market Depth Map on a 1 tick data series.

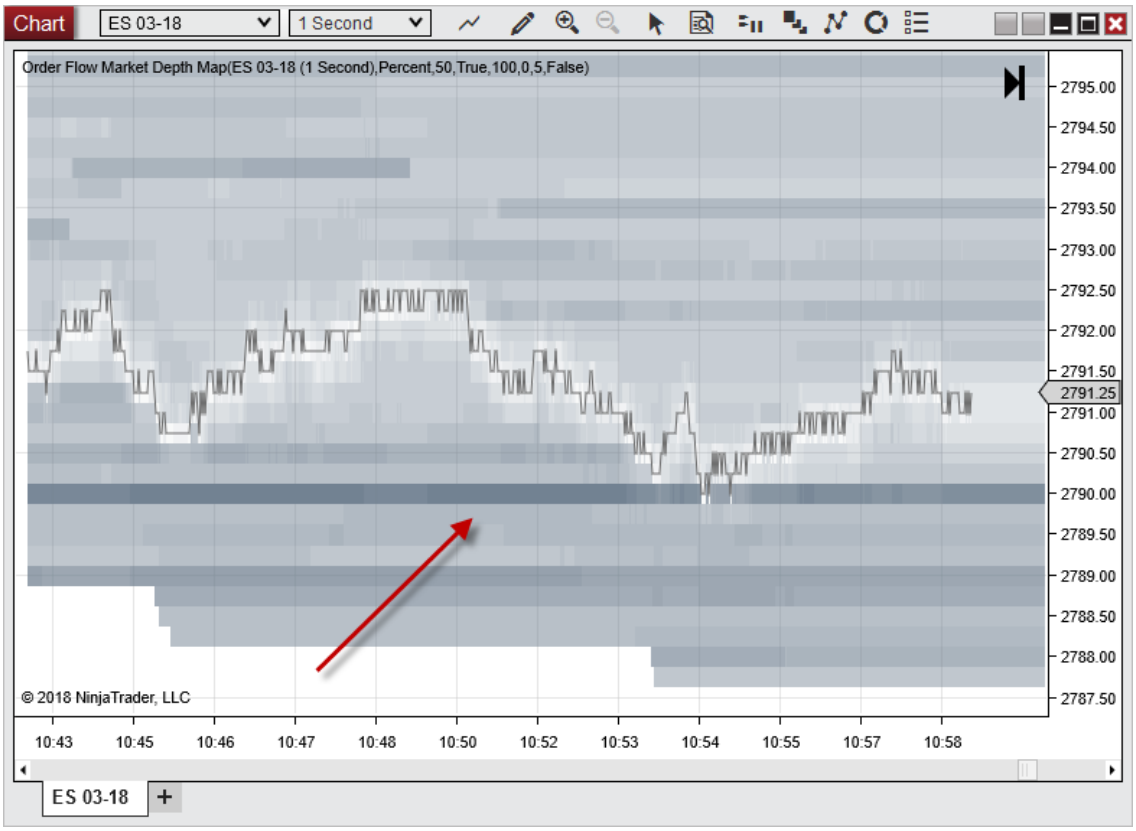


Using the Order Flow Market Depth Map

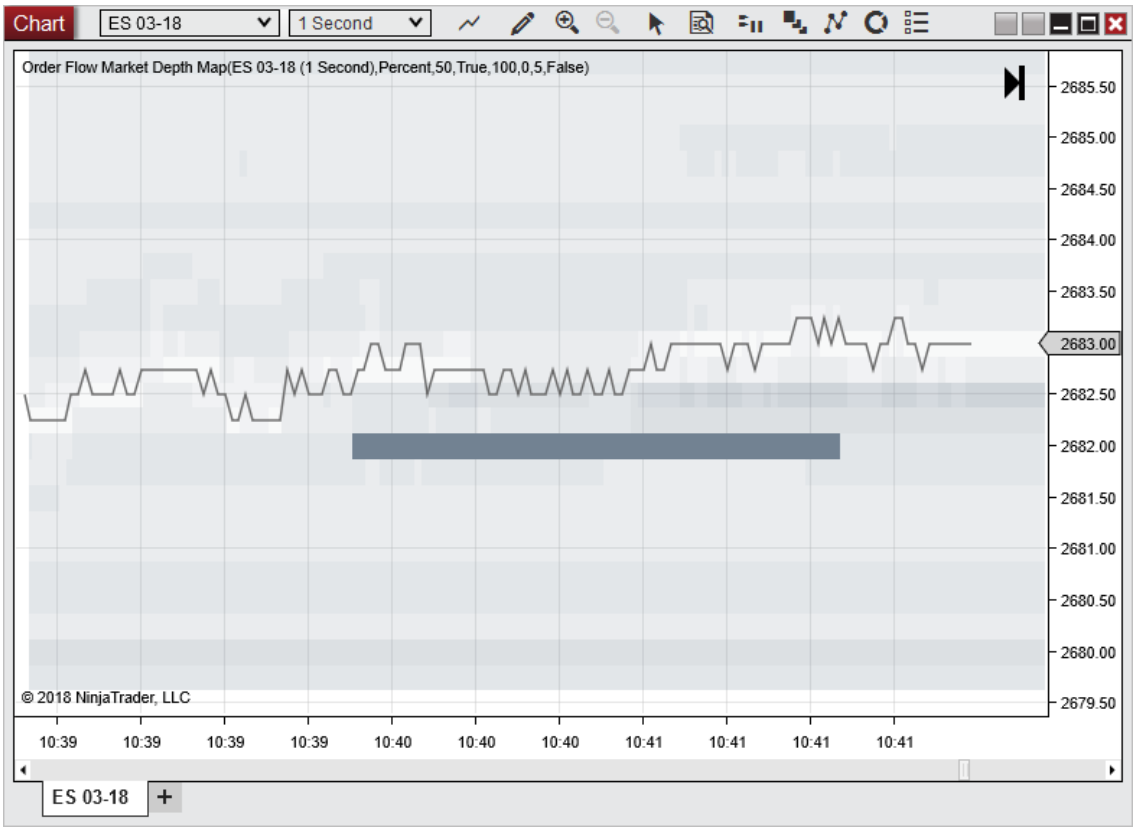
The Order Flow Market Depth Map is able to easily display high and low levels of market depth volume, both historical and real time.



Being able to see high market depth volume can help show potential areas of support and resistance. In the image below we can see there is some large volume at 2790.00 which has created an area of support.



Additionally the Order Flow Market Depth Map makes it easy to see when large volume has been added and removed as seen in the image below.



Notes: Level II data is required for this indicator to function.

Critical: Equity instruments are not supported. There is a limitation in which only one ECN or market marker will have depth provided per level.

Order Flow Market Depth Map Parameters

Base volume range on	Percent	Distributes the opacity levels of the map based on the depths percentage in comparison to the exact volume entered for the

	<table><tr><td></td><td>Maximum volume range and Minimum volume range</td></tr><tr><td>Exact volume</td><td>Distributes the opacity levels of the map based on the depths volume in comparison to the exact volume entered for the Maximum volume range and Minimum volume range</td></tr></table>
Maximum volume range	Input that indicates what values would have the most opacity
Minimum volume range	Input that indicates what value would have the least opacity
# of market depth levels	Input that indicates how many levels of depth on the bid and ask side to show. How many levels can be displayed will be limited by how many levels the data provider supplies.
Opacity distribution	<table><tr><td>Low</td><td>More values have less opacity</td></tr><tr><td>Normal</td><td>Opacity levels are evenly split between the highest and lowest depth volume received.</td></tr><tr><td>High</td><td>More values have more opacity</td></tr></table>
Display mode	Sets which parts of the depth map are displayed, possible values are: • Realtime and Historical Depth (default)

	• Realtime Depth • Historical Depth
Real time depth width	The margin of pixels (between the last price and time axis) to display the currently building realtime depth map
Extend last known volume	If enabled, areas where no market depth was reported, but a market depth was reported to the previous bar's price level, the value will be carried over to the next bar. It will continue carrying over until a new market depth is received at that price.
Color for bid depth	Color for depth on the bid side of the market
Color for ask depth	Color for depth on the ask side of the market
Show bid/ask line on close	If enabled show the line on close for the bid and ask prices

10.6.22 Tick Replay

What is Tick Replay?

Tick replay is a property that can be optionally enabled on NinjaScript indicators and strategies which will ensure that the market data (bid/ask/last) that went into building a bar is loaded in the exact sequence of market data events. This guarantees that your indicators and strategies are historically calculated tick-per-tick exactly as they would have been if the indicator/strategy was running live during a period. **Tick replay** can be enabled for indicators used in **Charts**, **Market Analyzers**, and **Strategies**.

Warning: It is important to note that this property implies that more PC resources are used to calculate your indicators and strategies and as a result will lead to a performance impact. The **tick replay** setting should only be reserved for indicators and strategies which would truly benefit from the additional resources dedicated to arrive at these calculations.

For example, a simple Pivot indicator which just uses the current and previous daily price levels would not see any advantage from using **tick replay**. In contrast, a Volume profile indicator which relies on the exact sequence of trades to calculate various levels would greatly benefit from using **tick replay**.

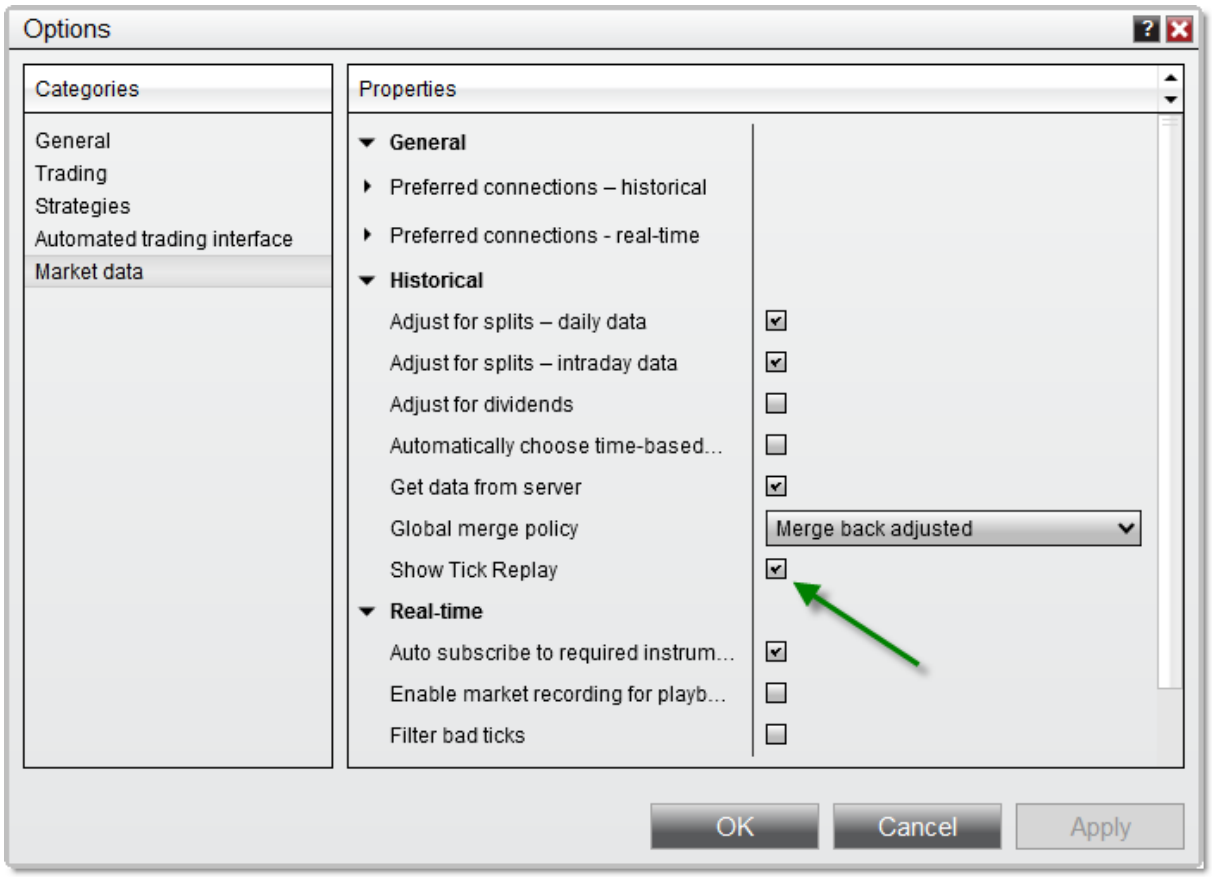
Note: Tick Replay is not intended to function in NinjaScript strategy backtests, and will not provide the same results as running a strategy on live data with Tick Replay enabled. For greater order-fill resolution in strategy backtests, you can use the High Fill Resolution in the Strategy Analyzer.

Indicators and Strategies will only be able to take advantage of **tick replay** if they have been explicitly programmed to calculate these market data events. If you are a programmer and would like to learn how to use Tick Replay with your custom scripts, please see the using tick replay section of our NinjaScript Help Guide.

Setting up Tick Replay

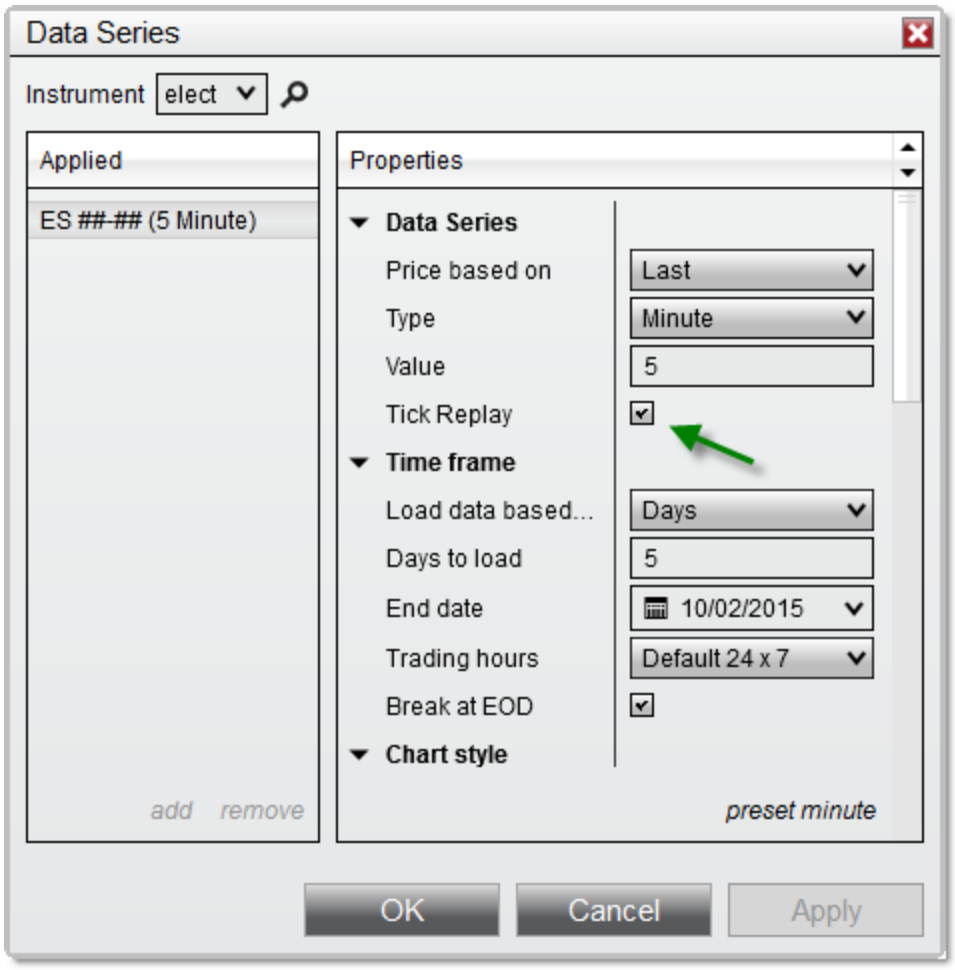
By default, tick replay will not be enabled. In order to expose this property for your indicators and strategies, you will first need turn on the global tick replay option:

- Navigate to the **Control Center** > **Tools** > **Options** menu, and under the **Market data** category, check "**Show Tick Replay**"



Once the "**Show Tick Replay**" option has been enabled from the **Market data** category of the Options menu, you will find a "**Tick Replay**" option which you can select when setting up your indicators or strategies, or when running a strategy in the **Strategy Analyzer**.

Note: The system bar types "Line Break" and "Renko" cannot be used with **Tick Replay** and as a result, the **Tick Replay** option will be disabled when configured with those bar types. There may be other 3rd party bar types which may also disable **Tick Replay** by design. If you are a developer, please see the property IsRemoveLastBarSupported for more information.



10.6.22.1 Tick Replay Indicators

NinjaTrader includes several real-time volume based indicators that are designed to function in real time only. However, Tick Replay allows these indicators to be used on historical data by simulating real-time price movements within historical bars.

Note: When **Tick Replay** is disabled, these indicators function only on real-time data, and therefore do not plot any values on historical data. If you change any property, interval, or instrument on a chart with **Tick Replay** disabled, these indicators will be reloaded and any accumulated real-time data plots will be lost. However, with **Tick Replay** enabled, they will plot historical data, although due to the way that **Tick Replay** processes data, the historical plots may not display precisely the same values as they would have if they had been running in real time.

- Buy Sell Volume

BuySellVolume Indicator

The BuySellVolume indicator displays a horizontal histogram of volume categorized as Buy or Sell trades. Trades are categorized as a Buy when they occur at the Ask or above, and as a Sell when they occur at the Bid or below. Trades that occur between the Bid and Ask are ignored.

1) In the image above, the **BuySellVolume** indicator has just been applied with **Tick Replay** enabled, allowing it to display historical data.

2) In the image above, the **BuySellVolume** indicator has just been applied with **Tick Replay** disabled, limiting it to only displaying what has been calculated in real time.

Buy Sell Pressure

BuySellPressure Indicator

The BuySellPressure indicator displays the current bar's buying and selling pressure as percentage values, and categorizes trades as either Buys or Sells. Trades are categorized as a Buy when they occur at the Ask or above, and as a Sell when they occur at the Bid or below. Trades that occur between the Bid and Ask are ignored.

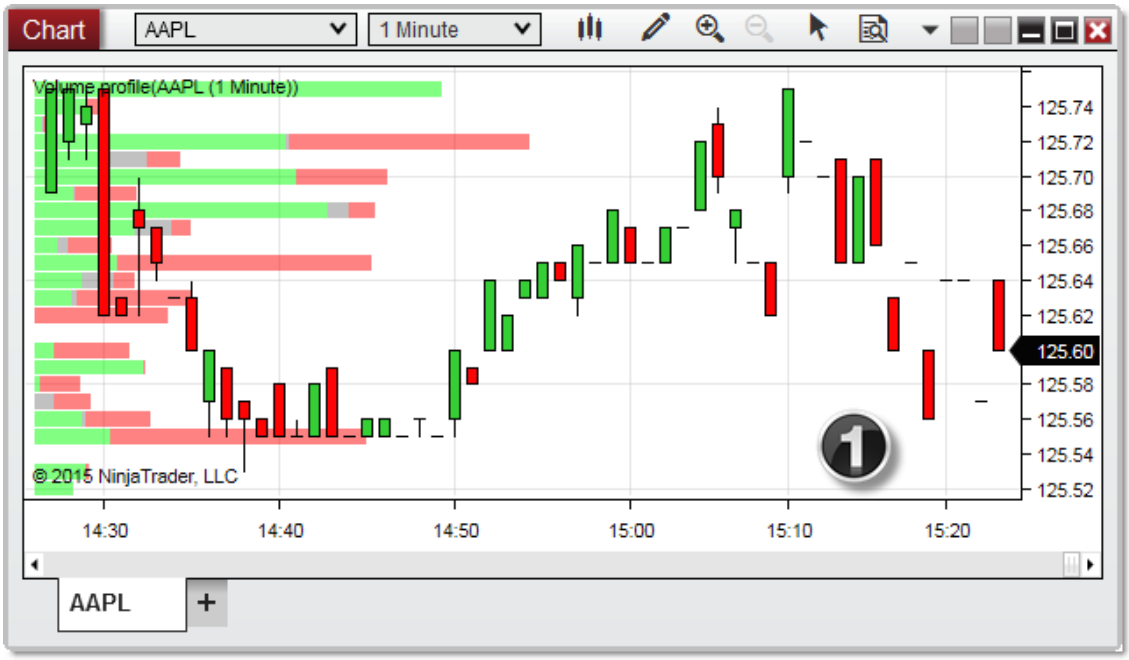
1) In the image above, the **BuySellPressure** indicator has just been applied with **Tick Replay** enabled, allowing it to display historical data.

2) In the image above, the **BuySellPressure** indicator has just been applied with **Tick Replay** disabled, limiting it to only displaying what has been calculated in real time.

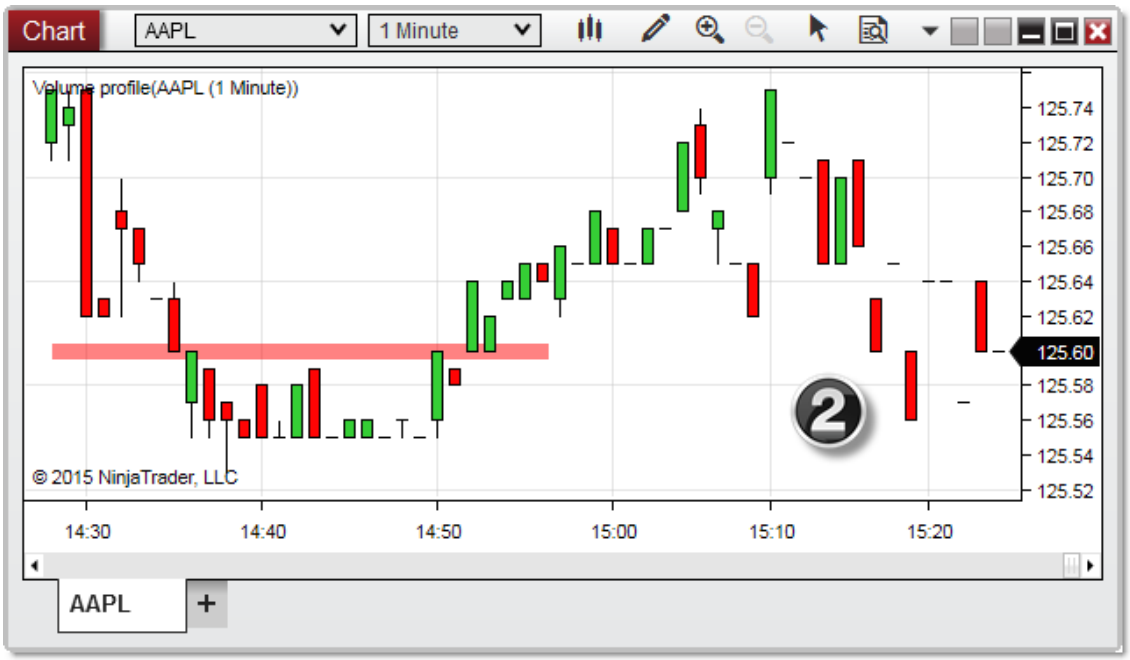
Volume Profile

Volume Profile Indicator

The **VolumeProfile** indicator plots a real-time volume profile as a vertical histogram on a chart. Each bar represents the volume (number of trades) that accumulate at each bar from the time the indicator is started or re-started on the chart. Bars are color coded to represent the number of Buys (trades at the Ask or higher), Sells (trades at the Bid or lower) and neutrals (trades between the market). This indicator provides you with instant feedback to identify support and resistance levels and determine whether accumulation or distribution is taking place at those levels. A cyan colored diamond is automatically drawn at the starting bar of the **VolumeProfile** indicator.



1) In the image above, the **VolumeProfile** indicator has just been applied with **Tick Replay** enabled, allowing it to display historical data.



2) In the image above, the **VolumeProfile** indicator has just been applied with **Tick Replay** disabled, limiting it to only displaying what has been calculated in real time.

10.6.23 COT

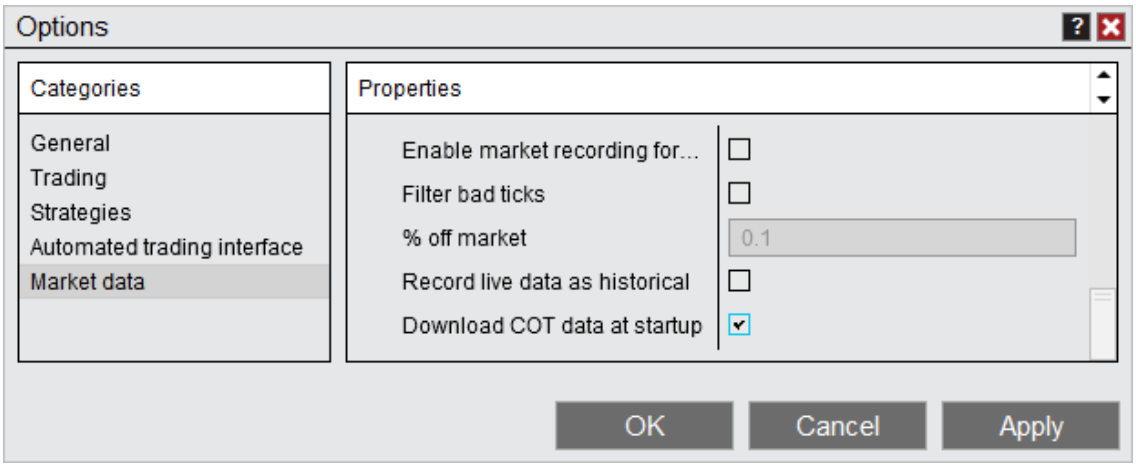
Description

The COT indicator plots weekly data from the Commitment Of Traders report, indicating holdings of different participants in the U.S. futures market.
The report is released every Friday at 3:30pm and contains position data supplied by reporting firms from that Tuesday.

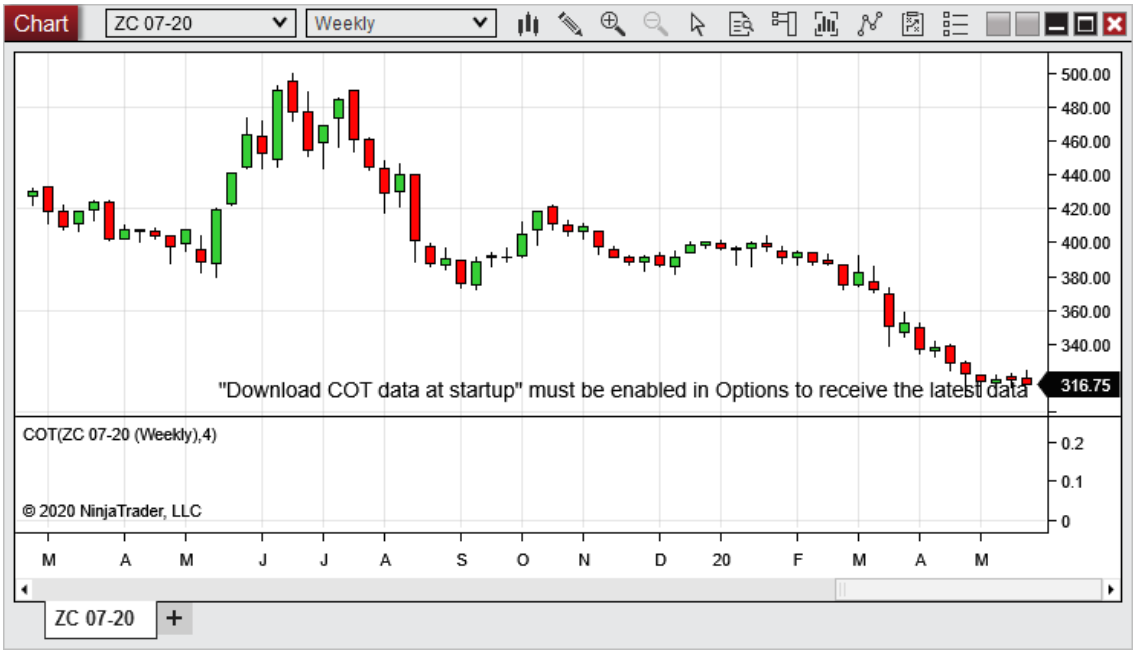
▼ COT Overview

Getting Data

In order to plot the COT indicator you must first enable COT data to be download.
To do this, within the Control Center go to Tools> Options> Market Data then check **Download COT data at start up** and restart NinjaTrader.



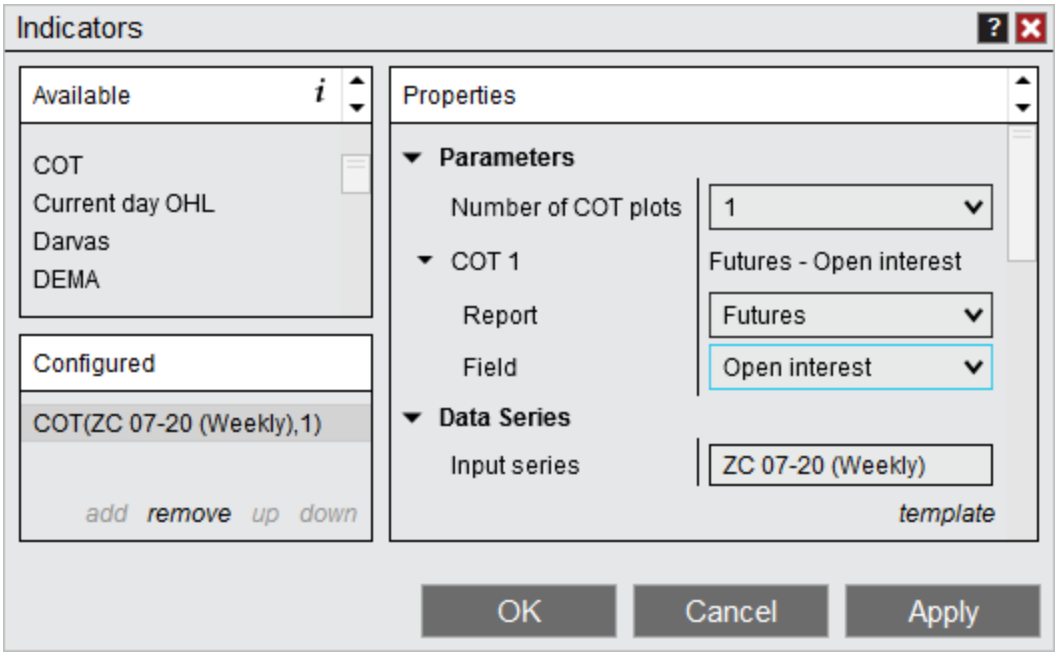
If this option is not enabled and you attempt to apply the COT indicator to a chart, a message will display on the chart saying **"Download COT data at startup" must be enabled in Options to receive the latest data**.



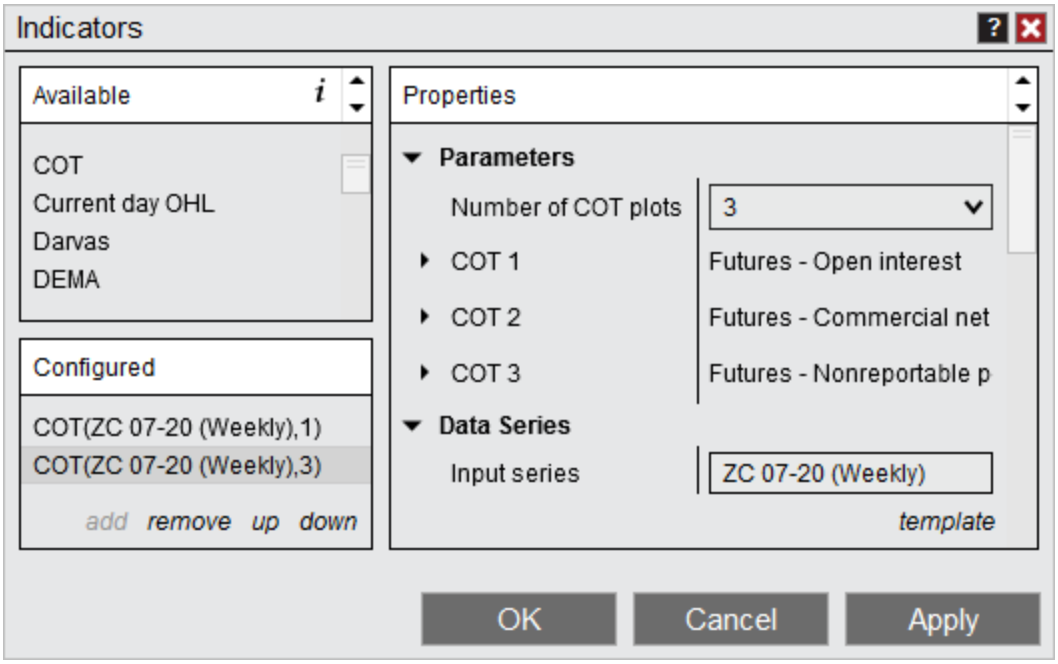
Display

The COT indicator is able to display 1-5 plots at a time from a single instance of the indicator for many U.S. futures instruments. By default it loads 4 plots with some of the most frequently used report fields.

The Open Interest plot typically has a value much larger than the others and it may be preferable to plot Open Interest on it's own.
To do this, within the indicator properties change the **Number of COT plots** to 1 and change **COT 1** to the **Open interest Field**. Adjust plot colors and values as desired.



Next add another COT indicator and set **Number of COT plots** to **3**.



Now you will have a chart with Open Interest on its own panel and the net position report fields on their own panel to increase visibility.

Using the COT

COT data is often times used to compare what market participants are long vs short. This information can then be used to help determine a position per your own trading method.

Here we can see that the red **Futures - Commercial net** plot is 558k short, indicated by the negative value, where as the blue **Futures - Non-commercial net** plot is 541k long.

Another common use is to see when a market participant reverses their position or when market participants plots cross, potentially indicating a change in market direction. Here we can see as the red commercials moved to long and crossed the blue non-commercials the price started to move in a bullish direction.

Note: Since the COT reports are weekly data, the most current data will continue to plot forward until new data is available. After the new report is released on Friday at 3:30pm Eastern time, NinjaTrader will need to be restarted to download the updated data. Since the data is from that weeks Tuesday, the plots will then be updated from that Tuesday forward.

COT Parameters

Number of COT plots	Indicates how many plots you would like the indicator to have from 1-5
COT #	<table><tr><td>Report</td><td>Indicates which report to get data from</td></tr><tr><td>Field</td><td>Indicates what field from the report you want to plot</td></tr></table>

Note: Since this indicator is a historical weekly report and does not change based on new prices or volume, there is no benefit to having this indicator **Calculate** **On price change** or **On each tick**.

COT Values NinjaScript access

For information on how to access the COT values in NinjaScript, please see the Commitment Of Traders (COT) page in the NinjaScript section of the Help Guide.

10.6.24 Wiseman

The Wiseman indicators group are provided by Profitunity and were developed by Bill M. Williams. Profitunity is an educational partner and offers courses on using the Wiseman indicators on their website www.profitunity.com

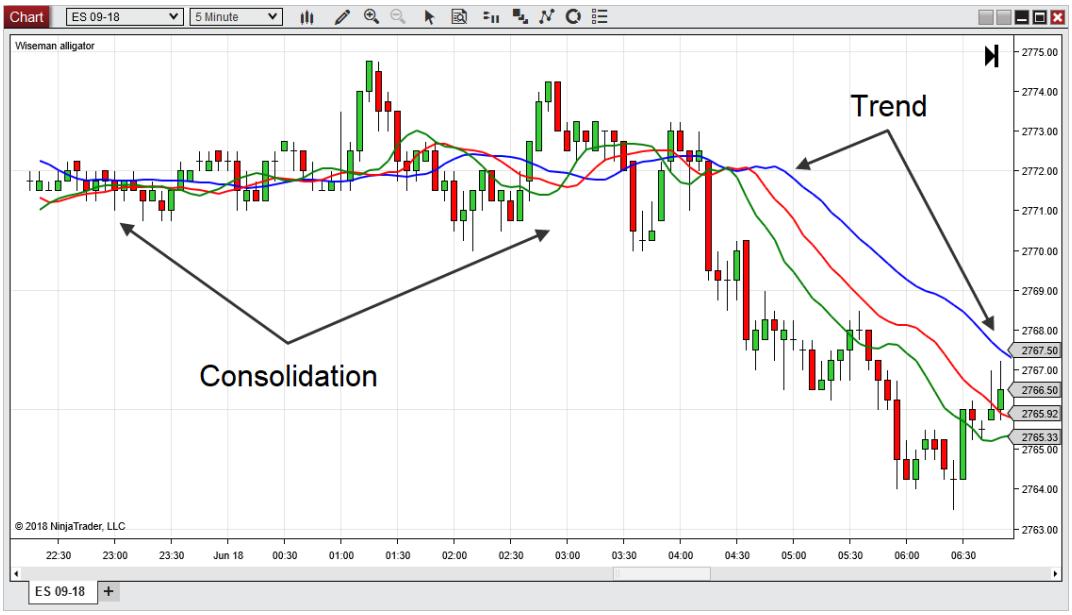
Wiseman Alligator

Description

The Wiseman Alligator is an indicator that consists of 3 moving averages with offsets applied to identify trend absence, formation, and direction.

Display and using the Wiseman Alligator

With the default values the blue line is the jaw, the red line is the teeth, and the green line is the lips. As these lines are intertwined, the alligator is sleeping and the market is consolidating. As they spread apart and continue to move in the same direction, a strong trend is indicated in the market.



Wiseman Alligator Properties

Jaw period	The moving average period of the blue jaw line
Teeth period	The moving average period of the red teeth line

Lips period	The moving average period of the green lips line
Jaw offset	The offset of the blue jaw line
Teeth offset	The offset of the red teeth line
Lips offset	The offset of the green lips line

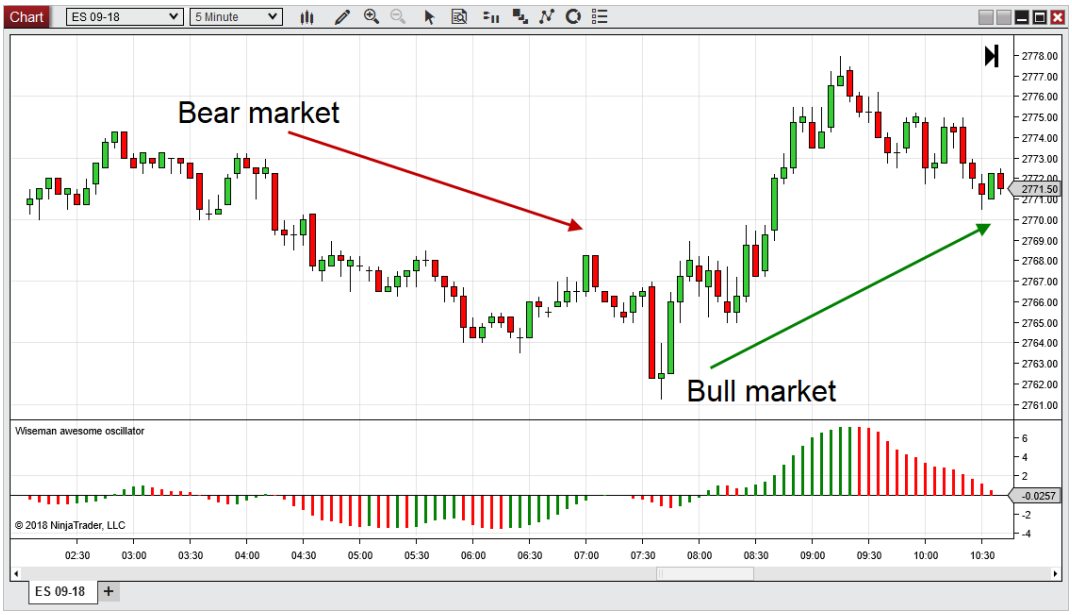
Wiseman Awesome Oscillator

Description

The Wiseman Awesome Oscillator is a momentum indicator to identify trends and reversals.

Display and using the Wiseman Awesome Oscillator

As the oscillator is below zero the market is considered a bear market. As the oscillator is above the zero it is a bull market. The oscillator is colored green when the value is greater than the previous bar and it is colored red when the value is less than the previous bar.



Wiseman Awesome Oscillator Properties

Width	The width of the bars
Color up	Color for bar values higher than the previous bar
Color down	Color for bar values lower than the previous bar

Note: For information on the Bar color properties, please contact www.profitunity.com

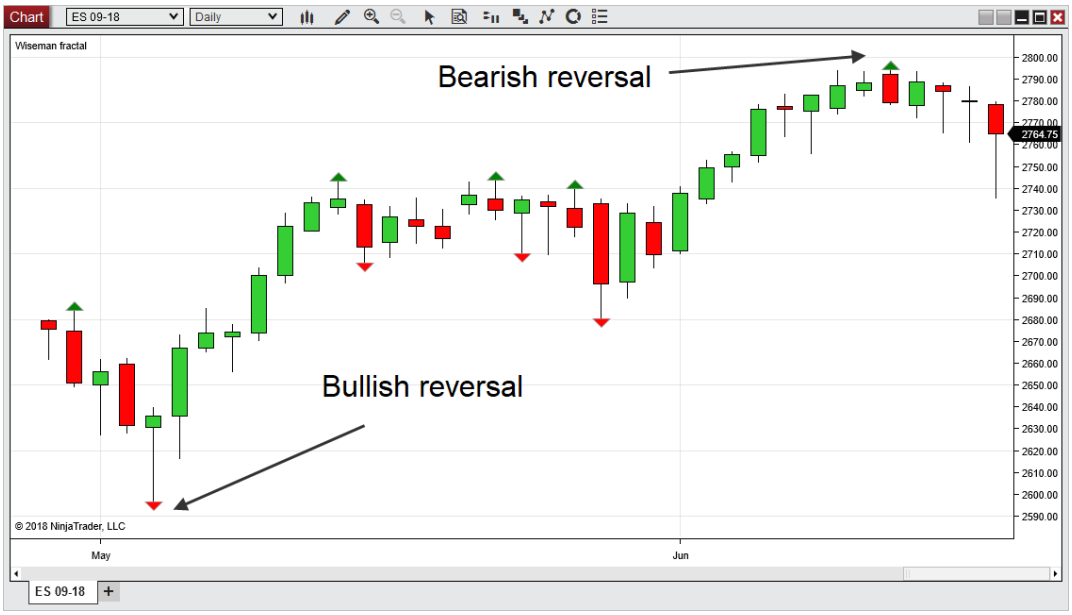
Wiseman Fractal

Description

The Wiseman Fractal identifies areas of reversals. This is signaled by highs surrounded by bars with lower highs or lows surrounded by bars with higher lows.

Display and using the Wiseman Fractal

A green arrow is formed when there is a high surrounded by lower highs per the Strength input.
A red arrow is formed when there is a low surrounded by higher lows per the Strength input.



Wiseman Fractal Properties

Strength	How many bars out to the sides with lower highs of a high bar or higher lows of a low bar to indicate a reversal
Offset (ticks)	How many tick from the high/low to place the pivot marker
High pivot color	Color for bearish reversal pivots
Low pivot color	Color for bullish reversal pivots

10.7 Commissions

Commissions Overview

The **Commissions** window allows you to define global and per instrument commission rates to be used on local Simulation or Live account. Commissions applied are visible inside the **Trade Performance** window. Setting up **commissions** in a local simulation account will help you make sure that you factor in the cost of trading into your **Trade Performance** analysis and can also be used in live accounts so that you can keep track of your actual PnL less commissions, which typically is not factored into PnL values provided by brokers.

› Working with Commission Templates
› Applying Commission Templates

10.7.1 Working With Commission Templates

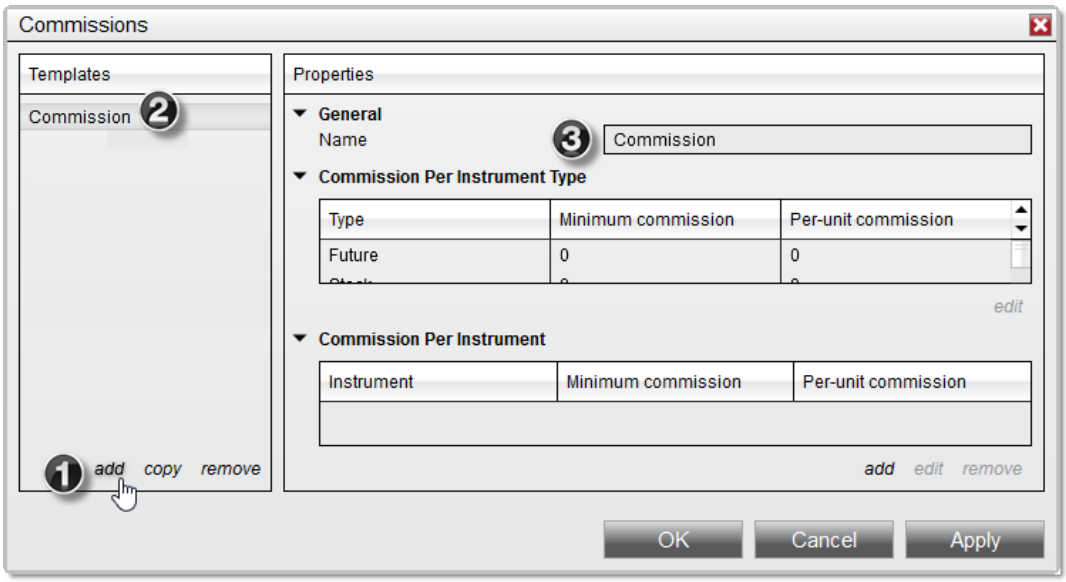
The **Commissions** window allows you to create and manage **Commission Templates** to be applied to different trading accounts configured in NinjaTrader. These templates can be used to set minimum and per-unit commissions for all instruments of a certain type, or to set specific commissions for individual instruments, which will override any commissions set for the instrument type.

▸ Managing commission templates

Adding Commission Templates

The **Commissions** window includes several pre-built **Commission Templates**, based on the currently available NinjaTrader software license types. Additional templates can be set up by clicking the **add** button at the bottom of the "Templates" section of the window. A new template will be created with a default name, and specific commissions can then be saved for the template using the steps in the sections below.

Once a new template has been added, you can edit it's name by selecting it in the list in the "Templates" section, then entering a name in the "Name" field within the "Properties" section.



1) The **add** button is clicked

2) A new template with a default name is added to the list

3) The template name can be changed in the "Properties" section

Copying Commission Templates

There may be an instance in which you need to maintain two copies of a **Commission Template** with a few small differences between the two. Rather than creating a second version from scratch, you can copy an existing template in the **Commissions** window, then make any needed changes to the new copy. To do this, first select a template in the list of configured templates, then click the **copy** button. A new copy will appear in the list, allowing you to make any necessary changes.

Removing Commission Templates

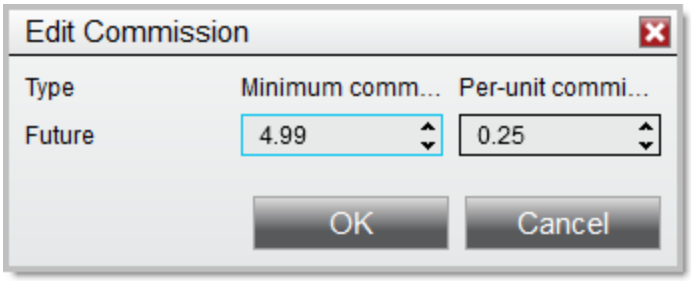
To remove a **Commission Template**, first select one in the list of configured templates, then click the **remove** button.

Managing commissions per instrument type

Adding Commissions Per Instrument Type

To add a commission for an entire asset class (instrument type), first select an asset class listed in the "Commission Per Instrument Type" grid, then click the

edit button. Alternatively, you can double-click on any row in the grid to open the **Edit Commissions** dialogue. Enter a value in the "Minimum Commission" field, the "Per-Unit Commission" field, or both, then click **OK**.

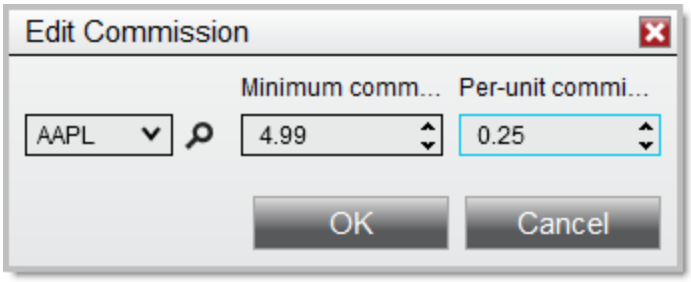


Notes:
1. "Minimum Commission" is applied per trade and per side, regardless of trade quantity, but is only applied if the total applied "per-unit" commissions are less than the minimum value. "Per-unit commission" applies to each unit in a trade, and is applied per-side.
2. When configuring **Forex instrument types**, "Per-unit commission" should be divided by the accounts FX lot size per trade. For example, if your commissions were $0.06 per 1000 FX lot, you would use "0.00006" as the Per-unit commission value (e.g., 0.06 / 1000)

Managing instrument-specific commissions

Adding Instrument-Specific Commissions

To add commissions for specific instruments, first click the **add** button below the "Commission Per Instrument" section to open the **Edit Commission** window. Select an instrument in the "Instrument" dropdown menu, or click the magnifying glass icon to search available instruments. Enter a value in the "Minimum Commission" field, the "Per-Unit Commission" field, or both, then click **OK**.



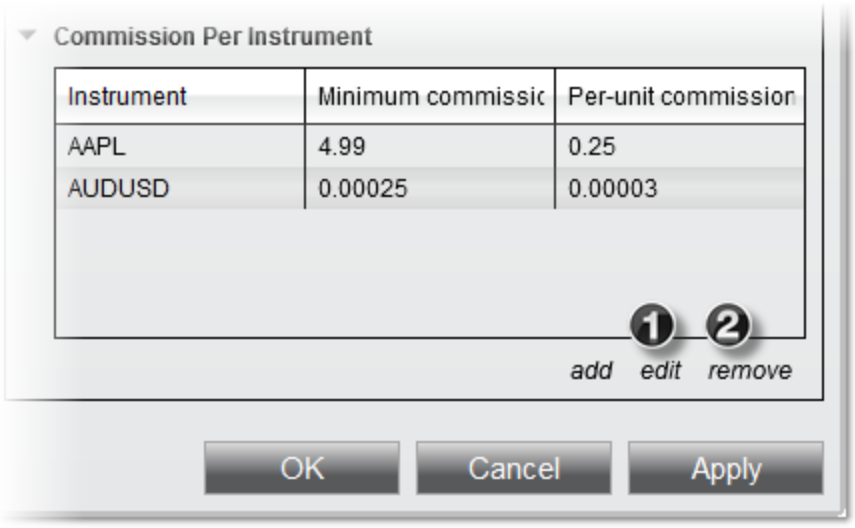
Notes:
1. Commissions entered for specific instruments will override settings for that instrument's type. For example, setting instrument-specific commissions for the E-Mini S&P 500 futures contract will override any commissions set for all Futures instruments.
2. "Minimum Commission" is applied per trade and per side, regardless of trade quantity, but is only applied if the total applied "per-unit" commissions are less than the minimum value.. "Per-Unit Commission" applies to each unit in a trade, and is applied per-side.
3. When configuring **Forex instrument types**, "Per-unit" commission should be divided by the account FX lot size per trade. For example, if your commissions were $0.06 per 1000 FX lot, you would use "0.00006" as the Per-unit commission value (e.g., 0.06 / 1000)

Editing Instrument-Specific Commissions

To edit an instrument-specific commission, first select it in the list of instrument-specific commissions for your chosen **Commission Template**, then click the **edit** button. You can then follow the process outlined above to change the instrument or commission values.

Removing Instrument-Specific Commissions

To remove an instrument-specific commission, first select it in the list of instrument-specific commissions for your chosen **Commission Template**, then click the **remove** button.



1) The **edit** button can be used to edit an existing instrument-specific commission.

2) The **remove** button can be used to remove an instrument-specific commission.

10.7.2 Applying Commission Templates

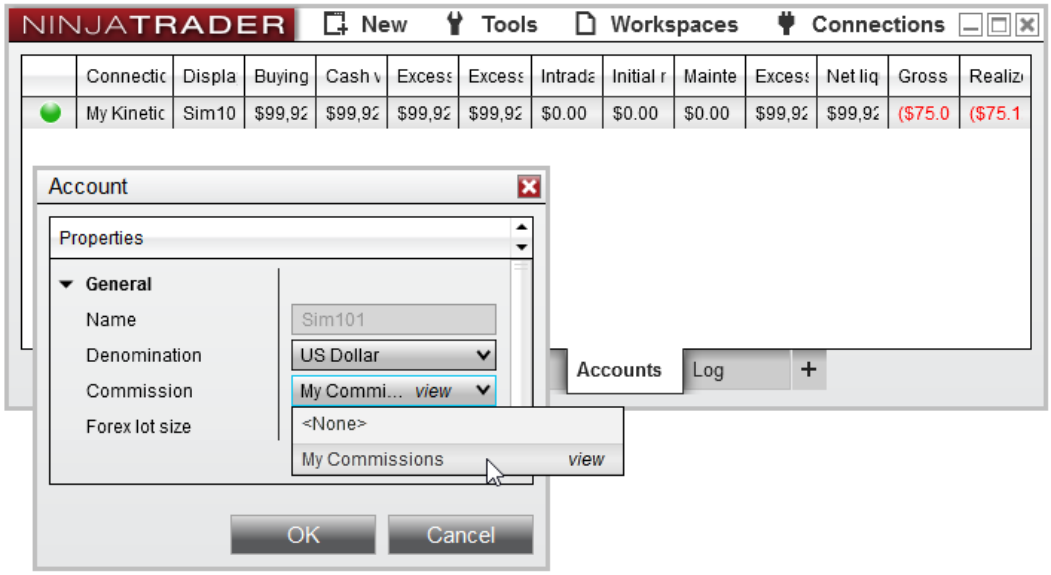
Once a Commission Template has been created, it must be applied to an account, whether it be a live brokerage account connected via a data provider, or a simulation account used for paper trading.

▶ Applying commission templates to accounts

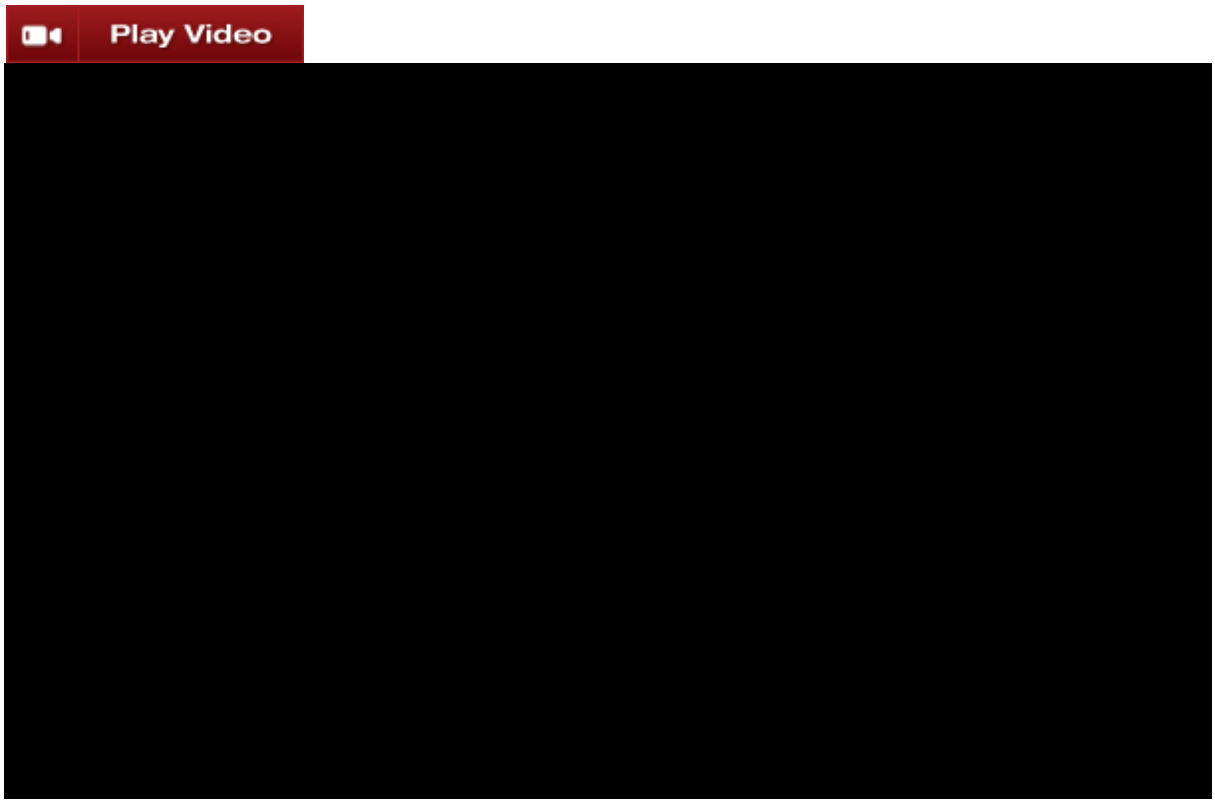
The Accounts Tab

Commission Templates can be applied directly in the Accounts tab of the Control Center. To apply a **Commission Template**, first right-click any account listed in the tab, then click the **Edit Account** menu item. In the **Account** window, select your chosen template in the "Commission" field, then click the **OK** button. Commissions will now be applied to any displayed unrealized PnL related to that account, as well as the Trade Performance window.

> **Note**: In case of the Playback101 account, it's settings including the applied **Commission Template** is dictated by the Sim101 it duplicates.



10.8 Control Center



Control Center Overview

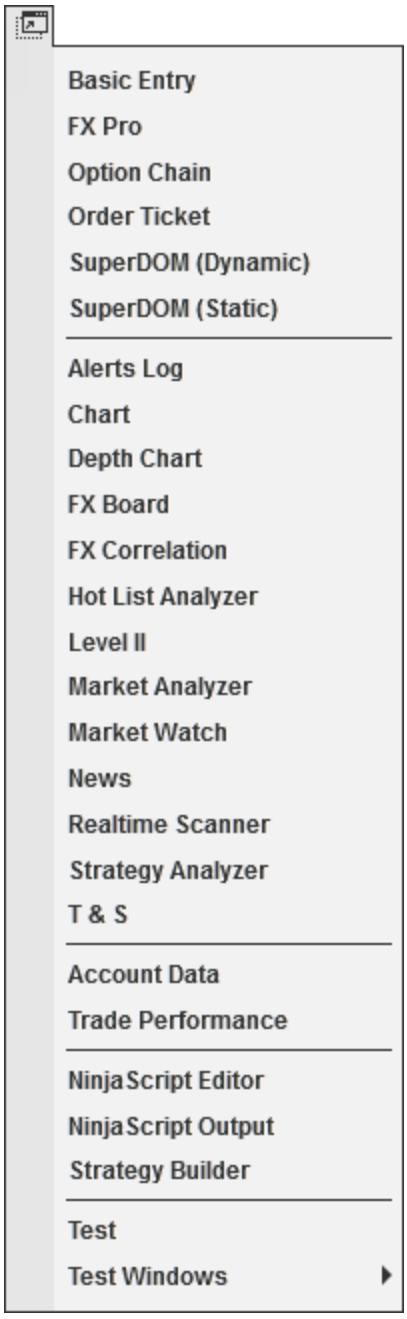
The **Control Center** window is the default window which appears when NinjaTrader is first installed and will always be displayed while NinjaTrader is running.

The NinjaTrader Control Center provides a centralized view of account, execution, order, historical log, and position information. It also provides access to all of the various NinjaTrader function windows and enables/disables global application features and commands.

Menu System

10.8.1 New Menu

The following menus and items are available via the **New** menu of the NinjaTrader Control Center.



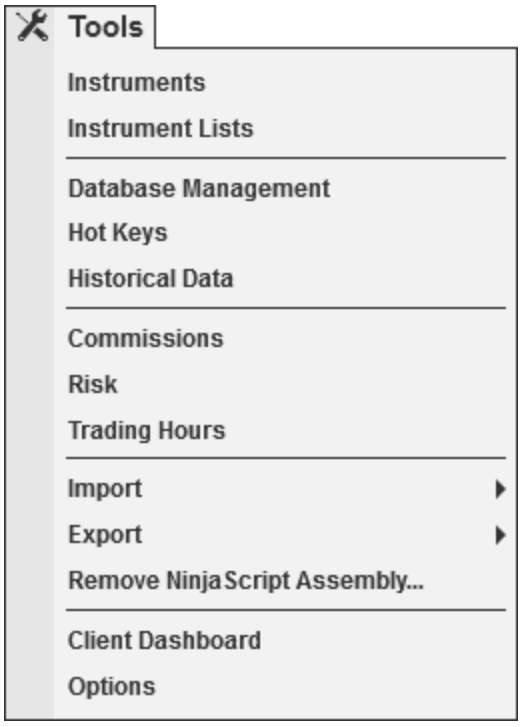
Basic Entry	Creates new Basic Entry window
FX Pro	Creates a new FX Pro window
Options Chain	Created a new Option Chain window

Order Ticket	Creates a new Order Ticket window
SuperDOM (Dynamic)	Creates a new SuperDOM (Dynamic) window
SuperDOM (Static)	Creates a new SuperDOM (Static) window
Alerts Log	Creates a new Alerts Log window
Chart	Creates a new Chart window
FX Board	Creates a new FX Board window
Hot List Analyzer	Creates a new Hot List Analyzer window
Level II	Creates a new Level II window
Market Analyzer	Creates a new Market Analyzer window
Market Watch	Created a new Market Watch window
News	Creates a new News window
Strategy Analyzer	Creates a new Strategy Analyzer window
T & S	Creates a new Time & Sales window
Account Data	Creates a new Account Data window
Trade Performance	Creates a new Trade Performance window

NinjaScript Editor	Creates a new NinjaScript Editor window
NinjaScript Output	Opens the NinjaScript output window (this includes the NinjaScript Utilization Monitor subwindow)
Strategy Builder	Creates a new Strategy Builder window
AddOn Framework	This is an example for a custom NinjaScript AddOn installed

10.8.2 Tools Menu

The following menus and items are available via the **Tools** menu of the NinjaTrader Control Center.



Instruments	Opens the Instruments window
Instrument Lists	Opens the Instrument Lists window

Database Management	Opens the Database Management window
Hot Keys	Opens the Hot Keys window
Historical Data	Opens the Historical Data window
Commissions	Opens the Commissions window
Risk	Opens the Risk window
Trading Hours	Opens the Trading Hours window
Vendor Licensing	Opens the Vendor Licensing window
Import	Opens the Import Sub Menu; Backup File, Historical Data, NinjaScript Add-On, Stock Symbol List
Export	Opens the Export Sub Menu; Backup File, Historical Data, NinjaScript Add-On
Remove NinjaScript Assembly	Opens the Remove NinjaScript assembly window
Global Simulation Mode	Enables or Disables Global Simulation Mode (**Note**: Global simulation mode can only be enabled with a live NinjaTrader License)
Client Dashboard	Opens the web view of the Client Dashboard
Options	Opens the Options window

10.8.3 Workspaces Menu

The following menus and items are available via the **Workspaces** menu of the NinjaTrader Control Center.

- A workspace named "Untitled1" will load automatically
- You can have multiple workspaces open simultaneously
- Open workspaces are indicated by the rectangle icon, if there is no icon then the workspace is closed
- The currently active workspace has a filled green rectangle. You can only have one active workspace
- You can toggle the currently displayed workspace by selecting the workspace you wish to display from the **Workspaces** menu or using the Hot Key SHIFT + F3
- On application shut down you will be given the opportunity to save changes in all open workspaces

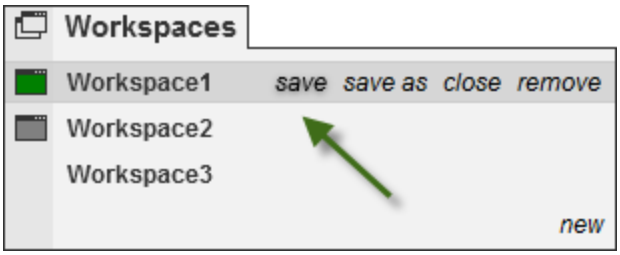
Creating a workspace

Create a Workspace

1. From the NinjaTrader Control Center select the menu **Workspaces**
2. Select "new"
3. You will be prompted to type in a name for the new workspace.
4. On "OK" you will be switched to the new workspace.

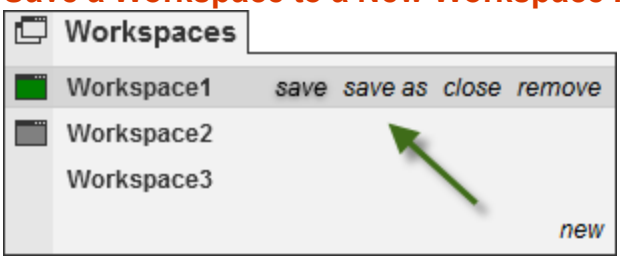
Saving a workspace

Save a Workspace



1. From the NinjaTrader Control Center select the menu **Workspaces**
2. Move your mouse over the name of the workspace you want to save.
3. Select "save"
4. Any changes made to the currently displayed workspace will be saved

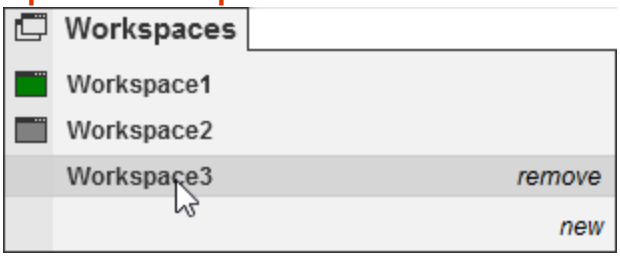
Save a Workspace to a New Workspace File



1. From the NinjaTrader Control Center select the menu **Workspaces**
2. Move your mouse over the name of the workspace you want to save to a new workspace file
3. Select **save as**
4. You will be prompted to type in a name for the new workspace file.
5. On "OK" you will be switched to the new workspace, the old workspace will persist with no changes and the workspace will be saved to the new Workspace file

Opening a workspace

Open a Workspace

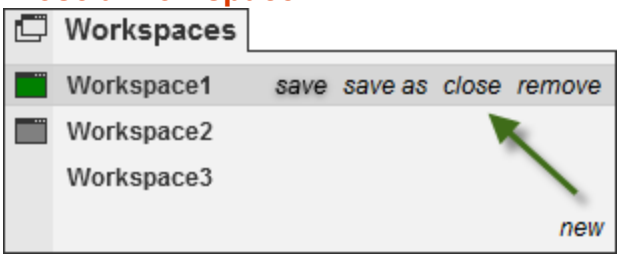


1. From the NinjaTrader Control Center select the menu **Workspaces**

2. Move your mouse over the workspace that you would like to open and left mouse click. In the screenshot above "Workspace3" is closed and clicking on "Workspace3" will open it and make it the active currently displayed workspace.

Closing a workspace

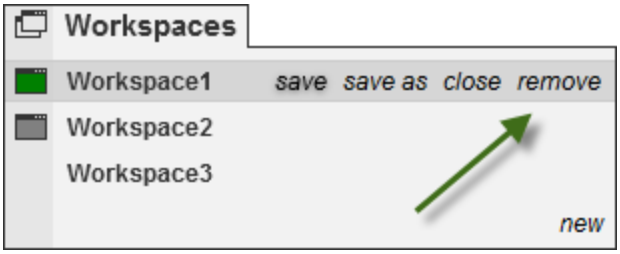
Close a Workspace



1. From the NinjaTrader Control Center select the menu **Workspaces**
2. Move your mouse over the workspace you would like to close
3. Select "close". The selected workspace will be closed and can be reopened at any time.

At least one workspace must remain open, if you close the last workspace a temporary workspace will be created.

Removing a workspace



Remove a Workspace

1. From the NinjaTrader Control Center select the menu **Workspaces**
2. Move your mouse over the workspace you would like to remove
3. Select **"remove"**. You will get a dialog asking you to confirm the delete as any remove operation cannot be undone

> **Note**: Removing a workspace will also remove previous versions from the restore utility. However, it will not remove it from backups

▼ How to quickly switch between workspaces

Quickly Switching Between Workspaces

Pressing SHIFT+F3 keys together will cycle to the next open workspace.

▼ How to recover workspaces

Restore a Prior Version of Workspaces

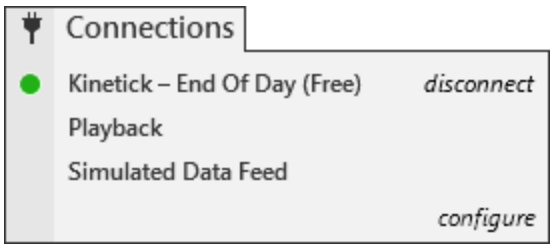
If you need to restore a previously saved version of a saved workpspace, you can do this within the Control Center by going to **Tools** and selecting **Database Management**. In here you will see a **Restore Workspace** section to complete the recovery. See the Database Operations section.

By default 10 previously saved versions of your workspaces will be retained. To modify how many recovery versions are available within the Control Center go to **Tools** and select **Options**. Within the **General** section go to **Preferences** and set the **Versions of recovery workspaces**. See the Options General section.

If you are unable to recover a prior version, if a backup was done you can restore from a backup. See the Backup & Restore section.

10.8.4 Connections Menu

The following menus and items are available via the **Tools** menu of the NinjaTrader Control Center.

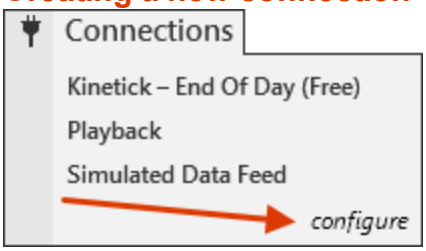


- You cannot remove the predefined constant connections: **Kinetick - End Of Day (Free)**, **Playback Connection**, **Simulated Data Feed**.
- You can connect to multiple connections simultaneously.

- The connection status is reported to the left of the connection name in the connections menu per provider. There is also an aggregated connection status in the bottom left hand corner of the **Control Center.** For more information please see the "Connection Status" section of the help guide.
- Connections menu will only show connections you are authorized to connect per your license key. If you need to connect to more connections or change the connection technology enabled on your license please contact platformsales@ninjatrader.com

Creating a new connection

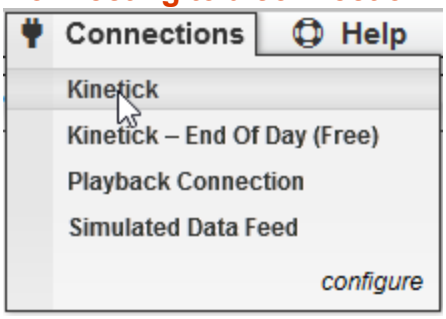
Creating a new connection



1. Click **configure** to define a new connection.
2. See the connection guide for your provider for detailed steps on how to setup your connection.

Connecting to a connection

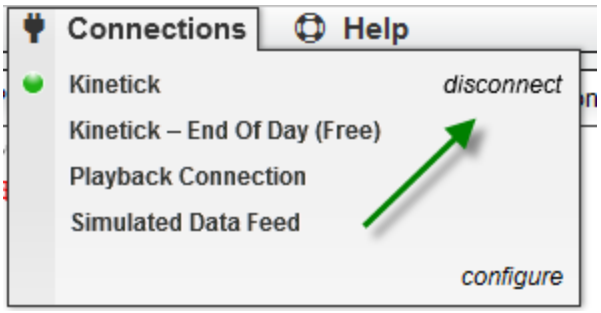
Connecting to a connection



1. Click on the connection name to connect to the defined connection.
2. See the section "*Connection Status*" below for more information on connection status.

Disconnecting from a connection

Disconnect a connection



1. When you are connected to a provider in the **Connections** menu you will see **disconnect** for each active connection.
2. Select "disconnect" to disconnect from the provider.

Understanding the Pre-Built Connections

Pre-Built Connections

Although you will need to configure your own connection to a real-time market data provider and your brokerage account, NinjaTrader comes pre-loaded with several connections which can be used for different purposes.

Kinetick - End of Day (Free)	Provided free of charge by Kinetick, offers daily End-of-Day updates for several instrument types, including stocks, forex, and futures
Playback Connection	Used to play Playback data at various speeds (data must be downloaded prior to using the Playback connection)
Simulated Data Feed	Creates simulated data locally on your PC (not based on real market movements)

10.8.5 Help Menu

The following menus and items are available via the **Help** menu of the NinjaTrader Control Center.

Resources	Opens the NinjaTrader Support web page
Help Guide	Opens the application help guide (or press F1)
Video Guides	Opens the New User Video Guides web page
Email Support...	Sends email support request
Remote Support...	Starts the NinjaTrader Support remote connection (only use this when instructed to do so by the support team)
Download	Download the latest version of NinjaTrader
Apps & Add-Ons	Takes you to the NinjaTrader Ecosystem web page
Live Events	Takes you to the live Events web page

3rd party licensing	Verification for 3rd party add on products
About...	About NinjaTrader

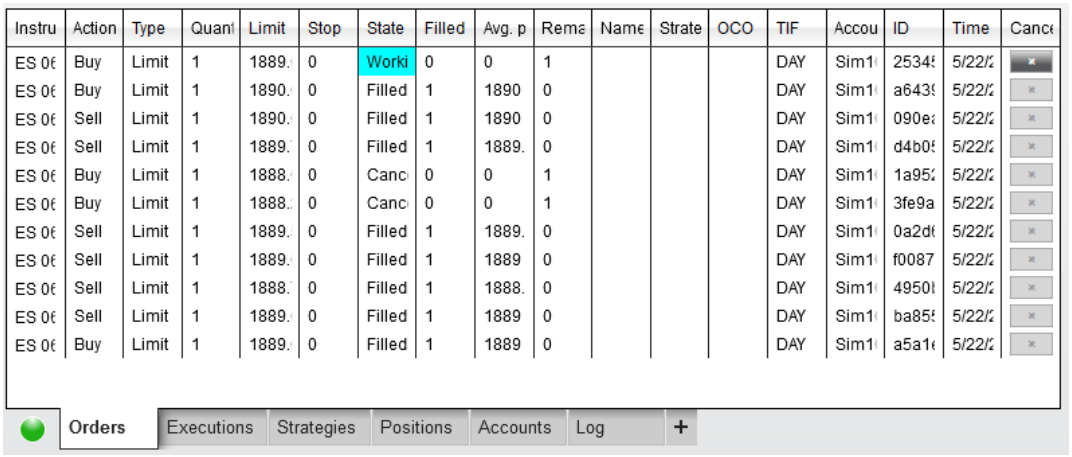
10.8.6 Orders Tab

The Orders tab by default shows all orders in a data grid. The order grid can be filtered by account and also be toggled to show inactive / active orders.

Understanding the orders tab

Order Grid

The order grid displays detailed information regarding the current day's orders. The grid is also active in that you can modify an order directly in it. The active order's State cell will be color coded for ease of use.



1. Modify the **Quantity** of an order by double left clicking your mouse in the QTY field for the order you wish to modify.You can increase/decrease the **Quantity** of an order by pressing the up/down arrow or scrolling the mouse wheel up/down. Holding the CTRL key will modify the Quantity in multiples of 10.
2. Modify the **Price** of an order by double left clicking your mouse in the **Limit** or **Stop** field. You can increase/decrease the **Price** of an order by pressing the up/down arrow or scrolling the mouse wheel up/down. Holding the CTRL key down will modify the order by 10 ticks.
3. **Cancel** an order by pressing on the "X" button.

Columns can be re-ordered and re-sized at will, and individual columns can be enabled or disabled via the **Properties** window accessible in the **Orders** grid's Right-Click menu. The following columns are displayed in the **Orders** grid by default:

Instrument	The instrument on which the order is placed
Action	Indicates whether the order is a Buy or a Sell
Type	The order type
Quantity	The quantity of the order
Limit	The limit price, if applicable. This column will display a zero for order types other than Limit or Stop Limit.
Stop	The Stop price, if applicable. This column will display a zero for order types other than Stop Market, Stop Limit, or Market if Touched.
State	The current order state. See the Order State Definitions page for a complete list of possible states and their definitions.
Filled	The quantity filled on a part-filled or fully-filled order
Avg. Price	The average fill price of a filled order. This column will display a zero for unfilled orders
Remaining	The quantity remaining on a part-filled order
Name	The name of an order placed by an ATM or NinjaScript strategy. This column will be blank for orders submitted manually without an ATM strategy.

Strategy	The name of the ATM or NinjaScript strategy which submitted the order. This column will be blank for orders submitted manually without an ATM strategy.
OCO	The One-Cancels-Other (OCO) identifier of the order, if applicable.
TIF	The Time-in-Force (TIF) setting applied to the order. Possible values are DAY, GTC (Good Till Cancelled), and GTD (Good Till Date)
Account Display Name	The Display Name of the account to which the order is submitted
ID	The NinjaTrader Order ID of the related order
Time	The time at which the order was submitted
Cancel	Contains buttons which can be used to cancel resting orders.

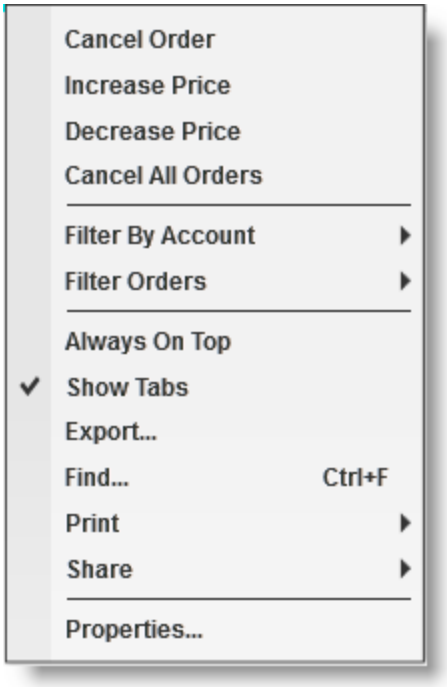
The following additional columns can be applied through the grid's **Properties** window:

Price	The price at which the order will be triggered. For Stop Limit orders, "Price" represents the order's Stop Price
Increase	Contains buttons which can be used to increase the price of an active resting order
Decrease	Contains buttons which can be used to decrease the price of an active resting order
Account Name	The "Account Name" -- not to be confused with the "Account Display Name." These two can differ for

	live brokerage accounts, and the "Account Display Name" tends to be more descriptive.

Right Click Menu

Right mouse clicking within the orders grid opens the following menu:

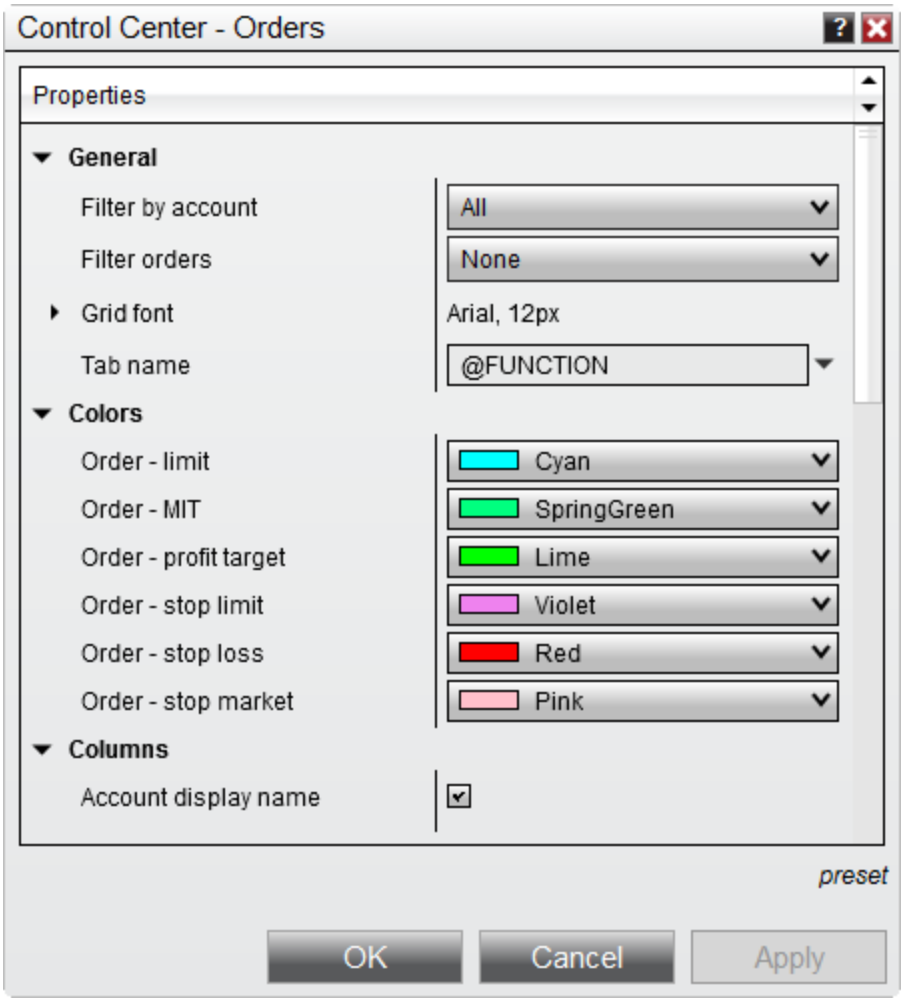


Cancel Order	Cancels the selected order
Increase Price	Increases the price of the order by 1 tick
Decrease Price	Decreases the price of the order by 1 tick
Cancel All Orders	Cancels all working orders

Filter By Account	Filters orders by selected account
Filter Orders	Filters shown orders, possible choices are - • None (show all orders) • Active Orders • Filled Orders • Rejected Orders
Always On Top	Sets if the window will be always on top of other windows
Show Tabs	Sets if the window should allow for tabs
Export...	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Select to print either the window or the order grid area.
Share	Select to share via your share connections.
Properties	Configure orders tab properties

Tip
You can have multiple order grid tabs open at once, this gives you the ability to have multiple order grids visible open and have each one filtering for a different account. This gives you the flexibility to setup an orders grid for each account you have for example so you can keep them totally separate.

Orders Tab Properties



Orders Tab Properties

General	
Filter by account	Filters orders by selected account
Filter orders	Filters shown orders, possible choices are - • None (show all orders) • Active Orders • Filled Orders • Rejected Orders

Grid font	Sets the font for the order grid
Tab name	Sets the tab name
Colors	
Order - limit	Sets the color used for background of the **State** column for working **limit** orders
Order - MIT	Sets the color used for background of the **State** column for working **MIT** orders
Order - profit target	Sets the color used for background of the **State** column for working **ATM profit target** orders
Order - stop limit	Sets the color used for background of the **State** column for working **stop-limit** orders
Order - stop loss	Sets the color used for background of the **State** column for working **ATM stop loss** orders
Order - stop-market	Sets the color used for background of the **State** column for working **stop-market** orders
Columns	Sets that columns are enabled or disabled in the order grid.

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original factory settings, you can left mouse click on the **preset** text and select the option to **restore** to return to the original factory settings - please note though that you cannot save a custom default to restore to.

> **Note**: A number of pre-defined variables can be used in the "Tab Name" field. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.8.7 Strategies Tab

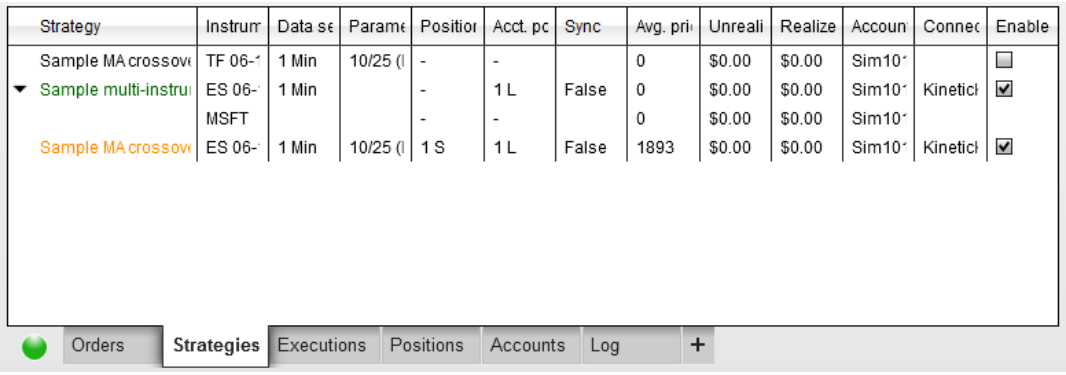
The Strategies tab displays running and terminated strategies in a data grid.

> **Note**: The IncludeTradeHistoryInBacktest property is set to `false` by default when a strategy is applied directly in the **Strategies** tab. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as SystemPerformance.AllTrades.Count that rely on references **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to `true` in the **Configure** state. For more information, see the Working with Historical Trade Data page.

Understanding the strategies tab

Strategy Display

Active and stopped strategies are listed as a grid and can be started/stopped by left mouse clicking the check box in the **Enabled** column.



- Green highlighted "Strategy" name indicates a currently running strategy.
- Orange highlighted "Strategy" name indicates the strategy is waiting until it reaches a flat position to be in sync with the account position before fully starting. (Please see the Syncing Account Positions section for configuration options)
- Black highlighted "Strategy" name indicates a disabled strategy.

Strategies using multiple instruments will be expandable so that each instrument's strategy position can be viewed. In the image above, the second strategy is using ES 06-14 as well as MSFT which is shown below it.

Columns can be re-ordered and re-sized at will, and individual columns can be enabled or disabled via the **Properties** window accessible in the **Strategies** grid's Right-Click menu. The following columns are displayed in the **Strategies** grid by default:

Strategy	The name of the strategy
Instrument	The instrument on which the strategy is enabled
Data Series	The interval type and value associated with the strategy's instrument
Parameters	The values of any user-defined parameter inputs used by the strategy
Position	The Strategy Position (independent of the Account Position)
Acct. Position	The Account Position (includes positions not entered by the strategy)
Sync	Compares the strategy position to the current real-world account position relative to the configured instrument. A value of true indicates the strategy position is currently in sync with the account position For Multi Instrument strategies, a small red flag can appear to the right of the sync value - this would alert the user to expand the row to check the sync for any additional instruments the strategy trades as well.

	Note: A strategy which is in "Wait until flat" (yellow) is considered "flat" regardless of the historical strategy position
Avg. Price	Average price of positions entered by the strategy
Unrealized	Any unrealized profit or loss of an open position entered by the strategy
Realized	Any realized profit or loss of positions entered by the strategy
Account Display Name	The Display Name of the account on which the strategy is enabled
Connection	The connection on which the strategy is running. This column will be blank for disabled strategies
Enabled	A checkbox indicating whether the strategy is enabled. This box can be checked or unchecked to enable or disable a strategy.

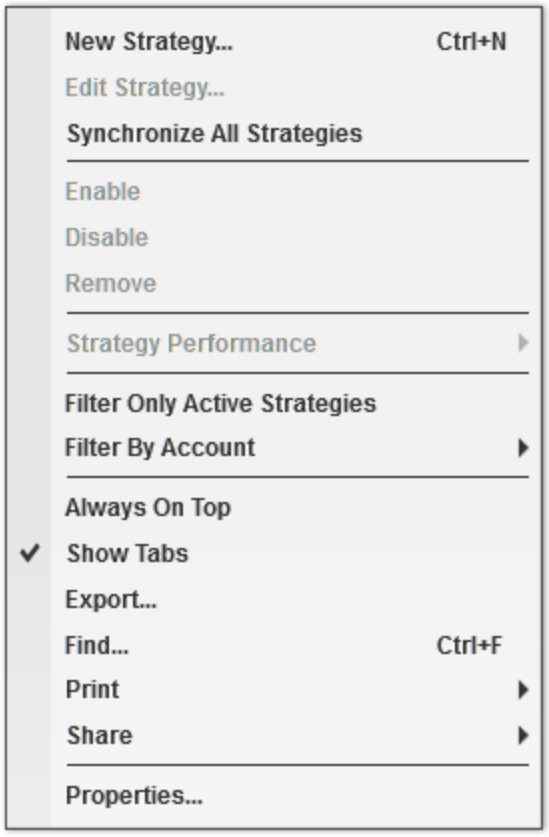
The following additional columns can be applied through the grid's **Properties** window:

Account Name	The "Account Name" -- not to be confused with the "Account Display Name." These two can differ for live brokerage accounts, and the "Account Display Name" tends to be more descriptive.

Tip: Please note the sync column compares only the individual strategy position to the account position, it will not generate a total strategy position for all strategies run on the same instrument / account combination.

Right Click Menu

Right mouse clicking within the strategies grid opens the following menu:



New Strategy...	Run a new automated NinjaScript strategy
Edit Strategy	Brings up the Edit Strategy window to edit the strategy parameters for the selected strategy. (Only disabled strategies can be edited)
Synchronize All Strategies	Will aggregate all strategy positions and syncs aggregate value to the accounts position for the instruments that have running strategies.
Enable	Enables the strategy
Disable	Disables the strategy
Remove	Removes the selected strategy from the grid

Strategy Performance	Generates a performance report for the selected strategy (See the "*How to view strategy performance*" section below)
Filter Only Active Strategies	Displays only active strategies. **Note:** If this item is checked and a new strategy is added that is not yet enabled, the strategy will not be displayed in this grid.
Filter By Account	Sets which strategies to display by account
Always On Top	Sets if the window should be always on top of other windows
Show Tabs	Sets if the window should allow for tabs
Export	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Select to print either the window or the order grid area.
Share	Select to share via your share connections.
Properties...	Configure the strategies tab properties

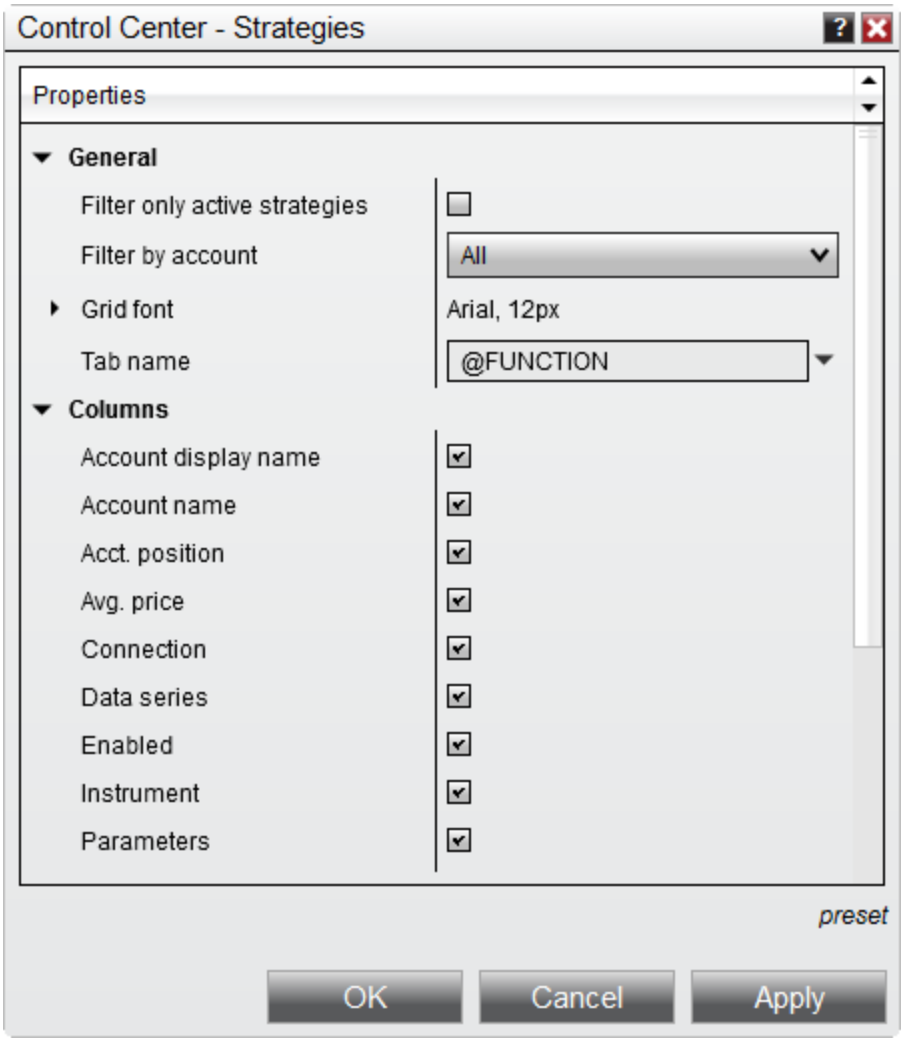
How to view strategy performance

Strategy Performance

While the Account Performance tab will generate performance report against your account's trade history, the Strategy Performance menu allows you to generate a performance report against the trades generated by the selected strategy.

- **Real-time** - Generates performance data for your real-time trades only (since the strategy started running) and will exclude historical trades. If your strategy held a virtual position (calculated against historical data) upon starting, a virtual execution representing the average price of this position will be injected into the real-time results to ensure that a trade pair can be created with the executions resulting from the closing of this position.
- **Historical & Real-time** - Generates performance data for both historical and real-time trade data.
- **Historical** - Generates performance data for historical data only.

Strategy tab properties



Strategy Tab Properties

General	
Filter only active strategies	Displays only active strategies
Filter by account	Displays only strategies running on the selected account
Grid font	Sets the font for the order grid
Tab name	Sets the tab name

Columns	Sets that columns are enabled or disabled

How to Save Property Presets

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original factory settings, you can left mouse click on the **preset** text and select the option to **restore** to return to the original factory settings - please note though that you cannot save a custom default to restore to.

Note: A number of pre-defined variables can be used in the "Tab Name" field. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

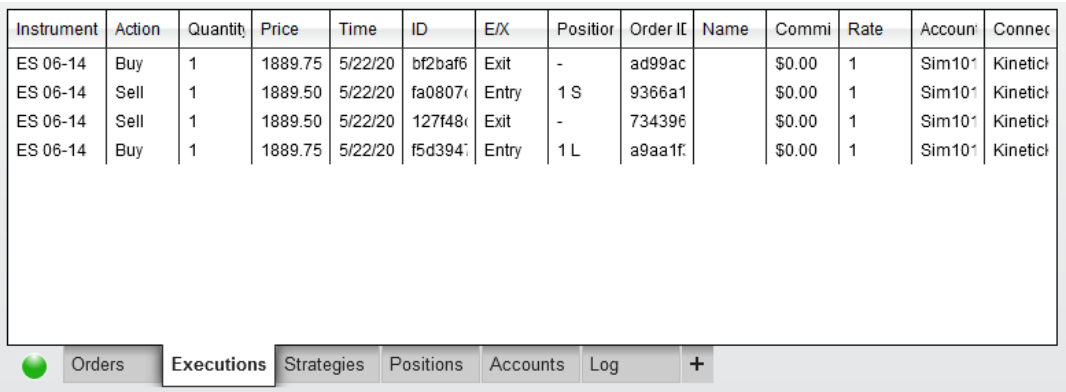
10.8.8 Executions Tab

The Executions tab displays all executions for the current day in the data grid.

▼ Understanding the executions tab

Executions Data Grid

The current day's execution information will be shown in the data grid when connected to your brokerage connection. Simulated trades (into any simulation account) will appear when connected to any data feed connection.



Columns can be re-ordered and re-sized at will, and individual columns can be enabled or disabled via the **Properties** window accessible in the **Executions** grid's Right-Click menu. The following columns are displayed in the **Executions** grid by default:

Instrument	The Instrument on which the execution took place
Action	Indicates whether the execution was from a Buy or Sell order
Quantity	The quantity of the execution
Price	The price at which the execution occurred
Time	The time at which the execution occurred
ID	A unique identifier for the execution
E/X	Indicates whether the execution was an Entry or Exit
Position	Indicates the Account Position after the execution
Order ID	The ID of the order executed
Name	The name of the order executed
Commission	The commission applied to the execution
Rate	The currency conversion rate used for the execution. A value of "1" indicates that no currency conversion took place
Account Display Name	The Display Name of the account to which the execution was placed

Connection	The connection through which the execution was routed

The following additional columns can be applied through the grid's **Properties** window:

Account Name	The "Account Name" -- not to be confused with the "Account Display Name." These two can differ for live brokerage accounts, and the "Account Display Name" tends to be more descriptive.

Right Click Menu

Right mouse clicking within the executions grid opens the following menu:

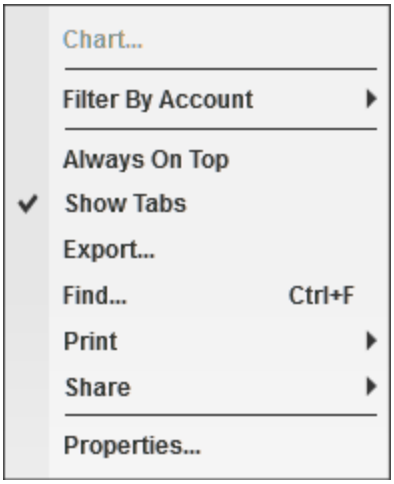


Chart	Opens a chart of the instrument at the time of the selected execution
Filter By Account	Sets which executions to display by account
Always On Top	Sets if the window will be always on top of other windows

Show Tabs	Sets if the window should allow for tabs
Export...	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Select to print either the window or the order grid area.
Share	Select to share via your share connections.
Properties...	Configure the executions tab properties

Forex Executions

Forex executions hold additional data such as **Rate** and **Account Lot Size**

Rate

Executions on currency pairs that do not contain USD will try to grab a conversion rate in real-time shown in the "Rate" column from your data provider. Should a suitable USD conversion rate not be available, a rate of 1 will be used. This Rate will be used in determining the PnL in USD for the forex trade in other areas like the Account Performance tabs. The approach NinjaTrader follows is the GAIN Capital approach, but may differ from what banks do since they base their conversion rates off of the prior session's closing price of the currency pair. This means that our calculation approach may result in slightly different PnL values than the ones reported from your brokerage.

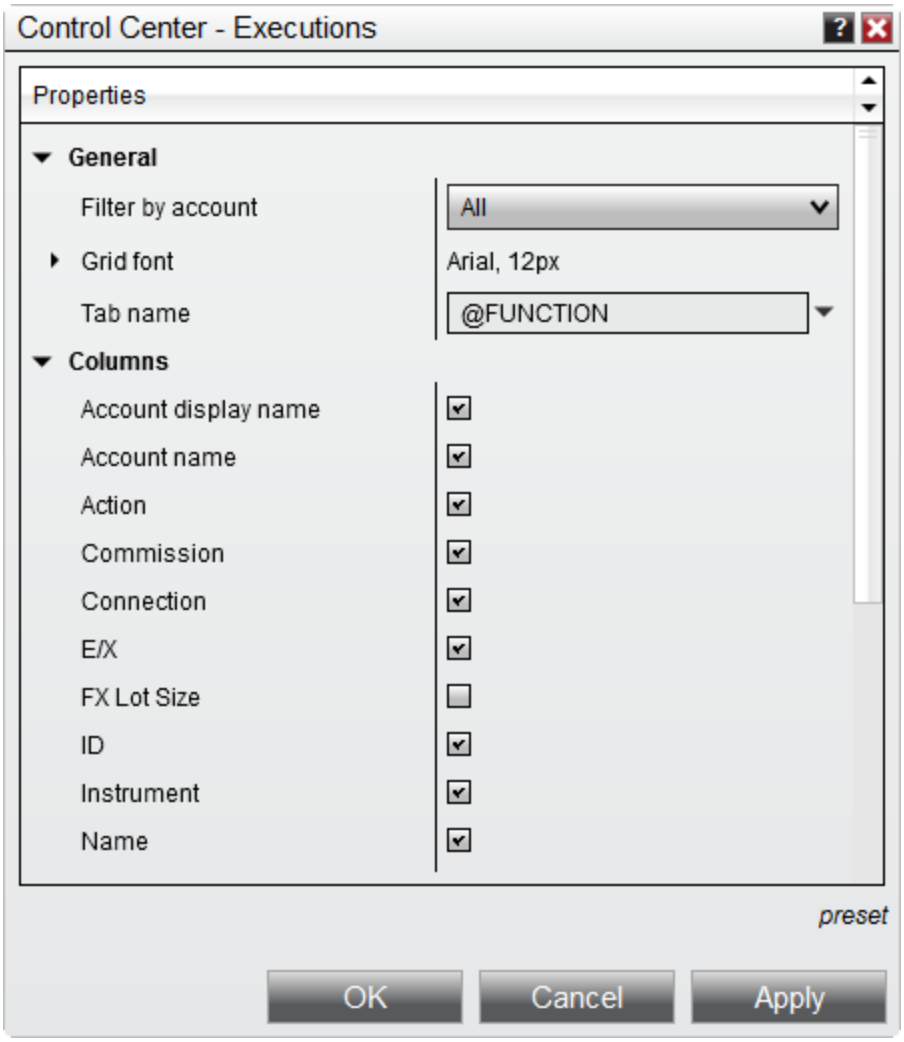
> **Note**: NinjaTrader connection setting "Auto subscribe to required instruments" must be enabled when creating your forex connection for NinjaTrader to automatically get the currency rates needed for PnL conversion when the trade execution occurs. For more information please see the connection guide for your provider on how to enable this property.

Account Lot Size

Executions track the Account Lot Size used for the account when the execution occured. This is used for accurate Pip PnL calculations as a 1 pip gain in EURUSD for a 10,000 QTY sized mini lot trader is different then a 1 pip gain in

EURUSD for a 1,000 QTY sized micro lot trader. Account Lot Size is used by NinjaTrader to normalize your Pip PnL reporting so that it is accurate to your accounts base Forex lot size. The Account Lot Size is normally provided from your broker automatically, however if the broker does not send the account lot size then the connection settings for the account in NinjaTrader will have an option for you to define the property "Forex lot size".

Executions tab properties



Executions Tab Properties

General	

Filter by account	Filters orders by selected account
Grid font	Sets the font for the order grid
Tab name	Sets the tab name
Columns	Sets that columns are enabled or disabled in the order grid.

How to set preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original factory settings, you can left mouse click on the **preset** text and select the option to **restore** to return to the original factory settings - please note though that you cannot save a custom default to restore to.

Note: A number of pre-defined variables can be used in the "Tab Name" field. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

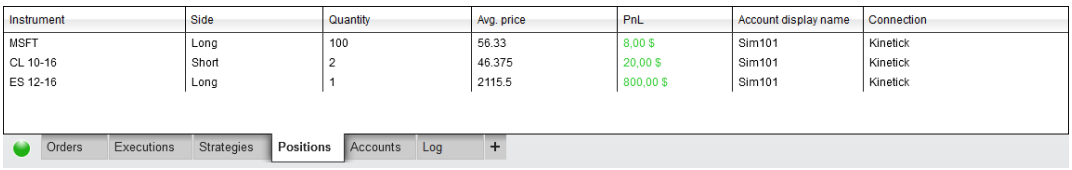
10.8.9 Positions Tab

The Positions tab displays the current open positions in a data grid.

Understanding the position tab

Positions Display

Open positions are displayed in the data grid.



The table below lists what connections natively or locally calculate positions.

Connectivity Provider	Position calculation
Continuum	Native
CQG	Native
cTrader	Native
Forex.com/City Index	Native
FXCM	Local
Interactive Brokers	Native
Rithmic	Native
TD Ameritrade	Native

Columns can be re-ordered and re-sized at will, and individual columns can be enabled or disabled via the **Properties** window accessible in the **Positions** grid's Right-Click menu. The following columns are displayed in the **Positions** grid by default:

Instrument	The instrument in which the position is held
Side	Indicates whether the position is held on the Long or Short side

Quantity	The quantity held in the position
Avg. Price	The average fill price of the entry orders filled to enter or increase the position
PnL	The current unrealized profit or loss of the position
Account Display Name	The Display Name of the account
Connection	The connection used to enter the position

The following additional columns can be applied through the grid's **Properties** window:

Account Name	The "Account Name" -- not to be confused with the "Account Display Name." These two can differ for live brokerage accounts, and the "Account Display Name" tends to be more descriptive.
Close	Contains a button which will allow you to close the position
Working Buys	The number of unfilled Buy orders currently resting on the account
Working Sells	The number of unfilled Sell orders currently resting on the account

Right Click Menu

Right mouse clicking within the positions grid section opens the following menu:

Apply ATM Strategy	Allows you to apply a predefined ATM Strategy Template to an open position using the current market price as the entry price.
Close Position*	Flattens the currently selected position in the grid and cancels any working orders associated to the position's instrument
Flatten Everything*	Flattens all open positions and cancels all working orders
Roll Position	Send a Market order to exit the position in the current contract month and send another Market order in the next contract month to roll your position.
Filter By Account	Sets which positions to display by account
Always On Top	Sets if the window will be always on top of other windows

Show Tabs	Sets if the window should allow for tabs
Export	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Select to print either the window or the order grid area.
Share	Select to share via your share connections.
Properties...	Configure the positions grid properties

*The Close Position and Flatten Everything functions are not guaranteed. (See the "*Risks of Electronic Trading with NinjaTrader*" section for more information)

Position tab properties

Position Tab Properties

General	
Filter by account	Filters orders by selected account
Grid font	Sets the font for the order grid

Tab name	Sets the tab name
Columns	Sets that columns are enabled or disabled in the order grid.
Window	Sets that window management features are enabled or disabled

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original factory settings, you can left mouse click on the **preset** text and select the option to **restore** to return to the original factory settings - please note though that you cannot save a custom default to restore to.

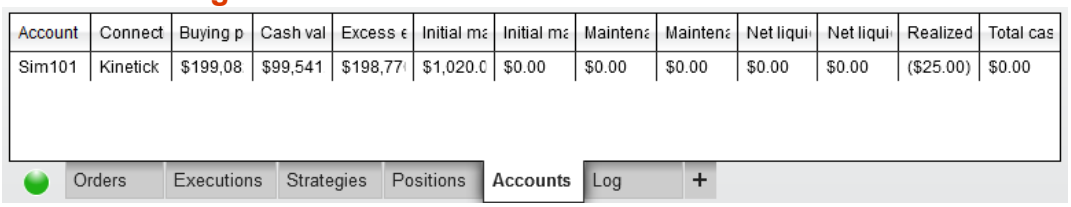
Note: A number of pre-defined variables can be used in the "Tab Name" field. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.8.10 Accounts Tab

The Accounts tab displays current account information in a data grid. The account values that are displayed is dependent on your connectivity provider. Not all connectivity providers transmit complete account data.

Understanding the accounts tab

Understanding the Accounts Tab



Columns can be re-ordered and re-sized at will, and individual columns can be enabled or disabled via the **Properties** window accessible in the **Accounts** grid's Right-Click menu. The following columns are displayed in the **Accounts** grid by default:

Connection	The connection through which the account is accessed
Display Name	The account's Display Name

Buying Power	The account's buying power, taking margin into account
Cash Value	The cash value of the account
Excess Intraday Margin	Available (unused) intraday margin on the account
Excess Initial Margin	Available (unused) initial margin on the account
Intraday Margin	The margin enforced for positions held intraday
Initial Margin	The percentage of the purchase price that must be covered by the account's cash value
Maintenance Margin	The minimum amount of cash that must be maintained in the account
Excess Maintenance Margin	The cash value of the account above the Maintenance Margin value
Net Liquidation	The total worth of the account's assets (including margin)
Gross Realized PnL	Realized profit or loss before commissions have been subtracted
Realized PnL	Realized profit or loss after commissions have been subtracted

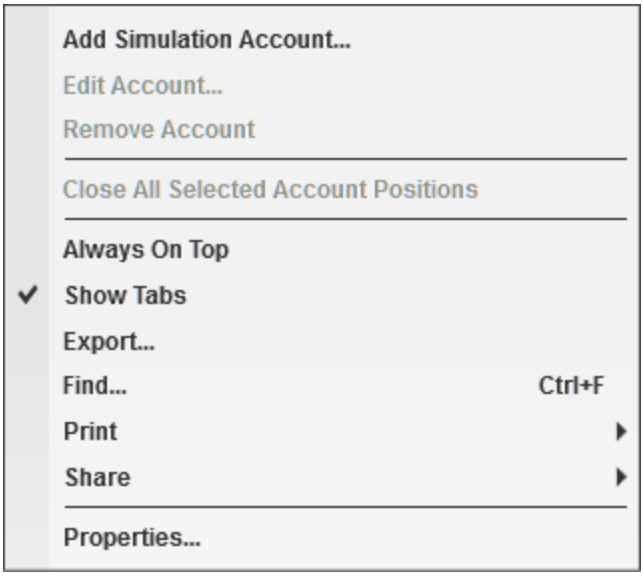
The following additional columns can be applied through the grid's **Properties** window:

Close	Contains a button which will allow you to close the position
Commission Name	The name of the Commission Template applied to the account
Filled Buys	The number of Buy orders filled in the current session
Filled Sells	The number of Sell orders filled in the current session
Look Ahead Maintenance Margin	Projected maintenance margin requirement as of the next period's margin change
Name	The name of the account. This can differ from the account's Display Name
Net Liquidation by Currency	Same as Net Liquidation, but for individual currencies
Position	The quantity held in all open positions on the account
Total Cash Balance	The total cash available in the account including any debits or credits
Total Commissions	The total commissions paid in the account for the session
Total PnL	The sum of Unrealized PnL + Realized PnL
Unrealized PnL	Total unrealized profit or loss for open positions on the account

Working Buys	The number of unfilled Buy orders currently resting on the account
Working Sells	The number of unfilled Sell orders currently resting on the account

Right Click Menu

Right mouse clicking within the positions grid section opens the following menu:

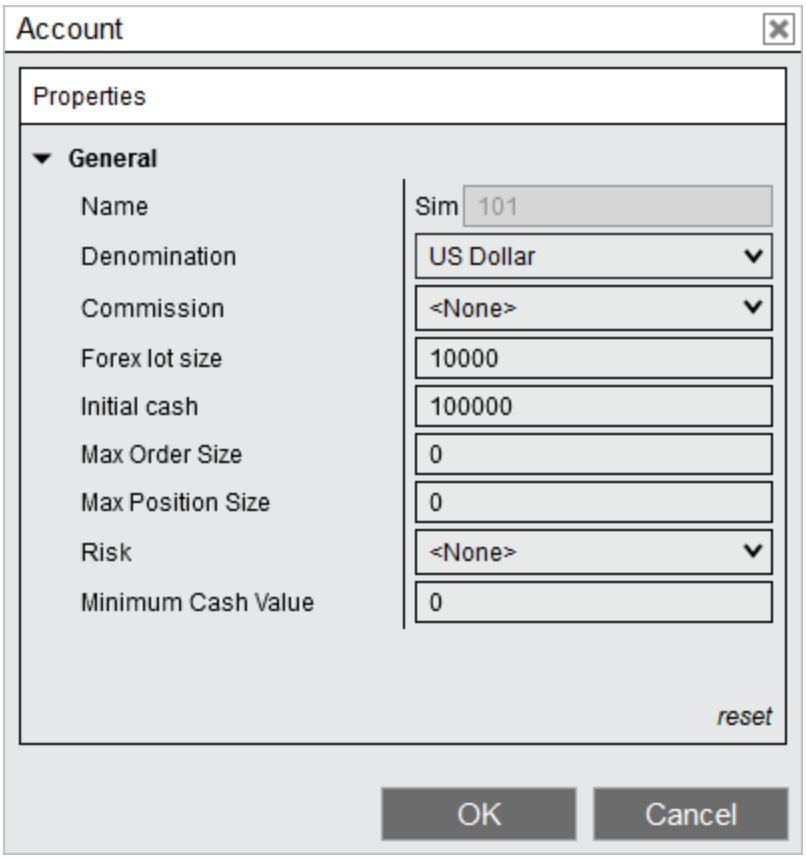


Add Simulation Account	Opens the Account window to configure a new simulation account
Edit Account	Opens the Account window to edit the selected account (See the Edit Account Menu section below)
Remove Account	Removes the selected account (**Note**: The Sim101 account cannot be removed)
Close All Selected Account Positions	Closes all selected positions on the selected account

Always on Top	Sets the Control Center window to always be on top of other windows
Show Tabs	Used to enable or disable tabs in the Control Center
Export	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Select to print either the window or the order grid area.
Share	Select to share via your share connections.
Properties...	Configure the positions grid properties

Edit Account Menu

Right mouse clicking within the positions grid section then selecting Edit Account opens the following menu:



Name	Displays the name of the account
Denominatio n	*Sets what currency to display the account values. Must be set to the currency the account is in.
Commission	Sets what commissions template to apply
Forex lot size	Sets the default order quantity for forex
Initial cash	Sets the initial cash balance for the simulation account
Max order size	Sets a limit for order size

Max position size	Sets a limit for position size
Risk	Sets what risk template to apply
Minimum cash value	Sets the minimum cash value that must be available to place an order

Notes:
The Playback101 account cannot be edited, it inherits its settings from the Sim101.
Adjusting the denomination requires a reconnect.
*Some providers will always display the denomination in US Dollars.
Not all properties display for all accounts.

Account Values Supported by Provider

The account values that are displayed depend upon your connectivity provider. Some connectivity providers transmit partial account data, while others do not transmit anything. Below is a table of the various account values displayed by different connectivity providers.

Connectivity Provider	Buying Power	Cash Value	Excess Intraday Margin	Excess Initial Margin	Intraday Margin	Initial Margin	Net Liquidation	Realized PnL	Unrealized PnL	Total PnL
NinjaTrader		✓	✓	✓	✓	✓	✓	✓	✓	
Local Simulation Accounts (values are all NinjaTra		✓	✓	✓	✓	✓	✓	✓	✓	✓

der calculated since its a simulation account)										
Continuum (values **exclude** commissions)		✓	✓*	✓*	✓*	✓*	✓	✓	✓*	
CQG (values **exclude** commissions)		✓					✓	✓		
FOREX.com		✓				✓		✓	✓	✓
FXCM		✓						✓		
Interactive Brokers (values **include** commissions)	✓	✓				✓	✓	✓		✓
Rithmic (values **exclude** commissions)		✓					✓	✓	✓	
TD Amerit	✓	✓						(NinjaTra		

rade (values **exclude** commissions)								der calculated)		

*With NinjaTrader Brokerage

Notes:
- If a Connectivity Provider supplies Unrealized PnL in NinjaTrader, but a risk template is applied (example: local simulation accounts) the Unrealized PnL will be calculated locally.
- Interactive Brokers reports forex trades in the instrument currency, not the account currency, without commissions applied. So, forex trades are not applied to Realized PnL.
- If a commissions template is applied, some values may include commissions, locally calculated.

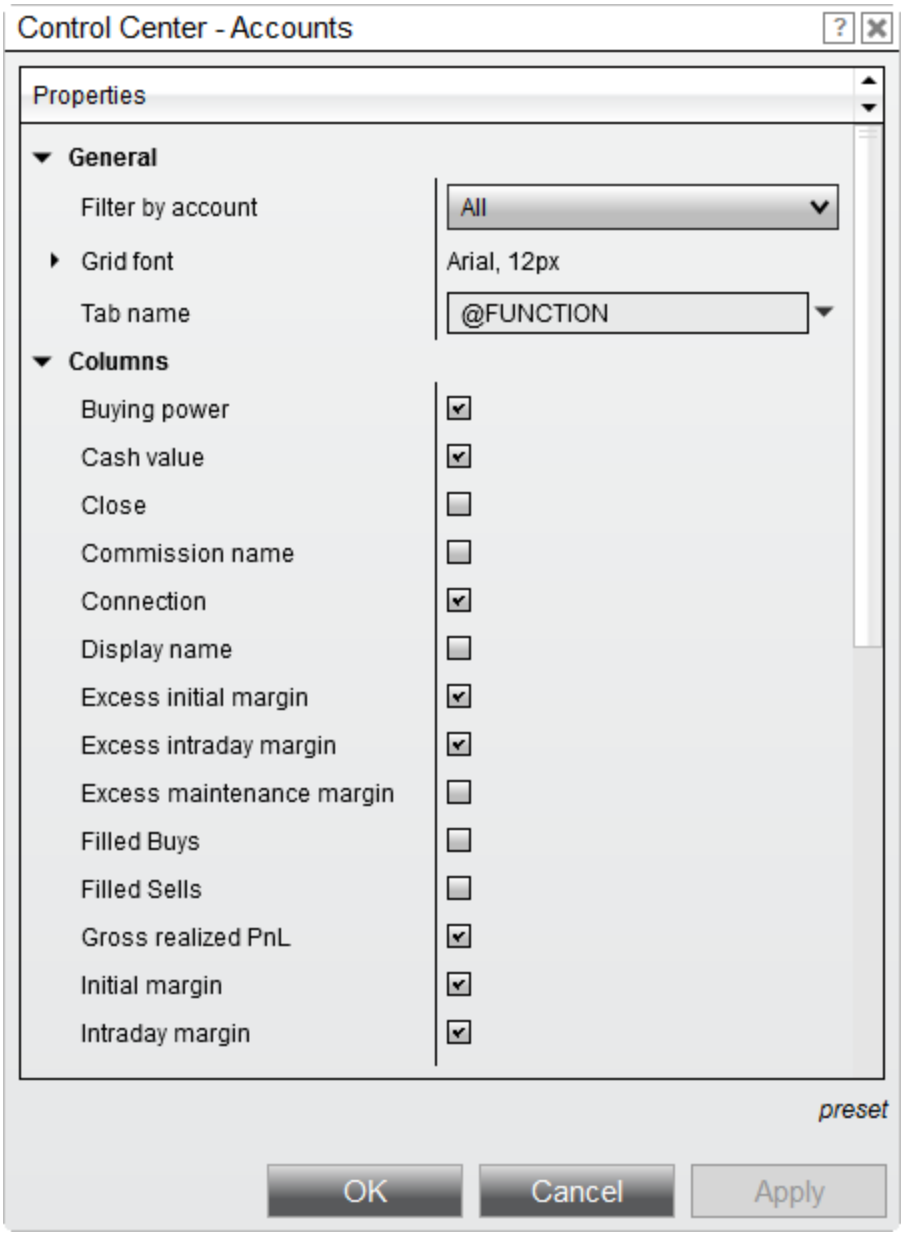
Understanding Currency Conversion

NinjaTrader will attempt to convert currency for forex and futures trades.

- Forex trades will be made for any currency pair that has a cross rate and that cross rate data is available on your data feed.
- Futures trades we use the CME FX futures (6A, 6B, 6E, etc) to make the conversion as long as you have access to that data from your data feed provider.

Notes:
- Commissions in the Account tab are calculated based on the Commission Template applied to the account in your installation of NinjaTrader, and not pulled from any Data Provider.
- Custom Margin Templates cannot be applied to live accounts
- Due to the CME FX Futures being mostly limited to US cross rates conversion will only occur to US Dollar account denomination for futures trades.

Accounts tab properties



Accounts Tab Properties

General	
Filter by account	Filters accounts display by selected account
Grid font	Sets the font for the accounts grid

Tab name	Sets the tab name
Columns	Sets that columns are enabled or disabled in the accounts grid.

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original factory settings, you can left mouse click on the **preset** text and select the option to **restore** to return to the original factory settings - please note though that you cannot save a custom default to restore to.

Note: A number of pre-defined variables can be used in the "Tab Name" field. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

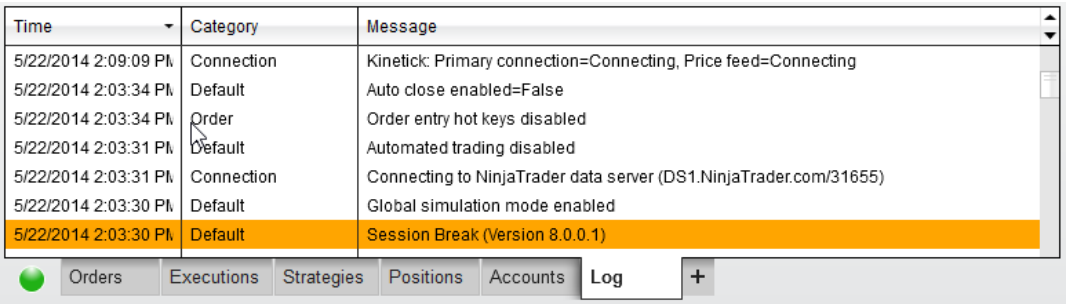
10.8.11 Log Tab

The Log tab displays historical application and trading events for the current day in a data grid.

Understanding the log tab

Log Display

Log events are categorized and color coded based on four distinct alert levels; Information, Warning, Error and Alert.



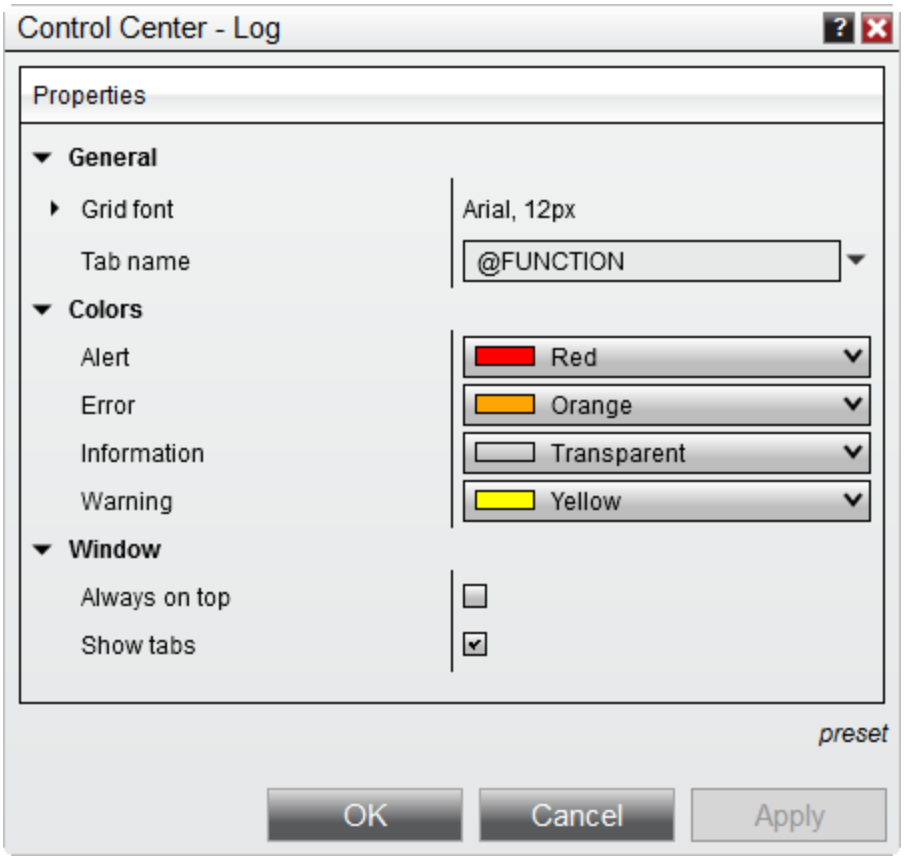
Each log event is displayed by date, category and message. In some cases, the length of the message may be larger than the width of the "Message" column. In this situation, you can hover your mouse above the message in order to have it display in a pop-up type window.

Right Click Menu

Right mouse clicking within the log display section opens the following menu:

Always On Top	Sets if the window will be always on top of other windows
Show Tabs	Sets if the window should allow for tabs
Export	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Select to print either the window or the log grid area.
Share	Select to share via your share connections.
Properties...	Configure the positions grid properties

Log tab properties



Log Tab Properties

General	
Grid font	Sets the font for the log grid
Tab name	Sets the tab name
Colors	
Alert	Sets the color used for background for **Alert** log messages
Error	Sets the color used for background for **Error** log messages

Information	Sets the color used for background for **Information** log messages
Warning	Sets the color used for background for **Warning** log messages
Window	Sets that window management features are enabled or disabled

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original factory settings, you can left mouse click on the **preset** text and select the option to **restore** to return to the original factory settings - please note though that you cannot save a custom default to restore to.

Notes:
- A number of pre-defined variables can be used in the "Tab Name" field. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.
- Columns cannot be resorted or removed.

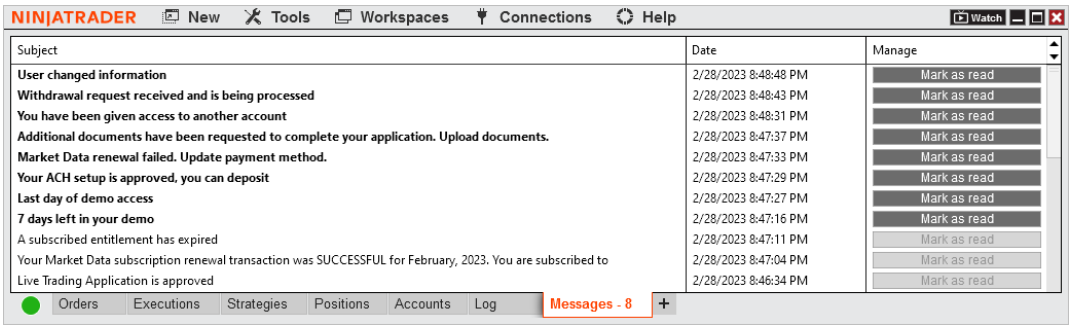
10.8.12 Messages Tab

The Messages tab displays messages in relation to your account in a data grid.

Understanding the Messages tab

Messages Display

Unread messages will be displayed in bold and have the **Mark as read button** enabled. The Date and time the message was sent is included. The tab will display in a highlighted color if there are unread messages and indicate how many unread messages there are.



Right Click Menu

Right mouse clicking within the log display section opens the following menu:

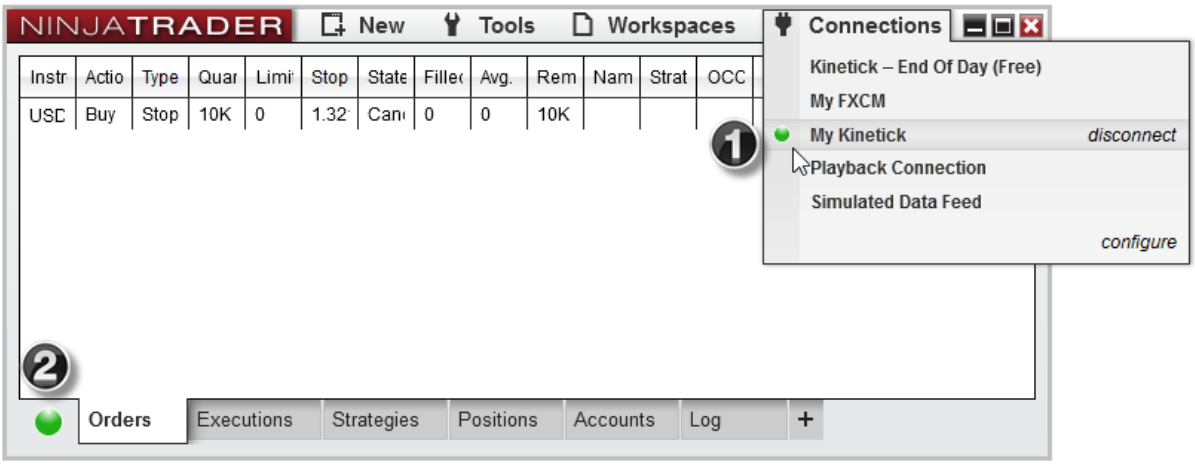
Mark All As Read	Marks all unread messages as read
Always On Top	Sets if the window will be always on top of other windows
Show Tabs	Sets if the window should allow for tabs
Export	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Select to print either the window or the log grid area.

Share	Select to share via your share connections.
Properties...	Configure the positions grid properties

10.8.13 Connection Status

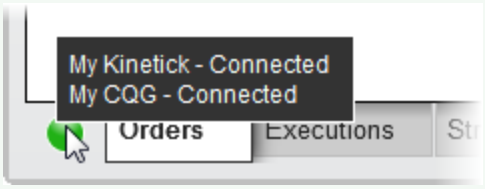
Connection Status

The connection status is reported in the connections menu per provider. There is also an aggregated connection status in the bottom left hand corner of the **Control Center.**



1. Connection Status icon inside the Connections menu
2. Connection Status icon displayed in the Control Center

Tip: If you're using multiple connections, hovering your mouse cursor above the connection status will show a tool tip which will give you the individual status of each connection.



Please see the following connection states:

	Connecte	Indicates that NinjaTrader is fully connected

	d	
	Connecting	Indicates that NinjaTrader is attempting to connect
	Connection Lost (Price Server)	Indicates that NinjaTrader has lost connection to the price server
	Connection Lost (Order Server)	Indicates that NinjaTrader has lost connection to the order server
	Disconnected	Indicates that NinjaTrader is not connected

10.9 Database

Database Window Overview

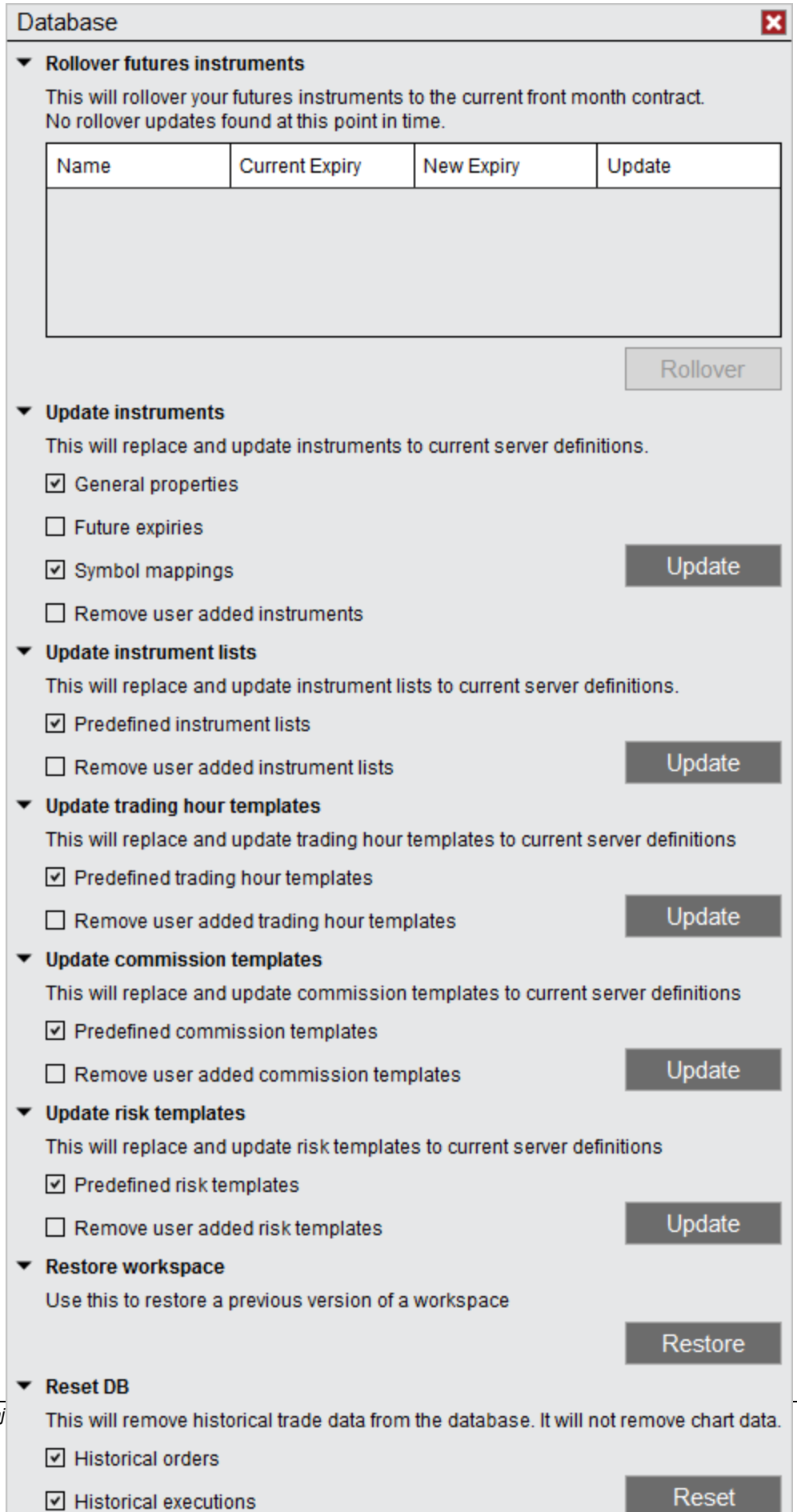
You can access the **Database** window by left mouse clicking in the **Tools** menu within the NinjaTrader Control Center and selecting the menu item **Database...**

The **Database** window allows for the centralized management of all database related functions.

› Database Operations

10.9.1 Database Operations

Various database operations can be performed from the **Database** window.


Database
Rollover futures instruments
This will rollover your futures instruments to the current front month contract.
No rollover updates found at this point in time.
Name
Current Expiry
New Expiry
Update
Rollover
Update instruments
This will replace and update instruments to current server definitions.
General properties
Future expiries
Symbol mappings
Remove user added instruments
Update
Update instrument lists
This will replace and update instrument lists to current server definitions.
Predefined instrument lists
Remove user added instrument lists
Update
Update trading hour templates
This will replace and update trading hour templates to current server definitions
Predefined trading hour templates
Remove user added trading hour templates
Update
Update commission templates
This will replace and update commission templates to current server definitions
Predefined commission templates
Remove user added commission templates
Update
Update risk templates
This will replace and update risk templates to current server definitions
Predefined risk templates
Remove user added risk templates
Update
Restore workspace
Use this to restore a previous version of a workspace
Restore
Reset DB
This will remove historical trade data from the database. It will not remove chart data.
Historical orders
Historical executions
Reset

Rollover futures instruments

Rollover Futures Instruments

This will rollover your futures instrument to the most recent expiries across all open workspaces. If there are instruments open in the workspace that are eligible to be rolled forward they will be shown in the grid. For more information on this process please see the rolling over a futures contract section of the help guide.

Update instruments

Update Instruments

This will replace and update all instruments to current server definitions.

The NinjaTrader data server maintains definitions for Instruments, NinjaTrader will update your local instrument general properties and symbol mappings should there be any changes on the server automatically. However if you make any changes to the either the general properties or the symbol mappings for an instrument then the instrument would no longer be automatically updated. Using this utility will remove any custom changes you have made locally and update the instrument definition to the servers version.

General properties	Sets if the General properties of each instrument will be replaced with the server definition
Future expiries	Sets if the Contract Months of each Future instrument will be replaced with the server definition
Symbol mappings	Sets if the Symbol mappings of each instrument will be replaced with the server definition
Remove user added instruments	Sets if instruments created by the user will also be

	removed

Note: If you would like to report an incorrect or missing server definition please send an email to platformsupport@ninjatrader.com and we will promptly correct the issue.

Update instrument lists

Update Instruments Lists

This will replace and update all instrument lists to current server definitions.

The NinjaTrader data server maintains definitions for the following instrument lists:

- Dow 30
- FOREX
- Futures
- Indexes
- NASDAQ 100
- SP 500

Any time there is an instrument added or removed to the above instrument lists they will be automatically updated by NinjaTrader. However if you add or remove instruments from the list manually using the **Instrument Lists** window then the instrument list will no longer continue to be updated automatically by NinjaTrader. Should you wish to update and reset your instrument lists manually then you would use the following update utility.

Predefined instrument lists	Sets if the General properties of each instrument will be replaced with the server definition
Remove user added instrument lists	Sets if instruments created by the user will also be removed

Note: If you would like to report an incorrect or missing server definition please send an email to platformsupport@ninjatrader.com and we will promptly correct the issue.

Update trading hour templates

Update Trading Hour Templates

This will replace and update all trading hour templates to current server definitions.

Update commissions templates

Update Commissions Templates

This will replace and update all commissions templates to current server definitions.

Update risk templates

Update Risk Templates

This will replace and update all risk templates to current server definitions.

Restore workspace

Restore Workspace

This can be used to restore a previous saved version of a saved workspace. When selecting **Restore** you will be directed to the location of the recovery workspaces. They will be organized in folders with the names of the workspaces. Double click the folder of the workspace you want to recover and then select the recovery file with the date/time that you want to recover.

By default 10 previously saved versions of your workspaces will be retained. To modify how many recovery versions are available within the Control Center go to **Tools** and select **Options**. Within the **General** section go to **Preferences** and set the **Versions of recovery workspaces**. See the Options General section.

Note: Deleted workspaces cannot be restored using this feature. However, if you have a backup you could restore it from there. See the Backup & Restore section.

Reset DB

Reset the Database

This will remove historical trade data from the database. It will not remove chart data or reset any Simulation accounts.

Historical orders	Sets if the historical orders stored in the database will be removed
Historical executions	Sets if the historical executions in the database will be removed

Repair DB

Repair the Database

This should only be used when directed by NinjaTrader support and performs a repair on the NinjaTrader database. Depending on the size of the database this can take a few moments to complete.

10.10 Data Grids

Data Grids Overview

Data grids are customizable tables which are used to display a multitude of information throughout various product features.

› Working with Data Grids

10.10.1 Working with Data Grids

All data grids found throughout NinjaTrader are customizable.

Data Grids

With a data grid you can:

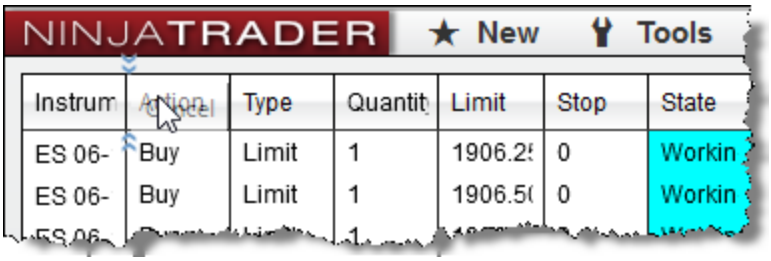
- Resize columns
- Enable and Disable columns
- Export data to Excel
- Save data as a CSV file
- Email the data as an image
- Print data
- Search data

Right Click Menu

Right mouse clicking within any data grid will bring up a menu with several grid actions. You may want to export your execution history to Excel for further analysis as an example. Simply go to any grid displaying execution history, right mouse click and select the **Export...** menu item. Here you can choose the file type and file name to export, select either "CSV" or "Excel".

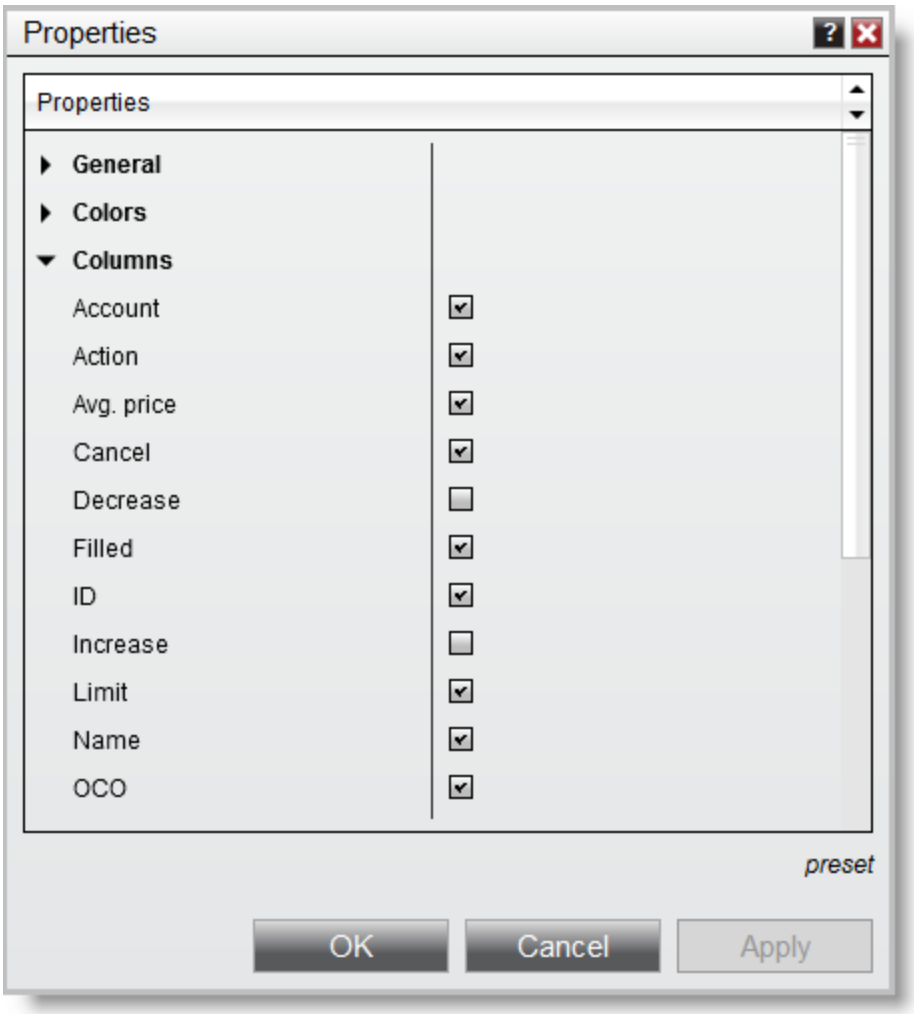
Moving Columns

To adjust the order of columns within a data gird, left mouse click and hold on the column header and drag it to the location you wish to place the selected column. Two blue arrows will appear above and below the location in the grid you will be moving the column to (see image below). Release the mouse button to place the column in the new location.



Enabling and Disabling Columns

Selecting the **Properties** menu item of the right mouse click of the window will give you thew windows properties. In the properties window under the "**Column**" category you can choose which columns you wish to show (make visible). Any column name with a check-mark next to it will be visible and un-checking a column name will take the column out of the data grid.



10.11 Depth Chart

Depth Chart Window Overview

You can access the **Depth Chart** window from within the NinjaTrader Control Center window by left mouse clicking on the menu **New**, and then selecting the menu item **Depth Chart**.

The **Depth Chart** window is available for futures and cryptocurrency instruments. It displays the current book bid and ask volume data in a cumulative manner. It is used to gauge cumulative strength and depth on either side of the market.

10.11.1 Using the Depth Chart Window

Selecting a Cryptocurreny Instrument

There are multiple ways to select an Instrument in the **Depth Chart** window.

- Right clicking on the **Depth Chart** window and selecting the menu **Instruments**.
- With the **Depth Chart** window selected begin typing the instrument symbol directly on the keyboard. Typing will trigger the **Overlay Instrument Selector.**

For more Information on instrument selection and management please see Instruments section of the Help Guide.

Understanding the layout of the Depth Chart window

1 Quotes

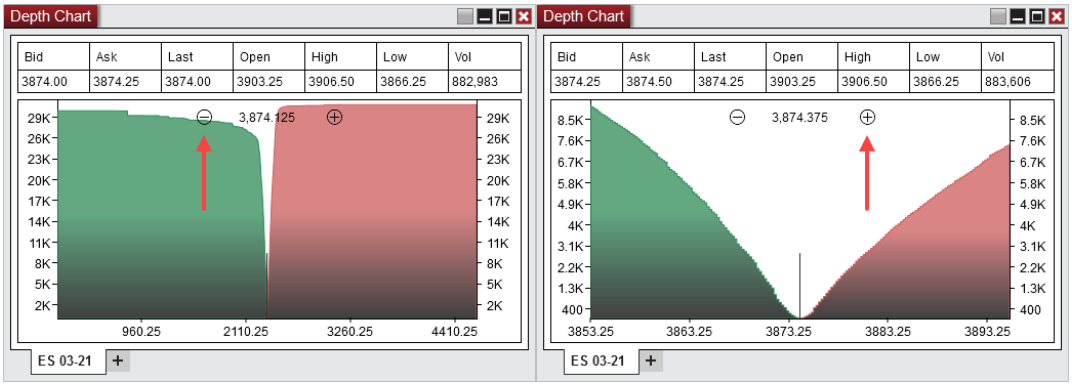
The Quotes section displays various market data items.

Bid	The current bid price
Ask	The current ask price
Last	The current last traded price
Open	The current sessions open price
High	The current sessions high price
Low	The current sessions low price
Vol	The current sessions total volume.

You can disable the Quotes section by clicking on your right mouse button and deselecting the menu item **Show Quotes**.

2 Graph

The Graph displays the cumulative buy and sell orders through the full available book, so the vertical Y-axis value at any point is the result of summing all bids (asks) from the best bid (ask) to the price value on the horizontal X axis.



You can use the +/- zoom buttons on the chart to set the zoom level (right click on the chart > Reset zoom to reset the level to default).

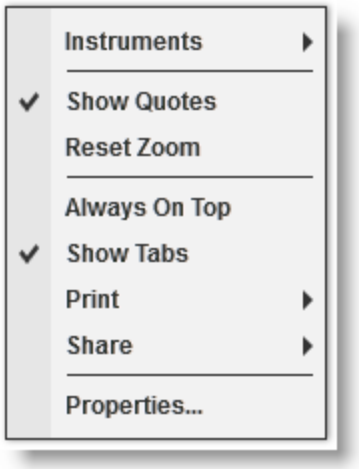
3 Tooltip

The tooltip displays:

- the selected Depth chart x-axis value
- the associated cumulative volume up to that value
- the currency cost of the cumulative volume

Right Click Menu

Right mouse click on the **Depth Chart** window to access the right click menu.



Instruments	Selects the instrument
Show Quotes	Sets if the quotes section is displayed
Reset Zoom	Resets the zoom level to default
Always On Top	Sets if the window should be always on top of other windows
Show Tabs	Sets if the window will allow for tab support
Print	Displays Print options
Share	Displays Share options
Properties...	Sets the Depth Chart Properties

Using Tabs

The **Depth Chart** window is a tabbed interface, this gives you the ability to have multiple **Depth Chart** tabs configured in the same window. Please see the Using Tabs section of the help guide for more information.

10.11.2 Depth Chart Properties

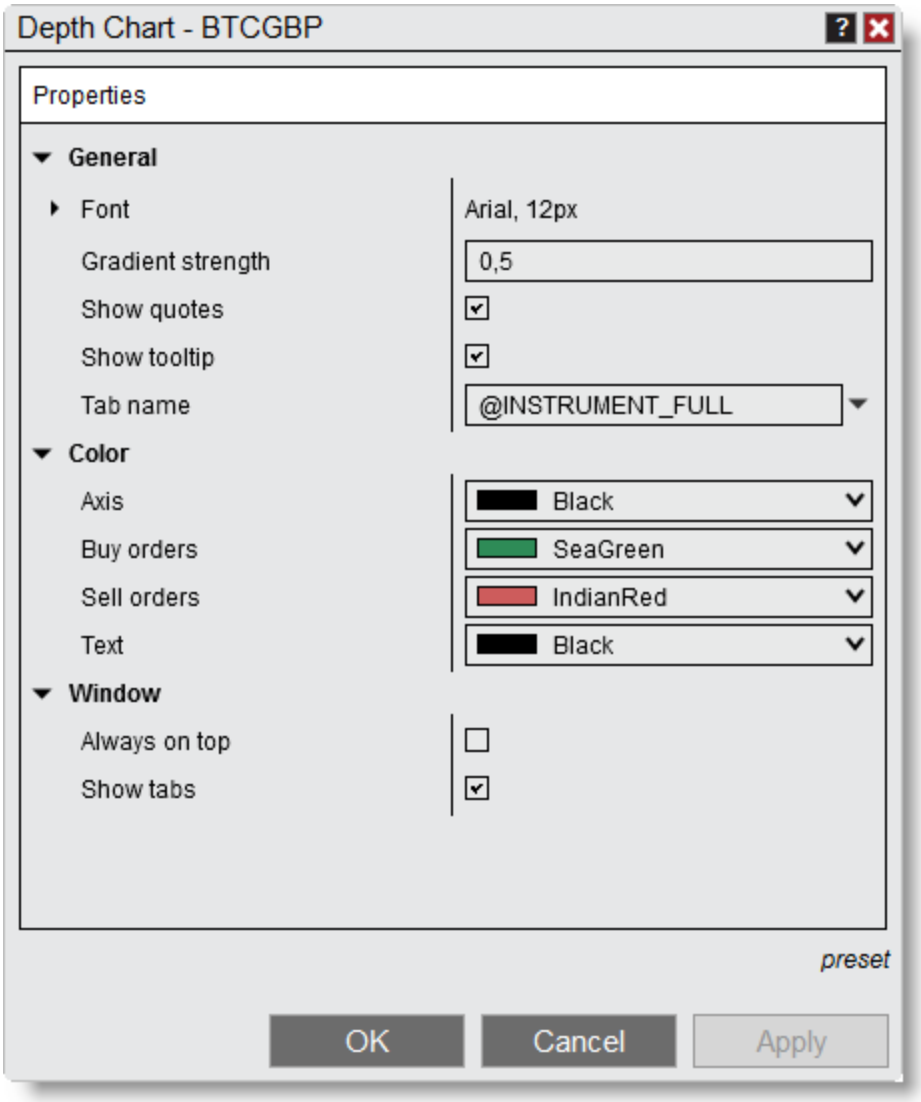
The Depth Chart window can be customized through the **Depth Chart Properties** window.

How to access the Depth Chart Properties window

You can access the Depth Chart properties dialog window by clicking on your right mouse button and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the Depth Chart Properties window:



Property Definitions

General	
Font	Sets the font options
Gradient strength	Sets how strong the gradient coloring effect should be, default is 0.5 - possible range of values is 0.0 - 1.0 (0.0 deactives the gradient)

Show quotes	Sets if the quotes section is displayed
Show tooltip	Sets if the data tooltip is displayed
Tab name	Sets the name of the tab, please see Using Tab Name variables for more information.
Color	
Axis	Sets the axis color
Buy orders	Sets the color for tracking buy orders
Sell orders	Sets the color for tracking sell orders
Text	Sets the text color
Window	
Always on top	Sets if the window will be always on top of other windows.
Show tabs	Sets if the window will allow for tab support

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

Using Tab Name Variables

Tab Name Variables

A number of pre-defined variables can be used in the "Tab Name" field of the **Depth Chart Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.11.3 Window Linking

Please see the Window Linking section of the Help Guide for more information on linking the **Depth Chart** window.

10.12 FX Correlation

FX Corerlation Window Overview

You can access the **FX Correlation** window from within the NinjaTrader Control Center window by left mouse clicking on the menu **New**, and then selecting the menu item **FX Correlation**.

The **FX Correlation** window is available for Forex instruments and displays instruments move in a similar direction, opposite direction, or have no correlation.

- Using the FX Correlation Window
- FX Correlation Properties
- Window Linking

10.12.1 Using the FX Correlation Window

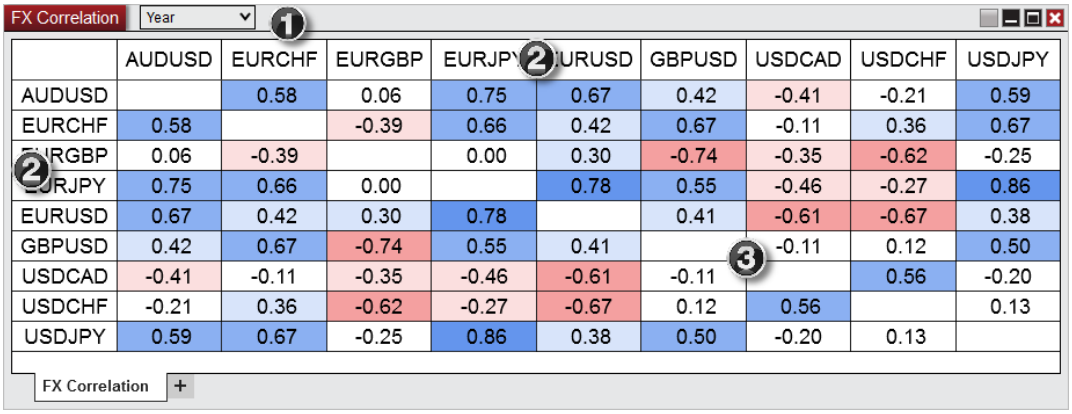
Selecting a Forex Instrument

There are multiple ways to select a Forex Instrument in the **FX Correlation Chart** window.

- Right clicking on the **FX Correlation** window and selecting the menu **Add Instrument(s)**.
- With the **FX Correltion** window selected begin typing the instrument symbol directly on the keyboard. Typing will trigger the **Overlay Instrument Selector.**

For more Information on instrument selection and management please see Instruments section of the Help Guide.

Understanding the layout of the FX Correlation window



	AUDUSD	EURCHF	EURGBP	EURJPY	EURUSD	GBPUSD	USDCAD	USDCHF	USDJPY
AUDUSD		0.58	0.06	0.75	0.67	0.42	-0.41	-0.21	0.59
EURCHF	0.58		-0.39	0.66	0.42	0.67	-0.11	0.36	0.67
EURGBP	0.06	-0.39		0.00	0.30	-0.74	-0.35	-0.62	-0.25
EURJPY	0.75	0.66	0.00		0.78	0.55	-0.46	-0.27	0.86
EURUSD	0.67	0.42	0.30	0.78		0.41	-0.61	-0.67	0.38
GBPUSD	0.42	0.67	-0.74	0.55	0.41		-0.11	0.12	0.50
USDCAD	-0.41	-0.11	-0.35	-0.46	-0.61	-0.11		0.56	-0.20
USDCHF	-0.21	0.36	-0.62	-0.27	-0.67	0.12	0.56		0.13
USDJPY	0.59	0.67	-0.25	0.86	0.38	0.50	-0.20	0.13	

1 Range

The Range drop down menu indicates over how long of a period to calculate the correlation for the instruments. It is based of of minute data so selecting **Year** would be 1 minute data over the past year.

2 Correlation instruments

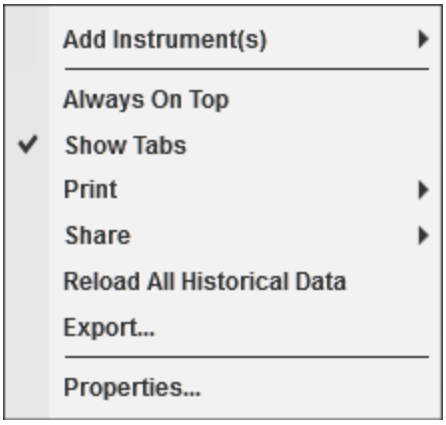
The correlation instruments are the instruments that will be compared against each other.

3 Correlation values

Correlation values closer to 1 indicate that the instruments move in the same direction. Values closer to -1 indicate that the instruments move in the opposite direction. Value closer to 0 indicate no correlation.

Right Click Menu

Right mouse click on the **FX Correlation** window to access the right click menu.



Add Instruments	Selects the Forex instrument to add
Always On Top	Sets if the window should be always on top of other windows
Show Tabs	Sets if the window will allow for tab support
Print	Displays Print options
Share	Displays Share options
Reload All Historical Data	Reloads the historical bar data used for Indicator calculations
Export...	Exports the **FX Correlaion** contents to "CSV" or "Excel" file format
Properties...	Sets the FX Correlation Properties

Using Tabs

The **FX Correlation Chart** window is a tabbed interface, this gives you the ability to have multiple **FX Correlation Chart** tabs configured in the same window. Please see the Using Tabs section of the help guide for more information.

10.12.2 FX Correlation Properties

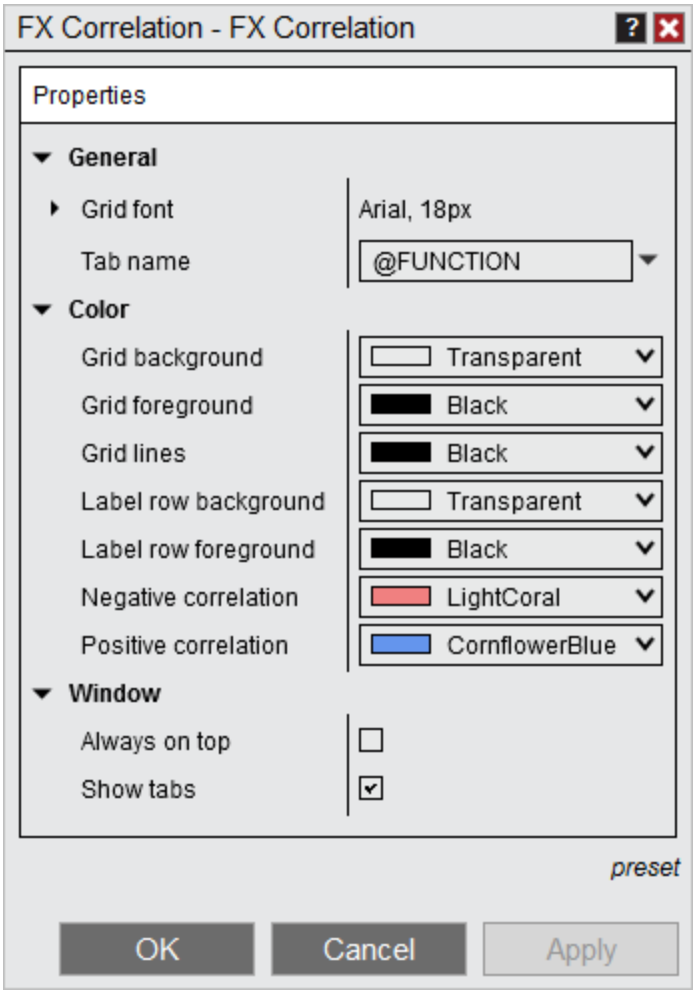
The FX Correlation window can be customized through the **FX Correlation Properties** window.

How to access the FX Correlation Properties window

You can access the FX Correlation properties dialog window by clicking on your right mouse button and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the FX Correlation Properties window:



Property Definitions

General	
Grid Font	Sets the font options
Tab name	Sets the name of the tab, please see Using Tab Name variables for more information.
Color	
Grid background	Sets the default color of the display grid background
Grid foreground	Sets the default color of the text in a cell
Grid lines	Sets the color of grid lines
Label row background	Sets the default color for the Label row background
Label row foreground	Sets the default color for the Label row foreground
Negative correlation	Sets the color for negative value correlations
Positive correlation	Sets the color for positive value correlations
Window	
Always on top	Sets if the window will be always on top of other windows.

Show tabs	Sets if the window will allow for tab support

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

Using Tab Name Variables

Tab Name Variables

A number of pre-defined variables can be used in the "Tab Name" field of the **FX Correlation Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.12.3 Window Linking

Please see the Window Linking section of the Help Guide for more information on linking the **FX Correlation** window.

10.13 Historical Data

Historical Data Window

The **Historical Data Window** can be accessed by left mouse clicking the **Tools** menu within the Control Center and selecting the menu item **Historical Data..**

The **Historical Data Window** provides access to all historical data and Market Replay data used in NinjaTrader as supplied from your historical market data provider and/or collected from a real-time data feed. The option to import, export, edit and download historical data are available within the **Historical Data Window** window.

- Importing
- Exporting
- Editing
- Download

10.13.1 Loading Historical Data

NinjaTrader has 3 levels of data access: From provider, from cache, and from memory.

1. From provider is naturally the slowest, since data has to be transferred over the internet from your connected data provider.
2. From cache is much faster, since we cache the data to the hard drive and load from there on any subsequent request.
3. From memory is the fastest and typically results in "instant" chart loads, which is possible since the data is already in use by NinjaTrader so we don't have to make a trip to the cache or the provider.

The reality is for any chart load, there typically will be data returned from multiple levels of data access. Where NinjaTrader will load as much as possible from the memory or cache and make a request to fill any gap to the provider.

▼ The NinjaTrader memory and speeding up the loading of data

Data that is currently being used will be in the memory and will first be used to populate your charts. Additionally, to minimize the need to load data and to speed up chart load times, NinjaTrader maintains an internal cache of your prior accessed data. When data is in this cache, NinjaTrader will use it to populate your charts instead of loading from your data provider.

There are two ways to ensure that the memory contains data for your instrument of interest:

1. Load the instrument into a Market Analyzer window along with an indicator column with the same chart parameters you plan to be loading.
2. Open and maintain a chart with the same data type and days to load that you plan on loading.

When does NinjaTrader download historical data?

NinjaTrader loads data from your data provider whenever it determines it could potentially not have all the data pertaining to the requested time period.

NinjaTrader will load data when:

1. The **End date** parameter of the Data Series window contains the current day (this results in the current and prior day downloading)
2. The most recent day of data in your data request is not available in the data repository (this results in the most recent day from your data request and prior day downloading)
3. When the oldest day of data in your data request is not available in the data repository or it only goes up to that date (this results in all requested historical trading day data downloading and the prior day)

The prior day is included as many instruments trading days starts on the prior day.

What historical data is loaded from provider?

Examples of when NinjaTrader will fetch data if the data repository contains data from 1/2/14 to 1/5/14 and the current date is 1/6/14:

1. Chart of 1/2/14 to 1/6/14 -> load data request for 1/6/14, use data stored in memory/cache for other dates
2. Chart of 1/2/14 to 1/5/14 -> load data request for 1/5/14, use data stored in memory/cache for other dates
3. Chart of 1/2/14 to 1/4/14 -> use data stored in memory/cache for all dates
4. Chart of 12/27/13 to 1/4/13 -> load data request for all dates

Expanded example for a more detailed explanation:

Historical tick data in the repository from 12/1/2013 until 1/1/2014 2:00 PM
Historical minute data in the repository from 1/1/2013 until 1/1/2014 2:00 PM
Historical daily data in the repository from 1/1/2013 until 1/1/2014
Today is the 1/2/2014 9:00 AM and the Trading Hours Template is "CME US Index Futures ETH"

Scenario	Requested from Local Repository	Downloaded From Provider
1 tick chart 3 days back with end date of today	12/31/2013 7:00 PM to 1/1/2014 2:00 PM	1/1/2014 2:00 PM to 1/2/2014 9:00 AM
1 minute chart 5 days back with end date of today	12/29/2013 7:00 PM to 1/1/2014 2:00 PM	1/1/2014 2:00 PM to 1/2/2014 9:00 AM
1 day chart 365 days back with end date of today	1/2/2013 to 1/1/2014	2/1/2014

Critical: All NinjaTrader historical data requests are handled using calendar days format, please keep this in mind when for example interpreting loading results that include weekends or holidays periods where the may be no actual data - the affected calendar day would still be counted as a 'day' in the request.

10.13.2 Data by Provider

Understanding the data provided by your connectivity provider

NinjaTrader, LLC is not a market data provider. Historical data is provided by our connectivity providers that offer historical data as part of their service. The table below displays all NinjaTrader supported connectivity providers as well as the historical and real-time data provided by each:

Connectivit	Real-Time	Historical Tic	Historical	Historical Bid	Historical	Historical Mi	Historical	Real-	Instru	Real-	TickR	Bid/As	DailyB	Settle

y Provider	Data	k Data	Bid/Ask Minute Data	/Ask Daily Data	Bid/Ask Tick Data	nute Data	Daily Data	Time Timestamp	ments Supported	Time News	eplay	k Stamped Tick Data	ars Trading Hours	mentadjusted Close Price For Daily Bars
NinjaTrader	YES	YES	YES	YES	YES	YES	YES	Native	F, I	NO	YES	YES	Extended Trading Ho	YES

												urs	
Kinetick www.kinetick.com	YES (subscription only)	YES (subscription only)	NO	NO	*YES	YES (subscription only)	YES	Native	E(O), F(O), FX, I	YES	YES	YES	SymbolMapSpecific	YES
BarChart	YES	YES	NO	NO	NO	YES	YES	Native	E, F, FX, I	NO	YES	NO	SymbolMapSpecific	YES
Coinbase	YES	YES	YES	YES	YES	YES	YES	Native	CC	NO	NO	NO	UTC	N/A
Continuum/CQG	YES	YES	YES	YES	YES	YES	YES	Native	F, I	NO	YES	YES	Extended	YES

												TradingHours	
Continuum/CQGWebAPI	YES	YES	YES	YES	YES	YES	YES	Native	F(O), I	NO	YES	YES	ExtendedTradingHours	YES
cTrader	YES	YES	NO	NO	YES	YES	YES	Native	C	NO	YES	NO	CFD	N/A
eSignal	YES	YES	NO	NO	YES	YES	YES	Native	E, F, FX, I	NO	YES	NO	SymbolMapSpe	NO

												cific	
FOREX.com/CityIndex	YES	YES	NO	NO	NO	YES	YES	Native	FX, C	NO	YES	N/A	Forex	N/A
FXCM	YES	YES	YES	YES	YES	YES	YES	Native	FX, C	NO	YES	N/A	Forex	N/A
InteractiveBrokers	YES	NO	YES	YES	NO	YES (live account only)	YES (live account only)	Local	C, E(O), F(O), FX, I	NO	NO	NO	Extended Trading Hours	NO
IQFeed	YES	YES	NO	NO	*YES	YES	YES	Native	E(O), F(O), F	YES	YES	YES	SymbolMapS	YES

									X, I				pecific	
Rithmic	YES	YES	YES	YES	YES	YES	YES	Native	F	NO	YES	YES	Extended Trading Hours	YES
TD Ameritrade	YES	NO	NO	NO	NO	YES	YES	Local	E (O), I	NO	NO	NO	Regular Trading Hours	NO

C = CFD
CC = CryptoCurrency
E = Equities
F = Futures
FX = Forex
I = Indexes

(O) = Options on underlying instrument type

Notes:
There are various limitations on data from each provider and data is subject to change.
Options are not supported for NinjaScript use.
*Historical bid/ask comes from associated value at each historical last tick value.

Converting Real-Time Data into Historical Data

NinjaTrader by default will always loads historical data from your provider (Recommended). However if you enable the option 'Record live data as historical' in the Control Center > Tools > Options > Market Data Category then NinjaTrader will store real-time incoming tick data to your local PC if you have a Chart or **Market Analyzer** (must have an indicator column added) window open. This data can then be used as historical data. For example, if you open a chart and let it run all day long, the data collected today, will be available as historical data when you open the same chart tomorrow.

Warning: Recording live data uses more PC resources and is intended for connections which **DO NOT** provide historical data. Enabling this option while also using a historical data provider is not recommended as it may result in data gaps.

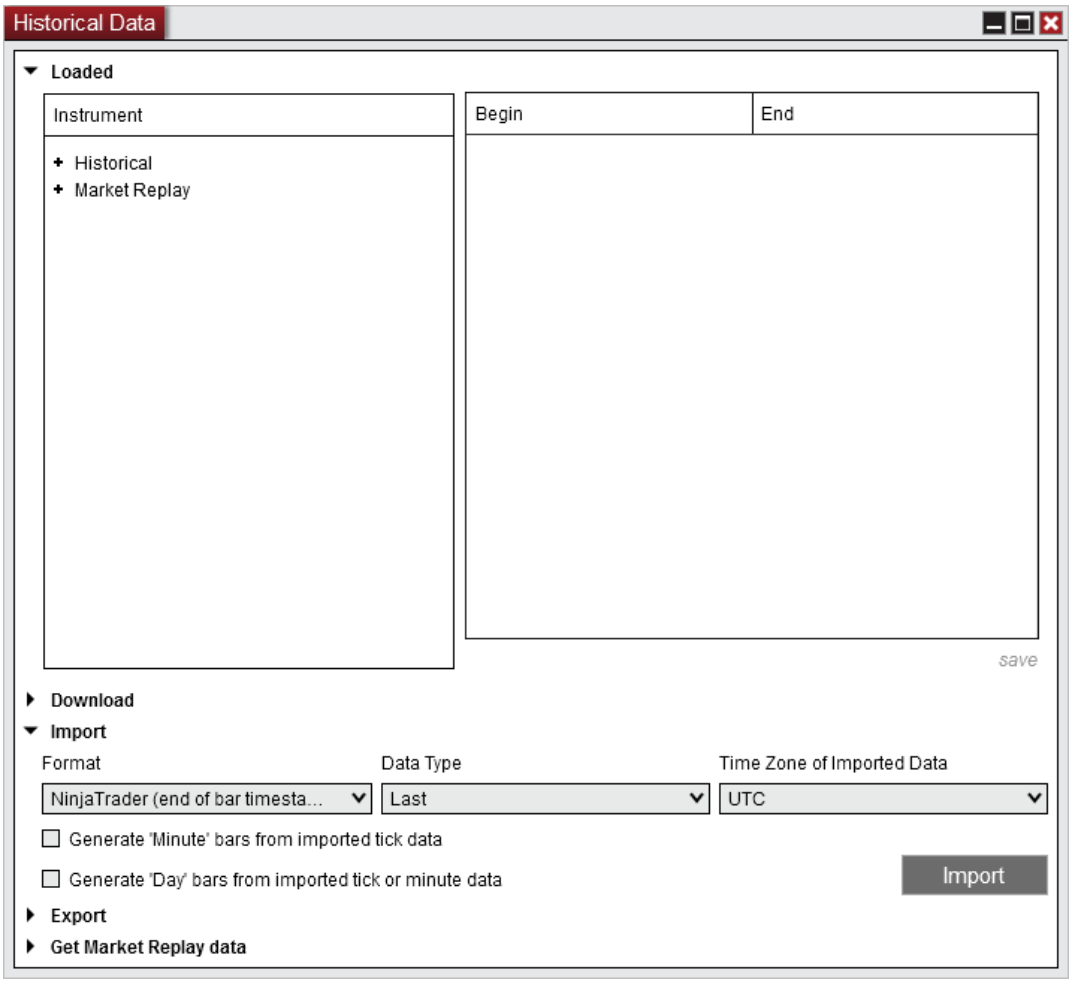
Connecting to your Broker and a Market Data Provider Simultaneously

If your broker technology does not support historical data, you can connect to a service like Kinetick at the same time as connecting to your broker so that you can receive historical data. Please see the topic on Multiple Connections for additional information.

10.13.3 Importing

Historical data can be imported from a text file with a ".txt" extension within the **Loaded** section of the Historical Data Window. Several formats and data types are supported and NinjaTrader can optionally build 'Minute' bars from tick data as well as 'Day' bars from tick or minute data.

▼ Understanding import options



Understanding import options

The following formats and options are available when importing a text file:

Format

Select one of three options available in the Format drop down menu:

1. NinjaTrader (timestamps in import file(s) represent end of bar time)
2. NinjaTrader (timestamps in import file(s) represent start of bar time)
3. Tick Data, LLC

Data Type

Select one of three options available for the data type:

1. Ask - Data values in the text file represent historical Ask prices
2. Bid - Data values in the text file represent historical Bid prices
3. Last - Data values in the text file represent historical Last prices (trades)

Time Zone of Imported Data
Select the time zone of the data you are importing (not the time zone you are importing to as all imported data will always be converted to local PC time). If you are importing data exported from NinjaTrader then this should be left as UTC because NinjaTrader exports are always done in the UTC time zone.

Generate 'Minute' Bars from Imported Tick Data:
Select this option to convert the tick data from the import file into historical 'Minute' data. This allows any 'Minute' interval to be available within NinjaTrader.

Generate 'Day' Bars from Imported Tick or Minute Data:
Select this option to convert the tick or minute data from the import file into 'Day' data. This allows the building of 'Day', 'Week', 'Month' and 'Year' bars within NinjaTrader. (See the "*Historical & Real-Time Data*" section of the Help Guide for more information on historical data.)

Note: Generating bars from imported tick data is done based off of the timestamps of the tick data. It is possible that the generated bars do not perfectly match minute or daily bars provided by the data provider as they may utilize a different timestamp granularity than your import data for their own bar generations.

Understanding import file and data formats

File Name
When using the NinjaTrader format, the name of the text file to be imported must be the NinjaTrader instrument name followed by a period and "Last", "Bid", or "Ask" depending on the data type. For example:

MSFT.Last.txt for Microsoft stock last price data
ES 12-09.Bid.txt for the S&P E-mini December contract bid price data
EURUSD.Ask.txt for the Euro/U.S. dollar currency pair ask price data

Daily Bars Format
Each bar must be on its own line and fields must be separated by semicolon (;). Only 1 day bars can be imported.

The format is:
yyyyMMdd;open price;high price;low price;close price;volume

Sample data:

20061023;1377.25;1377.25;1377.25;1377.25;86
20061024;1377.25;1377.25;1377.25;1377.25;27
20061025;1377.25;1377.25;1377.25;1377.25;24
20061026;1377.50;1377.50;1377.25;1377.25;82

Minute Bars Format

Each bar must be on its own line and fields must be separated by semicolon (;). Only 1 minute bars can be imported.

The format is:
yyyyMMdd HHmmss;open price;high price;low price;close price;volume

Sample data:
20061023 004400;1377.25;1377.25;1377.25;1377.25;86
20061023 004500;1377.25;1377.25;1377.25;1377.25;27
20061023 004600;1377.25;1377.25;1377.25;1377.25;24
20061023 004700;1377.50;1377.50;1377.25;1377.25;82

Tick Format (Second Granularity)

Each tick must be on its own line and fields must be separated by semicolon (;).

The format is:
yyyyMMdd HHmmss;price;volume

Sample data:
20061107 000431;1383.00;1
20061107 000456;1383.25;25
20061107 000456;1383.25;36
20061107 000537;1383.25;14

Tick Format (Sub Second Granularity)

You can also import tick granularity to the ten millionth of a second. Each tick must be on its own line and fields must be separated by semicolon (;).

The format is:
yyyyMMdd HHmmss fffffff;price;volume

Sample data: (Note: If you wanted to import in millisecond granularity data then each line must have the remaining "0"'s behind it to import correctly.)
20061107 000431 1000000;1383.00;1
20061107 000456 1000000;1383.25;25
20061107 000456 2000000;1383.25;36

20061107 000537 7000000;1383.25;14

> **Tip**: You can also import historical tick data to be used with Tick Replay, which includes the current bid and ask prices associated with the last price of that tick (not to be confused with Playback "Market Replay" data which CANNOT import manually). Importing tick replay data without sub-second granularity is less accurate.

Tick Replay Format (Sub Second Granularity)

Each tick must be on its own line and fields must be separated by semicolon (;).

The format is:
yyyyMMdd HHmmss fffffff;last price; bid price; ask price;volume

Sample data: (Note: If you wanted to import in millisecond granularity data then each line must have the remaining "0"'s behind it to import correctly.)
20061107 000431 1000000;1383.00;1383.00;1383.25;1
20061107 000456 1000000;1383.25;1382.50;1382.25;25
20061107 000456 2000000;1383.25;1383.25;1383.50;36
20061107 000537 7000000;1383.25;1383.25;1383.50;14

Tick Replay Format (Second Granularity)

Each tick must be on its own line and fields must be separated by semicolon (;).

The format is:
yyyyMMdd HHmmss;last price;bid price;ask price;volume

Sample data:
20061107 000431;1383.00;1383.00;1383.25;1
20061107 000456;1383.25;1382.50;1382.25;25
20061107 000456;1383.25;1383.25;1383.50;36
20061107 000537;1383.25;1383.25;1383.50;14

How to import historical data from a text file

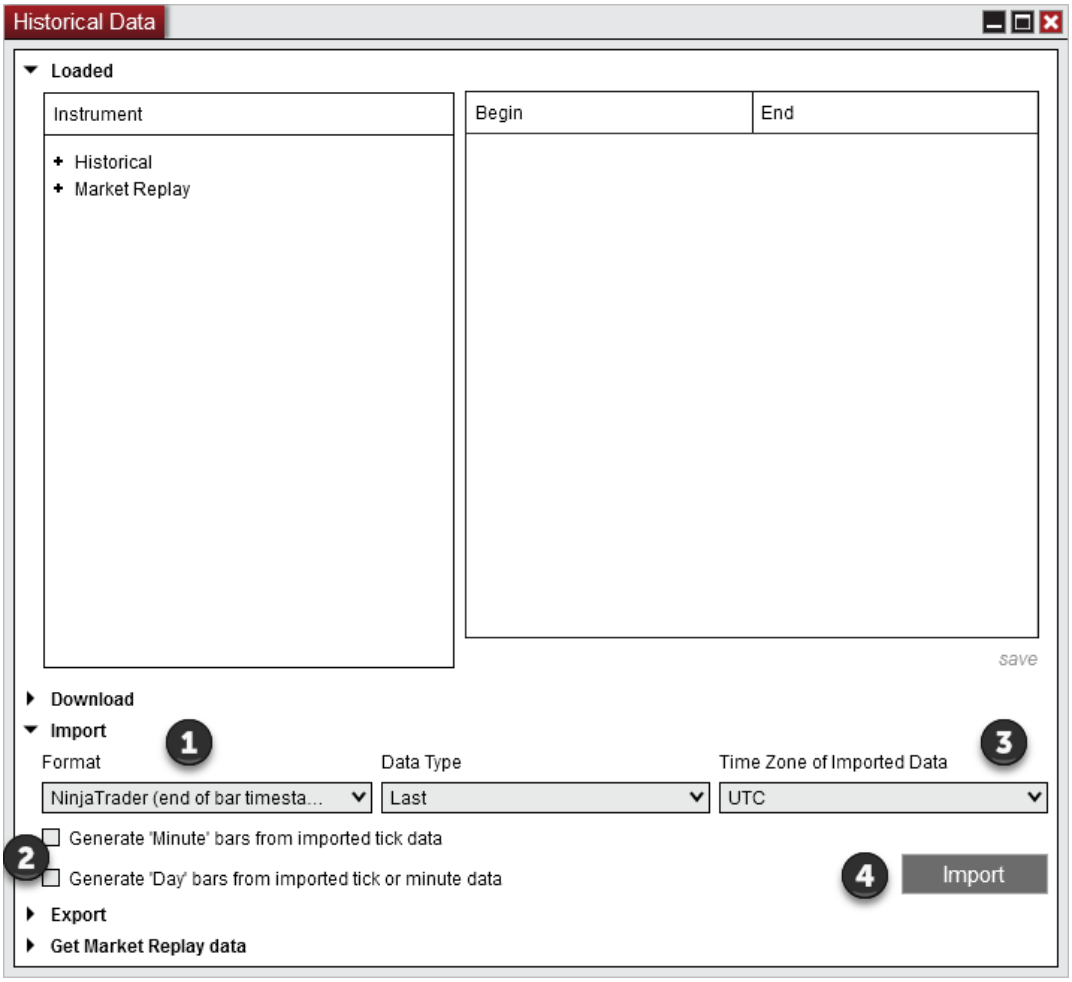
Importing Tips

Please review the following before importing:

- If you are importing historical data for a futures or forex instrument, the instrument MUST exist in the database. If it does not, you must add it first via the Instruments window.
- Any data imported where the instrument does not exist in the database will automatically be imported as a "Stock" instrument type
- Data points will be rounded to the instruments tick size as it is imported if the price is not evenly divisible by the instrument's tick size
- Imported data, regardless of time zone, will be converted to the local time zone.

Importing Historical Text Data

To import historical data from a text file into NinjaTrader:



1 Choose the Format and Data type that correctly represent the data in the import file (see the "*Understanding the import options*" section above)

2. Optionally select any of the Generate... choices to have NinjaTrader create other bar types from the import data
3. Select the **Time zone of the imported data** (Note: Any data exported from NinjaTrader is always exported in UTC time zone)
4. Press the Import button
5. Select the text file from your PC to import and press the "Open" button.

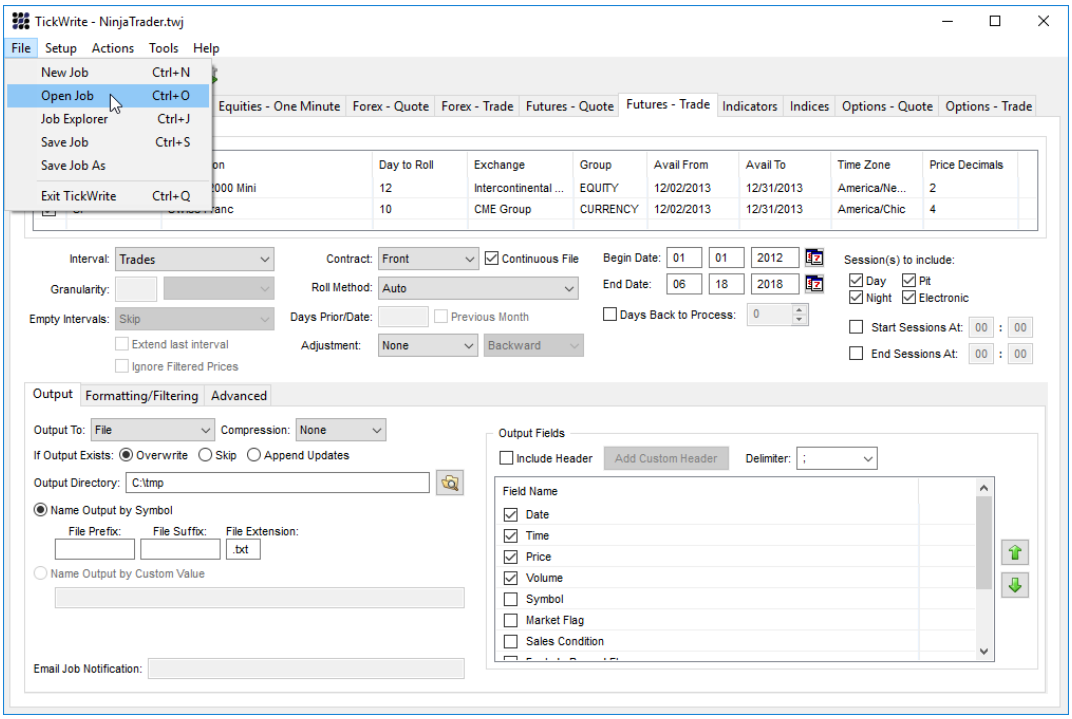
NinjaTrader will attempt to import the text file. If successful, a window will appear confirming this. If unsuccessful, an error window will appear and you should check the Log tab of the Control Center to view the error(s).

Formatting data from Tick Data, LLC

When exporting data from Tick Data, LLC with their TickWrite utility, the data must be exported in a format NinjaTrader can import and then the file will need to be properly renamed.

Formatting data

Download the following file then unzip it to access the NinjaTrader.twj file: TickDataLLC.zip.
Within TickWrite you can go to **File** and select **Open Job** to select the NinjaTrader.twj file.
Now you can configure what data you want to export and when you execute the job the data will be in the required format.



Formatting the File Name

When using the NinjaTrader format, the name of the text file to be imported must be the NinjaTrader instrument name followed by a period and "Last", "Bid", or "Ask" depending on the data type. For example:

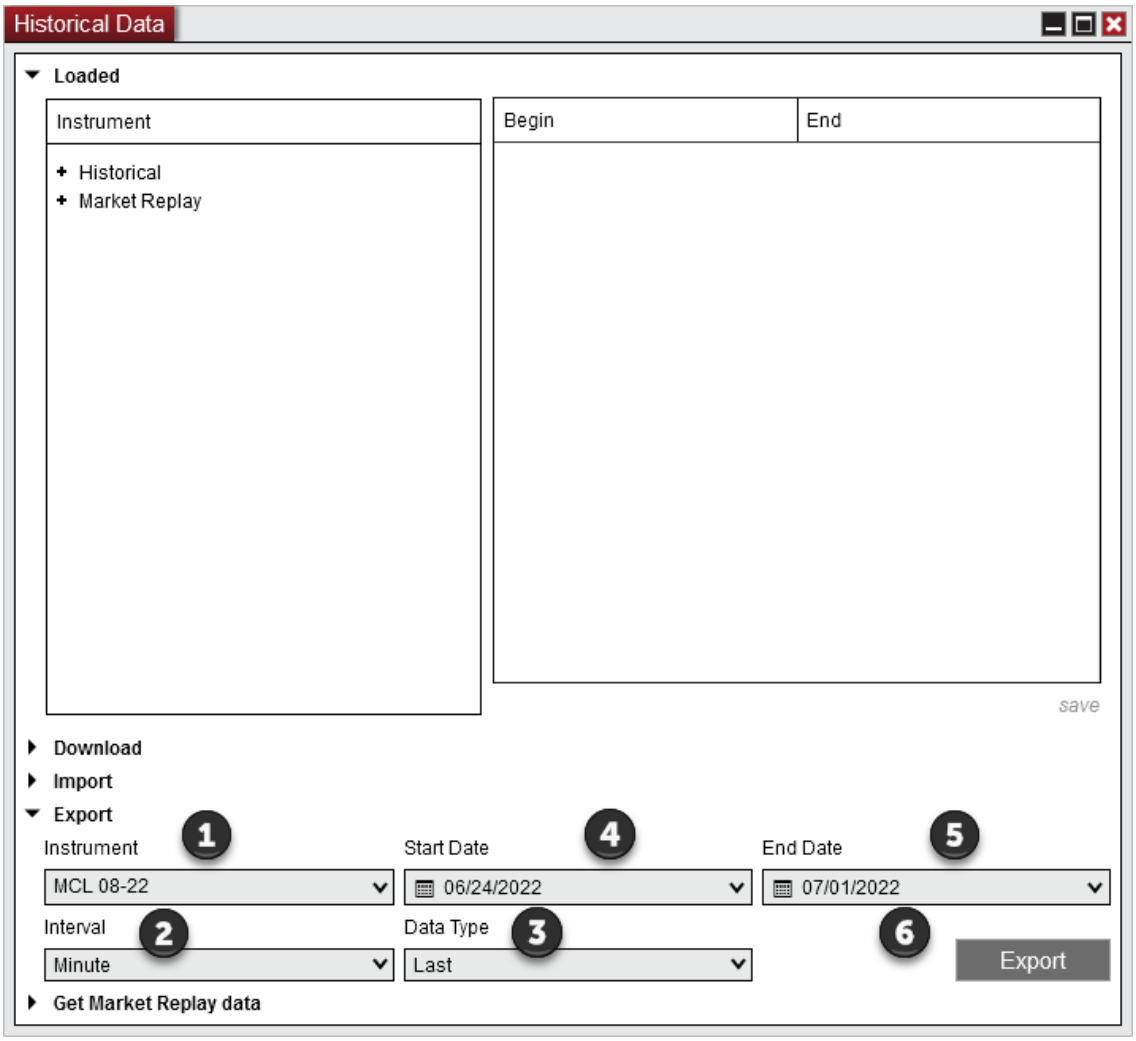
MSFT.Last.txt for Microsoft stock last price data
ES 12-09.Bid.txt for the S&P E-mini December contract bid price data
EURUSD.Ask.txt for the Euro/U.S. dollar currency pair ask price data

10.13.4 Exporting

Historical data stored within NinjaTrader can be exported to a text ".txt" file. This is done within the **Export** section of the **Historical Data Window** window.

How to Export Historical Data

It is important to understand that the historical data you wish to export must currently be saved in NinjaTrader as provided by the data provider or collected live. Please see the "*Historical & Real-Time Data*" section of the Help Guide for more information. If you do not have data, it can be downloaded from your data provider using the Download section of the **Historical Data Window Load** tab.



To export historical data to a text file:

1 Select the instrument to export. The instruments you have data for will be available for selection.

2 Select the interval type to export. The interval type(s) "Tick," "Minute," or "Day" are displayed if that type of data is available.

3 Select the Data type to export. The data type(s) "Ask," "Bid," or "Last" are displayed if that type of data is available.

4 Select the desired Start date.

5 Select the desired End date.

6 Click "Export" and select an location and file name for the exported file.

The historical data is exported with End of Bar time stamps to the chosen folder as a text file in the same format specified in the "*Understanding import file and data formats*" section of the Importing page. The exported data will be in the UTC time zone.

10.13.5 Editing

Historical data saved in NinjaTrader can be edited via the Edit section of the **Historical Data Window**.

▼ How to edit historical data

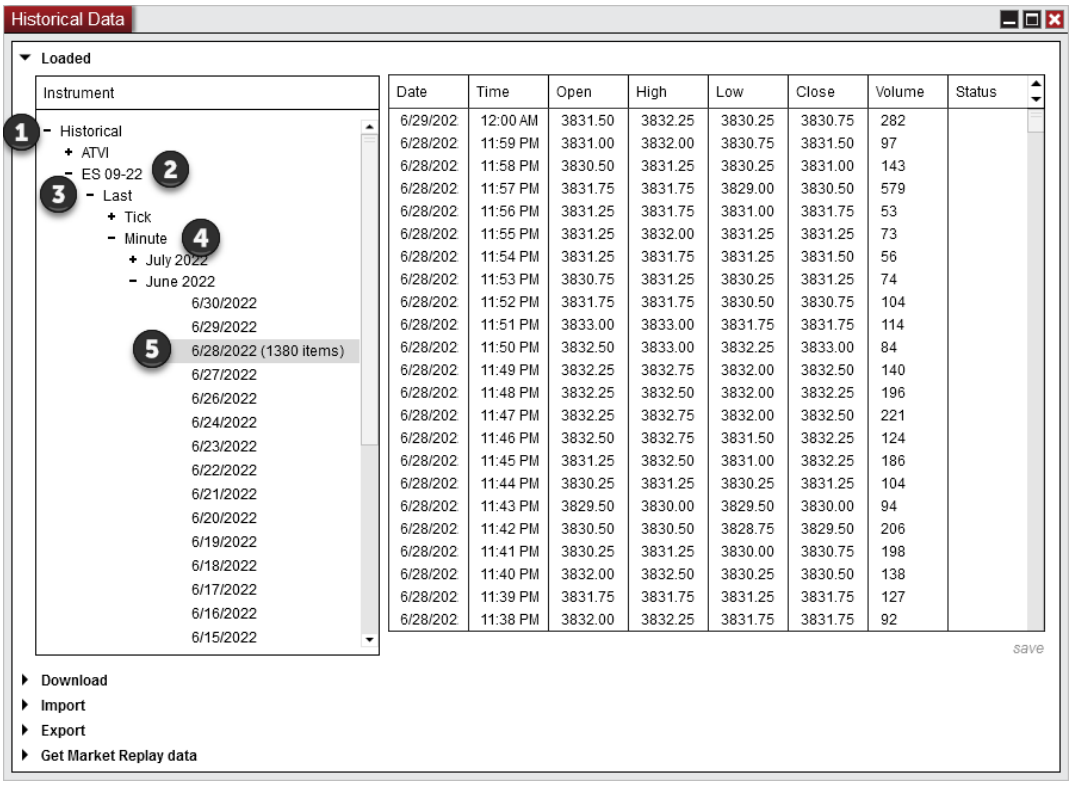
Editing Historical Data

It is important to understand that the historical data you wish to edit must currently be saved in NinjaTrader as provided by the data provider or collected live. Please see the "*Historical & Real-Time Data*" section of the Help Guide for more information. If you do not have data, it can be downloaded from your data provider if they offer it by using the Load tab of the **Historical Data Window**.

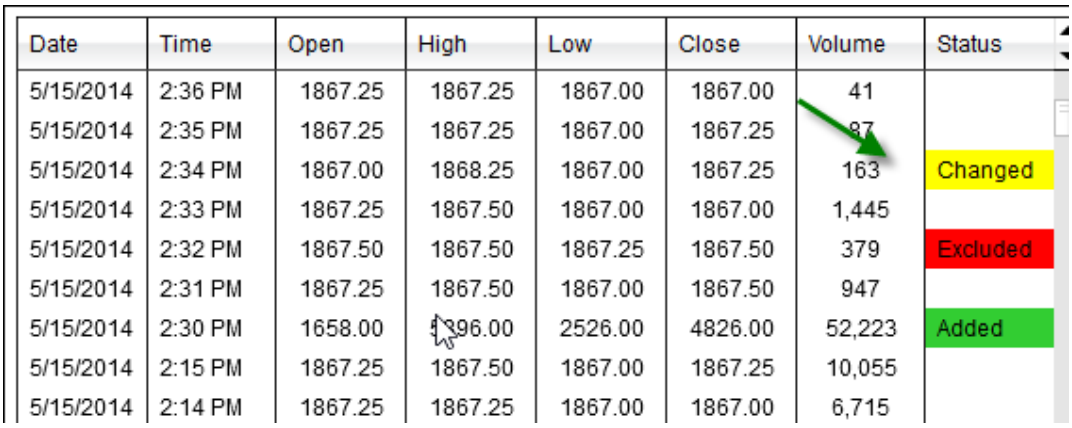
To edit historical data available within NinjaTrader:

1. Left mouse click the plus "+" for "**Historical**"
2. Select the "**+**" for the Instrument.
3. Select "Ask," "Bid," or "Last,"data type
4. Select the "**Tick**", "**Minute**", "**Day**" data type
5. Select the **date.**

The data for that data type and date will be shown in the data grid.



- Changing data - Double left mouse click on a cell in the Open, High, Low, Close or Volume column to edit the data value.
- Adding data - Left mouse click on a row to select it. Then right mouse click to access the options to **Add** a new data row.
- Excluding data - Right mouse click on the desired row and select the menu item **Exclude** to exclude the data. Excluded data is data that is intentionally ignored and not used. NinjaTrader will remember this excluded data on a historical data reload.
- Any changes that are made are both color coded as well as shown in the Status column. The status column will report when any data has been modified from original values.



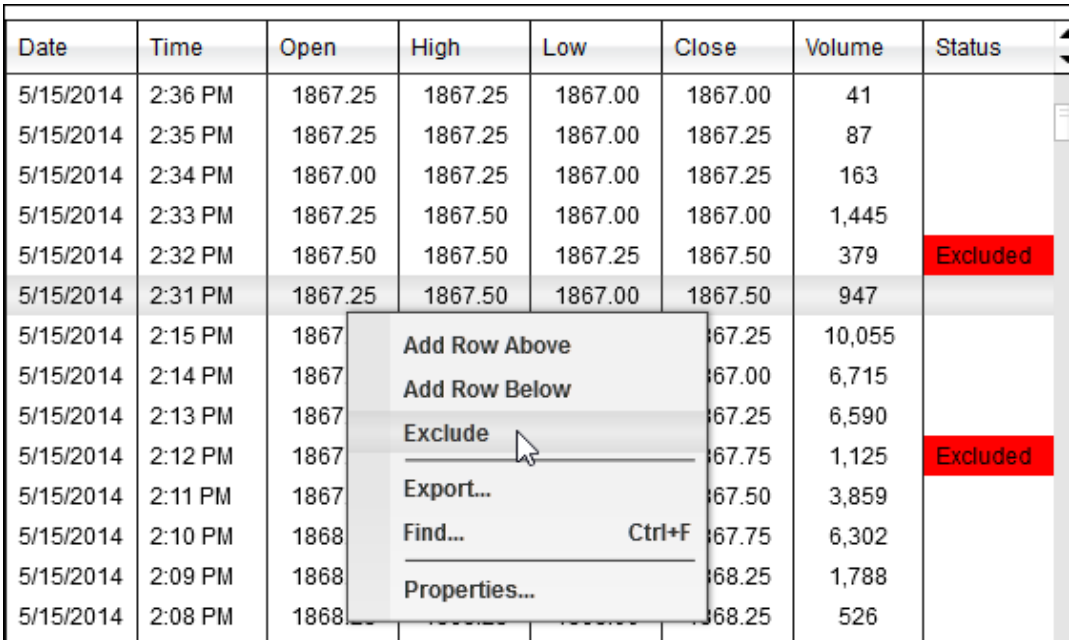
Date	Time	Open	High	Low	Close	Volume	Status
5/15/2014	2:36 PM	1867.25	1867.25	1867.00	1867.00	41	
5/15/2014	2:35 PM	1867.25	1867.25	1867.00	1867.25	87	
5/15/2014	2:34 PM	1867.00	1868.25	1867.00	1867.25	163	Changed
5/15/2014	2:33 PM	1867.25	1867.50	1867.00	1867.00	1,445	
5/15/2014	2:32 PM	1867.50	1867.50	1867.25	1867.50	379	Excluded
5/15/2014	2:31 PM	1867.25	1867.50	1867.00	1867.50	947	
5/15/2014	2:30 PM	1658.00	1896.00	2526.00	4826.00	52,223	Added
5/15/2014	2:15 PM	1867.25	1867.50	1867.00	1867.25	10,055	
5/15/2014	2:14 PM	1867.25	1867.25	1867.00	1867.00	6,715	

Once the desired changes are made, press "save" in the bottom right hand corner of the Edit tab to save the changes within NinjaTrader.

Note: If more than one row contains the same Date, Time and price values, all similar rows will be edited.

Excluding Data

To exclude data right click on the row of data to be excluded and select "Exclude". Note: All rows with the same date and time will be automatically excluded by NinjaTrader.

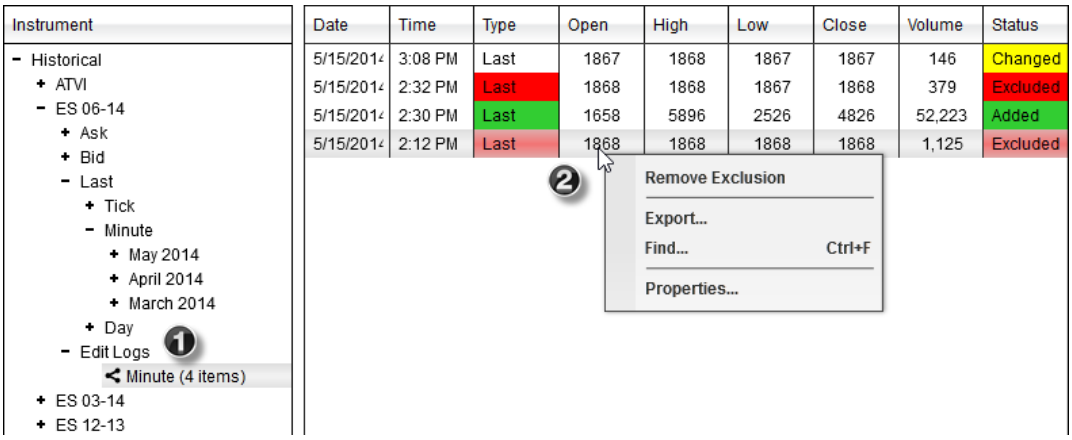


Date	Time	Open	High	Low	Close	Volume	Status
5/15/2014	2:36 PM	1867.25	1867.25	1867.00	1867.00	41	
5/15/2014	2:35 PM	1867.25	1867.25	1867.00	1867.25	87	
5/15/2014	2:34 PM	1867.00	1867.25	1867.00	1867.25	163	
5/15/2014	2:33 PM	1867.25	1867.50	1867.00	1867.00	1,445	
5/15/2014	2:32 PM	1867.50	1867.50	1867.25	1867.50	379	Excluded
5/15/2014	2:31 PM	1867.25	1867.50	1867.00	1867.50	947	
5/15/2014	2:15 PM	1867			67.25	10,055	
5/15/2014	2:14 PM	1867			67.00	6,715	
5/15/2014	2:13 PM	1867			67.25	6,590	
5/15/2014	2:12 PM	1867			67.75	1,125	Excluded
5/15/2014	2:11 PM	1867			67.50	3,859	
5/15/2014	2:10 PM	1868			67.75	6,302	
5/15/2014	2:09 PM	1868			68.25	1,788	
5/15/2014	2:08 PM	1868			68.25	526	

Using the Edit Logs

1 Once any changes are saved to the historical data by pressing "save", an Edit Logs node appears under the instrument node.

2 The Edit Logs node contains all edits made to historical data for a specific instrument. Edits can be undone by right mouse clicking on the change you wish to undue and selecting the menu item **Remove Exclusion.** All edits can be removed by right mouse clicking over the edit node and selecting the menu item **Remove All Edits**.

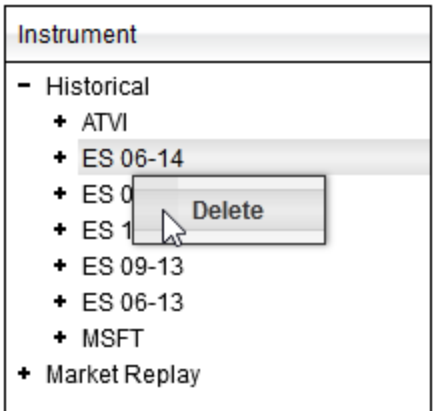


How to delete historical data

Historical data saved in NinjaTrader can be deleted via the **Loaded** section of the **Historical Data Window**.

Deleting Historical Data

It is important to understand that the historical data you wish to delete must currently be saved in NinjaTrader as provided by the data provider or collected live.



To delete historical data saved in NinjaTrader:

1. Left mouse click on any node available in the **Edit** tab of the **Historical Data Window** to select it.
2. Right mouse click and select the menu item **Delete** or press the 'Delete' key on your keyboard to delete all data contained in the node.

Note: Deleted historical data will be replaced when data is reloaded from the connectivity provider. Please see the "Excluding Data" sub-section of the "How to edit historical data" section above for more information on excluding data, which will remain excluded when reloading data from the data provider.

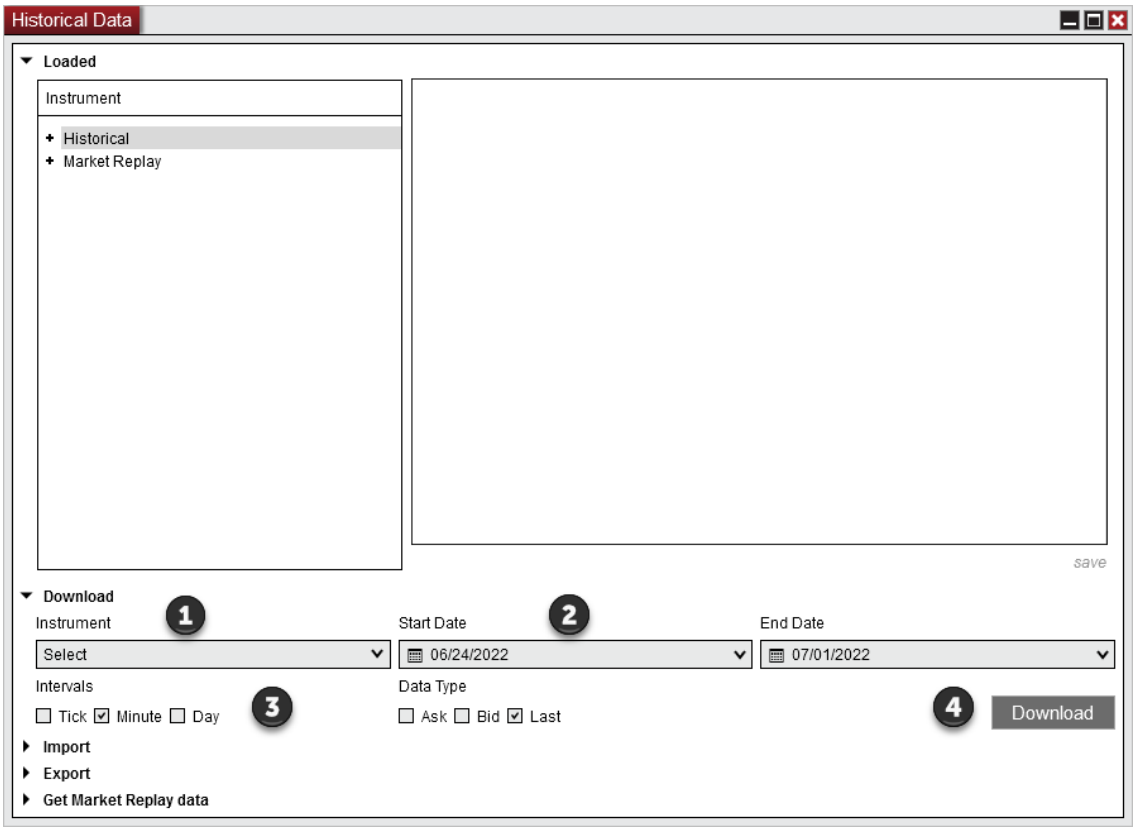
10.13.6 Download

Historical data can be downloaded from the data provider via the **Download** section.

How to Download Historical Data

To download historical data first make sure NinjaTrader is connected and historical data is available from your data provider.

1. Select an instrument for data to be downloaded. (**Tip**: You may also select an instrument list)
2. Select the desired Start and End date range
3. Select the desired Intervals and Data Types
4. Press the "Download" button to begin the download



A message in the bottom right of the **Historical Data Window** will appear and display the status of the download.

To cancel a historical data request close the Historical Data Window window.

Notes:

- If you already have historical data for an instrument, please be sure to only select a date range in which your data provider offers historical data. If you choose a range older than what your data provider offers you may lose any data you had stored on those dates in that range outside of what your data provider offers.
- Downloading historical data will function based on the **Merge Policy** being used. Using **MergeBackAdjusted** or **MergeNonBackAdjusted** will switch what contract month is being downloaded based on the rollovers occurring during the selected date range. To download data for just the selected contract the **Merge Policy** will need to be set to **DoNotMerge**. See the **Merge Policy** section for more information on Merge Policies.

10.14 Hot Keys

Hot Keys Overview

You can access the **Hot Key** window by left mouse clicking in the **Tools** menu within the NinjaTrader Control Center and selecting the menu item **Hot Keys...**

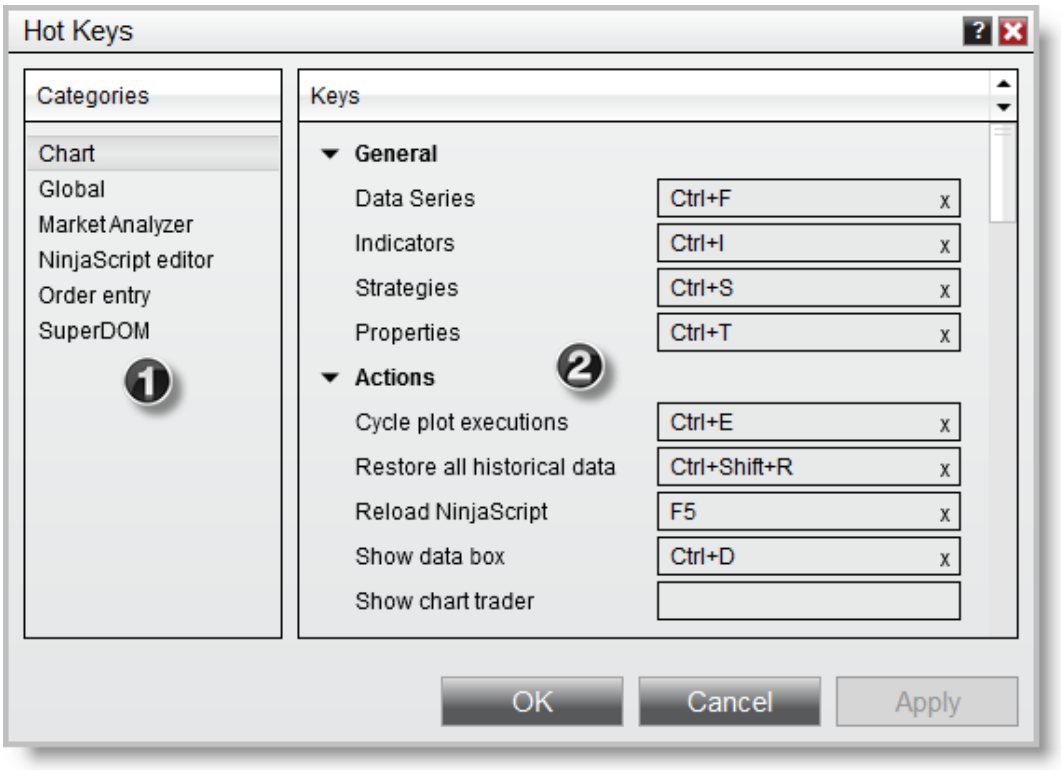
NinjaTrader allows you to assign specific key strokes as **Hot Keys** in order to quickly perform a task. **Hot Key** utilization includes, but is not limited to: opening new windows, performing tasks within open windows, and placing orders in an order entry window. The **Hot Key** window allows you to add and remove **Hot Key** assignments to various application actions.

- Working with Hot Keys
- Trading with Hot Keys

10.14.1 Working with Hot Keys

You can customize the Hot Keys by assigning the desired key stroke in the related action field. You also have the ability to print the full list of actions and their related Hot Keys for easy reference.

▼ Understanding the Hot Keys window



1. **Active Window Categories**
The Categories section displays a list of NinjaTrader windows where Hot Keys can be assigned. Please see the "Understanding when Hot Keys are active" section of this page for more information on the active window.

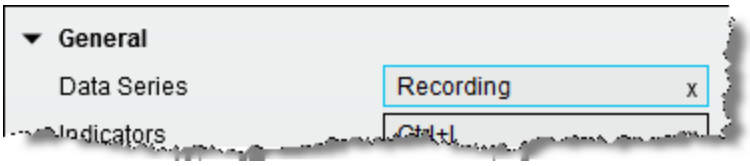
2. **Available Actions and Hot Keys**
The Keys section displays the actions available for Hot Key assignment within the selected active window.

Assigning and Removing Hot Keys

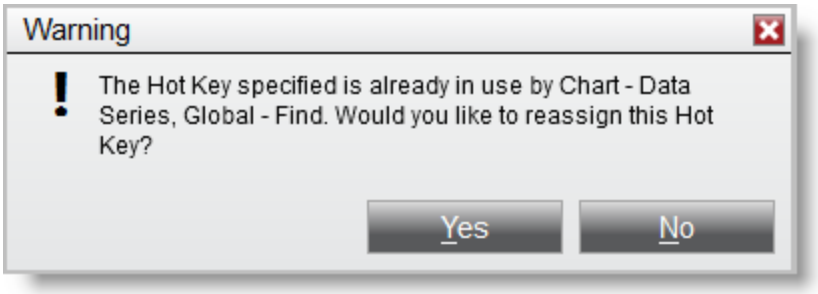
Assigning a Hot Key

You can assign a key stroke as a Hot Key to the desired action by completing the following steps:

1. Move your mouse over the action field where you want your Hot Key assigned, "Click to record hot key" should display
2. Left mouse click on the field to begin recording
3. Use the keyboard to select the Hot Key combination
4. Recording will finish as you input the hot key on your keyboard or press esc to cancel the recording

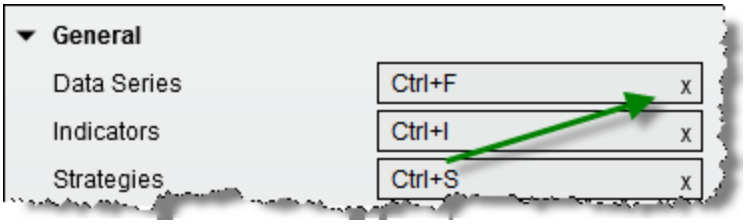


Note: If you try to assign a **Hot Key** that would conflict with an already defined **Hot Key** you will be asked to reassign.



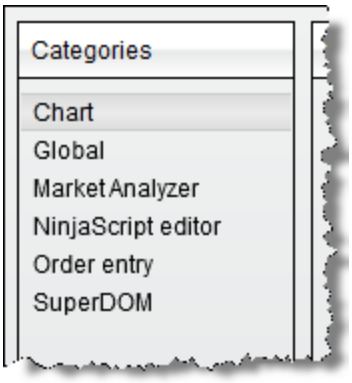
Removing a Hot Key

To remove a Hot Key left mouse click in the action field on the "X" icon.



Understanding when Hot Keys are active

Hot Keys are window sensitive. This means that Hot Keys will only work when the active window is selected. The name of the window that needs to be active is located in the left column of the Hot Keys window.



Global

Hot Keys assigned under the Global section are always active regardless of the active NinjaTrader window with the exception of a modal window having focus. See the "*Understanding the risks in using* Hot Keys *for order entry*" section of the Trading with Hot Keys page of the Help Guide for more information on the modal form exception.

Order Entry

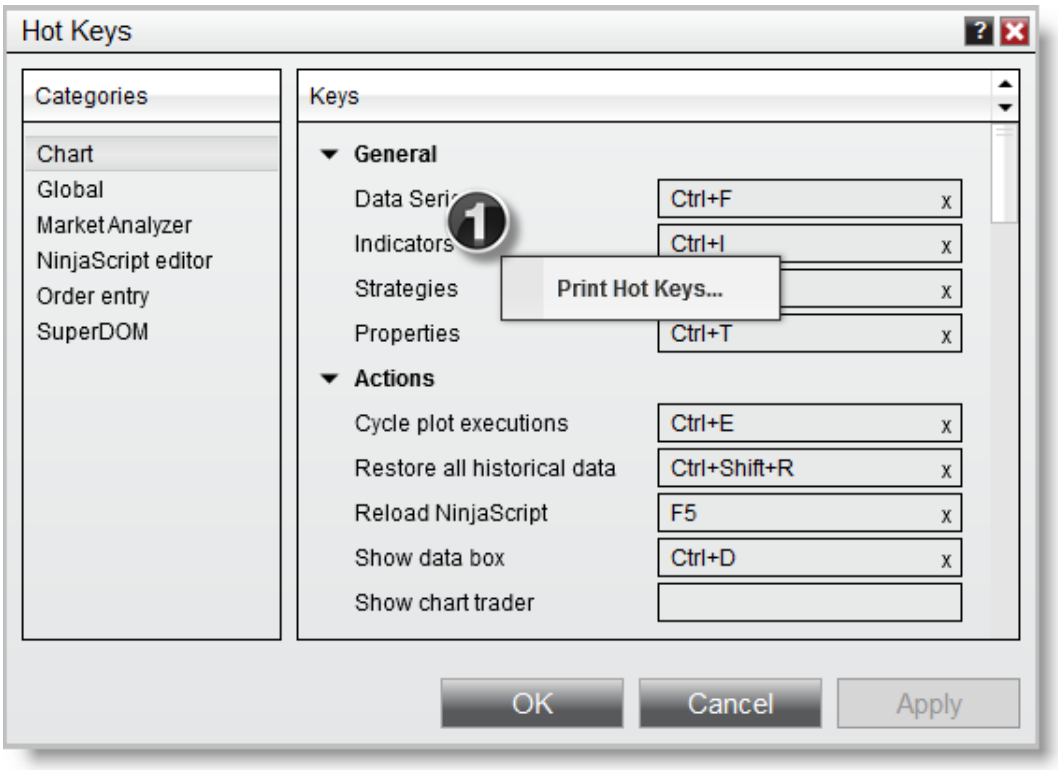
Hot Keys assigned under the Order Entry section are active whenever an order entry window is selected. Please see the Trading with Hot Keys section of the Help Guide for more information on this topic.

How to print your Hot Keys for reference

NinjaTrader gives you the ability to print your assigned Hot Keys for convenient reference.

Printing Hot Keys

1.To print a full list of your Hot Keys, right mouse click in the Hot Key Manager and select the **Print Hot Keys...** menu item.



10.14.2 Trading with Hot Keys

Hot Keys can be assigned to order actions and used to place orders within NinjaTrader order entry windows.

Understanding the risks in using Hot Keys for order entry

Hot Keys are a powerful and versatile trading tool. However, misuse can lead to unexpected trades and therefore loss of money. There are several features of the Hot Keys that you should become familiar with before using them for order entry to limit the risk of unexpected order placement.

Active Window

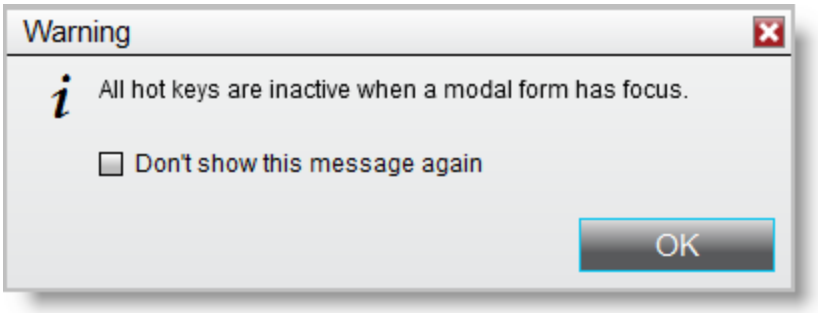
You must always be aware of the current active window when using Hot Keys for order entry. Order entry Hot Keys are window sensitive and will only execute an action to the active order entry window. Please see the "*Understanding where Hot Key order entry is active*" section on this page for more information on this topic.

Using the incorrect Hot Key

It is imperative that you know what Hot Key performs what action. It is easy to confuse Ctrl+B with Shift +B which may both enter different types of orders. For this reason, we recommend printing your Hot Keys after assigning for easy reference. Please see the "*How to print your Hot Keys for reference*" section of the Working with Hot Keys page of the Help Guide.

When Hot Keys are inactive

When you close the Hot Key window, you will see the message shown below. A modal form is a window that is always on top and always selected. (An example is the modal form message window itself.) It is important to understand that ALL Hot Keys are inactive any time a model form window is open.



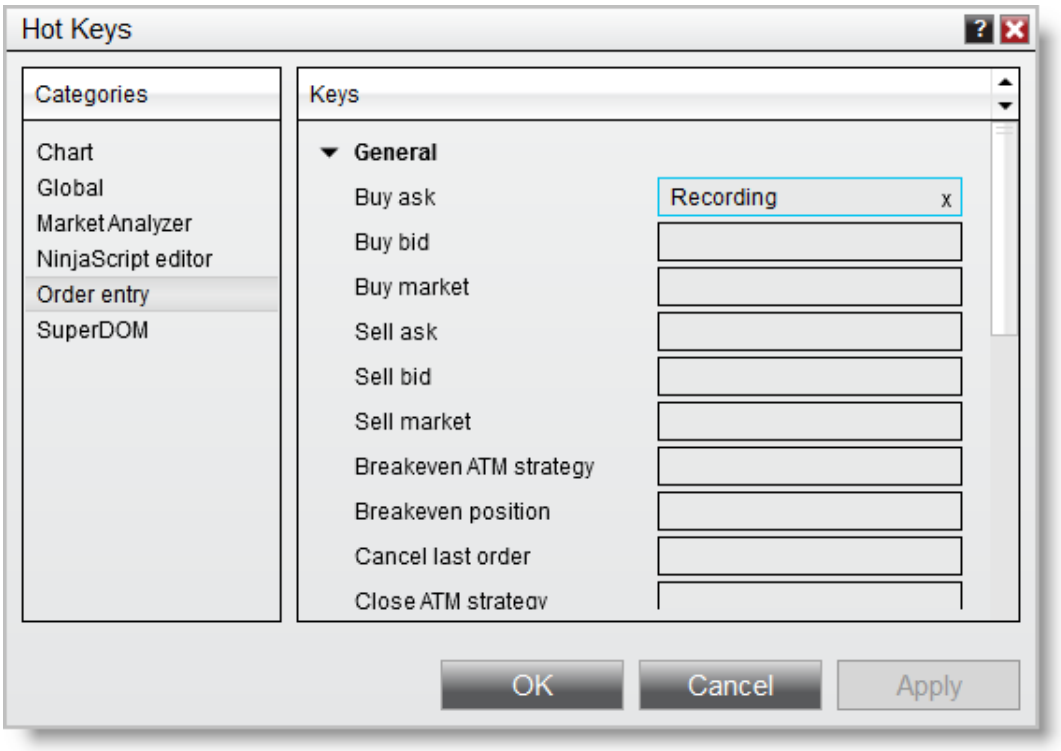
How to enable Hot Key order entry

To enable order entry Hot Keys

From within the Control Center window select the **Tools** menu and then select the menu name **Options**. Once in the Trading category, select "Use order entry hot keys"

Assigning Hot Keys

1. Move your mouse over the action field where you want your Hot Key assigned, "Click to record hot key" should display
2. Left mouse click on the field to begin recording
3. Use the keyboard to select the Hot Key combination
4. Recording will finish as you input the hot key on your keyboard or press esc to cancel the recording



Understanding where Hot Key order entry is active

Order Entry Hot Keys will only submit from the the active order entry window. This is important to understand, especially if using multiple order entry windows.

Order Entry Windows

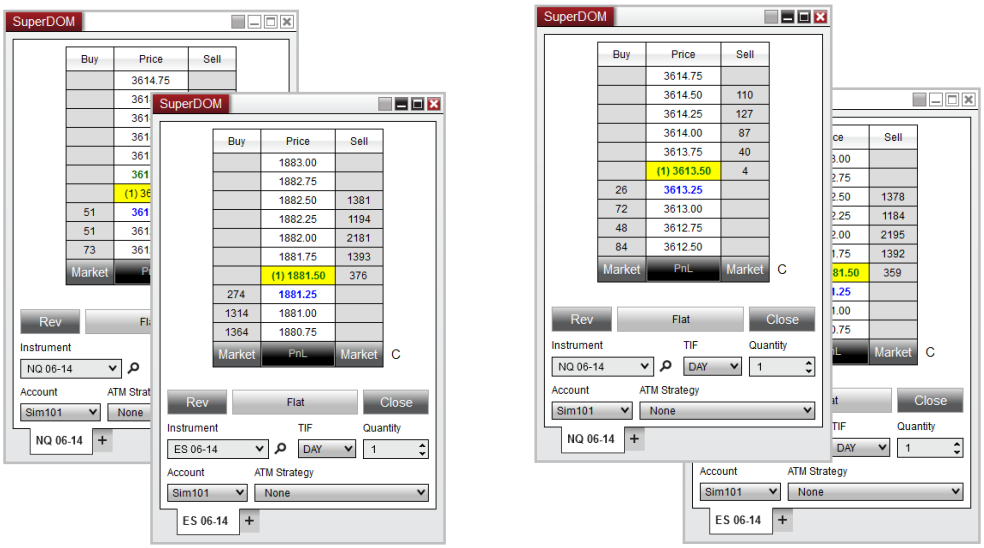
Below is a list of all of the order entry windows available in NinjaTrader.

Basic Entry
Chart Trader
FX Pro
FX Board
SuperDOM
Order Ticket

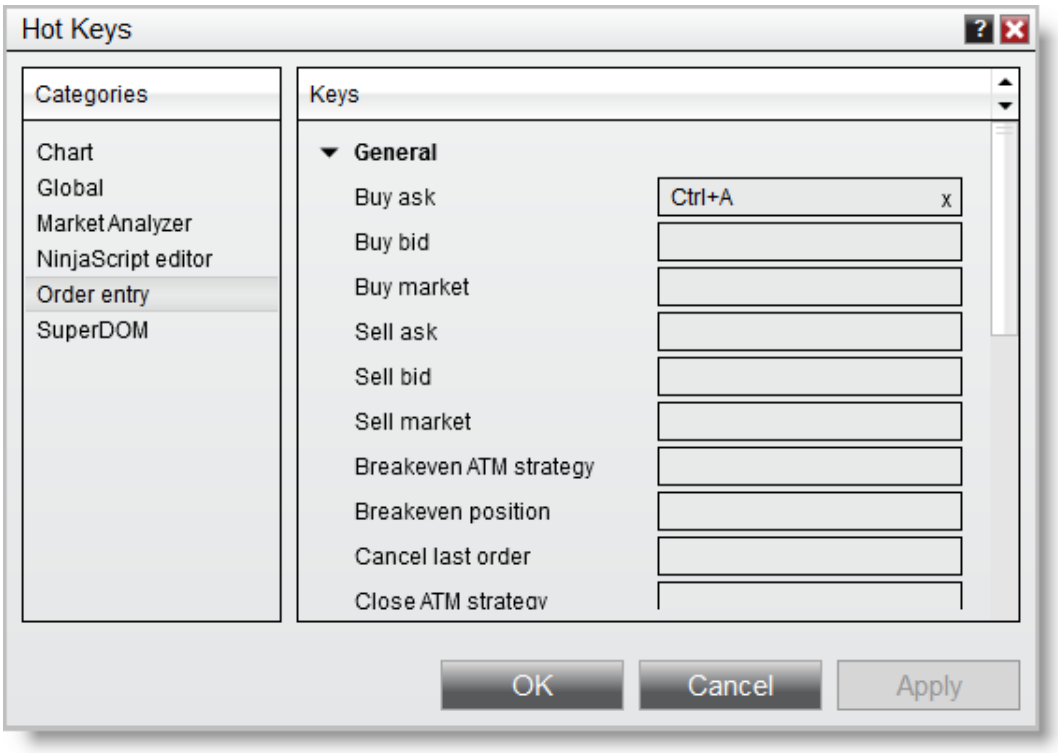
Identifying the Active Window

The active window is usually the window that was last clicked on and has the top most view. You will also notice that the active window's close button in the upper right hand corner is red compared to an inactive window that has a grey close button.

The screen capture below to the left shows the SuperDOM with **ES 06-14** selected as the active window whereas the screen capture in the right shows the SuperDOM with **NQ 06-14** selected and active. In the left screenshot any order **Hot Keys** would be submitted to the ES 06-14. In the right screenshot any order **Hot Keys** would be submitted to the NQ 06-14.



Pre-defined order actions and definitions



Pre-defined order actions

Buy Ask	Submits a buy limit order at the current ask price
Buy Bid	Submits a buy limit order at the current bid price
Buy Market	Submits a buy market order
Sell Ask	Submits a sell limit order at the current ask price
Sell Bid	Submits a sell limit order at the current bid price
Sell Market	Submits a sell market order

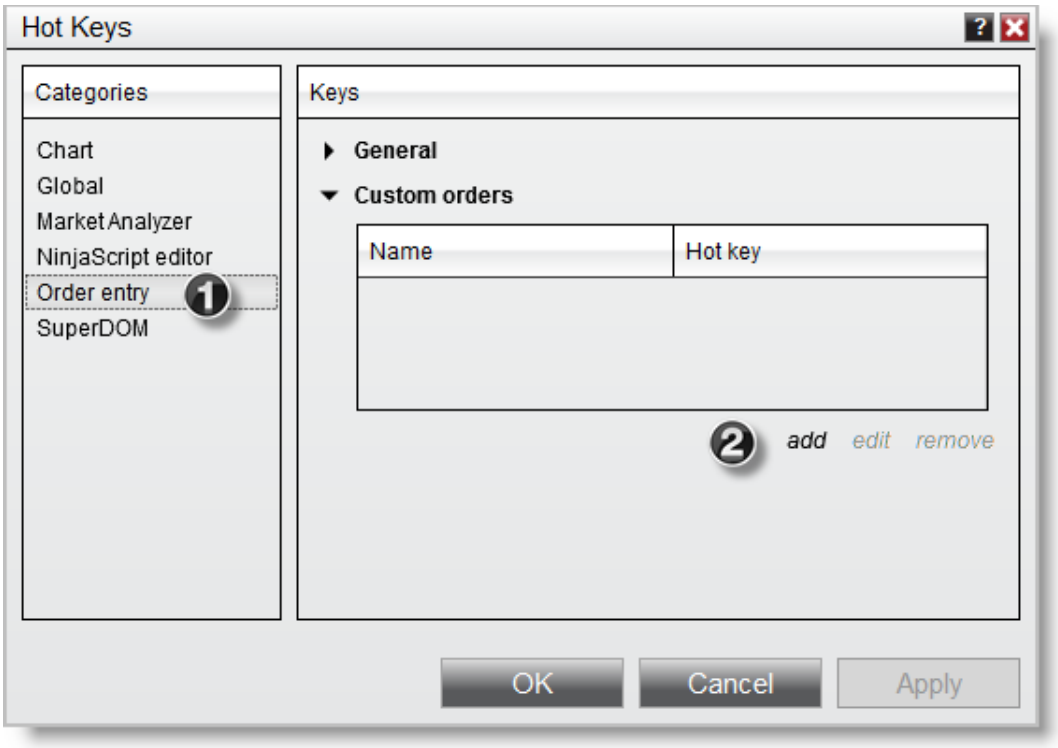
Breakeven ATM Strategy	Modifies your stop loss order to your break-even price for the ATM Strategy position on the active window
Breakeven Position	Modifies any stops to your break-even price for any open position.
*Cancel last Order	Cancels the last submitted order
Close ATM Strategy	Cancels any pending orders and exits any open positions activated by an ATM Strategy
Close Position	Closes any open position on the active order entry window
*Decrease Last Order Price	Decreases the price of the last submitted pending order by one tick
*Increase Last Order Price	Increases the price of the last submitted pending order by one tick
*Modify Last Order to Fill	Modifies the price of the last submitted pending order by 15 ticks past the last traded price in order to fill the order.

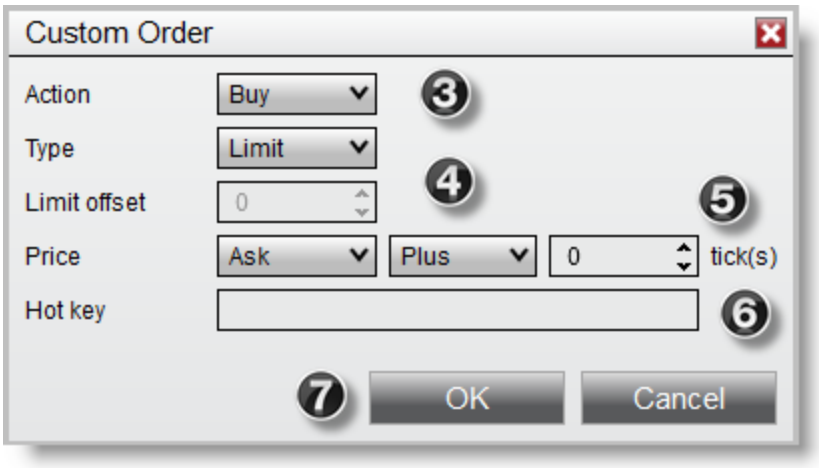
Reverse	Closes your open position and any related ATM orders and submits a market order in the opposite direction to reverse your open position.

*__Note__: For any **Hot Key** that references "last order," such as "Cancel last order," last order is defined as:
The last order submitted that is not a stop or target order generated by an ATM Strategy

How to create custom order actions

NinjaTrader allows you to create custom order actions within the Order Entry section of the Hot Key window.





To create a Custom Order action with an associated Hot Key

1. Select the Order Entry category of the Hot Key window.
2. Left click "add".
3. Select an Action.
4. Select an Order type (Limit offset allows you to enter the number of ticks your limit order will be submitted away from the stop order when using a StopLimit Order type).
5. Select the price the order will be submitted at. You can choose a number of ticks above (Plus) or below (Minus) the current Ask or Bid.
6. Select the hot key to use to submit the **Custom Order**.
7. Press the OK button.

To remove Custom Order actions

Left click the Custom Order and select "remove"

10.15 Hot List Analyzer

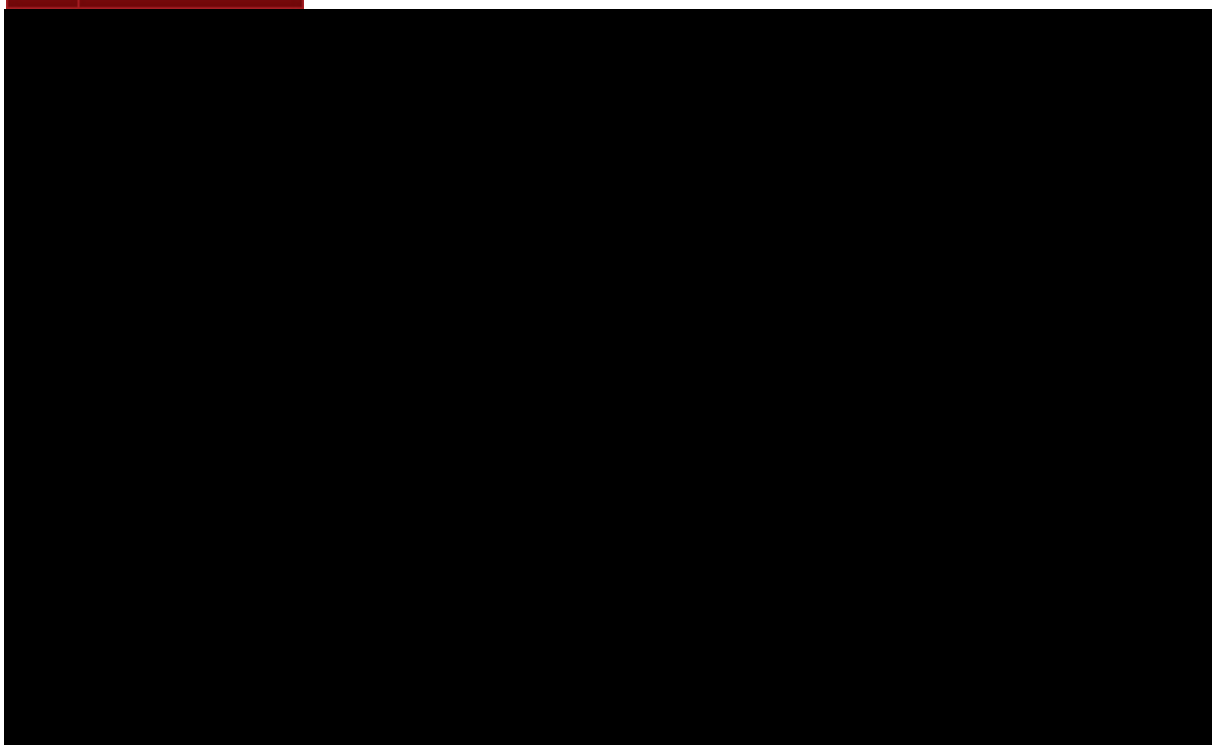
Hot List Analyzer Overview

You can access the **Hot List Analyzer** window from within the NinjaTrader Control Center window by left mouse clicking on the menu **New**, and then selecting the menu item **Hot List Analyzer**.

The **Hot List Analyzer** is designed to work with the same functionality as the **Market Analyzer**, with the added ability to dynamically add equity instruments based off of various **Hot Lists** supplied by your data provider.

10.15.1 Using the Hot List Analyzer

Play Video



▼ Understanding Hot Lists

What are hot lists?

Hot lists are a unique list of stocks which are constantly being monitored and updated by your data provider. These lists will give you valuable information which meet a specific criteria. For example, if you wanted to know which stocks trading on the NYSE had the highest amount of volume today, you could select the "*NYSE Most Actives"* hot list.

Who can I use hot lists with?

Hot lists can be used with the following data providers:

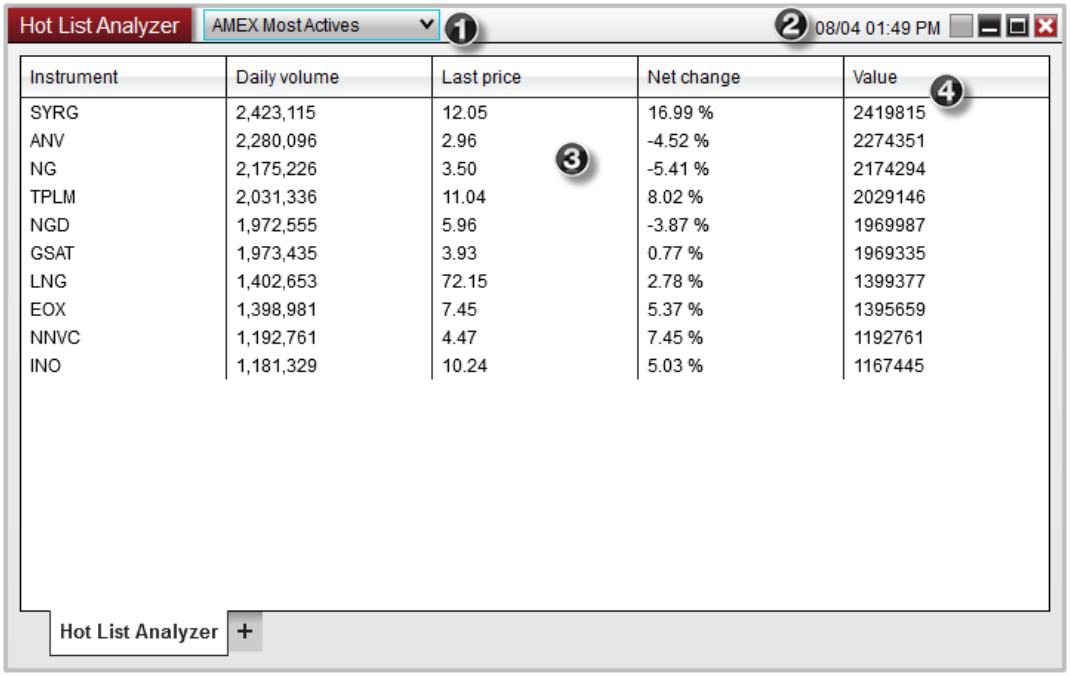
- Interactive Brokers
- TD AMERITRADE

What type of hot lists are there?

Hot lists are not hard coded into NinjaTrader and the type of lists that are available will vary depending on your data provider and are subjected to change. NinjaTrader's Hot list selector will display all current available hot lists from your provider. If you would like to know what types of hot lists you can get with your data provider, the best way to determine this information is to establish a connection to the data provider and browse the Hot List Selector on the title bar of the Hot List Analyzer.

Understanding the Hot List Analyzer Display

Display Overview



1. Hot List Selector	Sets the hot list as determined by the data provider.
2. Last Update Time	Displays the time that the hot list was last updated by the data provider
3. Hot List Grid	Grid displays various instrument related information, similar to the Market Analyzer Columns
4. Hot List Value Column	Displays the value of each instrument in the current selected hot list

Right Click Menu

Create Instrument List	Dynamically creates a list of all the current instruments in the Hot List Analyzer display which can be accessed from the Instrument Lists window
Alerts	Opens the Alerts window to configure user defined alerts to be armed
Columns	Opens the Columns menu to configure user defined columns to be displayed
Row Filter	Enables / Disables row filters
Send To	Loads the selected instrument into another NinjaTrader window
Always On Top	Sets the **Hot List Analyzer** window to always be on top of other windows

Show Tabs	Sets if the Hot List Analyzer displays tabs
Export	Exports the Hot List Analyzer contents to "CSV" or "Excel" file format
Find	Search for a term in the Hot List Analyzer
Print	Displays Print options
Share	Displays Print options
Reload All Historical Data	Reloads the historical bar data used for Indicator calculations
Reload NinjaScript	Reloads all of the NinjaScript columns to recalculate the current values
Templates	Access the templates menu to save / load custom Hot List Analyzer settings
Properties	Set the Hot List Analyzer properties

Hot List Analyzer vs Market Analyzer

What are the differences between the Hot List Analyzer and Market Analyzer?

The primary difference between the **Hot List Analyzer** and the **Market Analyzer** is that while the **Market Analyzer** allows you custom create rows of Instruments, the **Hot List Analyzer** does not. Any instruments that are added to the **Hot List Analyzer** are dynamically added based on the **Hot List** you have selected. The **Hot List Analyzer** also does not allow you to remove instruments from the current display.

Creating and customizing an Instrument List

If you would like to further customize a list of instruments based off a hot list you have displayed, you can create a custom list of instruments by right clicking on the **Hot List Analyzer** display and selecting **Create Instrument List.** Once the **Instrument List** has been created, you can open the Instrument List window to

customize the list of instruments. You can also add the **Instrument List** to the Market Analyzer for run analysis on your custom hot list.

10.15.2 Customizing the Hot List Analyzer

For information on how to configure and customize the display of the Hot List Analyzer, please refer to the help topics on the Market Analyzer

10.15.3 Hot List Analyzer Properties

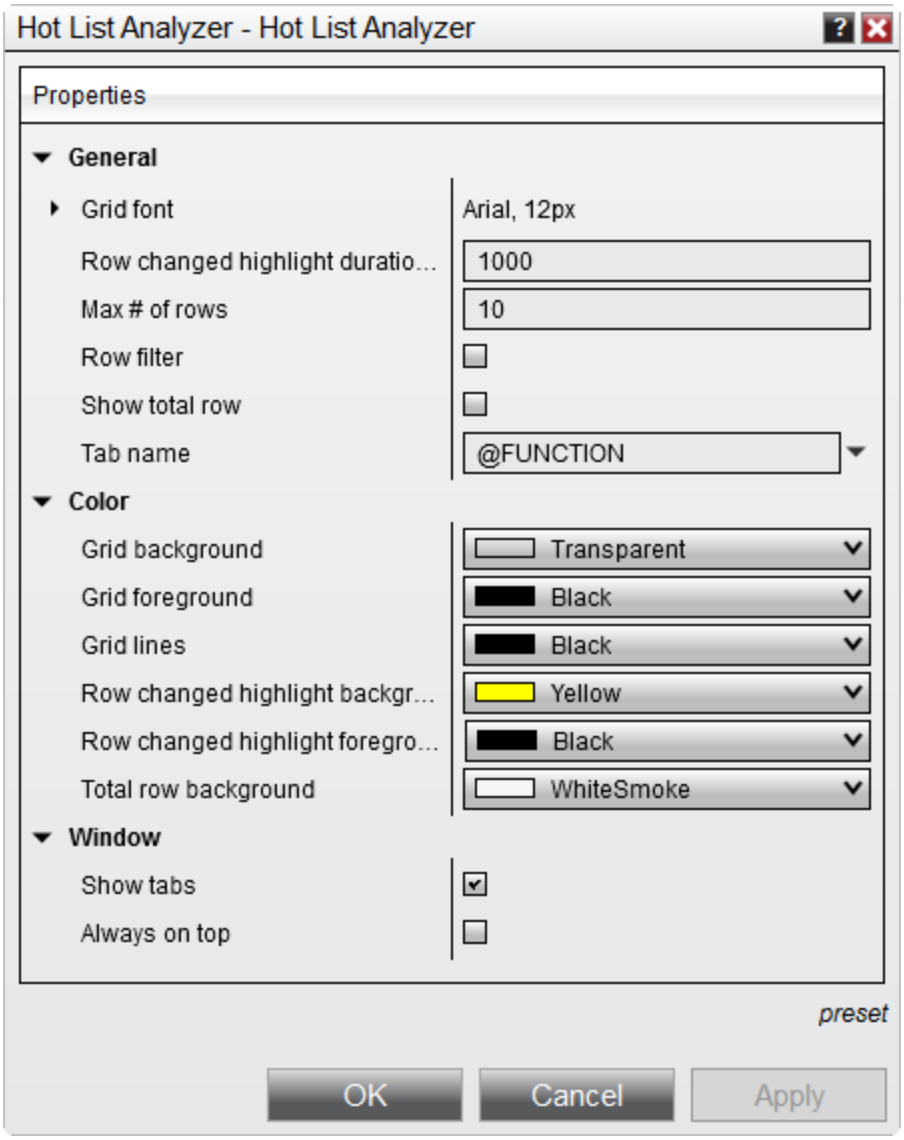
The Hot List Analyzer can be customized to your preferences in the Hot List Analyzer Properties window.

How to access the Hot List Analyzer properties window

To access the Hot List Analyzer Properties window, press down on your right mouse button inside the Hot List Analyzer window and select the menu **Properties...**

Available properties and definitions

The following properties are available for configuration within the Hot List Analyzer Properties window



Property Definitions

General	
Grid font	Sets the font
Row change highlight duration (ms)	Sets the duration (in seconds) the instrument cell will remain highlighted. A value of zero will disable highlighting.

Max # of rows	Determines how many instrument rows can be added to the Hot List Analyzer
Row filter	Enables/Disables the automatic filtering of rows from the grid display based on the Filter Conditions of the columns.
Show total row	Enables/Disables the Total row in the Hot List Analyzer window display grid
Tab name	Sets the tab name
Color	
Grid background	Sets the default color of the display grid background
Grid foreground	Sets the default color of the text in a cell
Grid lines	Sets the color of grid lines
Row changed highlight background	Sets the color for the row change highlight background
Row changed highlight foreground	Sets the color for the text in the row change highlight
Total row background	Sets the color of the Total row background
Window	
Show tabs	Enables/Disables the tab control
Always on top	Enables/Disables if the window will be always on top of other windows.

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

10.16 Instrument Lists

Instrument Lists Overview

Instrument Lists can be configured by left mouse clicking on the **Tools** menu within the NinjaTrader Control Center and selecting the menu item **Instrument Lists**.
NinjaTrader supports grouping together instruments into easy to access lists. There are several uses for an instrument list:

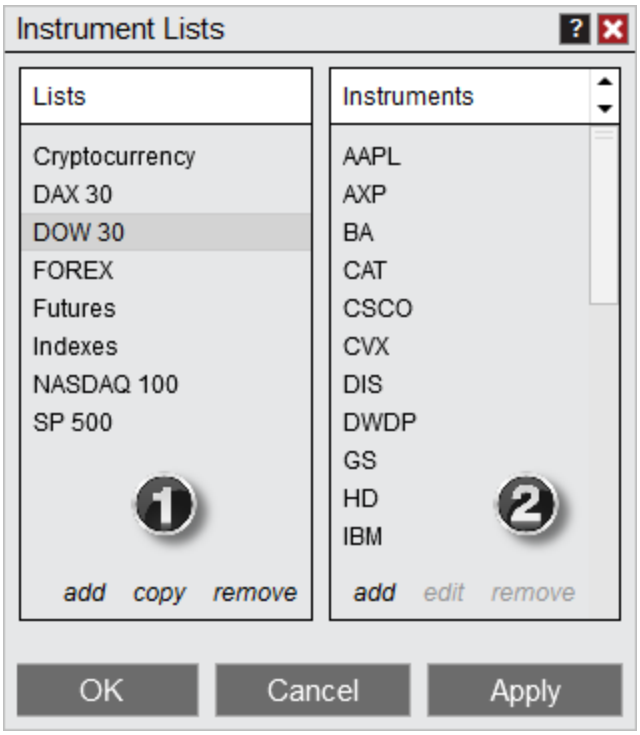
- Backtesting in the Strategy Analyzer
- Quickly adding multiple instruments to the Market Analyzer window
- Creating lists from the instruments from the Hotlist Analyzer window.
- Organizing instruments in the **Instrument Selector** for ease of access.

› Working with Instrument Lists
› Updating Splits and Dividends

10.16.1 Working with Instrument Lists

NinjaTrader comes predefined with a few instrument lists that are kept up to date on our server. You also can create and manage your own instrument lists via this dialog.

Understanding the Instrument Lists window



1. **Instrument Lists**

The **Lists** section displays a list of Instrument Lists that can be configured. Please see "*Adding or Removing **Instrument Lists***" section below for information on how to add and remove instrument lists.

2. **Instruments**

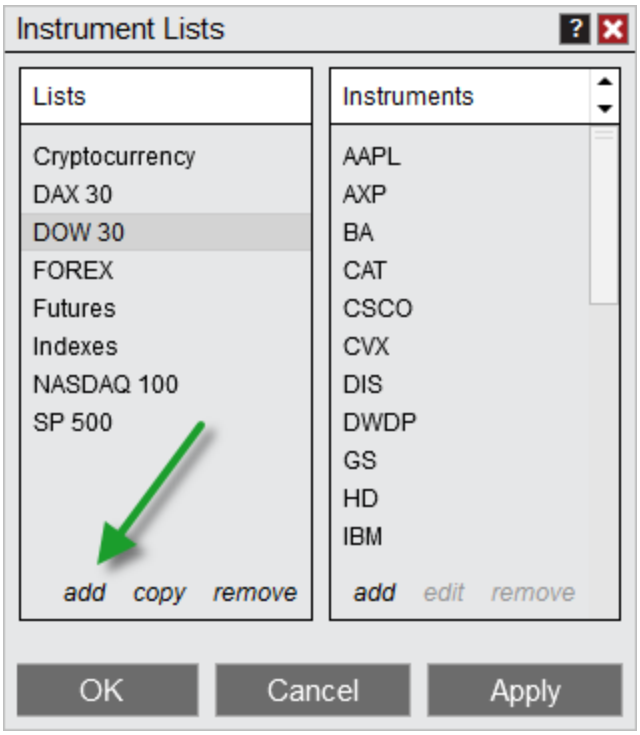
The **Instruments** section displays the selected lists instruments. Please see "*Adding or Removing instruments to a list*" section below for information on how to add and remove instruments to an instrument list.

Adding or Removing Instrument Lists

Adding an Instrument List

To create a new instrument list:

1. Select "add" in the Lists section of the **Instrument Lists** window
2. Type in the name of the instrument list you wish to add



Removing an Instrument List

To remove an instrument list:

1. Select the list you wish to remove in the Lists section of the **Instrument Lists** window.
2. Select **remove**

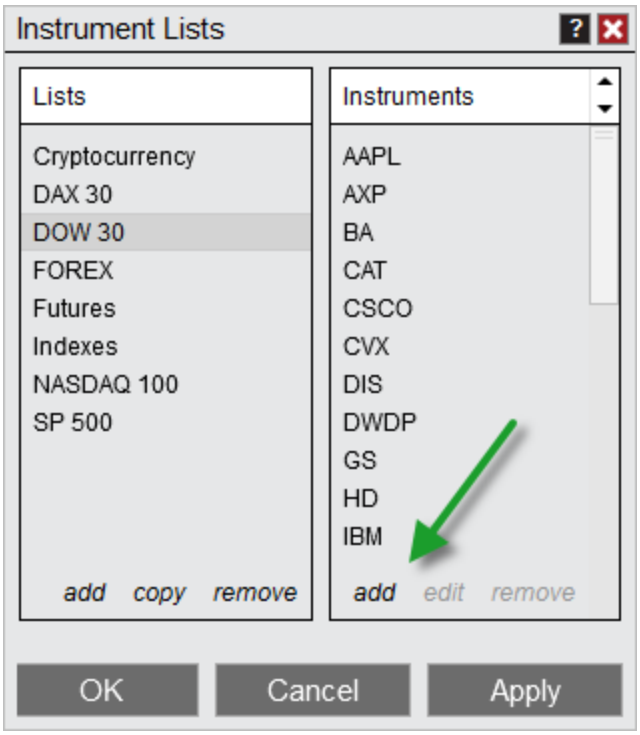
> **Note:** A predefined instrument lists **cannot** be removed.

Adding or Removing instruments to a list

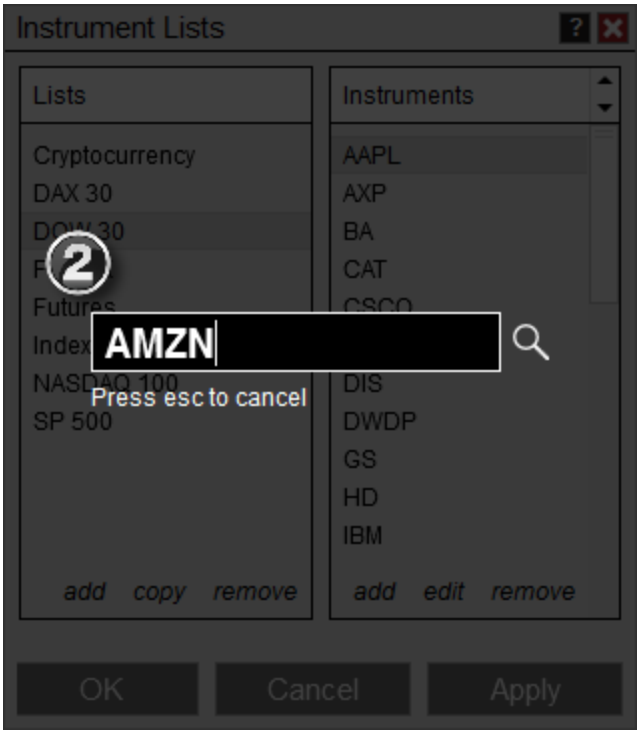
The collection of instruments that are associated to the selected instrument list are displayed in the "**Instruments**" section.

Adding an Instrument

To add an instrument to an instrument list



1. Start typing on the keyboard or select "add" for the overlay instrument selector to be triggered.



2. Type in the instrument that you want to add or select the magnifying glass to search for an instrument.

The instrument is added to the instrument list and will now be available throughout the NinjaTrader application.

Removing an Instrument

To remove an instrument from an instrument list:

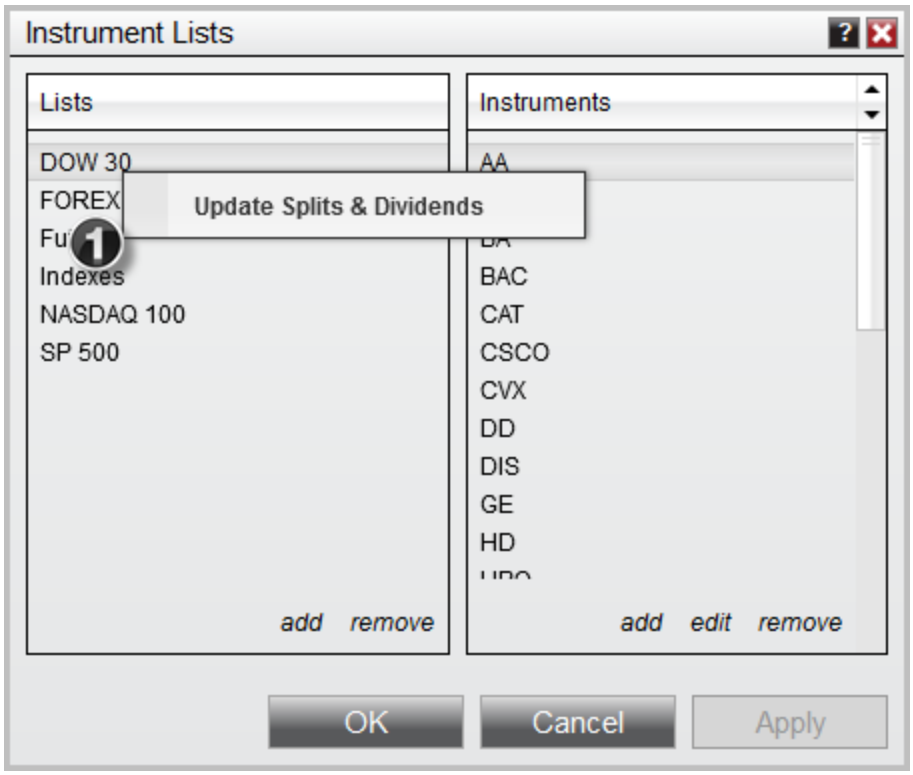
1. Left mouse click on the instrument you wish to remove from the instrument list in the right pane of the **Instrument Lists** window.
2. Press **remove**

10.16.2 Updating Splits and Dividends

You can quickly update all Splits and Dividends on an instrument list via the **Instrument Lists** window. Please follow the guide Adding Splits and Dividends for details on how to enable NinjaTrader to do updates on the selected instrument list.

Updating Splits and Dividends

To trigger and update of all Split and Dividend data on all instruments on a specific instrument list:



1. Right click on the Instrument List you want to trigger the mass update for and select **Update Splits & Dividends**.

NinjaTrader will now request historical splits and dividend information from your provider and populate the information in your local database.

10.17 Instruments

Instruments Window Overview

The **Instruments** window can be accessed by left mouse clicking on the **Tools** menu within the NinjaTrader Control Center and selecting the menu item **Instruments**. NinjaTrader supports multiple connectivity providers and therefore manages a single instrument instance (master instrument) which maintains the parameters required to establish market data and order permissions through the various connectivity sources. The **Instruments** window manages the instrument data and can add or remove instruments from the database. NinjaTrader maintains a predefined database of commonly traded instruments on our server and your local database is updated automatically on NinjaTrader application startup.

10.17.1 Instrument Types

NinjaTrader supports the following Instrument types, although what your individual connected to provider can support will vary. Please consult this link for more info.

CFD (Contract for Difference)

- CFD instruments can be charted and traded
- Master instrument prefaced with a @ sign.
- The PnL will be currency converted to the account base denomination (execution tab "Rate" can be used to see conversion rate details), NinjaTrader uses the corresponding CFD FX instruments to make the conversion.

Cryptocurrency

- Cryptocurrencies can be charted but not traded.
- They have a dedicated Depth Chart window for analysis

Forex

- Forex instruments can be charted and traded

- These instruments do not trade on a centralized ("exchange") market, thus pricing differences between providers are to be expected.
- The PnL will be currency converted to the account base denomination (execution tab "Rate" can be used to see conversion rate details), NinjaTrader uses the corresponding Forex instruments to make the conversion.

Future

- Futures contracts can be charted and traded
- They need to be requested with the specific desired expiry
- Per default NinjaTrader will built continuous 'merged' contracts, for more info please consult this link.
- The PnL will be currency converted to the account base denomination, NinjaTrader uses the CME FX futures (6A, 6B, 6E, etc) to make the conversion.

Index

- Indices can be charted but not traded.
- Master instrument prefaced with a ^ sign.

Stock

- Stocks can be charted and traded.

Option

- Options can be charted (real-time only) and traded
- Options master instruments and properties are dynamically injected into the database and therefore are not search-able. Also they are removed as the database file is reset or deleted.
- Options are requested through the dedicated Options Chain window
- Options can be less liquid than futures, especially before major economic data releases or in less popular expiration's or strikes. Using market orders including use of Close or Flatten all could result in a fill worse than using limit orders.
- Options are not supported for NinjaScript usage

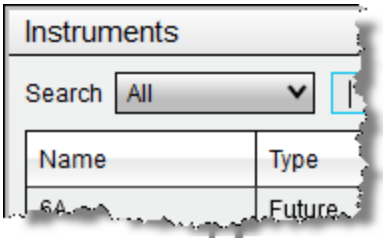
10.17.2 Searching for Instruments

NinjaTrader has a predefined database of commonly supported instruments that you can search through.

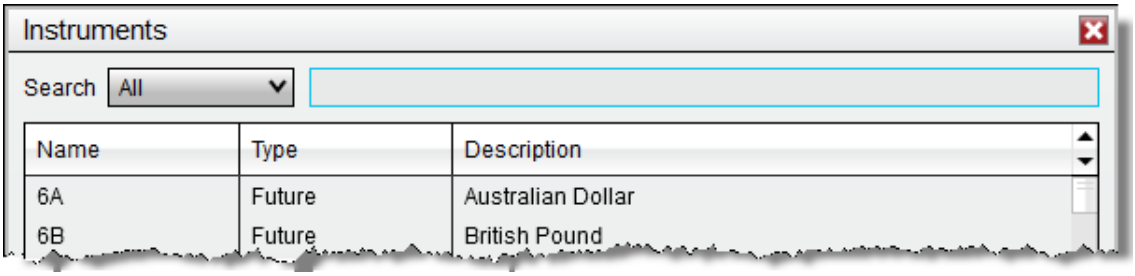
Searching Instruments

To search for an instrument within the database:

1. From the **Instruments** window, optionally select an available instrument type using the **Type** drop down menu to narrow down your search.

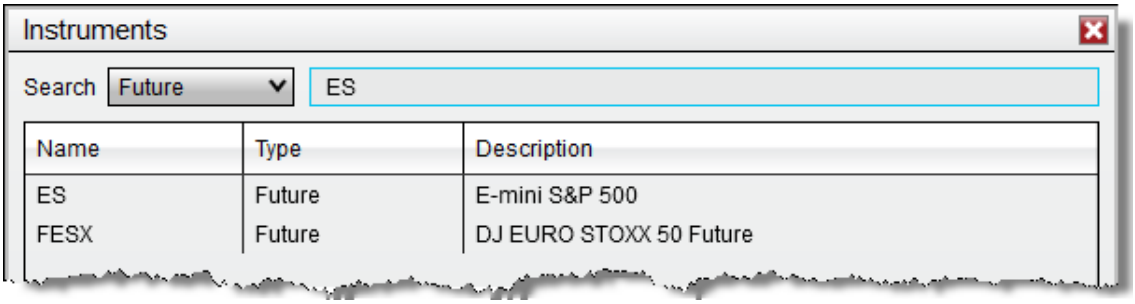


2. Enter any search parameters in either the **Search** field. Typing in capital letters will search for via the instrument name, typing in lowercase letters will search via the instrument description.



3. As soon as you begin typing the **Instruments** window will immediately begin filtering your results.

The image below displays the results of searching for any Futures with "ES" in the instrument name.



You can double left mouse click on any search result to bring up the **Instrument Editor** window for the selected instrument. Please see the Editing Instruments page for more information on how to edit instruments.

10.17.3 Managing Instruments

NinjaTrader installs with a predefined database of commonly traded instruments that is updated by the NinjaTrader data server automatically. There will be rare occasions where you may need to manage the instruments manually in the Instruments window.

Adding an Instrument

1. Press the "new" button which opens the Instrument window
2. Add instrument parameters including the symbol mapping for your connectivity provider(s)
3. Press the "OK" button

Equities can be alternatively added by typing the symbol name into an open chart or Market Analyzer and pressing the "Enter" key on your keyboard. Please see the "*How to change a Data Series*" section of the Working with Price Data page of the Help Guide for more information.

> **Note**: The **Master instrument** field can only contain letters, digits, and underscores.

Removing an Instrument

1. Select an instrument in the instrument grid
2. Press *"remove"*

Editing an Instrument

1. Select an instrument in the instrument grid
2. Double left click the Instrument or select the instrument and press "edit"

For more information on editing instruments please see the Editing Instruments section of the help guide.

Editing or Removing Multiple Instruments at a Time

NinjaTrader supports editing or removing multiple instruments at once. In the **Instruments** window select multiple instruments by holding CTRL to toggle selection of instruments one at a time or SHIFT to toggle selection of all instruments between the first selected instrument and the next selected instrument.

When editing multiple instruments any property that is not the same between all selected instrument will display as blank, allowing you to override the setting for all selected

instruments or leave if you do not want to change that specific property. If a property is the same between all selected instruments the selected property will show its common property setting.

> **Note**: Although most database operations are instant, please be prepared for longer database saving times when editing larger selections of instruments.

Resetting Instrument Defaults

If you ever needed to restore the Instruments to default settings you can do so via the NinjaTrader **Control Center** `Tools` menu and accessing `Database`. In the **Database** window use the **Update instruments** function. For more information on the Database window please see the Database section of the help guide.

10.17.4 Editing Instruments

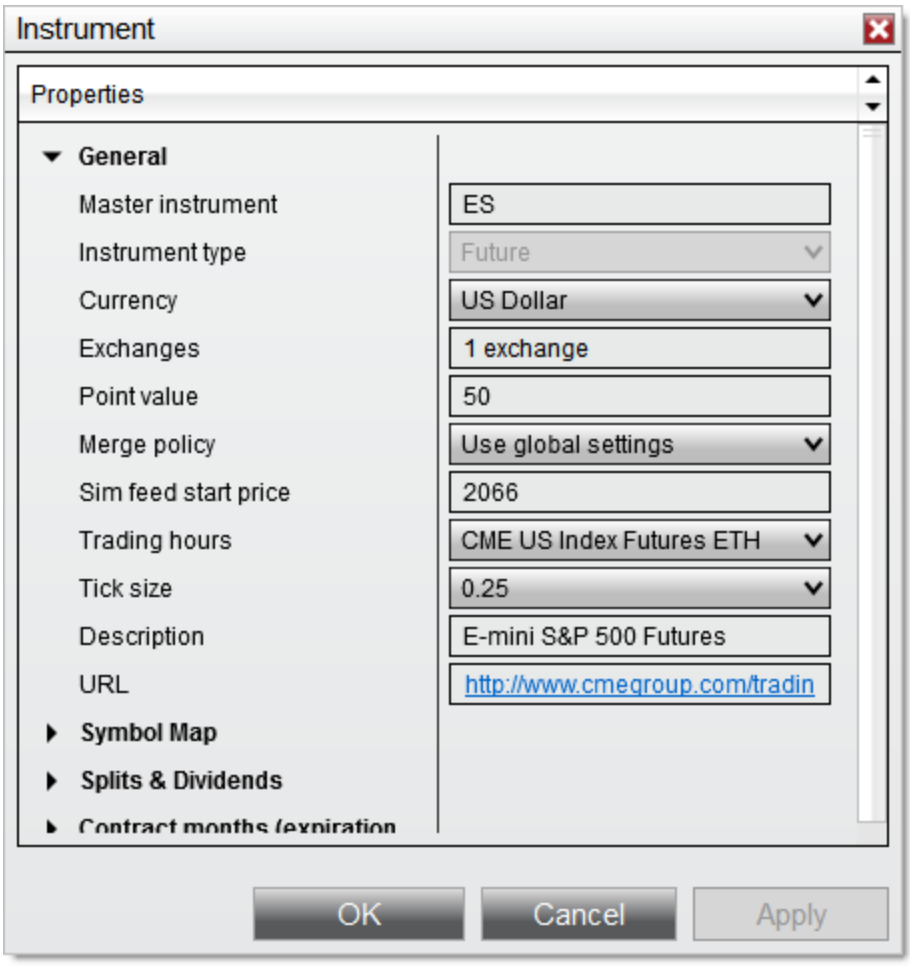
The **Instrument** window displays all parameters that define an instrument including symbol mappings to your connectivity provider and symbol level commission values. The editor allows you to change or add parameters to an instrument's profile. In general, instruments that are predefined in NinjaTrader do not require any parameter modification. However, you may want to override your global commission settings if a particular symbol has a unique commission structure.

In the **Instruments** window, once an instrument is selected in the instrument grid, you can double left mouse click or press the **edit** button to open the **Instrument** window.

▼ Understanding General section

General Section

The **General Section** in the Instrument window displays parameters that uniquely define an instrument.



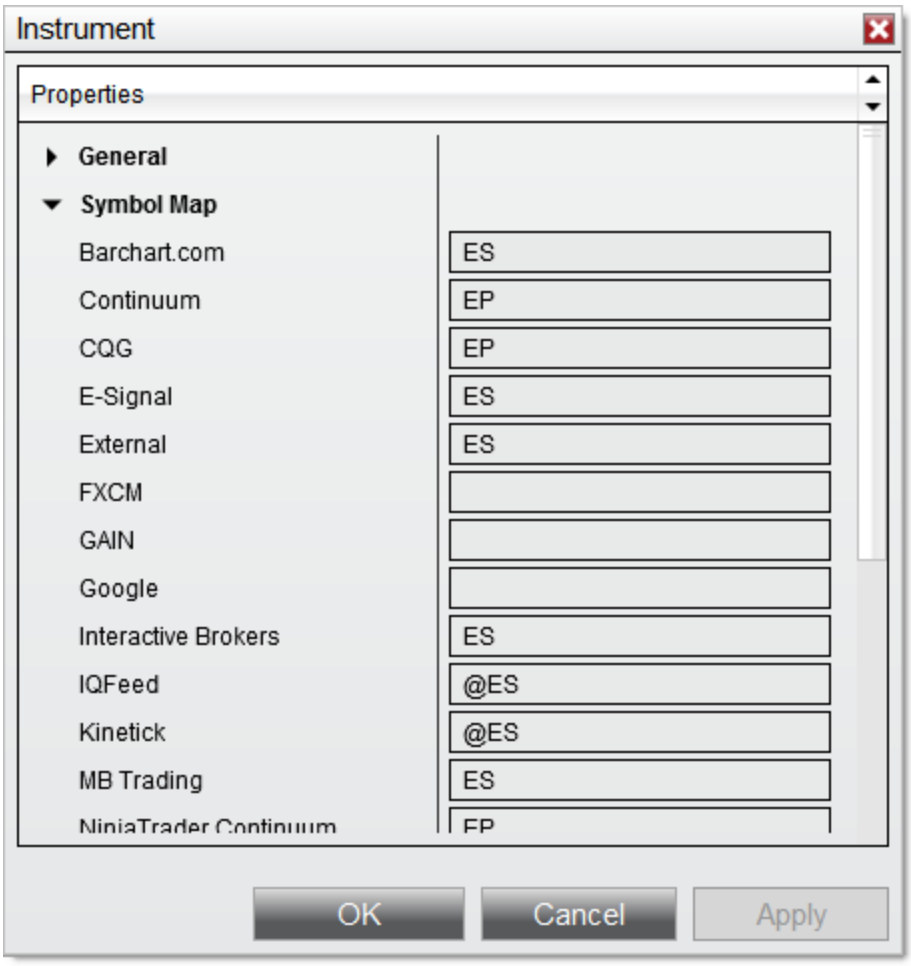
Master instrument	The NinjaTrader master name of the instrument
Instrument type	The instrument type (asset class)
Currency	The currency the instrument trades in
Exchanges	Click to select what exchanges the instrument trades on
Point value	The currency value of 1 point of movement for the instrument

Merge policy	The merge settings applied to historical data. (See the Data Tab section of the Help Guide for more information on merge policies and to set the global merge policy)
Sim feed start price	The starting price for the internally generated data feed (Simulated Data Feed connection). The price is automatically set by NinjaTrader using the last seen price from a live data feed connection.
Trading hours	Sets the default trading hours for the instrument. (See the Session Manager section of the Help Guide for more information)
Tick size	The increment value the instrument trades in
Description	Description of the instrument
URL	The website address of the instrument definition

Understanding the Symbol Map section

Symbol Mapping

If you add a new instrument that is not already in the NinjaTrader instrument database, you will need to map the new instrument to the symbol used for the connectivity provider (broker or data feed) that you will be requesting data from. Most instruments in the database are already mapped.

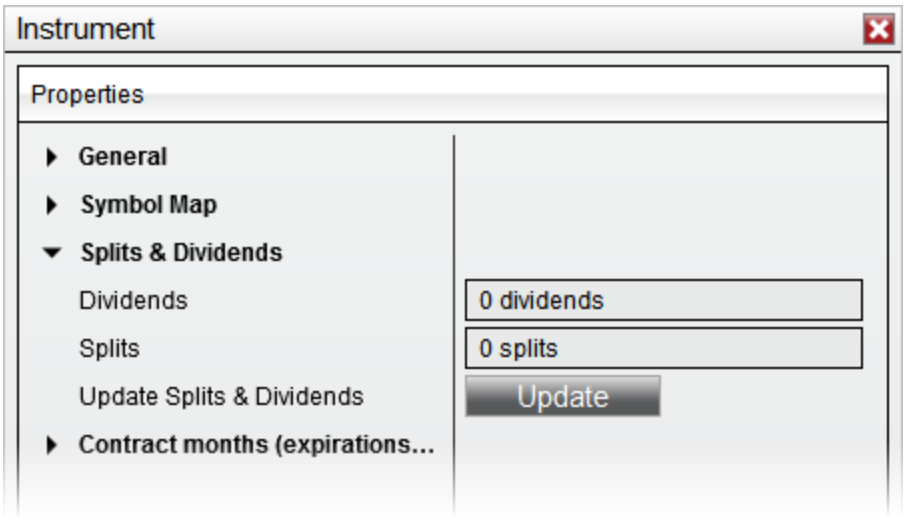


Understanding Splits & Dividends section

Splits & Dividends

With an equity instrument selected, the Splits & Dividends section will be available for editing. NinjaTrader will split and dividend adjust historical chart data based on the information defined per instrument and if options are enabled to do so. Some market data providers provide already adjusted data while others do not. Please see the Adding Splits and Dividends page of the help guide for more information on adding split and dividend data to an instrument.

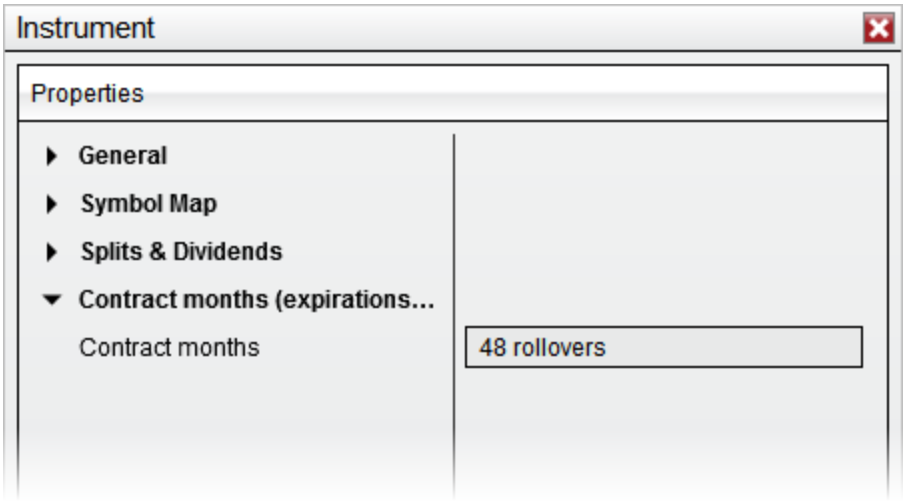
Clicking the **Update** button will attempt to download Split & Dividend data directly from your provider so that manual entry is not necessary.



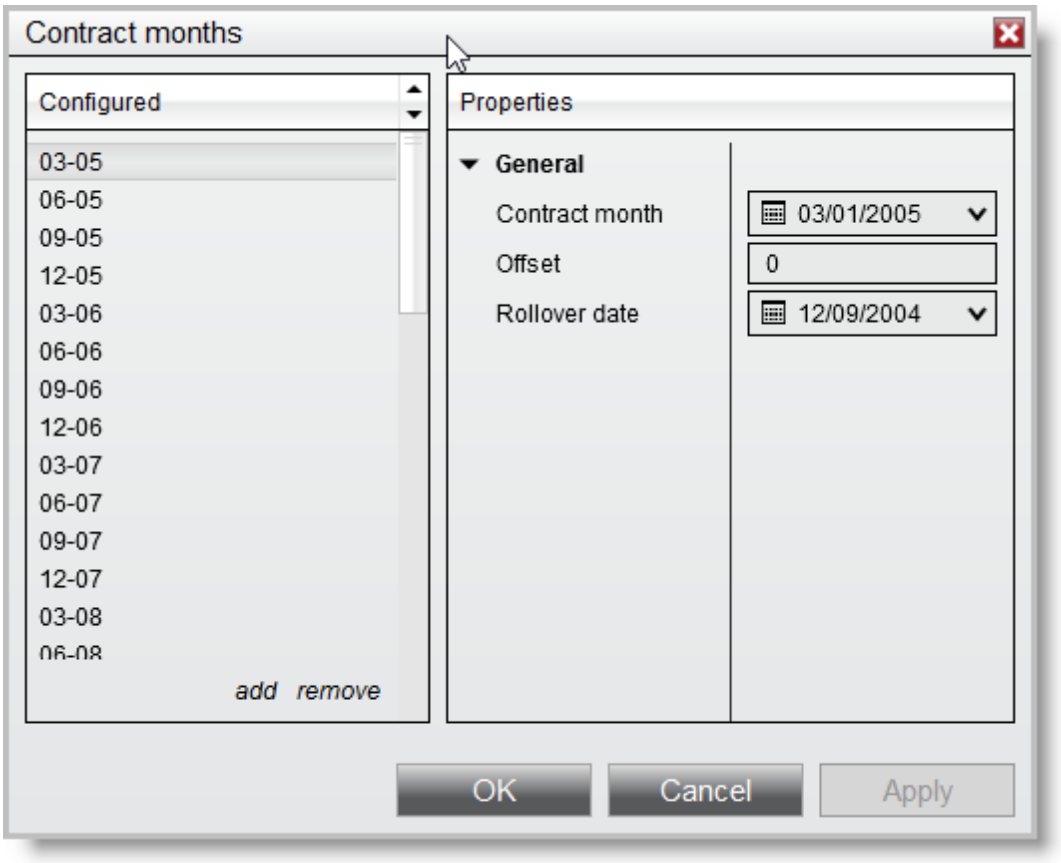
Understanding the Contract months section

Contract Months

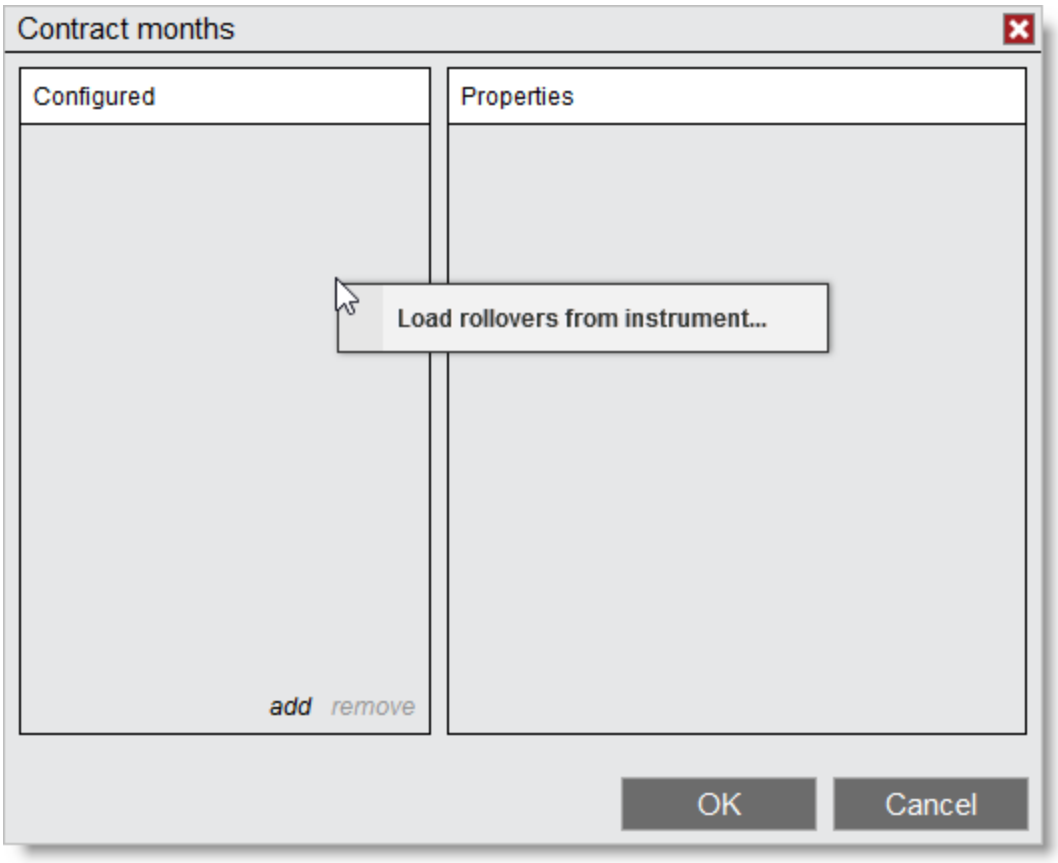
The **Contract months** section shows the contract months with associated rollover dates. This information is automatically downloaded from the NinjaTrader server whenever you are connected to your live data feed or the Simulated Data Feed.



You can open up the defined contract months by left mouse clicking in the **Contract months** field.



You can add and remove contract months by selecting the **add** and **remove** buttons in the bottom of the Configured section.
You can also copy contract months that are defined in another instrument by right clicking in the Configured section then selecting **Load rollovers from instrument...**



Contract Month Properties

Once a contract is selected in the Configured section you may edit it properties. The **Contract month**, **Offset value**, and **Rollover date** are used when NinjaTrader automatically merges historical data.

The **Offset** value is used to connect the last value of a contract month with the next one.

Although NinjaTrader will attempt to download the **Offset** values from the data server, if they do not exist on the data server, they will be calculated locally. **Offsets** are only downloaded when the "Offset" field is left blank and the rollover date matches the date defined on the server.

When NinjaTrader will calculate the **Offset** value locally:

- The Offset field in the Contract Months window is blank
- Historical data exists in the database for both the new and old contract near the rollover date

- The Merge Back Adjusted policy must be selected in the **Market data** category of the Options menu
- You must be connected to your data provider and requesting data for the instrument

How NinjaTrader will calculate the **Offset** value locally:

- Request the old and new expiry's daily price data for calculations
  - If daily data is not offered by the data provider, use minute data
  - If minute data is not offered by the data provider, default **Offset** value will be 0
- One day prior to the rollover date, calculate the difference between the close price of the new expiry and the close price of the old expiry. This is the **Offset** value.
  - If you wish to overwrite the calculated **Offset** value you can input in your own
  - When using minute data, the close price at **the ending time as defined in the default session template for the instrument will be used**

Notes:
1. If you inputted your own **Offset** value, it will be overwritten by values downloaded from the data server if it exists there. To prevent this you will need to ensure that your rollover date is not the same as the ones coming from the data server.
2. The rollover date is the date to roll into the selected contract month and **NOT** out of.

10.17.5 Rolling Over Futures Contracts

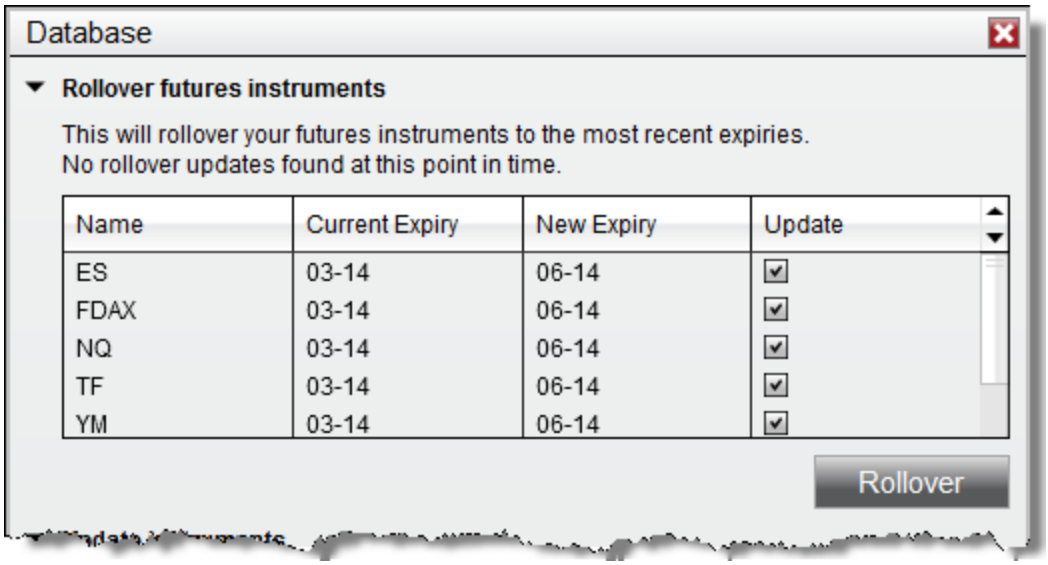
Batch Rollover

NinjaTrader allows batch rollover of the contract expiry of all instruments across all instrument lists and windows on all open workspaces. To perform this batch rollover please see the steps below.

1. Select the **Tools** menu within the Control Center followed by the **Database Management** menu item.
2. The grid for the Rollover futures instruments section will show each instrument that is eligible to be rolled over. A contract is eligible to be rolled when today's date is greater then or equal to the rollover date defined for the instruments next contract month.
3. When selecting "Rollover" any instrument with a check mark in the Update column will be updated to the contract month in the New Expiry column.

Note: Rolling over futures instruments will update the expiry of the instruments across all instrument lists and windows using the instruments on all open workspaces. These changes on workspaces will need to be saved should you wish to preserve them. If there are areas you still wish to use the old expiry with please be sure to switch them back to the old expiry or do not rollover at this time.

Note: NinjaScript strategies are not rolled forward and must be manually rolled over.



Manual Rollover

You can choose to manually rollover each window to the next contract month. This may be useful for when you want to only partially roll over your workspace.

Manually rolling the contract is done by typing in the next contract expiry in the windows instrument selector.For example: "ES 09-16" to "ES 12-16".

Rollover of Drawing Objects

To rollover your drawing objects you can complete either a batch rollover or a manual rollover. Closing your chart and opening a new chart would result in the drawing objects being lost.

When rolling over, your drawing objects will be kept at the same price level they were originally placed. If a **Merge Policy** of **MergeBackAdjusted** is being used, this will result in the adjusted bars moving the price away from the original placement of the drawing objects. Using a **Merge Policy** of **MergeNonBackAdjusted** will keep the previous contracts prices the same, keeping the drawing object's placement with the bars.

If you have a strategy that places drawing objects, the drawing objects will be redrawn when rolling over based on the bar's placement after rollover.

For more information on merge policies, see the Merge Policy section of this guide.

10.17.6 Adding Splits and Dividends

You can automatically update an instrument with historical split adjustment data from within the **Instrument** window. You can choose to update split information from the following connections:

1. IQFeed
2. Kinetick (you must have a subscription - the free Kinetick EOD does **NOT** provide splits and dividend information)

Adding Splits via the instruments window

To automatically update an instrument with historical split data follow the steps below. If you have already defined one of the connections above then you may skip step 1.

1. Create a connection to one of the providers above, see the Connection Guide for more information
2. Connect to the provider by left mouse clicking on the menu **Connect** and selecting your connection.
3. Open the **Instruments** window by left mouse clicking on the menu **Tools** and select the menu item **Instruments**.
4. Select a single stock or select multiple stock that you wish to update with historical split data Note: Hold down **CTRL** to individually toggle instrument selection or **SHIFT** to toggle selecting a group of instruments.
5. Right click on one of the selected stocks and left mouse click the menu **Update Splits**.

NinjaTrader will now request historical splits information from your provider and populate the information in your local database.

> **Notes**:
> The **Update Splits** menu item is only enabled when you are connected to one of the providers mentioned above.
> At this time, no supported connections provider dividends. Dividends must be manually added.

Adding Splits for a predefined instrument list

You can perform the same steps above on a predefined instrument list by going to the NinjaTrader **Control Center** **Tools** menu and selecting **Instrument Lists**. Here you can right click on the instrument list name and select **Update Splits**.

10.17.7 TradeStation Symbol Mapping

The following section outlines the requirements for proper TradeStation to NinjaTrader symbol mapping when using the **Automated Trading Interface** (both DLL or Email interface).

Note: Mapping is NOT required for stocks or Forex symbols.

How to map an individual futures contract

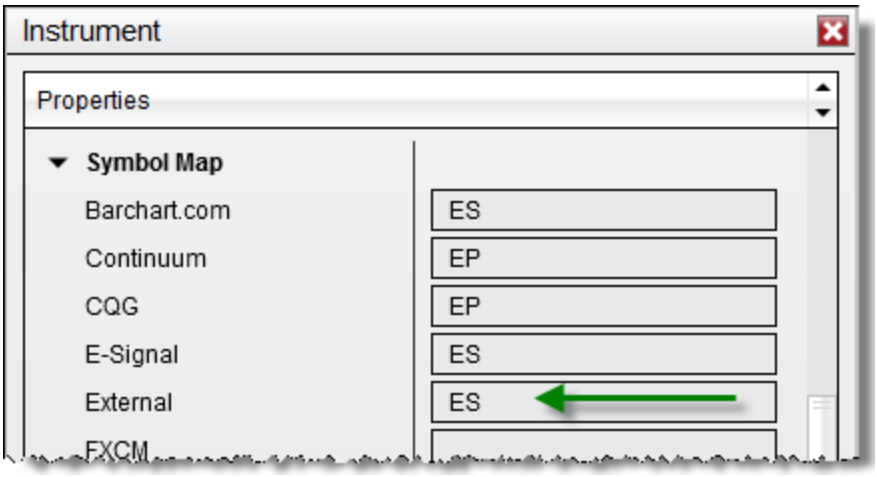
Mapping an Individual Futures Contract

To send orders via the NTExternalFeed strategy through the ATI to NinjaTrader from an individual futures contract such as the Emini S&P June contract "ESM14" in TradeStation you have to correctly set up mapping within NinjaTrader.

For this example, let's map the "ESM14" contract.

1. From the NinjaTrader Control Center window select the menu **Tools** and select the **Instruments** menu item.
2. Highlight the ES contract from the main grid which is the NinjaTrader S&P 500 Emini contract.
3. Press the "edit**"** button to bring up the **Instrument** window.
5. In the Symbol Map category for the **External** data feed set the value to "ES".
6. Press the "**OK"** button.

* The symbol map name "ES" in the image below needs to be the TradeStation symbol base name.



This procedure would be repeated for any other symbols you wish to map between TradeStation and NinjaTrader.

* Most popular futures contracts already have mapping set up

How to map a continuous futures contract

Mapping a Continuous Futures Contract

NinjaTrader can map continuous contracts in one of two ways:

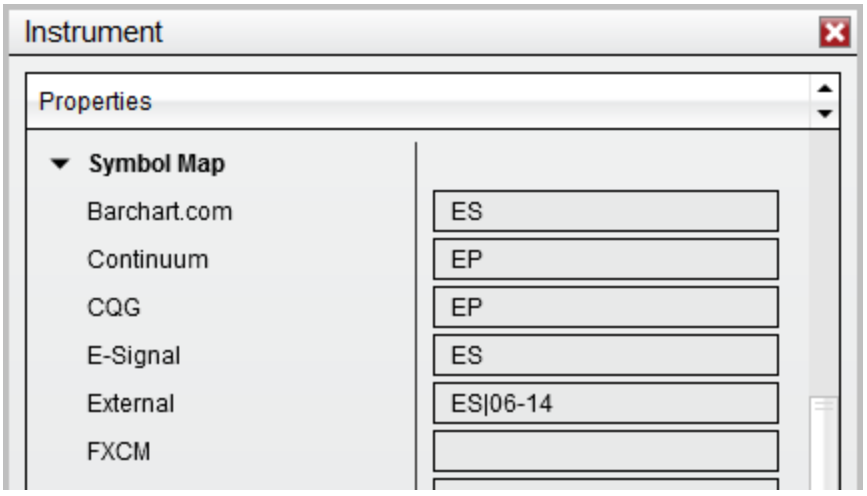
- Automatically map to the next closest expiration date
- User defined contract mapping

For automatic mapping, follow the instructions above for "How to map an individual futures contract" otherwise follow the instructions below.

If you run the TradeStation Automated Trading Interface through the email protocol follow the instructions below. We will use the "@ES" continuous contract symbol and front month of June 2014 for example purposes.

1. From the NinjaTrader Control Center window select the menu **Tools** and select the **Instruments** menu item.
2. Highlight the ES contract from the main grid which is the NinjaTrader S&P 500 Emini contract.
3. Press the "edit**"** button to bring up the **Instrument** window.
5. In the Symbol Map category for the **External** data feed set the value to "ES|06-14".
6. Press the "**OK"** button.

* The symbol map name "ES|06-14" in the image below needs to be the TradeStation symbol base name.



Automated Trading Interface - Orders generated for "@ES" will now be routed to the NinjaTrader "ES 06-14" contract.

Please remember to change this when the contract rolls over.

10.17.8 Importing a List of Stock Symbols

Importing a Stock List

Importing a list of stock symbols is an efficient way to add instruments to the instruments database in bulk.

Within the Control Center window select the **Tools** menu. Then select the menu item **Import** and left mouse click on the menu item **Stock Symbol List...**

1. Press the **Load** button to open a text file that contains your symbol list or type each symbol into the editor manually

The text file must contain valid symbols separated by either -

- User defined character such as a semicolon or comma
- White space
- Carriage return

2. The symbols for import are listed in the editor
3. Select the exchange the instruments are traded on
4. Select the currency the instruments are traded in
5. Optionally add the instruments to an Instrument List (optionally create a new one by selecting "**New**" in the combo box.)
6. Select a Session template for the instruments
7. Enter any user defined separator characters
8. Press the **OK** button to import

Note: Instruments with illegal characters such as a period will be converted to use an underscore instead automatically when running through the import or migration process.

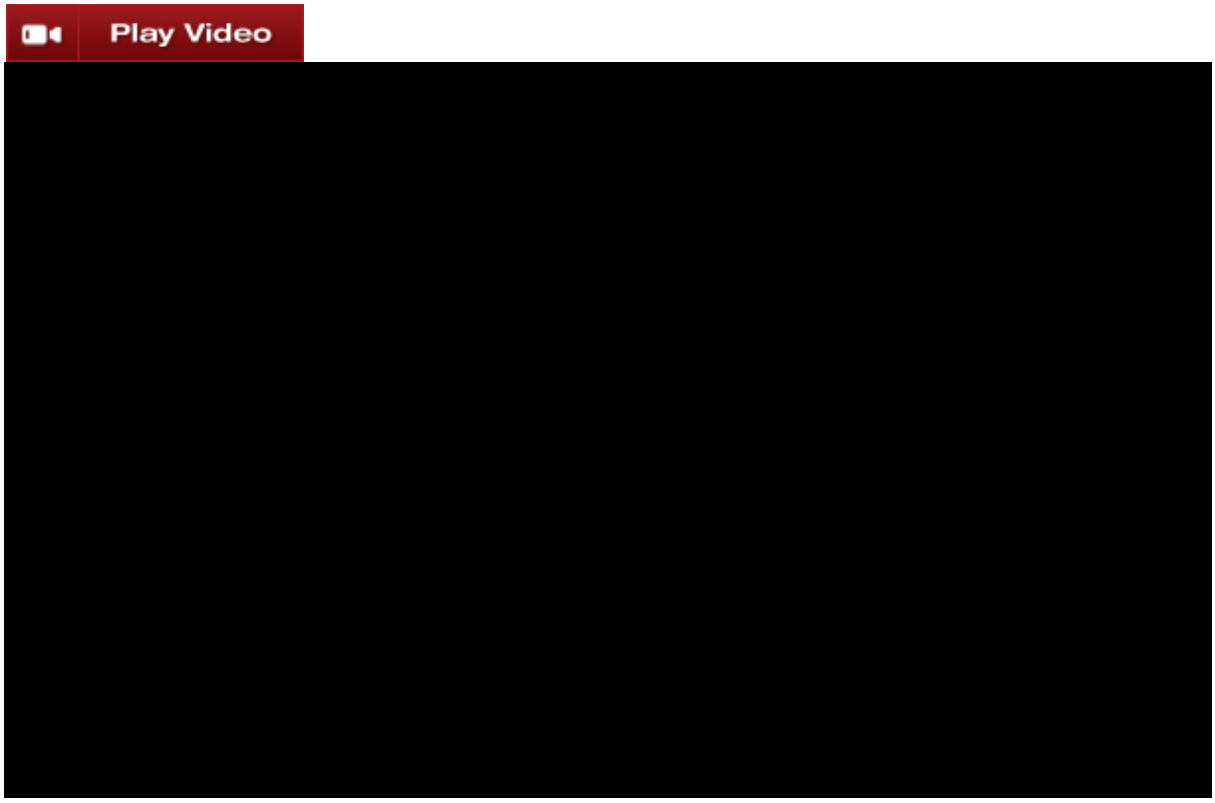
10.18 Level II

Level II Window Overview

You can access the **Level II** window from within the NinjaTrader Control Center window by left mouse clicking on the menu **New**, and then selecting the menu item **Level II**.

The **Level II** window displays bid and ask data color coded by price. It is used to gauge strength and depth on either side of the market. Each price row in the Details section shows a Market Maker or ECN for that price level. For non-Nasdaq stocks, market depth is displayed for the regional exchange the market is traded.

10.18.1 Using the Level II Window



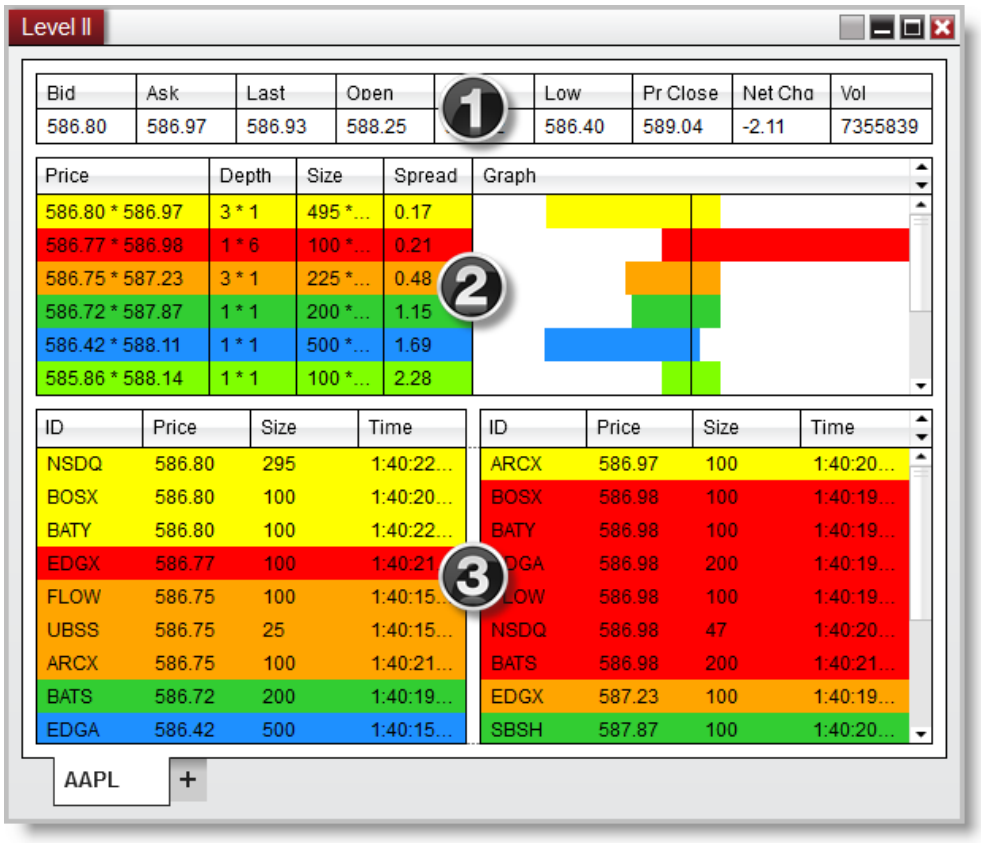
▼ Selecting an Instrument

There are multiple ways to select an Instrument in the **Level II** window.

- Right clicking on the **Level II** window and selecting the menu **Instruments**.
- With the **Level II** window selected begin typing the instrument symbol directly on the keyboard. Typing will trigger the **Overlay Instrument Selector.**

For more Information on instrument selection and management please see Instruments section of the Help Guide.

Understanding the layout of the Level II window



1 Quotes

The Quotes section displays various market data items.

Bid	The current bid price

Ask	The current ask price
Last	The current last traded price
Open	The current sessions open price
High	The current sessions high price
Low	The current sessions low price
Prior Close	Yesterdays sessions close price
Net Chg.	Calculated net change in points from the prior close to the current last traded price
Vol	The current sessions total volume.

You can disable the Quotes section by clicking on your right mouse button and deselecting the menu item **Show Quotes**.

Summary

The Summary section displays total size per price level.

Price	The bid price by ask price
Depth	Number of market participants on the bid by ask price
Size	The total number of shares/contracts on the bid by ask price
Spread	The spread between the bid and ask price
Graph	Visual display of either Size or Depth (number of market participants)

	You can change the graph type via the Level II properties dialog window.

You can disable the Summary section by clicking on your right mouse button and selecting the menu **Show Summary**.

3 Details

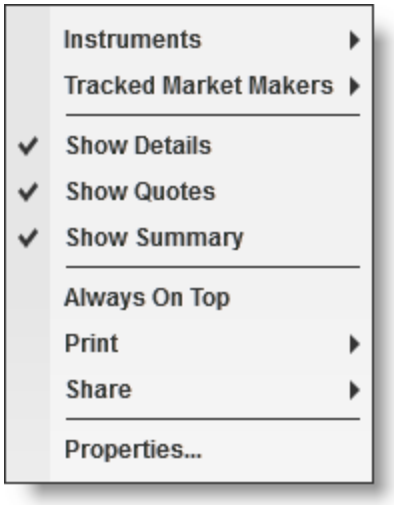
The Details section displays bid data on the left and ask data on the right.

ID	The Market Maker or ECN identification
Price	The bid or ask price
Size	The number of shares/contracts at that price level available for buy or sell by the specific Market Maker or ECN
Time	The last time the bid/ask was refreshed by the Market Maker or ECN

You can disable the Details section by clicking on your right mouse button and de-selecting the menu item **Show Details**.

Right Click Menu

Right mouse click on the **Level II** window to access the right click menu.



Instruments	Selects the instrument
Tracked Market Makers	Selects Market Maker ID's to be tracked
Show Details	Sets if the details section is displayed
Show Quotes	Sets if the quotes section is displayed
Show Summary	Sets if the summary section is displayed
Always On Top	Sets if the window should be always on top of other windows
Print	Displays Print options
Share	Displays Share options
Properties...	Sets the Level II properties

Using Tabs

The **Level II** window is a tabbed interface, this gives you the ability to have multiple **Level II** tabs configured in the same window. Please see the Using Tabs section of the help guide for more information.

10.18.2 Level II Properties

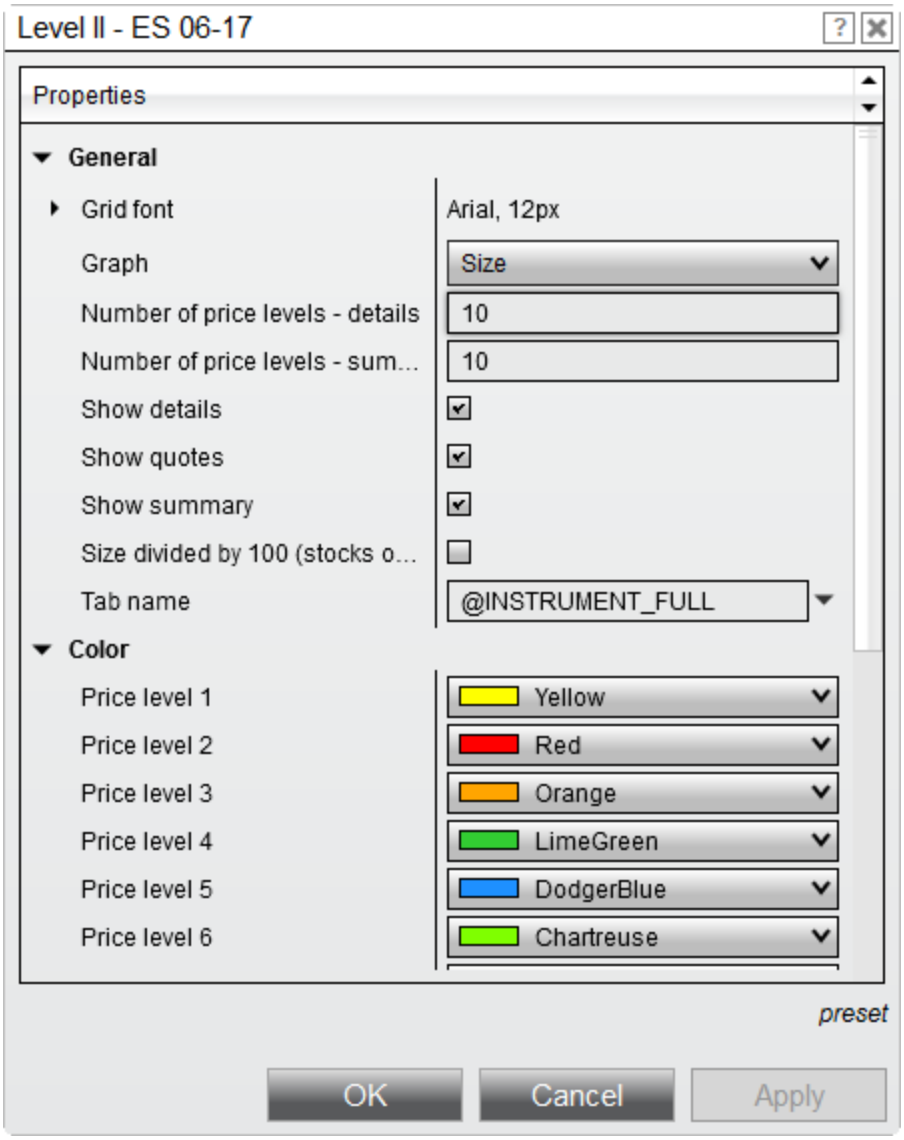
The Level II window can be customized through the **Level II Properties** window.

How to access the Level II Properties window

You can access the Level II properties dialog window by clicking on your right mouse button and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the Level II Properties window:



Property Definitions

General	
Grid font	Sets the font options
Graph	Graphs the total size at a price level or depth which is number of market participants

Number of price levels - detailed	Sets the number of visible price levels in the details section of the Level II window
Number of price levels - summary	Sets the number of visible price levels in the summary section of the Level II window
Show details	Sets if the details section is displayed
Show quotes	Sets if the quotes section is displayed
Show summary	Sets if the summary section is displayed
Size divided by 100 (stocks only)	Displays the the size column values divided by 100 for stock instruments only
Tab name	Sets the name of the tab, please see Managing Tabs for more information.
Color	
Price level X	Sets the background color for a specific price level
Tracked market makers background	Sets the background color for tracked market makers
Tracked market makers foreground	Sets the foreground color for tracked market makers
Up tick background	Sets the background color for the ask,bid, and last cells on uptick.

Down tick background	Sets the background color for the ask,bid, and last cells on downtick.
Window	
Always on top	Sets if the window will be always on top of other windows.
Show Tabs	Sets if the window will allow for tab support

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

Using Tab Name Variables

Tab Name Variables
A number of pre-defined variables can be used in the "Tab Name" field of the **Level II Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.18.3 Window Linking

Please see the Window Linking section of the Help Guide for more information on linking the **Level II** window.

10.19 Market Analyzer

Market Analyzer Overview

You can access the **Market Analyzer** window from within the NinjaTrader Control Center window by left mouse clicking on the menu **New**, and then selecting the menu item **Market Analyzer**.

The **Market Analyzer** window is a high powered quote sheet that enables real-time market scanning of multiple instruments based on your own custom criteria. You can use the **Market Analyzer** to display indicator, market and trade data in a highly customizable manner.

Management

- Creating a Market Analyzer Window
- Working with Instrument Rows
- Working with Columns
- Dynamic Ranking and Sorting
- Window Linking

Conditions

- Creating Cell and Filter Conditions

Performance

- Performance Tips
- Reloading Indicators & Columns

10.19.1 Creating a Market Analyzer Window

Understanding the Market Analyzer display

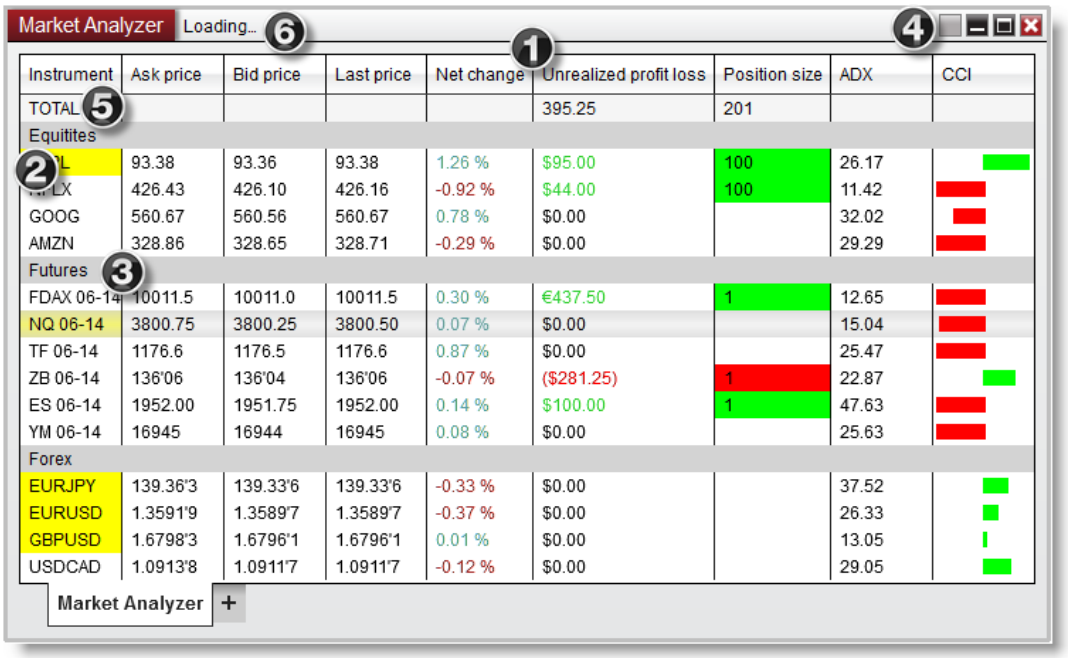
Market Analyzer Display Overview

Each NinjaTrader Market Analyzer is a free floating window that can be manually resized by dragging the edges of the window and moved by left mouse clicking and dragging in the upper most margin for arrangement within the open Workspace.

The image below shows some of the common features of a Market Analyzer window:

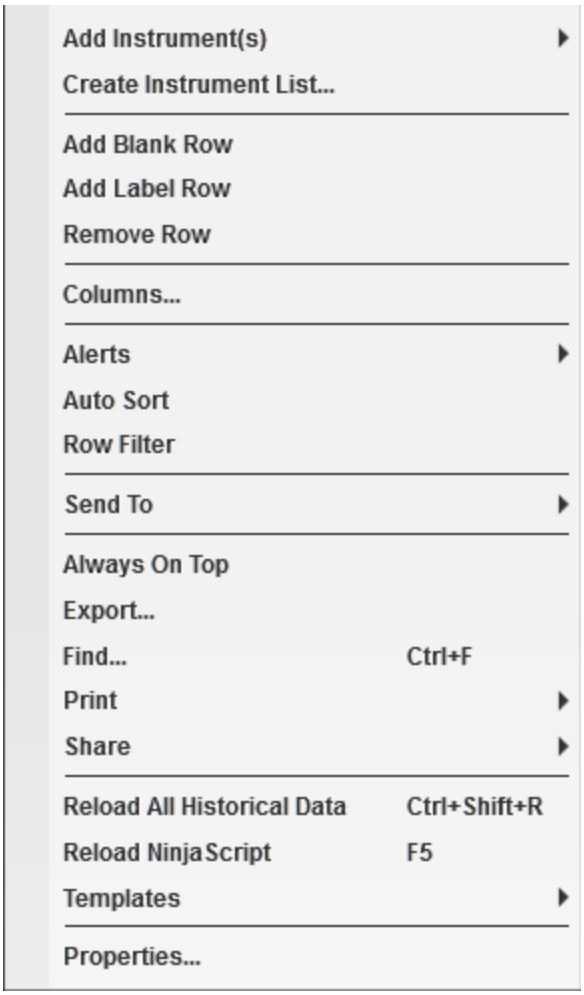
1. Columns	Displays the column name
2. Instrument row	Displays the instrument name

3. Label row	Displays a user defined label row.
4. Link button	Window linking links windows to the same instrument and can be applied to many NinjaTrader windows.
5. Total row	Displays the sum of all rows for a specific column. Can be applied in the Market Analyzer Properties window.
6. Loading dialog	Display a message indicating when an indicator or historical data is being loaded into the Market Analyzer



Right Click Menu

All functions of the Market Analyzer can be accessed by pressing on your right mouse button within the Market Analyzer window to bring up the right click menu.



Add Instrument (s)	Adds an individual instrument or list of instruments to the **Market Analyzer** display
Create Instrument Lists...	Dynamically creates a list of all the current instruments in the **Market Analyzer** display which can be accessed from the Instrument Lists window
Add Blank Row	Adds a blank row to the **Market Analyzer** display
Add Label Row	Adds a Label Row to the **Market Analyzer** display

Remove Row	Removes a selected row from the **Market Analyzer** display
Columns...	Opens the Columns menu to configure user defined columns to be displayed
Alerts	Opens the Alerts window to configure user defined alerts to be armed
Auto Sort	Enables/Disables the dynamic sorting and ranking
Row Filter	Enables/Disables row filters
Send To	Loads the selected instrument into another NinjaTrader window
Always On Top	Sets the **Market Analyzer** window to always be on top of other windows
Export...	Exports the **Market Analyzer** contents to "CSV" or "Excel" file format
Find...	Search for a term in the **Market Analyzer**
Print	Displays Print options
Share	Displays Share options
Reload All Historical Data	Reloads the historical bar data used for Indicator calculations
Reload NinjaScript	Reloads all of the NinjaScript columns to recalculate the current values
Templates	Access the templates menu to save/load custom **Market Analyzer** settings

Properties	Set the **Market Analyzer** properties

10.19.2 Working with Instrument Rows

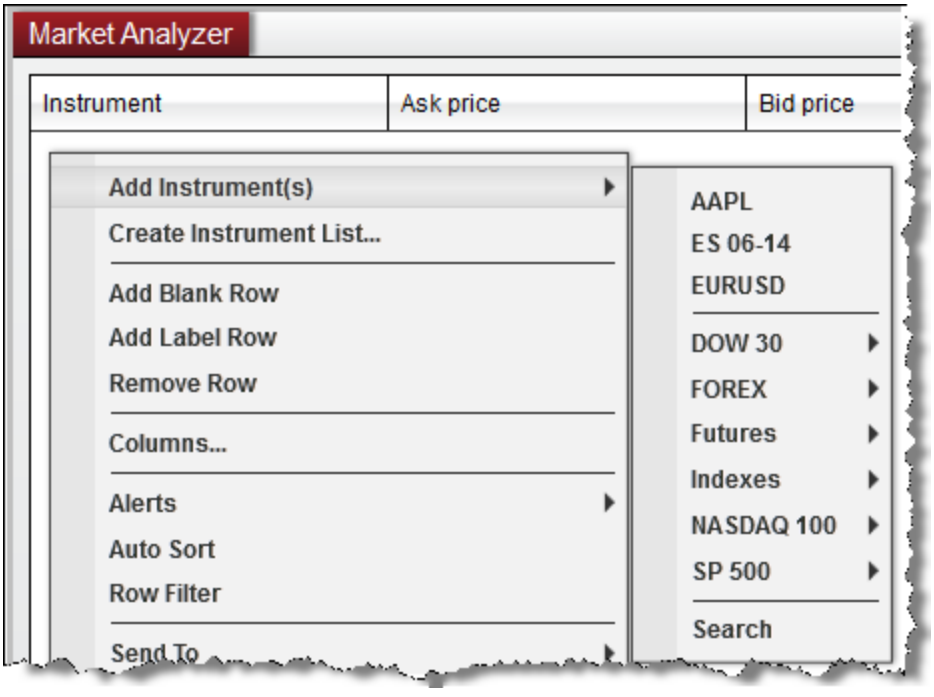
The Market Analyzer window allows you to display a variety of real-time quotes, indicator values, and position information on multiple instruments. You can add, remove, and organize individual instrument rows, Instrument Lists, Label rows, Blank rows, and a Total row with the instructions listed below.

▼ How to add instruments

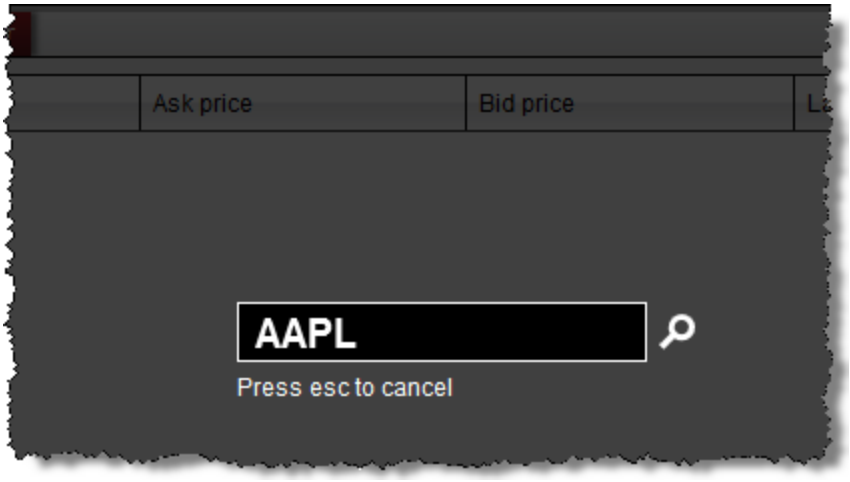
Adding an Instrument

You can add an individual instrument to the **Market Analyzer** through one of the techniques below:

- Press down on your right mouse button in the **Market Analyzer** window and select the menu item **Add Instrument(s)**. Through the **Instrument Selector** menu, you can navigate through various instrument lists to locate the instrument you desire, and left click on the instrument to add the individual instrument to the **Market Analyzer.**



- With the **Market Analyzer** window selected begin typing the instrument symbol directly on the keyboard. Typing will trigger the **Overlay Instrument Selector.**



- Double left mouse click in a Blank row under the Instrument column to get a flashing text cursor. After the cursor is showing in the row you can type in the symbol of your choice and press enter to add the instrument.

Editing an Instrument Row

To change an instrument, double click on an existing Instrument cell which will give you a flashing text cursor, allowing you to edit the instrument to a new instrument of your choice.

Adding an Instrument List

You can rapidly add a list of instruments to the Market Analyzer window.

- Press down on your right mouse button in the Market Analyzer window and select **Add Instrument(s)** > and then select the desired "**Instrument List"** and then **Select All**. Please see the Instrument Lists section of the user help guide for additional information on creating, editing, and deleting Instrument lists.

> **Tip**: It is more efficient to add instruments after defining the columns of your Market Analyzer window. This will minimize NinjaTrader re-loading historical data into the Market Analyzer window.

▼ How to Create an Instrument List from the Market Analyzer

Creating an Instrument List

If you have a Market Analyzer setup with a number of different instruments you would like to save for later, you can quickly add the entire display of instruments into an Instrument List for quick access.

- Press down on your right mouse button in the Market Analyzer window and select the menu **Create Instrument List**, then give the **Instrument List** a unique name and press OK.

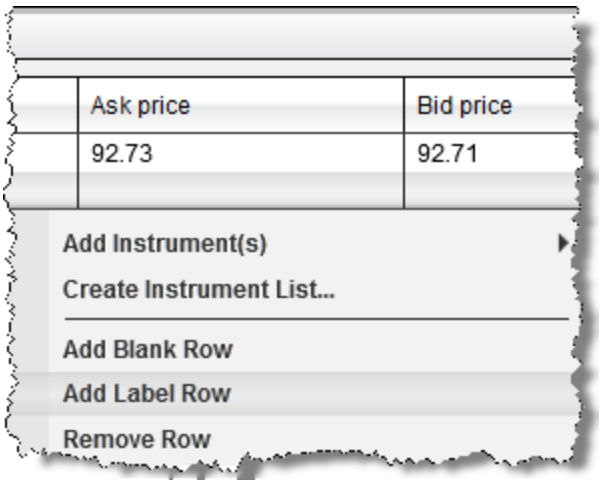
You will now be able to access this list from other features of NinjaTrader using the Instrument Selector. You can further edit this list by using the Instrument Lists window

How to add Label rows

Label rows are user defined and can be used to separate groups of instruments in any way (by asset class, instrument list, etc.).

Adding Label Rows in the Market Analyzer Window

Press down on your right mouse button inside the Market Analyzer window and select the menu **Add Label Row**. Once the Label row is added you can type in any user defined name.



Editing the Label Row Name

If you have an existing **Label** row you wish to go back and change the text, double clicking on the exiting **Label** row text will give you a flashing cursor, allowing you to type in a new name for the **Label** row.

Dynamic Sorting within Label Rows

Instruments you drag or add under a Label row will "auto-sort" with only the other instruments under the same Label row. For example, if you have one Label row for futures and one for stocks, when you sort the columns, the instruments listed under the futures label would be sorted only against other instruments under the futures label, while instruments under the stocks label would be sorted only against instruments under the stocks label. For more information on ranking and sorting within the Market Analyzer see the Dynamic Ranking and Sorting section of the user help guide.

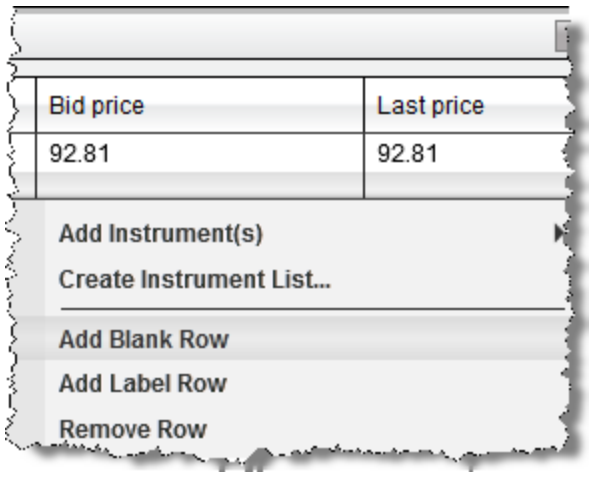
Aligning a Label Row

The label of a **Label Row** can be aligned to the left, center or right of the row. This is done by right mouse clicking within the **Market Analyzer** and selecting the **Properties** menu. Change the property **Label row text alignment** to either "Left", "Center", or "Right".

How to add Blank rows

Adding Blank rows to the Market Analyzer window

Blank rows can be used to create space between instruments in the Market Analyzer window or if you need to add more instruments. To add a Blank row press down on your right mouse button in the Market Analyzer window and select the menu **Add Blank Row**. The Blank row will be added above the row you right clicked in.



How to move Instrument, Label and Blank rows

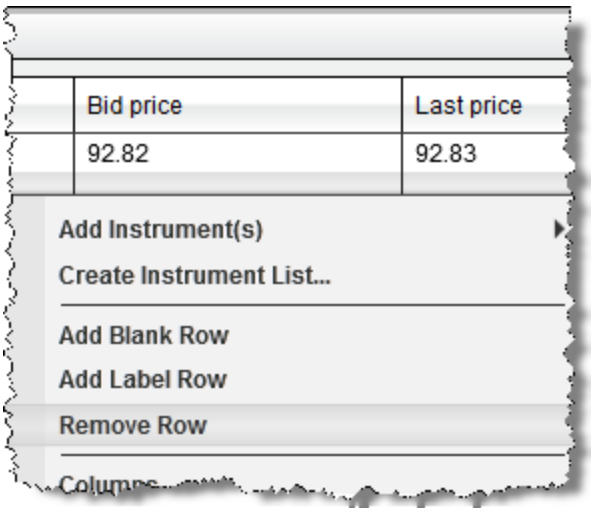
Moving Rows in the Market Analyzer Window

Instrument, Label and Blank rows can all be moved up or down within the Market Analyzer window. To move a row in the Market Analyzer window press down and hold on your left mouse button in the row you would like to move and drag it to the new location. When your cursor is hovering over the new desired location release your left mouse button to set the row down in the new location.

How to remove Instrument, Label and Blank rows

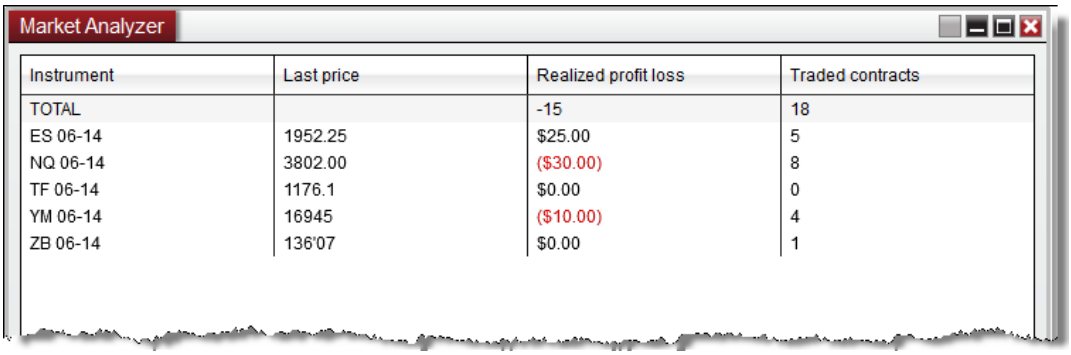
Removing Instrument, Label and Blank Rows

To remove an Instrument, Label or Blank row left mouse click on the row to select it and then press the delete button on your keyboard, or press down on your right mouse button within the row you want to remove and select the menu **Remove Row**.



How to add and remove a Total row

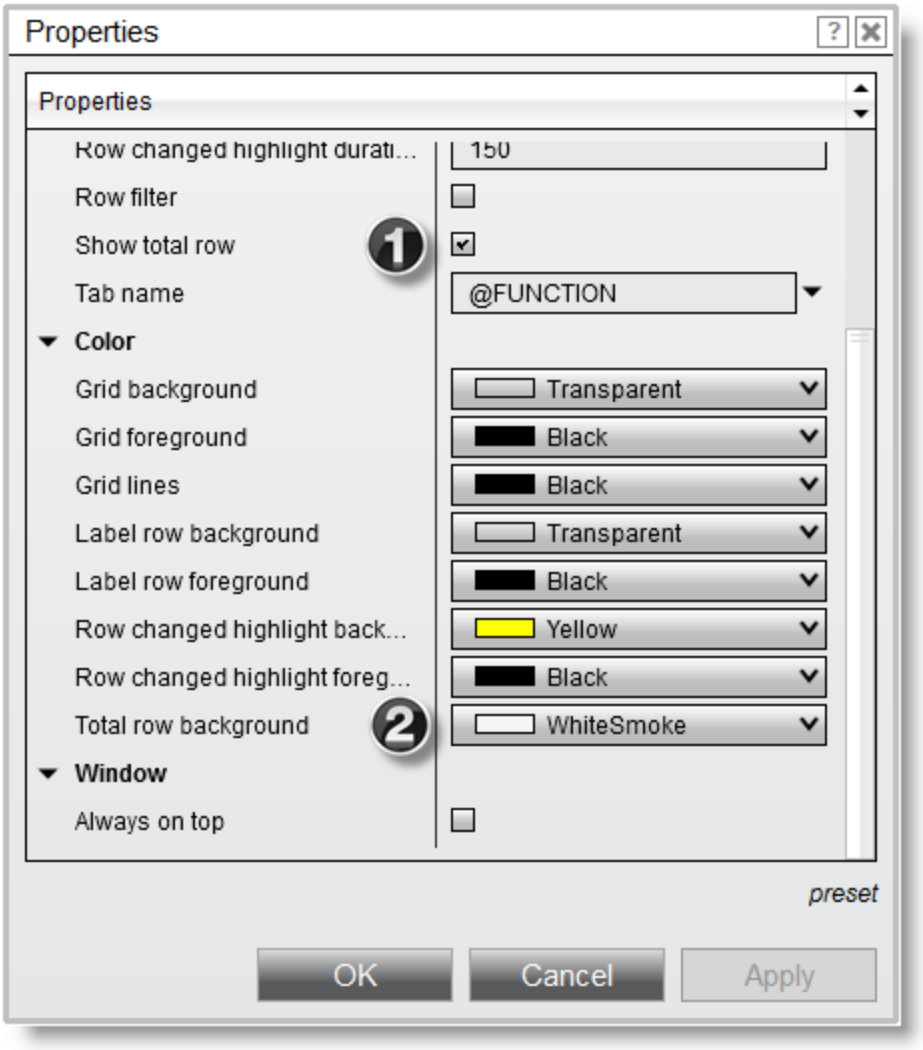
A Total row can total any column of values and is displayed at the top of the Market Analyzer window. For example, you could choose to display your total Realized PnL and total Traded Contracts for all instruments displayed in the Market Analyzer.



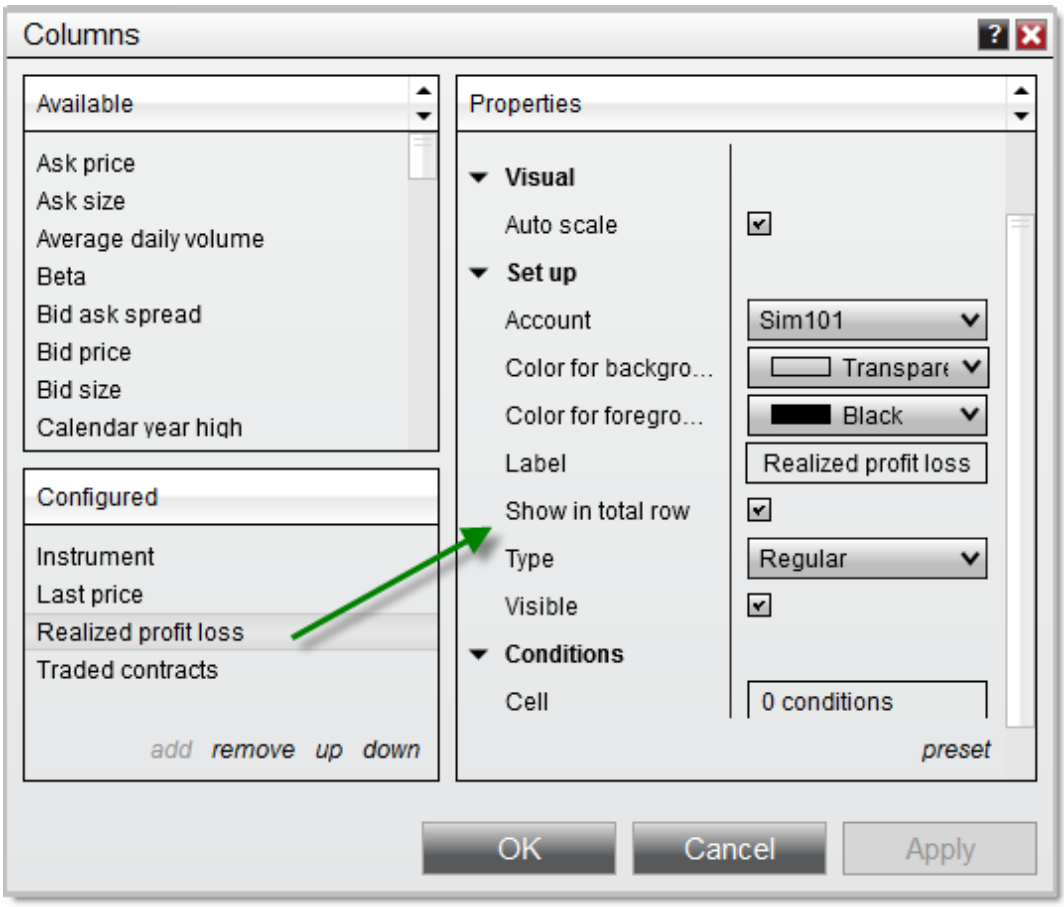
Adding the Total row to the Market Analyzer window

To add a Total row in the Market Analyzer you must enable both the Total row and the columns you would like totalled with the following steps:

1. Press down on your right mouse button in the Market Analyzer window and select the menu **Properties**.
2. In the Properties menu scroll down to the ❶ Total Row section and ***check*** the box to enable. You can also choose to customize the color of this row with the ❷ **Total row background** property.
3. Press the Apply button to apply the changes or press the OK button to apply the changes and exit the Properties menu.



4. To show each column's total in the Total row press down on your right mouse button inside the Market Analyzer window and select the menu **Columns**.
5. ***Check*** the Show in Total row property each column you want totaled in the Total row.



6. Press the Apply button to apply the changes or press the OK button to apply the changes and exit the Columns window.

Removing the Total row from the Market Analyzer window

To remove the Total row press down on your right mouse button inside the Market Analyzer window and select the menu **Properties**. Scroll down to the Total Row section and ***uncheck*** the property. Then press the Apply button to apply the changes or press the OK button to apply the changes and exit the Properties window.

Understanding Row Filtering

Row Filtering allows you to filter out (hide) rows from the Market Analyzer grid display based on a cell's value. Filter conditions can be setup for any column applied to the Market Analyzer.

To enable Row Filtering:

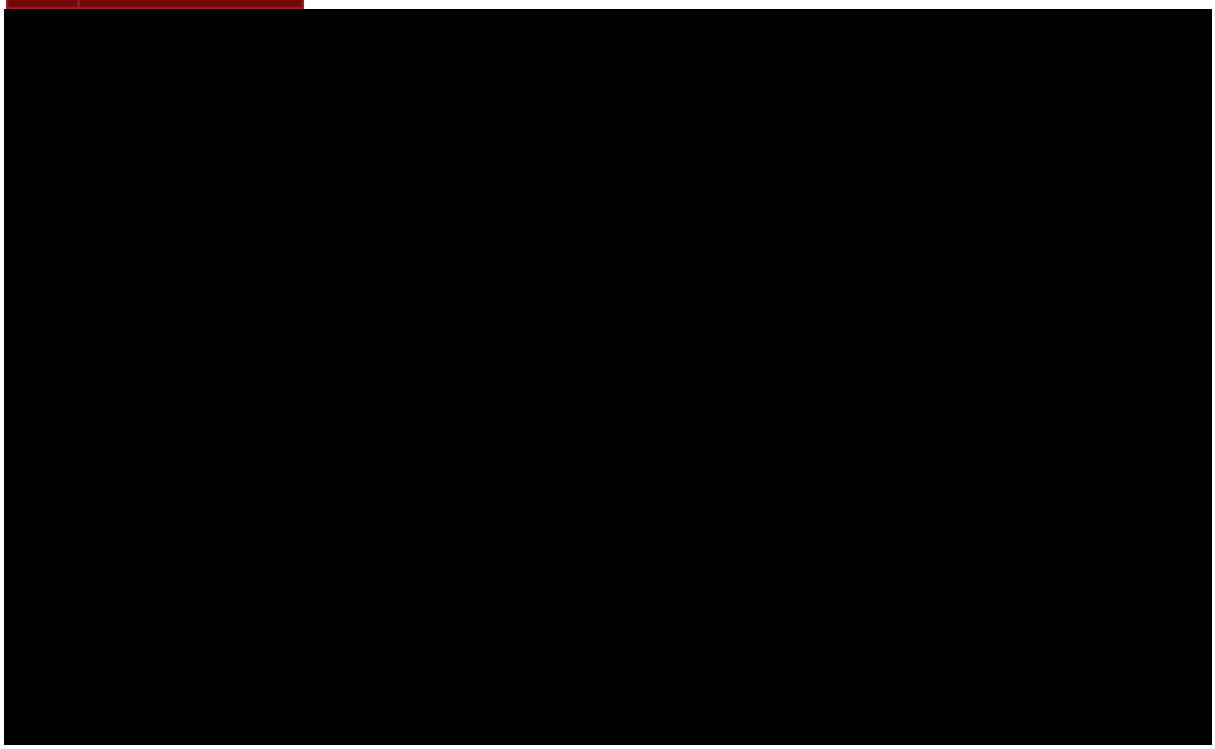
1. Press down on your right mouse button in the Market Analyzer window and select the menu **Row Filter**.
2. To access the Columns menu where you can add filtering conditions to each column press down on your right mouse button and select the menu **Columns**.

For more information on Row Filtering see the Creating Filter Conditions section of the user help guide.

10.19.3 Working with Columns

The Market Analyzer allows you to add a variety of columns ranging from indicators to position information. To add, remove, and customize columns in your Market Analyzer window please review the information below.

Play Video



Understanding the Columns window

The Columns window is used to add, remove, and edit columns within the Market Analyzer window.

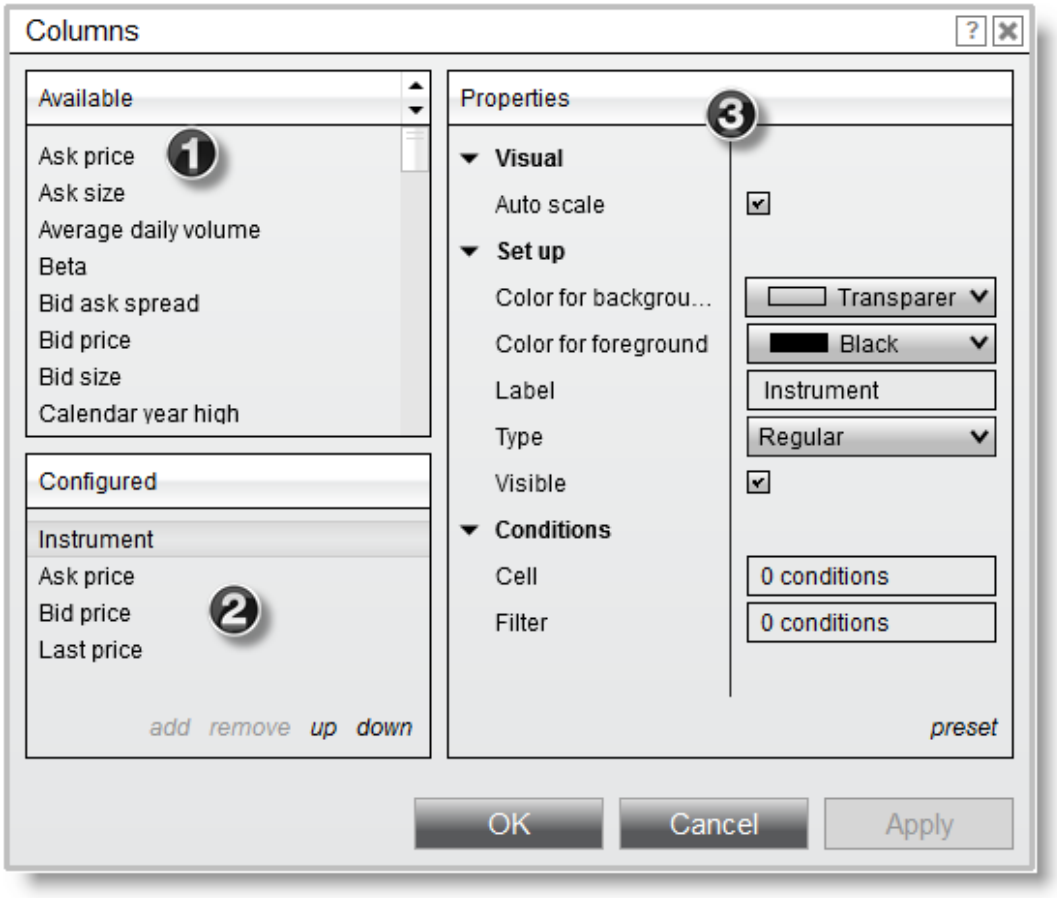
Accessing the Columns Window

To access the Columns window press down on your right mouse button in the Market Analyzer window and select the menu item **Columns...**

Sections of the Columns Window

The image below displays the four sections of the Columns window.

1. List of available columns
2. Current columns applied to the Market Analyzer
3. Selected column's parameters



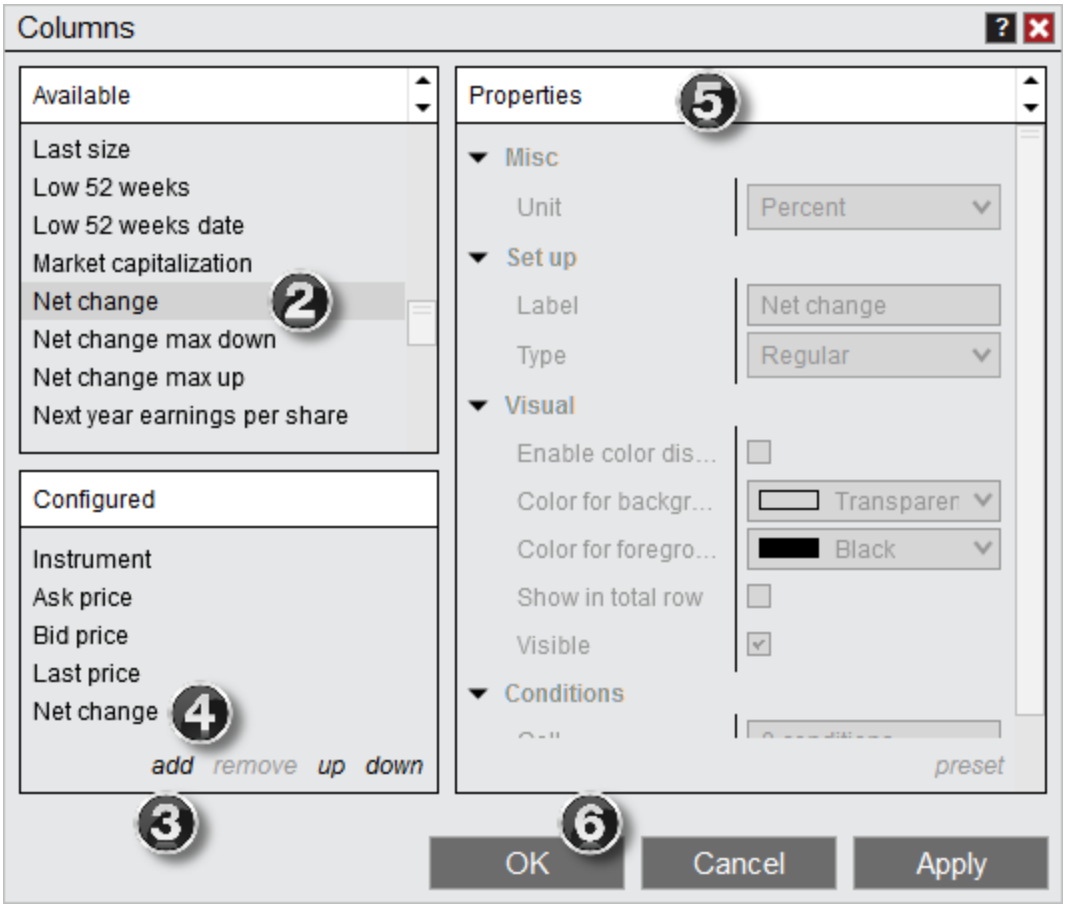
How to add columns

A wide variety of columns can be added to your Market Analyzer window allowing you to see indicator, position, or price information at a glance.

Adding columns to the Market Analyzer window

To add a column to the Market Analyzer window:

1. Open the Columns window *(see the "Understanding the Columns window" section above)*
2. Select the column you want to add from the list of available columns
3. Press the **Add** button or simply double click on the column you want to add
4. The column will now be visible in the list of applied columns
5. The column's parameters will be editable on the right side of the Columns window when the column is selected from the applied columns list *(see the "How to customize columns" section below)*
6. Press the OK button to apply the column(s) to your Market Analyzer, and exit the Columns window



Adding an Indicator Column

To add an indicator column to the Market Analyzer window:

1. Open the Columns window *(see the "Understanding the Columns window" section above)*
2. Left mouse click on the Indicator column and press the **Add** button or simply double click on it
3. The column will now be visible in the list of applied columns and listed as "ADL on 1 Min data"
4. You can now select the indicator of your choice from the Indicator parameter

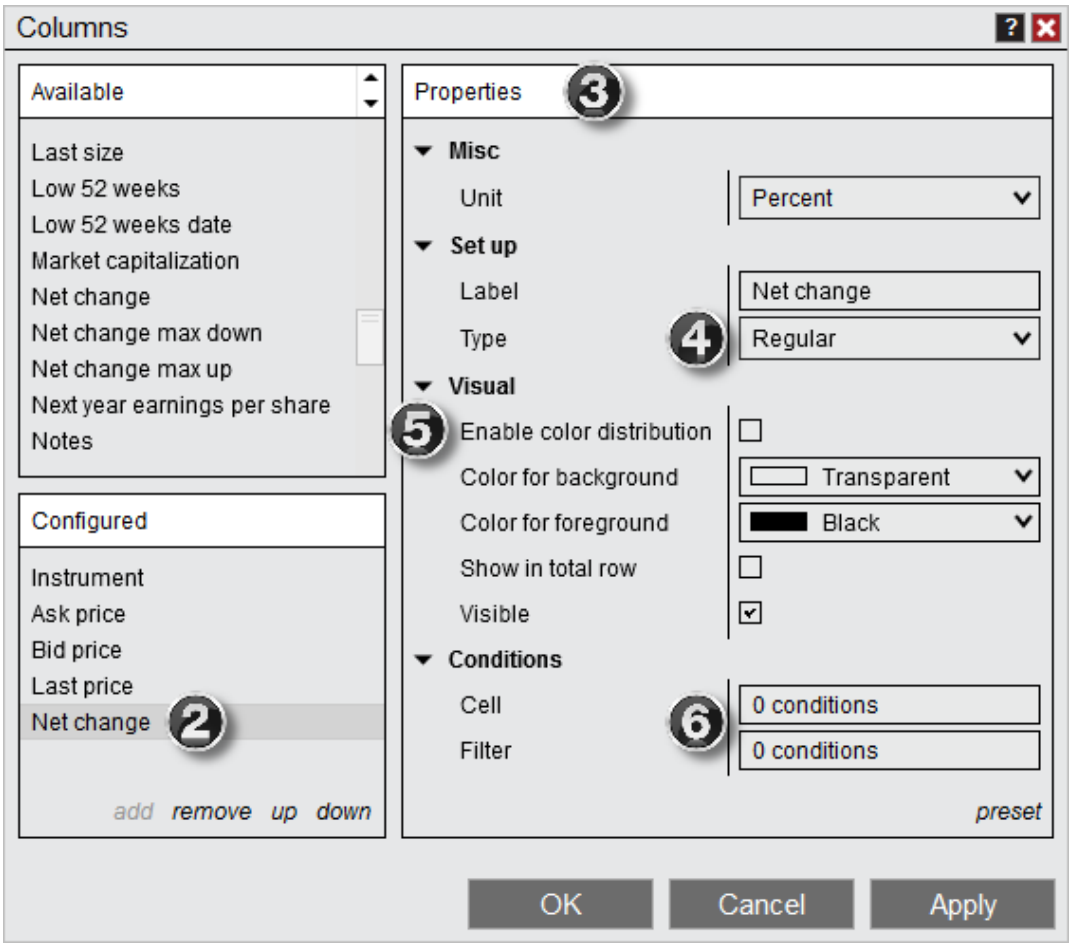
How to customize columns

Once you have added columns to your Market Analyzer window (*see the "How to add columns" section above*) you can customize the column by editing the column's parameters.

Editing a Column's Parameters

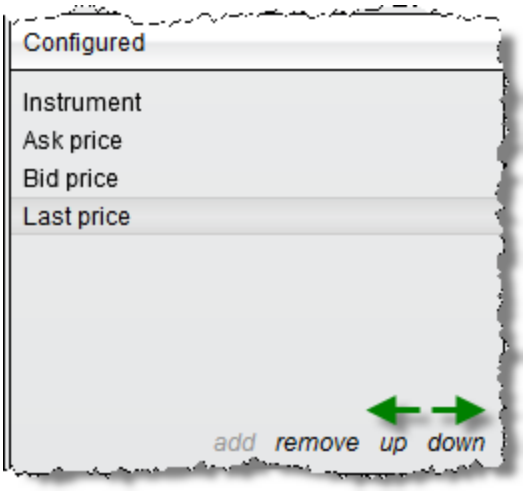
You can customize any column from the Columns window.

1. Open the Columns window *(see the "Understanding the Columns window" section above)*
2. Highlight the column you would like to edit in the list of **Configured** columns (as shown by the image below).
3. Once highlighted this column's parameters will be editable on the right hand side
4. You can choose to display the column Type as Regular or as a BarGraph
5. You can configure the color settings, including checking Enable color distribution to apply a range of colors based on their values
6. You can set Cell or Filter conditions for any column from the **Conditions** parameters section



Changing the Order and Width of Columns

To order columns in the Market Analyzer window you can use "***up***" or "***down***" in the **Configured** columns section.



- Left mouse click **"*up*"** to move the selected applied column *left* in the **Market Analyzer** window
- Left mouse click "***down***" to move the selected applied column *right* in the **Market Analyzer** window

Please see the Data Grids section of the user help guide for information on sizing and ordering columns.

Understanding Indicator Column Properties

An **Indicator column** has many unique properties used to determine how the indicator is calculated. It is important to understand how these properties will impact the resulting indicator value displayed in your **Market Analyzer** column.

Indicator	
Indicator	Selects the indicator used for the column
Plot	Selects which of the indicator's plot is used. Some indicators will have several plots.
Data Series	
Input Series	Selects the price type used. **Close** is the most common
Price based on	Selects the data type used. **Last** is the most common
Type	Selects the bar type which the indicator is calculated on
Value	Selects the interval used in correlation to the bar type
Time Frame	

Load data based on	Select from **Bars**, **Days** or a **Custom Range** in terms of historical data used for the indicator
Bars to load	Selects the number of bars (or days) used requested to calculate the indicator.
End date	Sets the last day used for calculation.
Trading hours	Sets the trading session used for calculation
Break at EOD	Sets if the indicator values are reset at the end of each session
Set up	
Calculate	Sets the frequency that the indicator calculates. **On bar close** will slow down the calculation until the close of a bar; **On price change** will calculate on when there has been a change in price; **On each tick** calculate the indicator's value which each incoming tick.
Maximum bars look back	Max number of bars used for calculating an indicator's value. The **TwoHundredFiftySix** setting is the most memory friendly.

Note: Setting the **Type** to **Bar Graph** under **Data Series** for text based columns will result in the column being blank. Text based columns require the **Type** to be **Regular**.

Saving a Customized Column Presets

Once you have an individual column properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you apply a new column.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

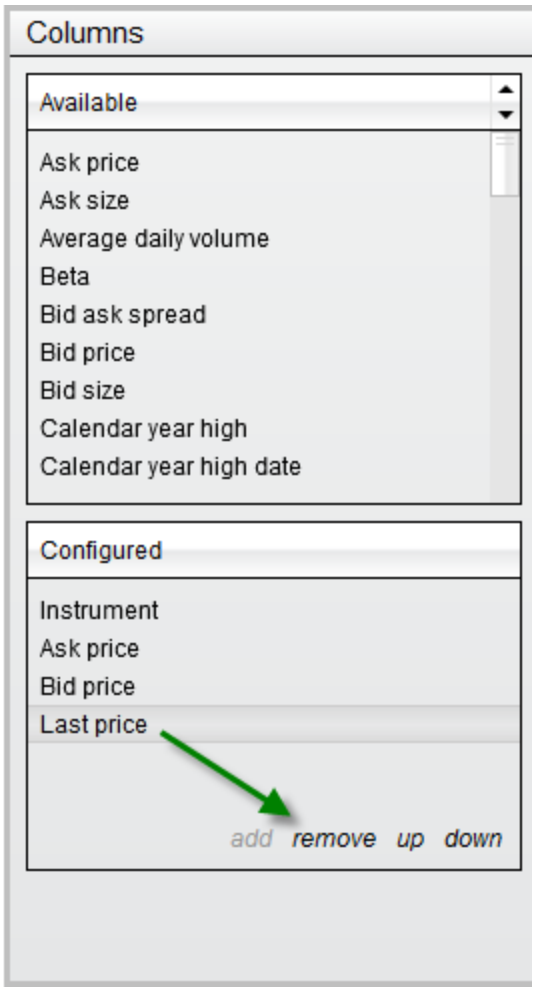
How to remove columns

Columns can be removed from the Columns window or from the Market Analyzer directly.

Removing Columns from the Market Analyzer Window

There are two ways to remove a column:

1. From the Market Analyzer window left mouse click on the column header and hold down the left mouse button to drag the column outside the Market Analyzer window, once the cursor changes to a black X release the left mouse button to remove the column.
2. Open the Columns window *(see the "Understanding the Columns window" section above).* Highlight the column you would like to remove in the list of **Configured** columns (as shown in the image below) then press the Remove button.



Dynamic ranking and sorting

See the Dynamic Ranking and Sorting section of the user help guide for information on sorting and raking your Market Analyzer columns.

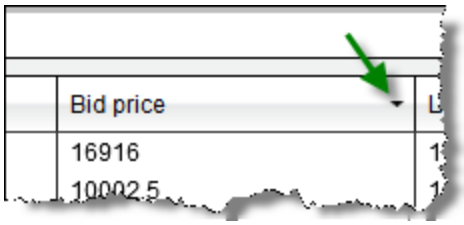
10.19.4 Dynamic Ranking and Sorting

The Market Analyzer window can automatically rank and sort the data rows.

How to Enable Automatic Ranking and Sorting

To enable ranking and sorting for a column:

1. To set the column you wish to sort press down on your left mouse button in the column header. You can set the column to sort in either descending (down arrow) or ascending (up arrow) order.



2. You can enable dynamic sorting by pressing down on our right mouse button inside the Market Analyzer and selecting the menu **Auto Sort**.

3. You can set the auto sort interval within the Market Analyzer Properties window.

10.19.5 Creating Cell and Filter Conditions

Market Analyzer columns can have cell and filter conditions applied to them for a more convenient display of information.

Understanding cell conditions

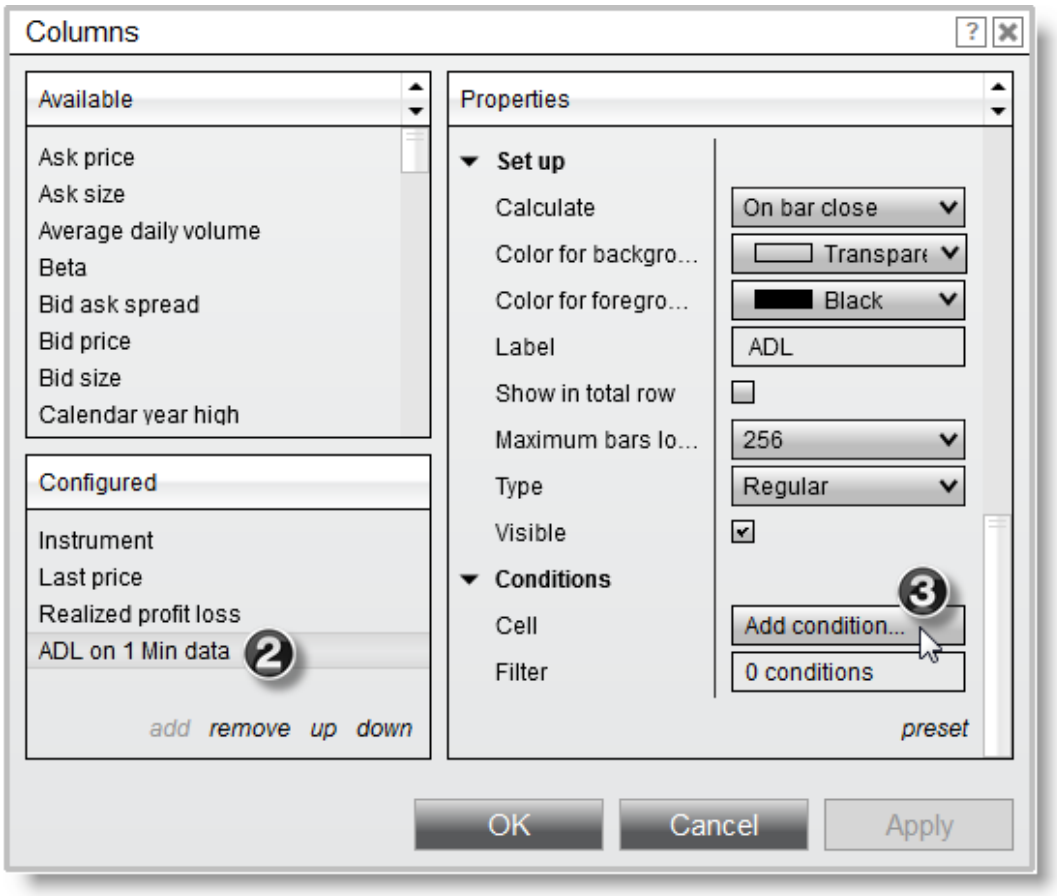
Cell Conditions

Cell Conditions allow you to define the display behavior of a cell based on the cell's value, and are defined per column. You can choose to alter both the color and text of a cell with Cell Conditions.

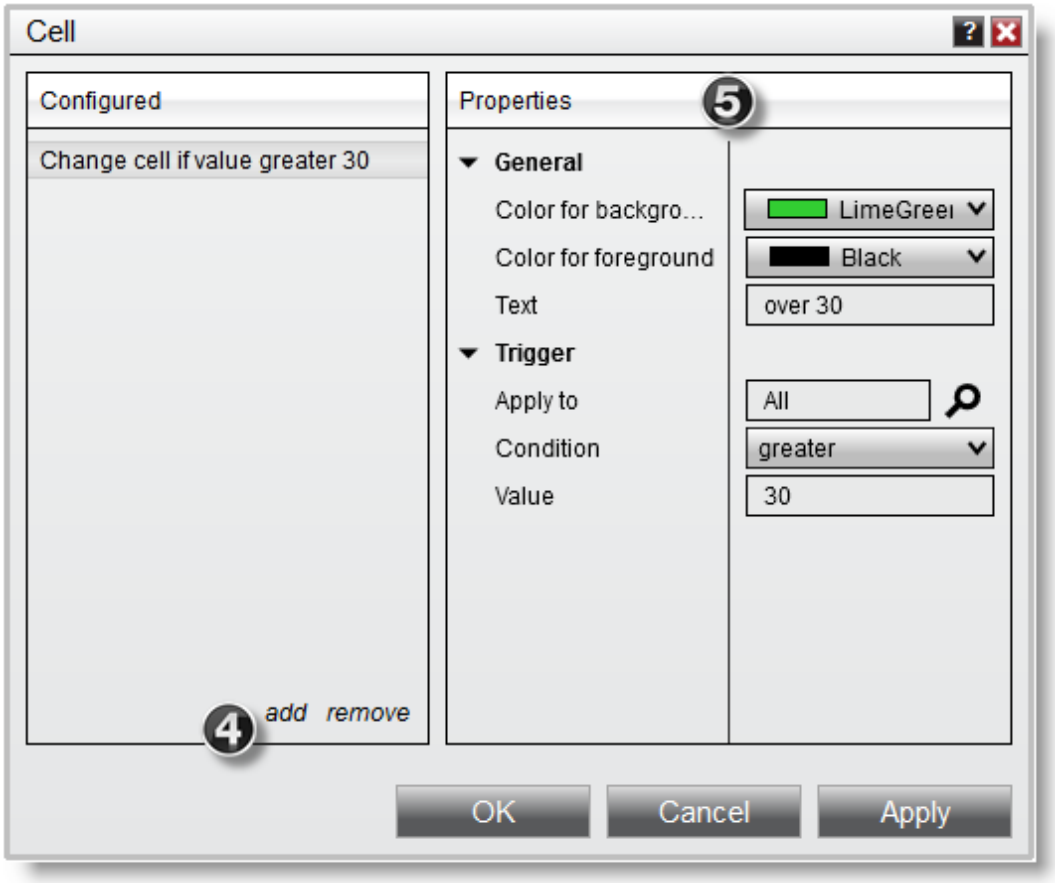
Creating Cell Conditions

To create a Cell Condition:

1. Open the Columns window
2. Select the column you would like to create a Cell Condition for in the applied column section.
3. Under the **Conditions** parameters section, move your mouse over the Cell field and then press the "**Add condition...**" button which will appear.



4. Press the Add button to add a new Cell Condition to the list of **Configured** conditions displayed in the left side of the Cell Conditions window
5. Set the Cell Condition properties in the right side of the Cell Conditions window



The example Cell Condition in the above image will:

- Trigger once the cell value is greater than 30
- Applies to "All" Instruments (please see the **Understanding the apply to trigger** section at the bottom of this page for more information)
- Display a lime green background with black text
- Display "over 30" as the text

You can remove a Cell Condition by pressing the Remove button.

Multiple Cell Conditions

Cell Conditions are evaluated from top to bottom.

Assume you have the following conditions defined:

Change cell if value is greater than 30
Change cell if value is greater than 100

In this example, if the value of the cell was greater than 100, the first condition of "greater than 30" would change the cell's color since its first in the list of conditions to be evaluated. The "greater than 100" condition would never trigger in this example since "greater than 30" will always trigger the color change first. To ensure that both conditions trigger a color change so that you get the desired alerting behavior you want, you have to list the conditions in this order:

Change cell if value is greater than 100
Change cell if value is greater than 30

This will guarantee that a cell value over 100 will fall in the "greater than 100" condition and cell values between 30 and 100 will be triggered by the "greater than 30" condition.

Understanding filter conditions

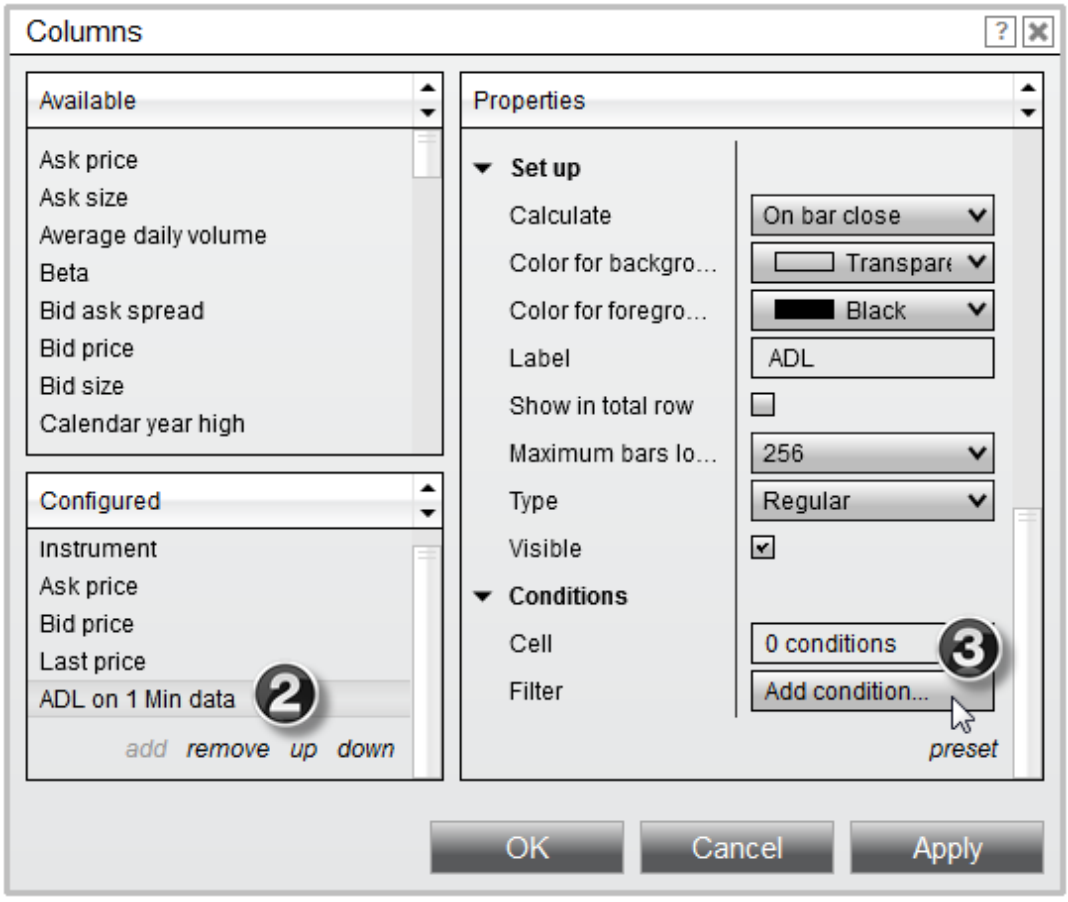
Filter Conditions

Filter Conditions allow you to define conditions that filter out rows from the Market Analyzer grid display based on the cell's value and are defined per column.

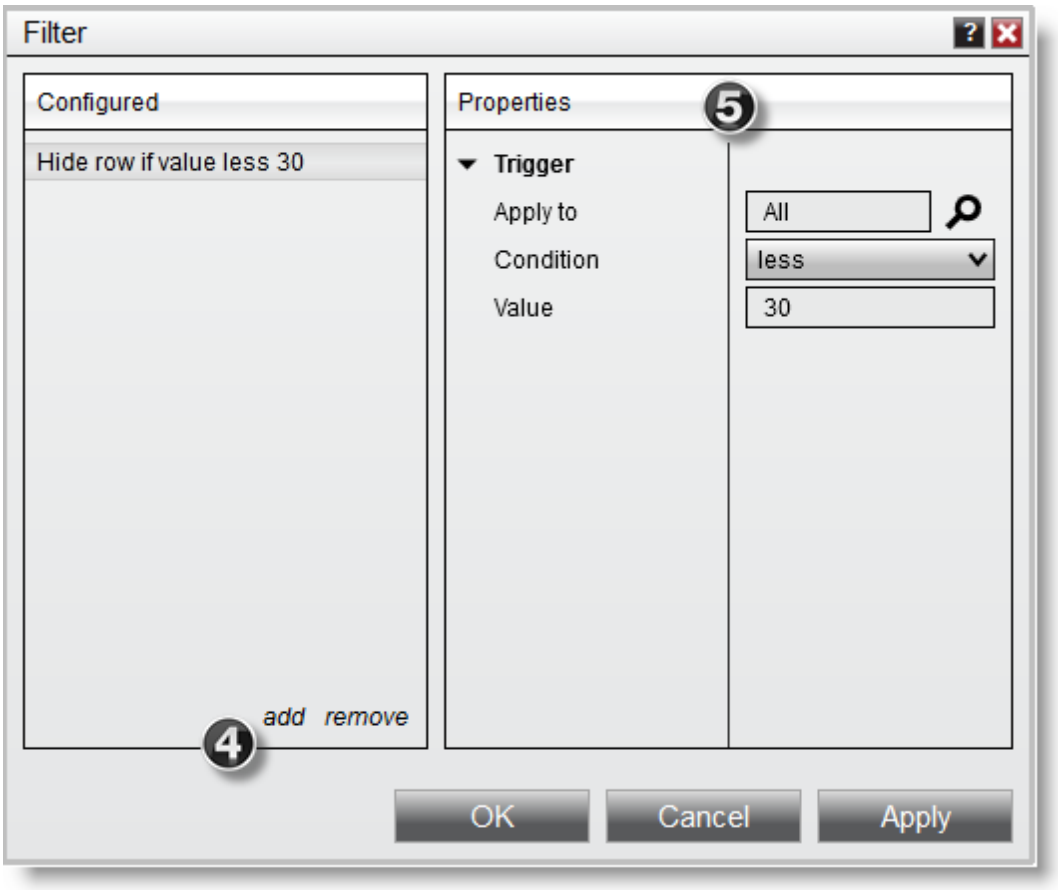
Creating Filter Conditions

To create a Filter Condition:

1. Open the Columns window
2. Select the column you would like to create a Filter Condition for in the applied column section.
3. Under the **Conditions** parameters section, move your mouse over the **Filter** field and then press the "**Add condition...**" button which will appear.



4. Press the Add button to add a new Filter Condition to the list of **Configured** conditions displayed in the left side of the Filter Conditions window
5. Set the Filter Condition properties in the right side of the Filter Conditions window



The example Filter Condition in the above image will:

- Filter out the row from the Market Analyzer grid display when the cell value is less than 30
- Applies to "All" Instruments (please see the **Understanding the apply to trigger** section at the bottom of this page for more information)
- The row will be displayed in the Market Analyzer grid display when the cell value is greater than or equal to 30

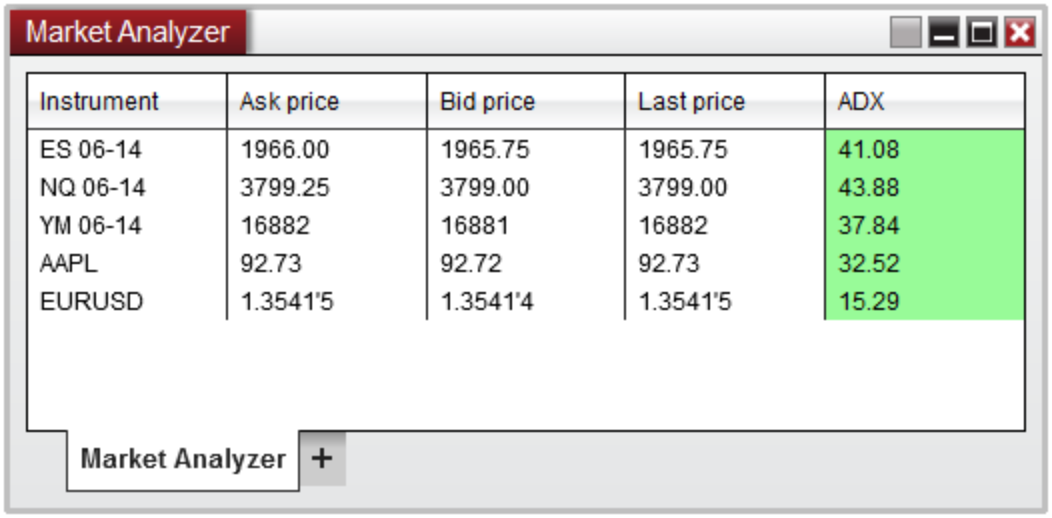
You can remove a Filter Condition by pressing the Remove button.

To enable/disable filtering press down on your right mouse button in the Market Analyzer window and select the menu **Row Filter**. When enabled, the Market Analyzer will filter out rows from the grid display based on the Filter Conditions of the columns.

Understanding the apply to trigger

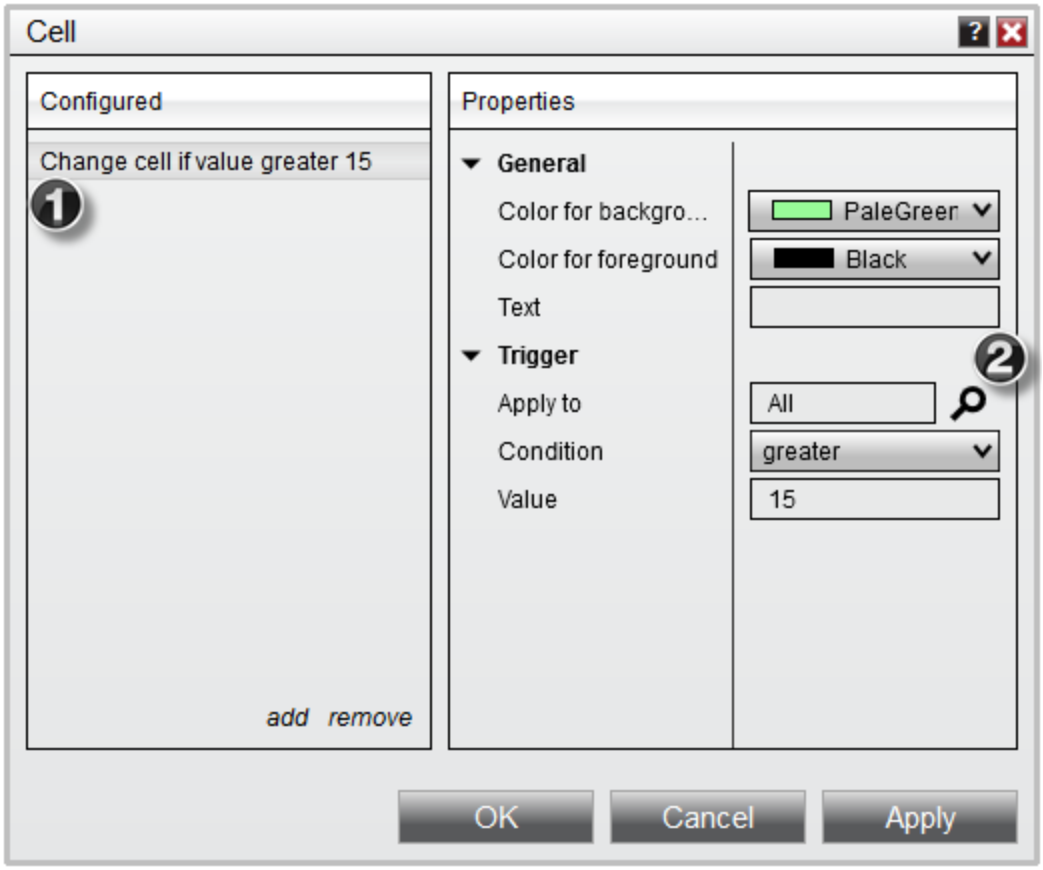
Applying conditions to specific instruments

When setting up **Cell** and **Filter conditions**, the default behavior is to apply these conditions to **all** instruments in the **Market Analyzer.**



However, you can optionally reconfigure these conditions to apply to instruments with specific names. For example, if you had a Market Analyzer setup with several different instruments (as per the screen shot above), but only wanted your **Cell conditions** to work on only the Futures instruments, you can redefine your conditions to only include those instruments by:

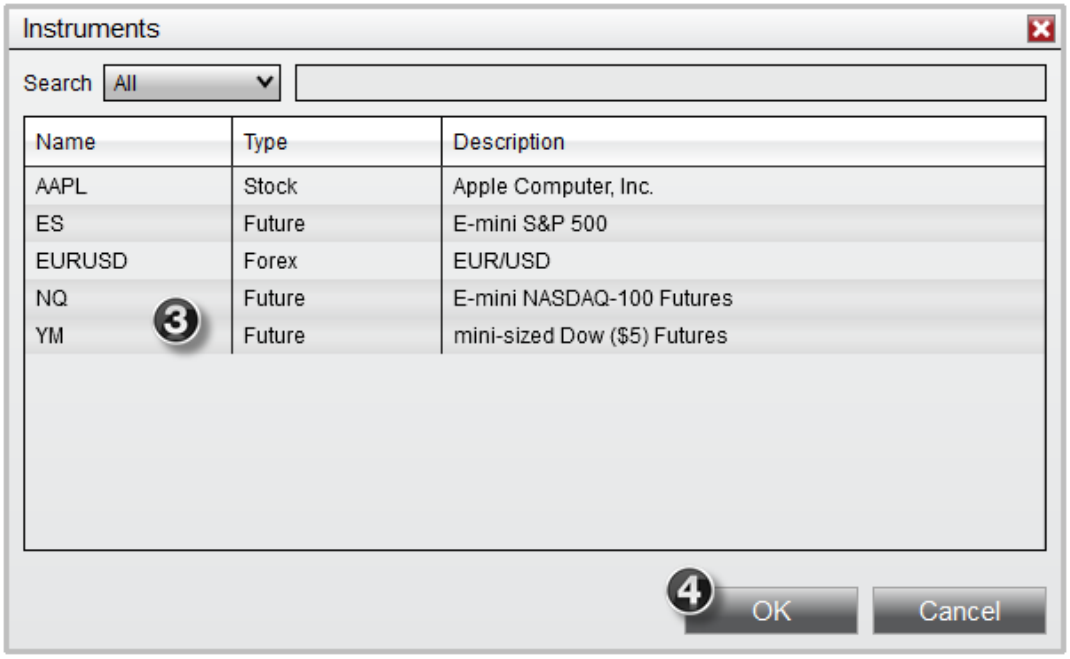
1. Select your **Configured** condition
2. Press the Magnify glass icon next to the **Apply** to field



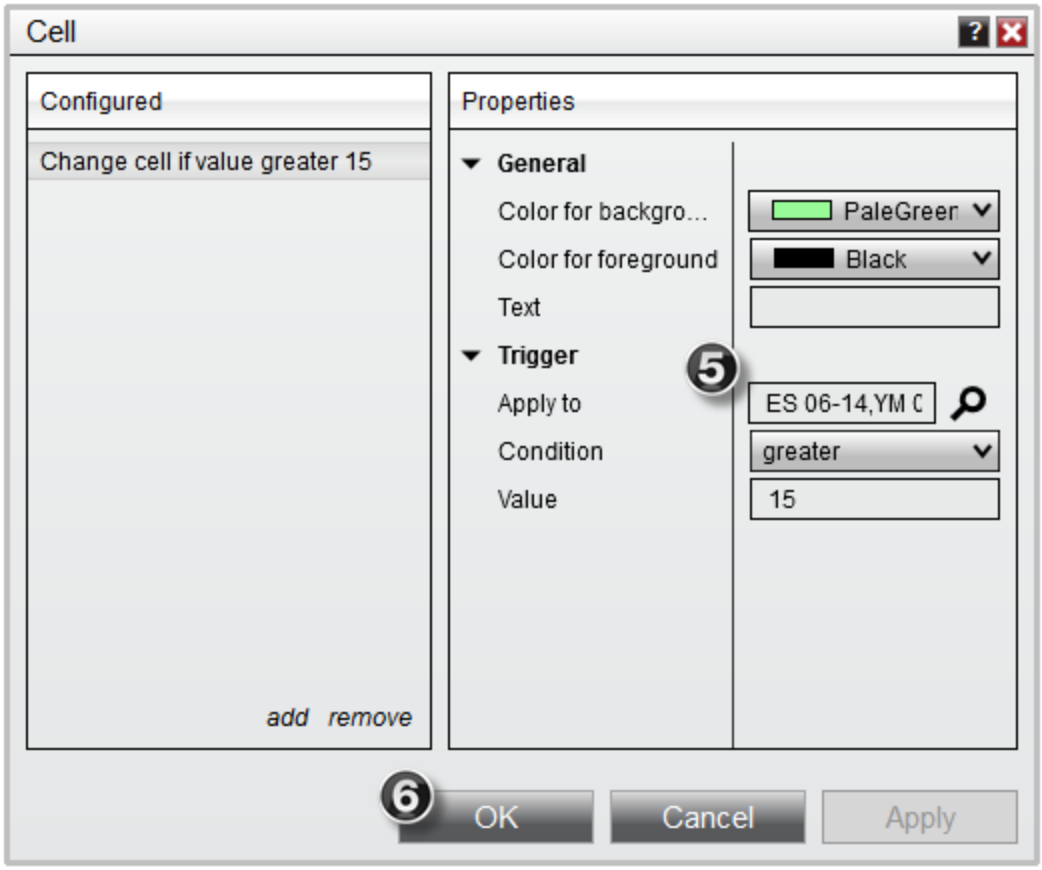
3. From the newly opened **Instruments window**, select the instruments you wish to apply the condition

> ***Tip***
> *Multi-select is supported in the Instrument window:*
> - *To select a consecutive instruments, click the first instrument, press and hold down the Shift key, and then click the last instrument.*
> - *To select non-consecutive instruments, press and hold down the Ctrl key, and then click each instrument that you want to select.*

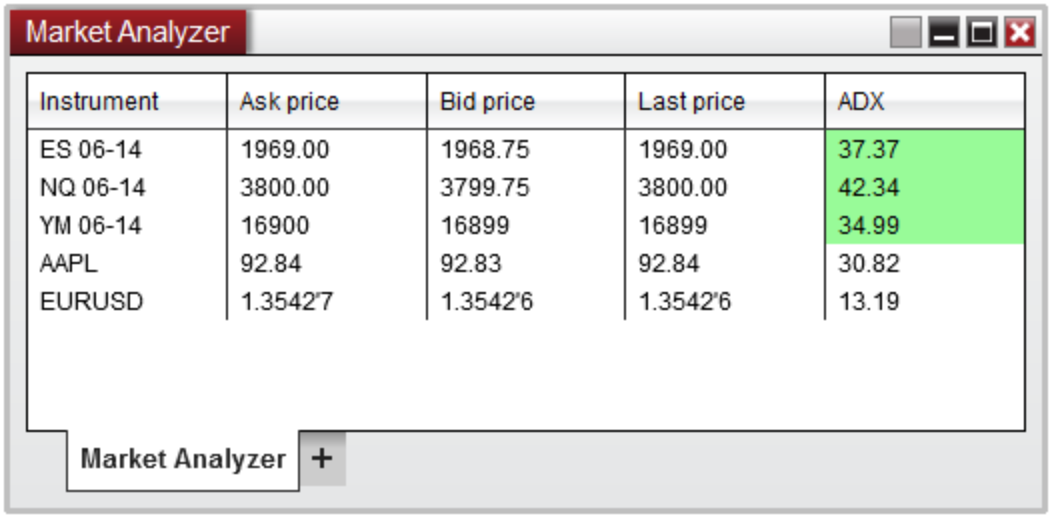
4. Press ***OK*** on the **Instruments window**



5. Your **Apply to** field will now list the instrument names you selected earlier, indicating that conditions will only be triggered on instruments contained in this list.
6. Press ***OK*** **Conditions** window



Your **Market Analyzer** window will now only apply these conditions to the instruments which mach the name you configured



10.19.6 Market Analyzer Properties

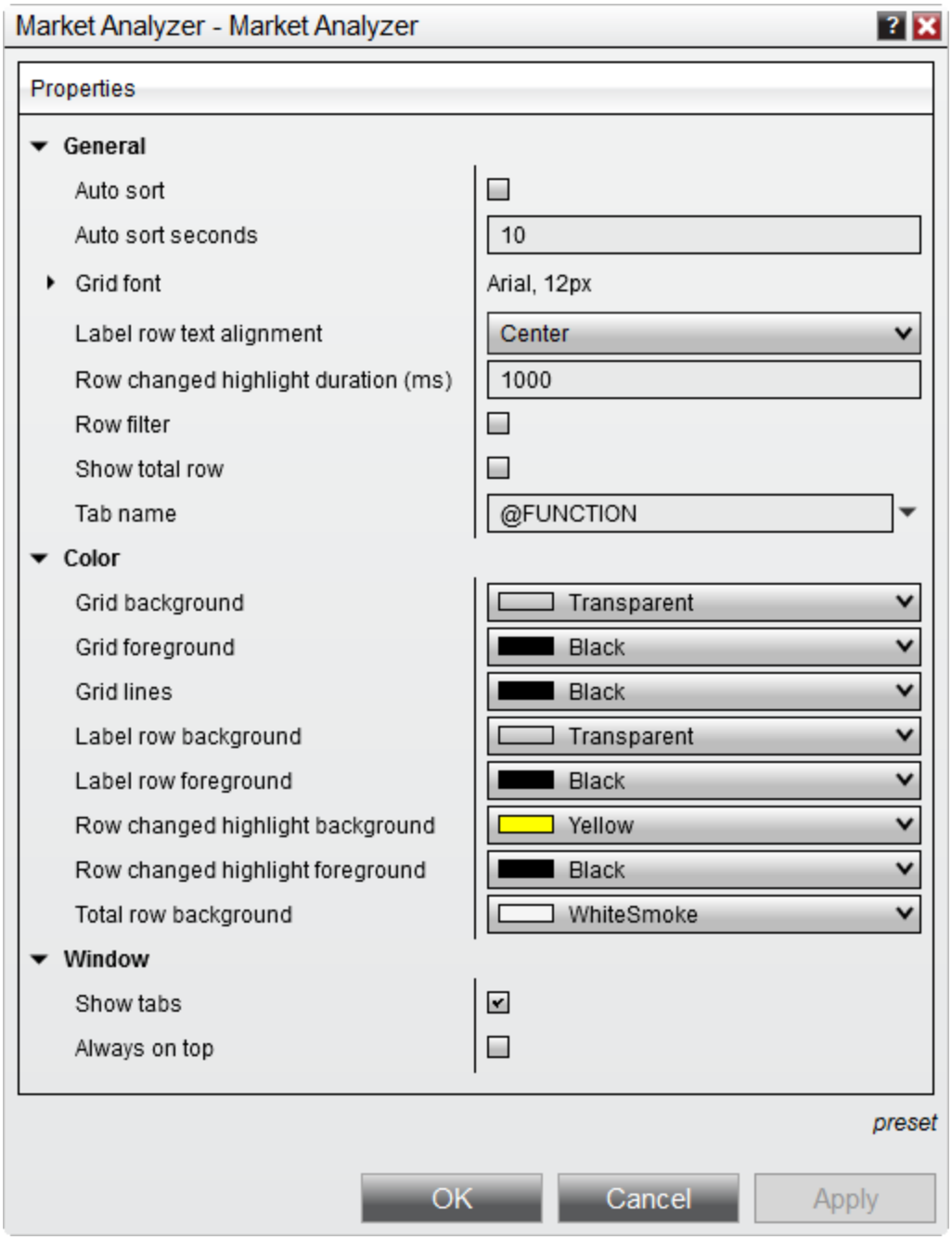
The Market Analyzer can be customized to your preferences in the Market Analyzer Properties window.

How to access the Market Analyzer properties window

To access the Market Analyzer Properties window, press down on your right mouse button inside the Market Analyzer window and select the menu **Properties...**

Available properties and definitions

The following properties are available for configuration within the Market Analyzer Properties window



Property Definitions

General	
Auto sort	Enables/Disables the automatic ranking and sorting of rows

Auto sort seconds	Sets the interval time in seconds between automatic resorting of rows
Grid font	Sets the font
Label row text alignment	Sets the alignment for the label rows
Row change highlight duration (ms)	Sets the duration (in milliseconds) the instrument cell will remain highlighted. **Note**: The lowest value which will take effect is 1000 (ms)
Row filter	Enables/Disables the automatic filtering of rows from the grid display based on the Filter Conditions of the columns.
Show total row	Enables/Disables the Total row in the Market Analyzer window display grid
Tab name	Sets the tab name
Color	
Grid background	Sets the default color of the display grid background
Grid foreground	Sets the default color of the text in a cell
Grid lines	Sets the color of grid lines
Label row background	Sets the default color for the Label row background
Label row foreground	Sets the default color for the Label row foreground

Row changed highlight background	Sets the color for the row change highlight background (set to transparent to disable)
Row changed highlight foreground	Sets the color for the text in the row change highlight (default is disabled, as set to grid foreground)
Total row background	Sets the color of the Total row background
Window	
Show tabs	Enables/Disables the tab control
Always on top	Enables/Disables if the window will be always on top of other windows.

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

Market Analyzer Columns along with custom properites can be saved within a Market Analyzer Template.

10.19.7 Working with Templates

NinjaTrader allows you to save your customized Market Analyzer layout as a template that can be loaded in an open Market Analyzer or set as the default for new Market Analyzer windows.

How to save a Market Analyzer Template

What is Saved

The following are saved within a Market Analyzer template:

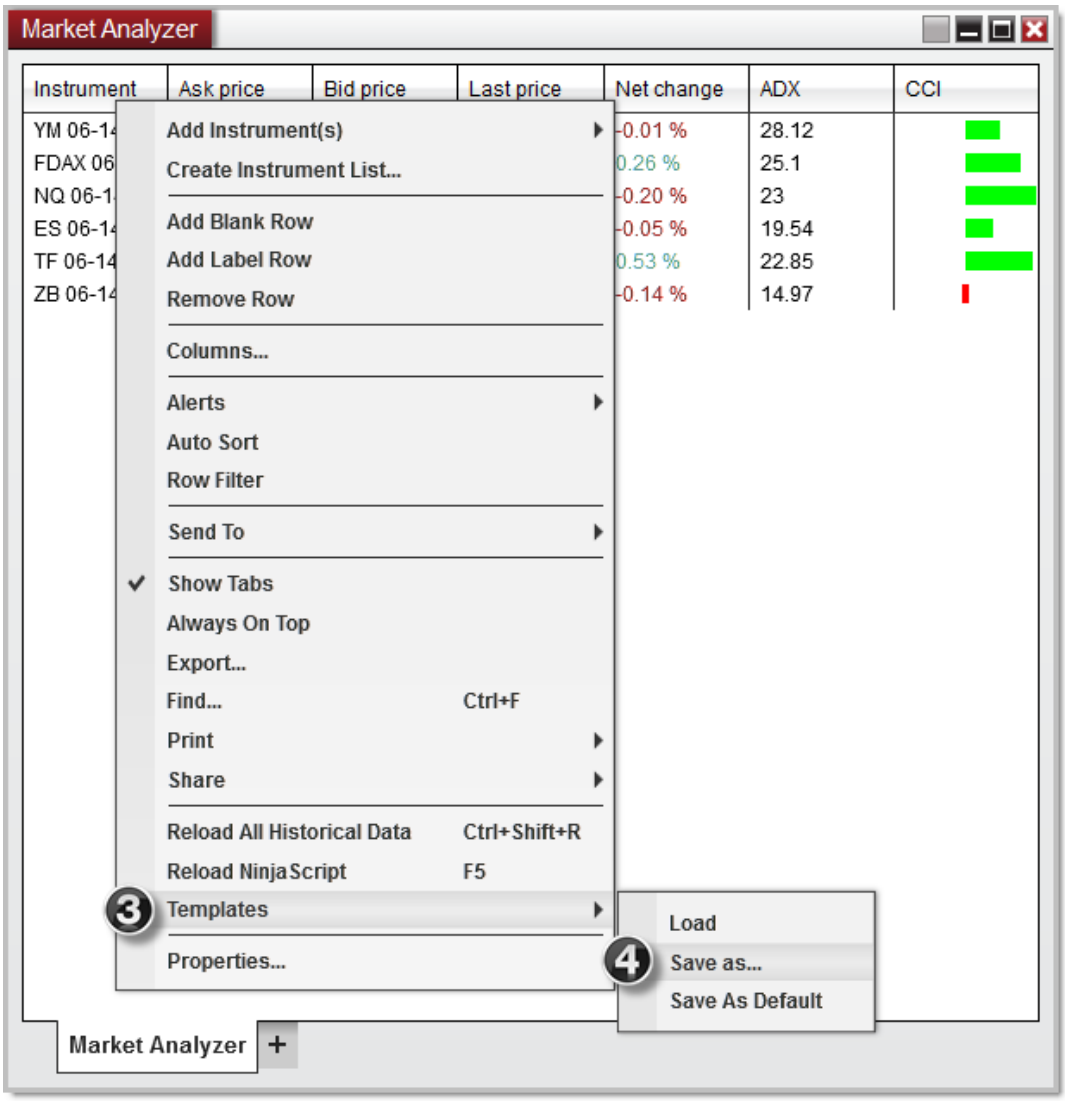
- Column layout
- Column parameters and conditions
- Market Analyzer properties
- Instrument Rows

Saving a Market Analyzer Template

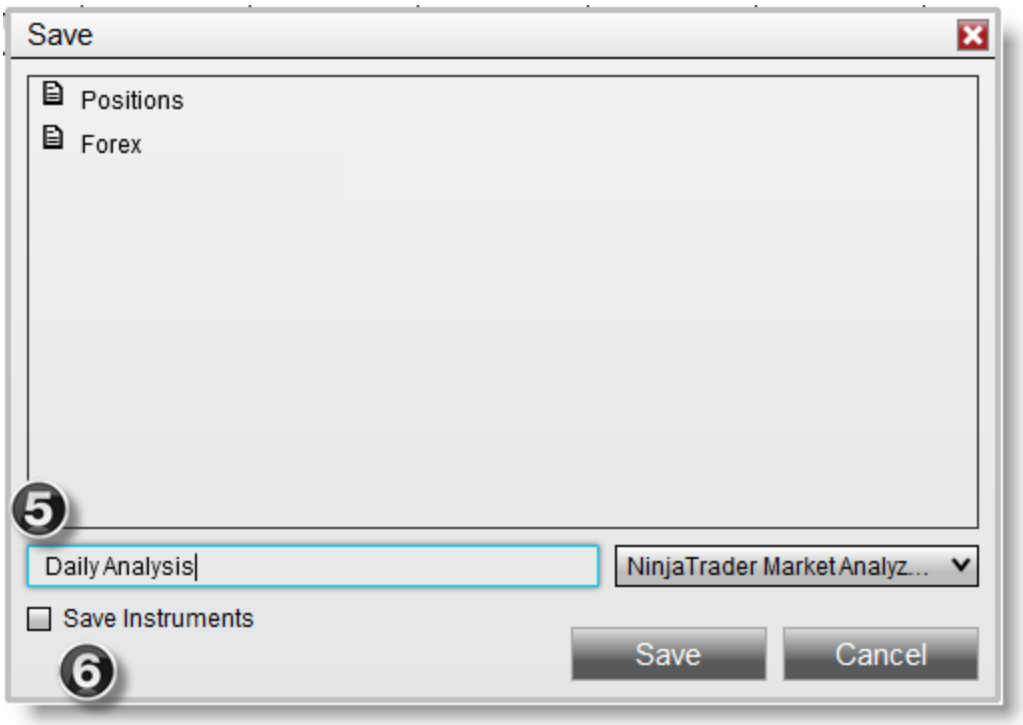
To save a Market Analyzer template (shown in the image below):

1. Configure your desired Market Analyzer columns and properties (see the "*Working with Columns*" and "*Market Analyzer Properties*" sections of the Help Guide for more information)
2. Right mouse click within the Market Analyzer
3. Select the menu item **Templates**
4. Select the menu item **Save As...**

(You can optionally select the menu item **Save As Default** to save the current settings as default. Any new Market Analyzer will load with these new default settings)



5. Enter a name for your Market Analyzer template
6. Optionally check **Save Instruments** to save the current display of instrument rows in the Market Analyzer template



7. Press the **Save** button

How to load a Market Analyzer Template

Loading a Market Analyzer Template

To load a saved Market Analyzer template:

1. Right mouse click within the Market Analyzer
2. Select the menu item **Templates**
3. Select the menu item **Load**
4. Select the template you wish to load from the Load dialog menu and press the OK button

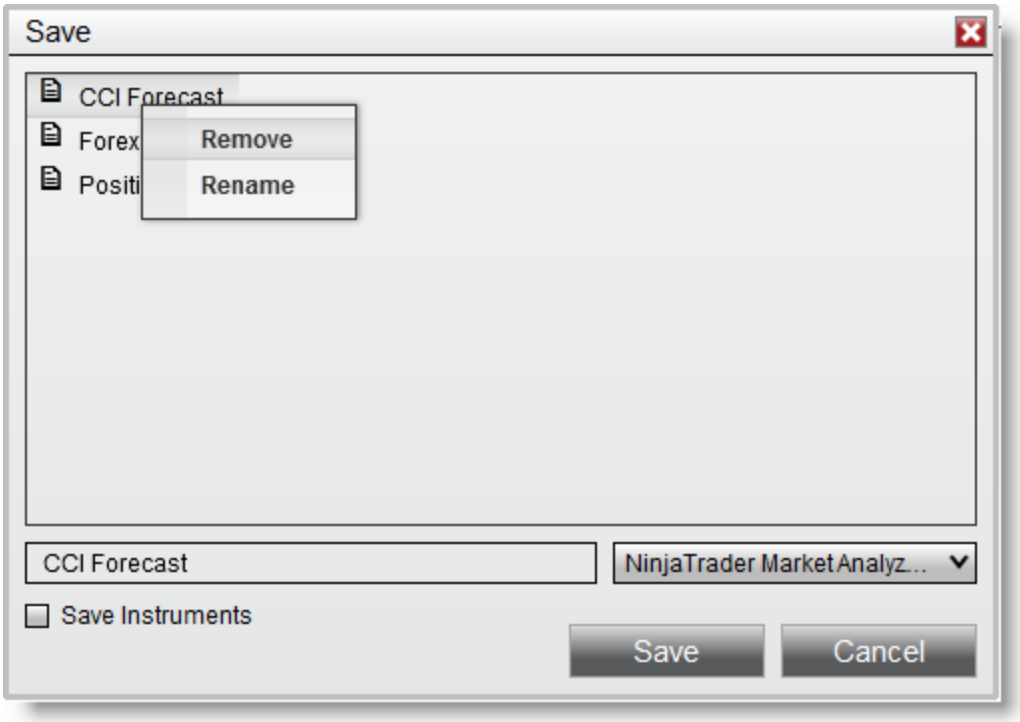
How to remove a Market Analyzer Template

Removing a Market Analyzer Template

To remove a saved **Market Analyzer** template:

1. Right mouse click within the **Market Analyzer**

2. Select the menu item **Templates**
3. Select the menu item **Load**
4. Right click on the template you wish to remove from the Load dialog menu and select the **Remove** menu item



> ***Note***
> *If you wish to rename an existing template, you can select* **Rename** *from the same menu*

10.19.8 Performance Tips

The following performance tips are specific to the **Market Analyzer** window.

Number of Instruments and Columns.

- The Market Analyzer has no limit to the number of instruments that can be added. It is important to monitor computer resources to understand your PC's limit.
- The Market Analyzer has no limit to the number of columns, specifically indicator columns, that can be added. Depending on the indicator and "**Time Frame**" property described below, it may take a few seconds to calculate the indicator. This time is multiplied by the number of instruments in the Market Analyzer which can result in a few minutes of loading time. Decrease the number of instruments or indicator columns to lessen this loading time.

Indicator Columns

- The **Time Frame** ("Bars to load", "Days back", "Custom range") property determines the minimum number of bars required to properly initialize each indicator column. The higher the number, the longer it will take to load data and the more memory (RAM) NinjaTrader will use to hold the data in memory.
- Each indicator has a Maximum bars look back parameter in the Columns window that determines how many historical indicator values are stored for access. It is set to TwoHundredFiftySix by default for optimal performance. Setting this to Infinite will take longer to calculate and NinjaTrader will use more memory (RAM) to hold the extra values in memory.

Indicator Calculate on Bar Close Parameter

- All indicators added to the Market Analyzer have the parameter "Calculate" set to "On bar close" as default which only calculates the indicator value on the bar close to help with PC performance. This parameter can optionally be set to "On price change" which will only calculate when there has been a change in price, or "On each tick" which allows for a tick-by-tick calculation (which will use more CPU resources).

Dynamic Ranking and Sorting Frequency

- Depending on the number of Instruments you have added to your **Market Analyzer** display, using a low "Auto Sort seconds" value can cause your CPU to spike as the auto sort feature continues to re-evaluate the ranking of the column you are sorting. For example, using a value of 1 second on 100 instruments could potentially overwork your CPU. Setting this to a higher value, such as every 10-30 seconds, will reduce the CPU workload, and still maintain dynamic sorting at a customizable interval. You should monitor your CPU workload to find the number of seconds that work for your system.

10.19.9 Reloading Indicators & Columns

When compiling custom NinjaScript indicators and columns, the Market Analyzer window will not automatically reload the changes. To force a reload of any changed indicators or columns you must select the menu item **Reload NinjaScript** via the right mouse button context menu or alternatively, press the "F5" Hot Key.

10.19.10 Window Linking

One of the most useful features of the **Market Analyzer** is the ability to link the instruments displayed in the **Market Analyzer** grid to any other window in the NinjaTrader application. This allows you to cycle through a custom list of instruments and quickly load the desired symbols in a **Chart**, **SuperDOM**, or any other feature which uses the Window Linking feature.

In order to accomplish this setup, please see the steps and image below:

1. Select a **Link Color** in the **Market Analyzer**
2. Select the same **Link Color** in any number of windows you wish to have updated

3. Using your mouse, left or right mouse click on any instrument row in the **Market Analyzer**

In the example image below, doing so will change the current instrument displayed in the **Chart** (*ADI*) with the instrument that was selected in the **Market Analyzer** (*ALTR*).

All windows that are linked by the same color will receive the same change of instrument request.

10.20 Market Watch

Market Watch Window Overview

You can access the **Market Watch** window from within the NinjaTrader Control Center window by left mouse clicking on the menu **New**, and then selecting the menu item **Market Watch**.

The **Market Watch** window displays tiles of the selected instrument(s) that display a 1 minute net change graph of the current session, last price, and net change in points & percent for a quick glance at the market(s).

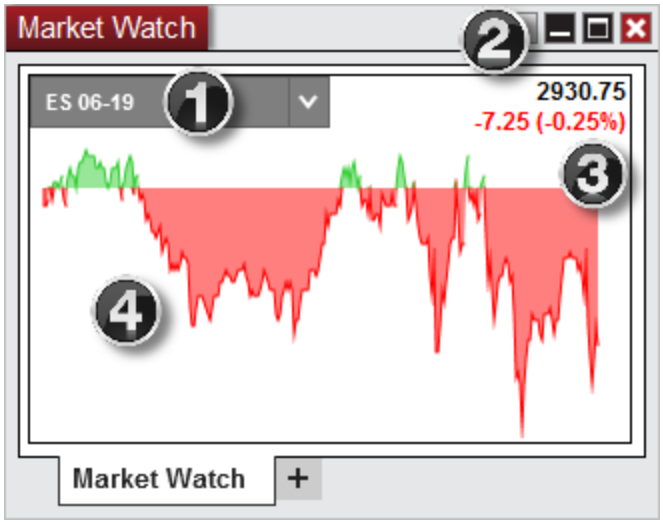
10.20.1 Display Overview

To open the **Market Watch** window, select the **New** menu from the NinjaTrader **Control Center**. Then left mouse click on the menu item **Market Watch**.

Instruments will need to be added to the **Market Watch** window to display the instrument tile. One of the ways to do this is by right clicking on the window and going to **Add Instrument(s)**. See the Working with Instrument Tiles section for more information.

Tile Display

1. The display for the selected instrument of the tile & instrument selector
2. The last price
3. The net change displayed in points and percent
4. The net change chart. With default settings the green area are prices that were above the last close and the red area are prices that were below the last close.



Notes:
1. Market Watch requires Last Close to be provided by the data connection. If the data connection does not support Last Close, Market Watch will not properly display (for example FXCM / Forex.com / eSignal are known to not provide Last Close)
2. Tiles do not have a time relation to one another. For each tile the left of the tile is the open of the session and the right is the end of the session per the default session template for the instrument.

10.20.2 Working with Instrument Tiles

The **Market Watch** can be setup for use with an unlimited number of **instrument tiles** which are used to display a net change chart, last price, and net change in points & percent.

▼ Managing instrument tiles

Adding an Individual Instrument

You can add as many individual **instrument tiles** to your **Market Watch** window as you would like.

- Press down on your right mouse button in the **Market Watch** window and select the menu item **Add Instrument(s)**. Through the **Instrument Selector** menu, you can navigate through various instrument lists to locate the instrument you desire, and left click on the instrument to add the individual instrument to the **Market Watch.**

Adding a List of Instruments

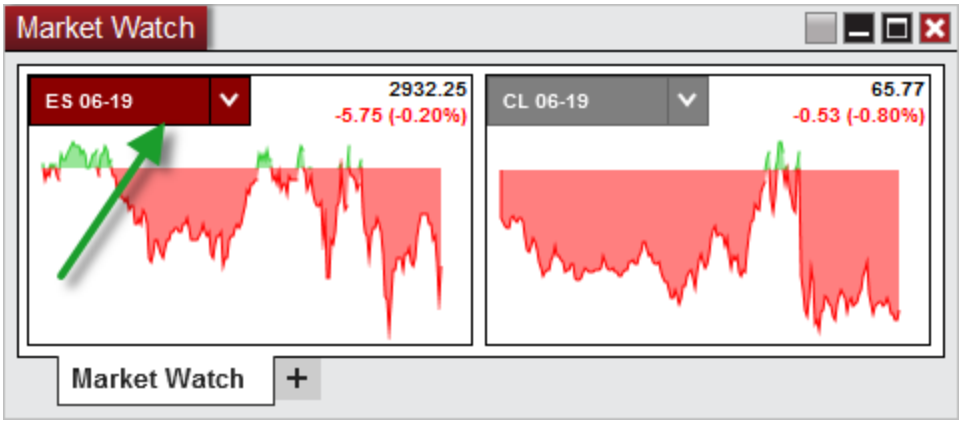
You can also rapidly add an entire list of predefined instruments to the **Market Watch** window.

- Press down on your right mouse button in the **Market Watch** window and select the menu item **Add Instrument List**. Then select the instrument list you would like to add to the **Market Watch**.

Please see the Instrument Lists section of the user help guide for additional information on creating, editing, and deleting instrument lists.

Changing Instruments

Once an **instrument tile** has been added to the **Market Watch** display, you can quickly change the instrument by using the **Instrument Selector**.



Arranging Tiles

You can customize the arrangement in which each **instrument tile** is displayed by left clicking and dragging the **instrument tile** to the desired location.

Removing Instruments

To remove an instrument tile, simply right click on the desired tile and select **Remove Tile.**

Creating instrument lists from the Market Watch

Creating an Instrument List

If you have a **Market Watch** setup with a number of different instruments you would like to save for later, you can quickly add the entire display of instruments into an **Instrument List** for quick access.

- Press down on your right mouse button in the **Market Watch** window and select the menu item **Create Instrument List**, then give the **Instrument List** a unique name and press **OK**.

You will now be able to access this list from other features of NinjaTrader using the Instrument Selector. You can further edit this list by using the Instrument Lists window

10.20.3 Market Watch Properties

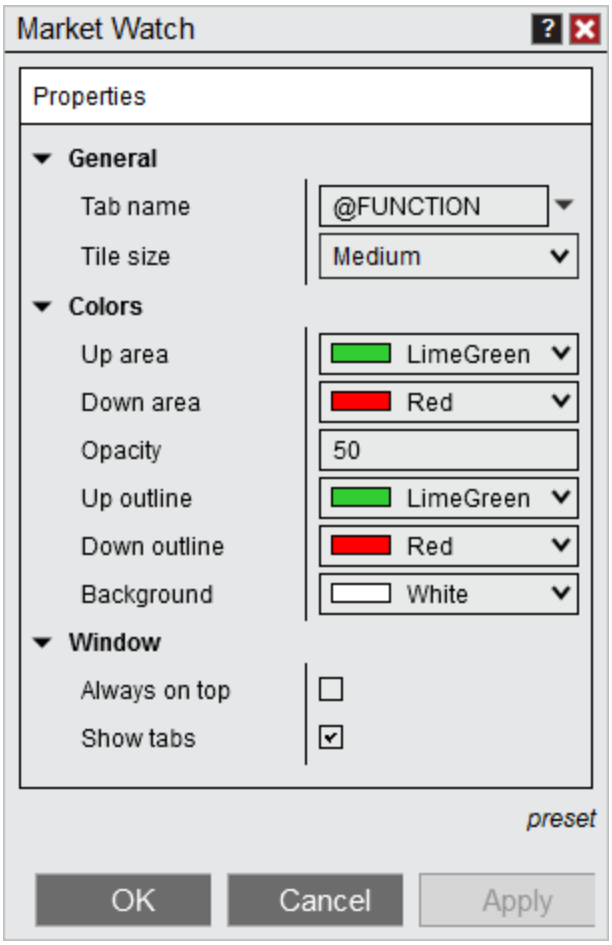
The Market Watch window can be customized through the **Market Watch Properties** window.

How to access the Market Watch Properties window

You can access the Market Watch properties dialog window by clicking on your right mouse button and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the Market Watch Properties window:



Property Definitions

General	
Tab name	Sets the name of the tab, please see Managing Tabs for more information.
Tile size	Select the size used for all tiles in the Market Watch display
Colors	
Up area	Sets the area color for prices above last close

Down area	Sets the area color for prices below last close
Opacity	Sets the area opacity
Up outline	Sets the outline color for prices above last close
Down outline	Sets the outline color for prices below last close
Background	Sets the background color for the tile
Window	
Always on top	Sets if the window will be always on top of other windows.
Show Tabs	Sets if the window will allow for tab support

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

Using Tab Name Variables

Tab Name Variables

A number of pre-defined variables can be used in the "Tab Name" field of the **Market Watch Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.21 News

News Window Overview

The **News** window can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting the **News** menu item.

The **News** window allows you to display, filter and create alerts for real-time news. You will receive real-time news if are subscribed to a news service through a market data vendor or broker.

10.21.1 News Window

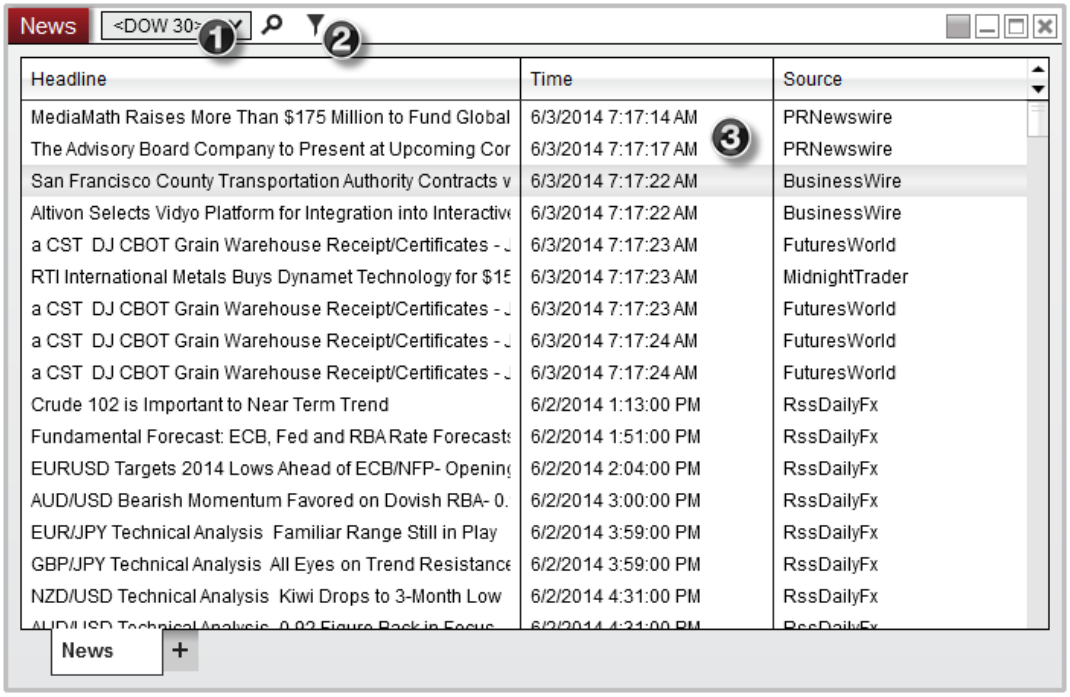
News sent from the connectivity provider is displayed in the **News** window. Alerts and filters can be configured based on keywords in the news headline.

▼ Understanding the News Window

News Window Display

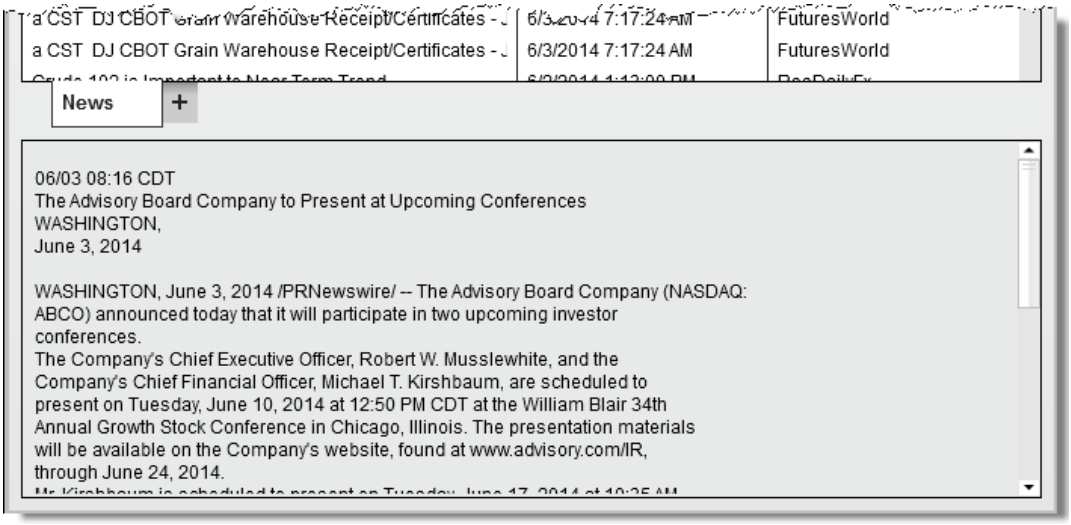
The **News** window give you the ability to:

1. Filter news based on individual instruments or instrument list
2. Setup user defined Keyword filters
3. View a list of real-time news headlines



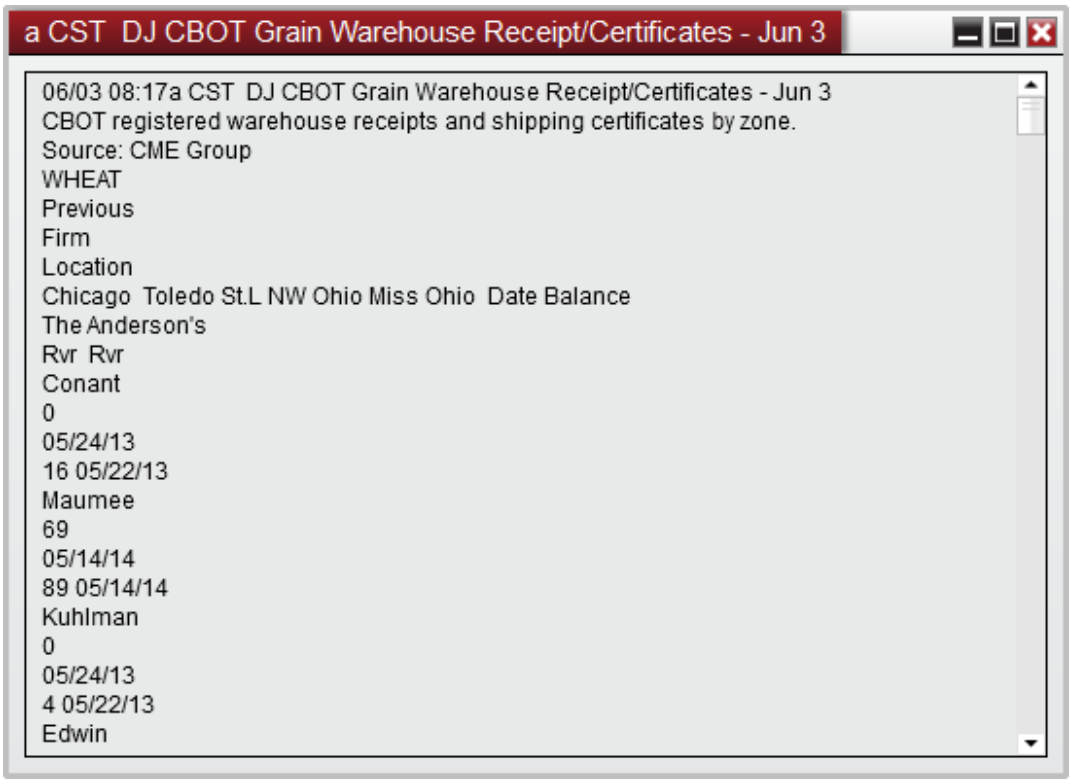
Reading Pane

And optional Reading Pane can be enabled below the list of real-time news headlines by right clicking on the News window and selecting **Show Reading Pane**.



Headline Window

Double clicking on a headline will open the Headline Window which will display the content of the news article.



Note: For the News window to populate information, you must be connected to a data provider that supports real-time news. See the Data by Provider section for information on what providers support news.

How to create a filter condition

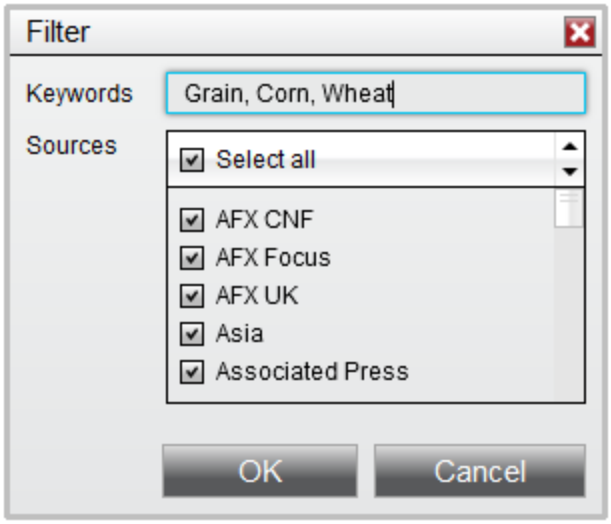
Creating a Filter Condition

The News window gives you the ability to filter News Articles based off of user-defined Keywords

To enable this type of conditional filtering:

1. Select the Filter Icon from the News window

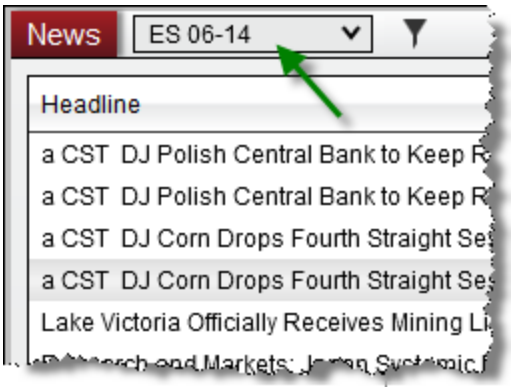
2. From the Filter window, insert the Keywords you wish to filter (multiple keywords can be separated by commas)
3. Optionally uncheck any Sources you may wish to exclude from your results
4. Press OK



Filtering on specified instrument(s)

You can also define news filters based on a specific instrument, or even a list of predefined instruments.

To enable this type of filtering, simply select the desired Instrument or Instrument List from the **Instrument Selector** of the News Window



How to create an alert on news article

An alert will visually and audibly notify you when a new article is received.

Enabling News Alert

To turn on/off the Alerts feature:

Right click on the News window and check or uncheck **Alert on New Article**

You can customize the sound file, priority, and colors of the Alerts which are generated from the News Properties

Alerts will be sent to the Alerts Log window

10.21.2 News Properties

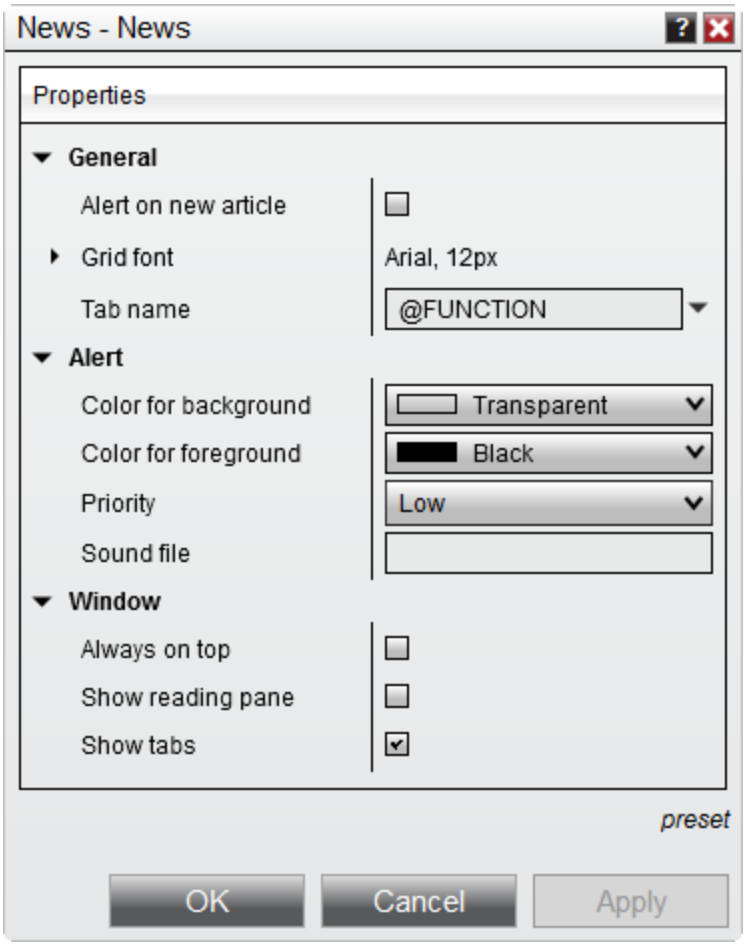
The News window can be customized through the **News Properties** window.

How to access the News Properties window

You can access the News properties dialog window by clicking on your right mouse button and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the News Properties window:



Property Definitions

General	
Alert on new article	Sets the option to receive alerts when new article is received
Grid font	Sets the font options
Tab name	Sets the name of the tab, please see Managing Tabs for more information
Alert	
Color for background	Sets the alert background color

Color for foreground	Sets the alert text color
Priority	Sets a user defined priority
Sound file	Sets the sound file that will play when the alert is triggered
Window	
Always on top	Sets if the window will be always on top of other windows.
Show reading pane	Sets if the Reading Pane is displayed

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

10.22 Option Chain

The **Option Chain** window can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting the **Option Chain** menu item.

Option Chain Overview

The Option Chain window displays a listing of all available options contracts, along with current quotes, for the selected security.

Display

› Display Overview

Misc

› Properties

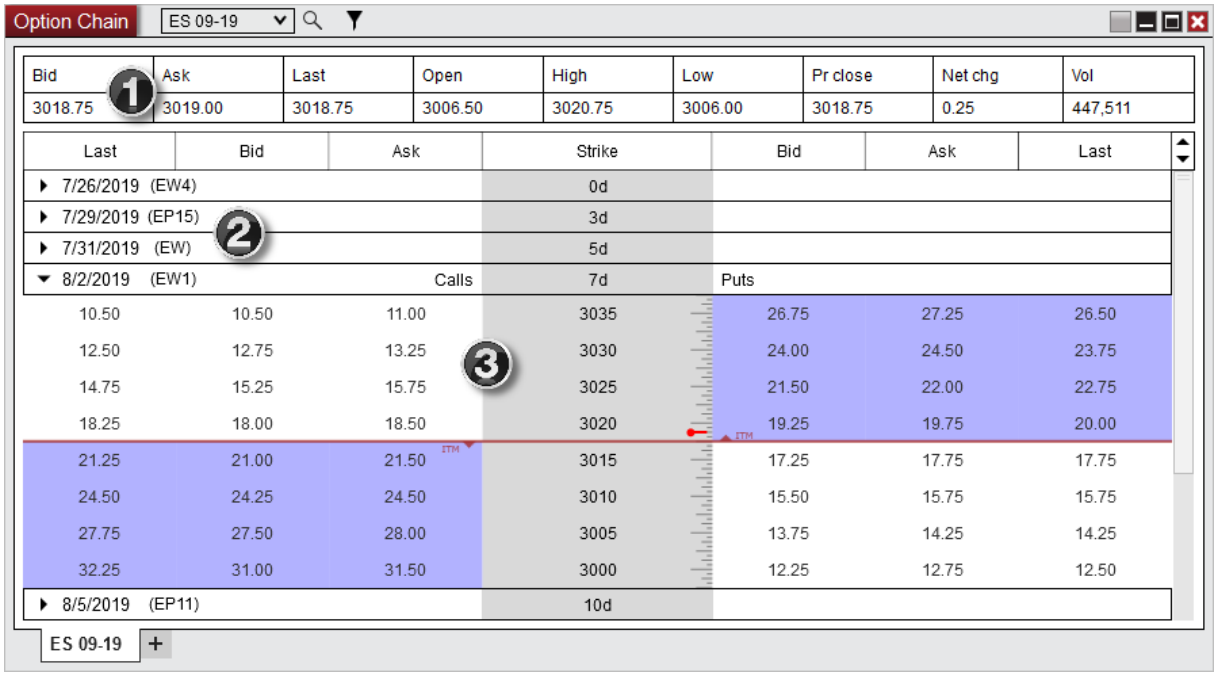
Order and Position Management

› Submitting Orders

10.22.1 Display Overview

To open the **Option Chain** Window, select the **New** menu from the NinjaTrader Control Center. Then left mouse click on the menu item **Option Chain**.

The image below shows the three sections in the of the **Option Chain** window:

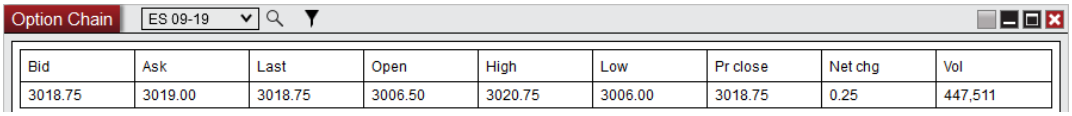


1) Quotes grid
2) Ticker list
3) Options quotes

Understanding the quotes grid section

Quotes Grid

The **Quotes Grid** displays current day quotes for the underlying security of the **Option Chain** window.



Bid	Ask	Last	Open	High	Low	Pr close	Net chg	Vol
3018.75	3019.00	3018.75	3006.50	3020.75	3006.00	3018.75	0.25	447,511

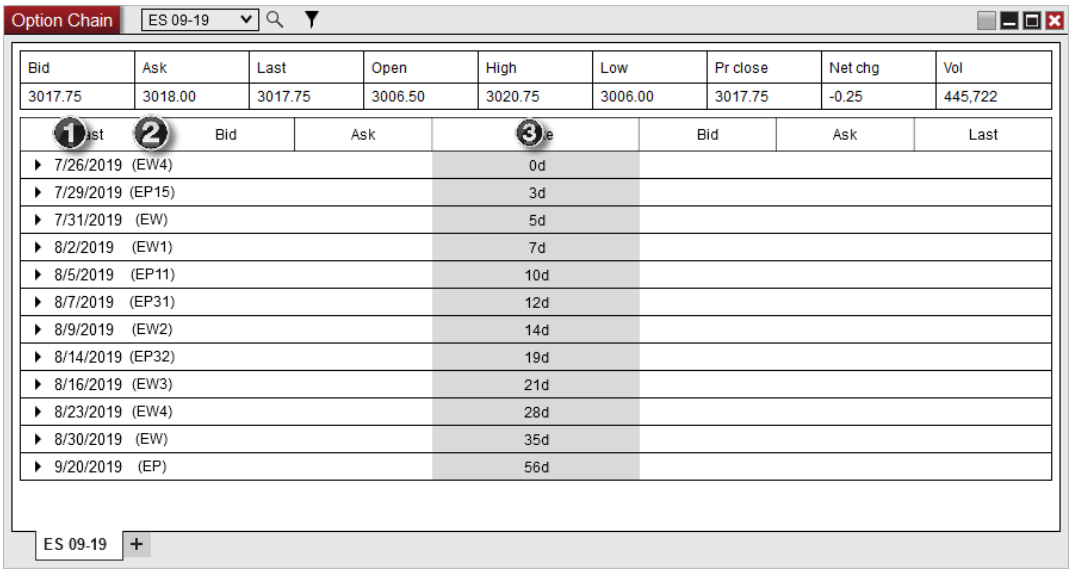
Column Definitions

Bid	The bid price
Ask	The ask price
Last	The last price
Open	The daily open price
High	The daily high price
Low	The daily low price
Pr close	The prior day close price
Net chg	The net change since the prior day close in points
Vol	The daily volume

Understanding the ticker list section

Ticker List

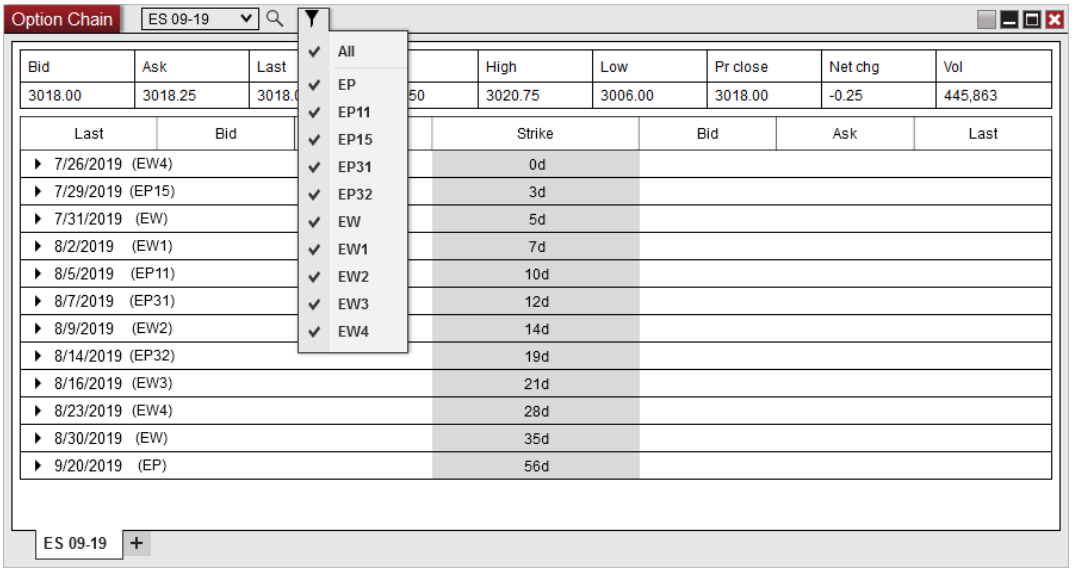
The **Ticker List** displays the active options tickers for the selected underlying security in the **Option Chain** window.



1) Expiration date
2) Ticker name
3) Days until expiration

Filtering Options Contracts

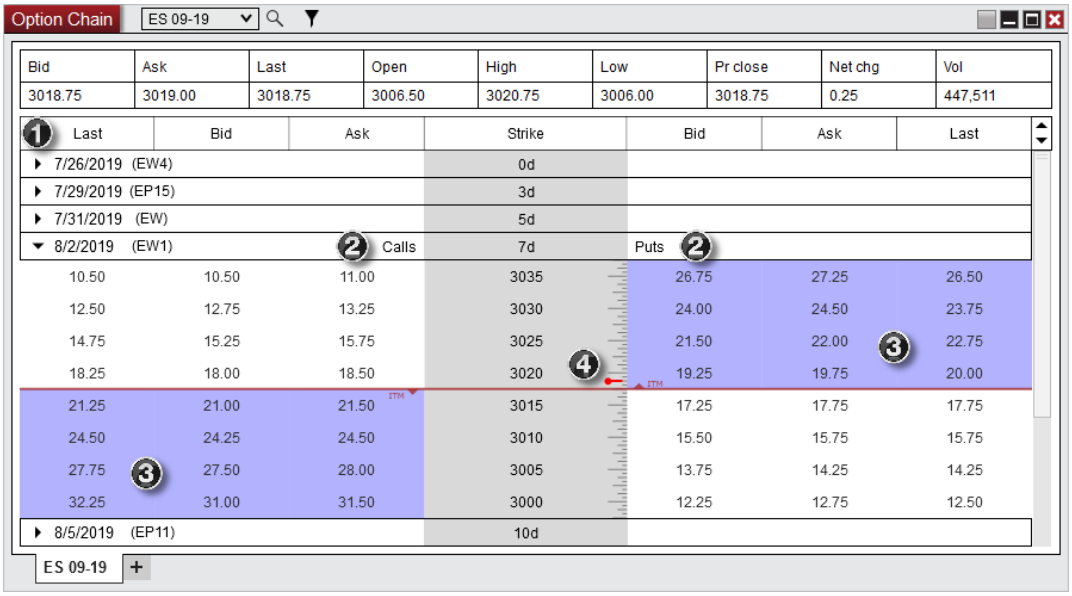
With the filter button at the top of the window you can check or uncheck what tickers you want the **Option Chain** window to display.



Understanding the option quotes grid section

Option Quotes

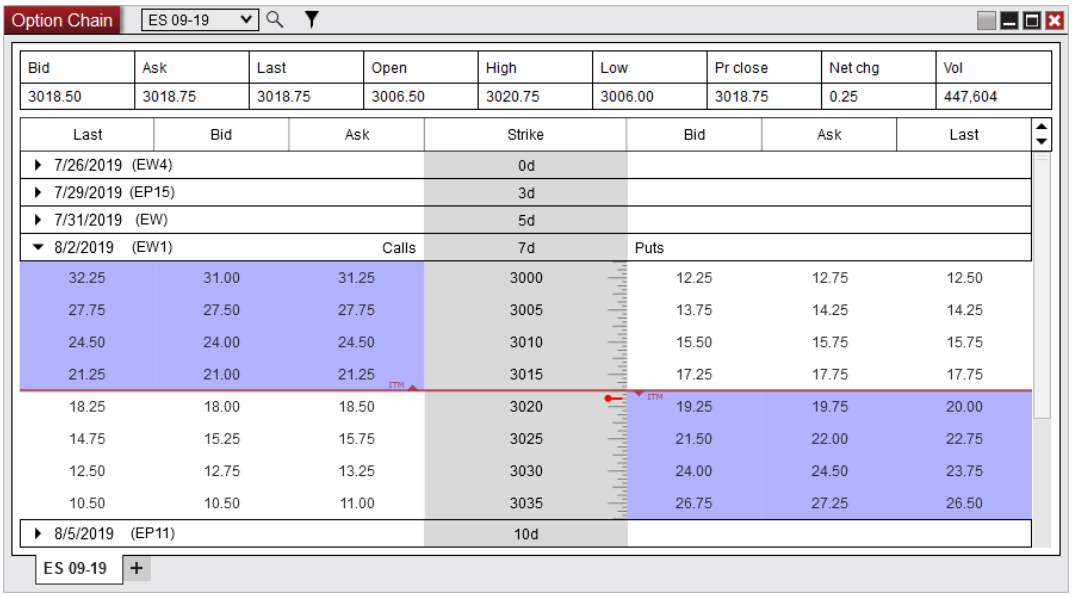
The **Option Quotes** displays calls and puts quotes for each strike price of the expanded options ticker in the **Option Chain** window.



1) Columns display - the last, bid, ask, and strike columns (additional columns can be added in the **Properties**)
2) Calls & Puts quotes - call quotes are on the left and put quotes are on the right
3) In The Money (ITM) options - indicated by the shaded area the ITM triangle is pointing towards on the calls and puts section
4) Last price for the underlying security - indicated by the marker on top of the strike prices

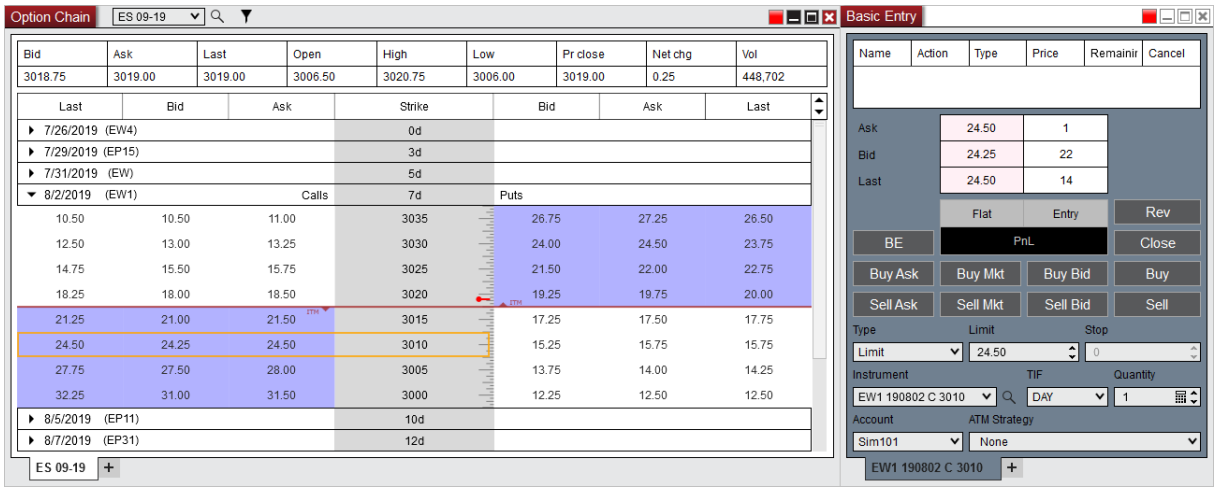
Changing Sort Direction of Strike Prices

Clicking on the **Strike** column header will change the sort direction of the strike prices. This can also be adjusted in the **Properties**.



10.22.2 Submitting Orders

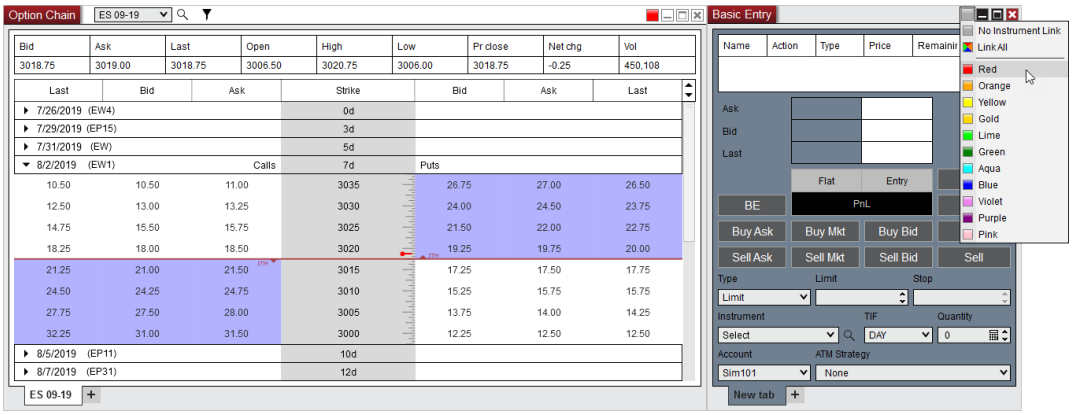
Submitting options orders is done by linking an order entry window then clicking on a bid or ask price within the options quotes section of the **Option Chain**. For more information on the order entry windows please see the Order Entry section of the user help guide.



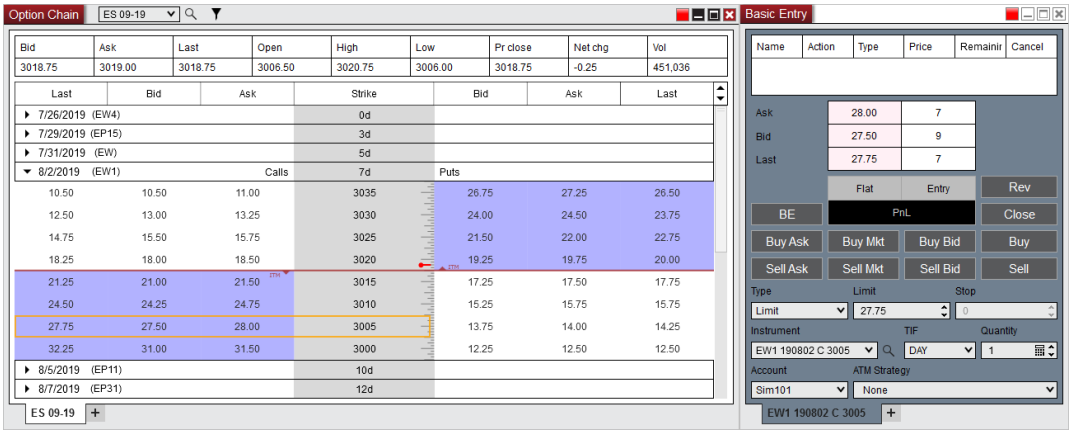
How to link the Option Chain to an order entry window

You can link the **Option Chain** window to an order entry window by
1) Setting a link color in the **Option Chain** window
2) Then setting a matching link color in an order entry window

For more information on window linking please see the Using Window Linking section of the user help guide.



To load the desired options in the order entry window
3) Click on the **Bid** or **Ask** price within the options quotes at the desired **Strike** price on the **Calls** or **Puts** side
4) The options instrument will then automatically be populated into the **Instrument** drop down menu of the order entry window



10.22.3 Properties

The Options Chain window is highly efficient by design but can also be customized to your preferences through the Options Chain Properties menu.

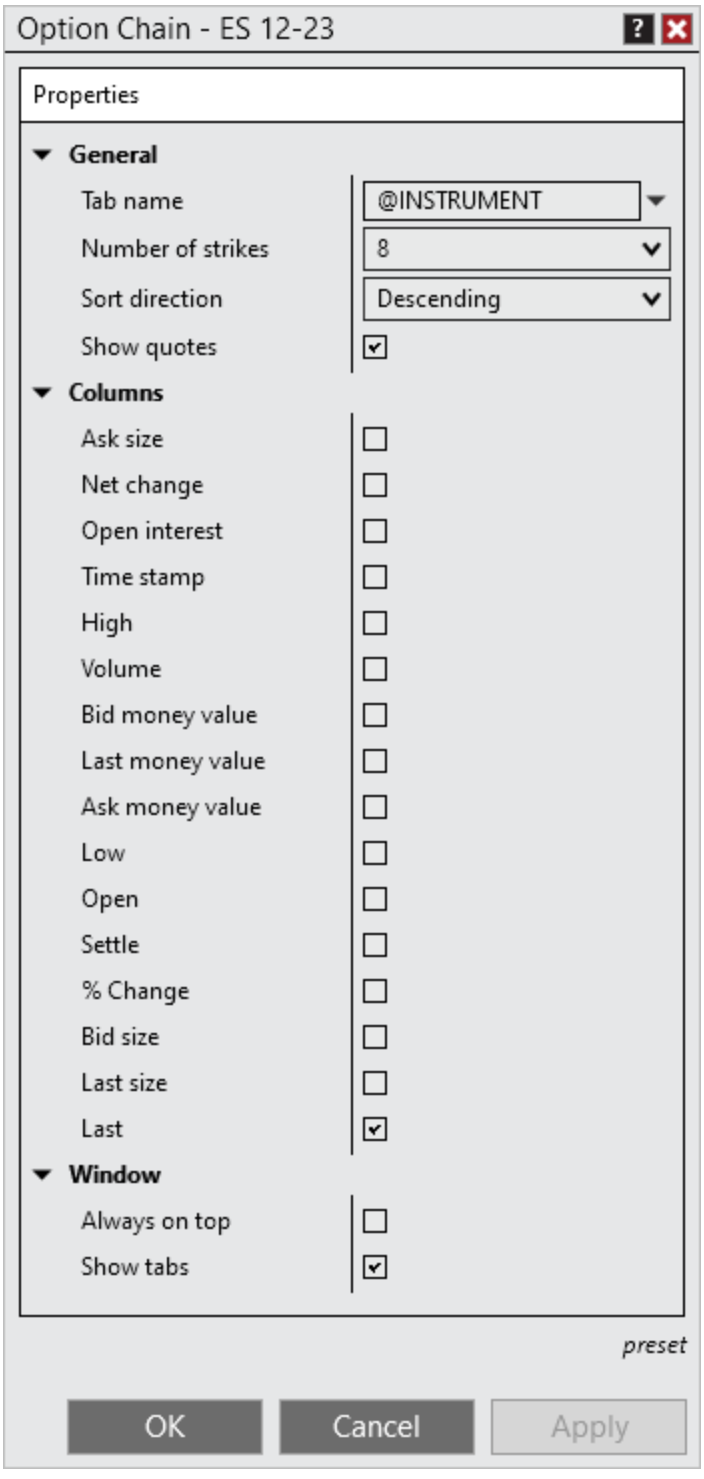
How to access the Option Chain properties window

You can access the **Option Chain** properties dialog window by clicking on your

right mouse button within the **Option Chain** window and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the **Option Chain** properties window:



Property Definitions

General

Tab name	Sets the tab name
Number of strikes	Sets the number of strikes to display
Sort direction	Sets if the strike prices are listed in descending or ascending order
Show quotes	Enables and disabled the quotes grid for the underlying security
Columns	
Columns checkboxes	Enables/disabled what columns to display on the Option Chain
Window	
Always on top	Sets if the window will be always on top of other windows
Show tabs	Sets if the window should allow for tabs

How to set the default properties

Once you have your properties set to your preference, you can left mouse click on the "**preset**" text located in the bottom right of the properties dialog. Selecting the option "**save**" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "**preset**" text and select the option to "**restore**" to return to the original settings.

Using Tab Name Variables

Tab Name Variables

A number of pre-defined variables can be used in the "Tab Name" field of the **Option Chain Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.23 Order Entry

Order Entry Overview

Various **Order Entry** windows can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting the name of order entry window.

NinjaTrader provides six graphical interfaces for order, position, and ATM Strategy management. These interfaces provide complete functionality for the management of orders, positions and discretionary exit and stop strategies in a highly visual and efficient manner. The majority of your time using NinjaTrader will be spent in one of these six interfaces if you are primarily a discretionary trader.

Order Entry Windows

- Basic Entry
- Chart Trader
- FX Pro
- FX Board
- Order Ticket
- SuperDOM

Misc

- Trade Controls
- Simulated Stop Orders
- Order State Definitions
- FIFO Optimization

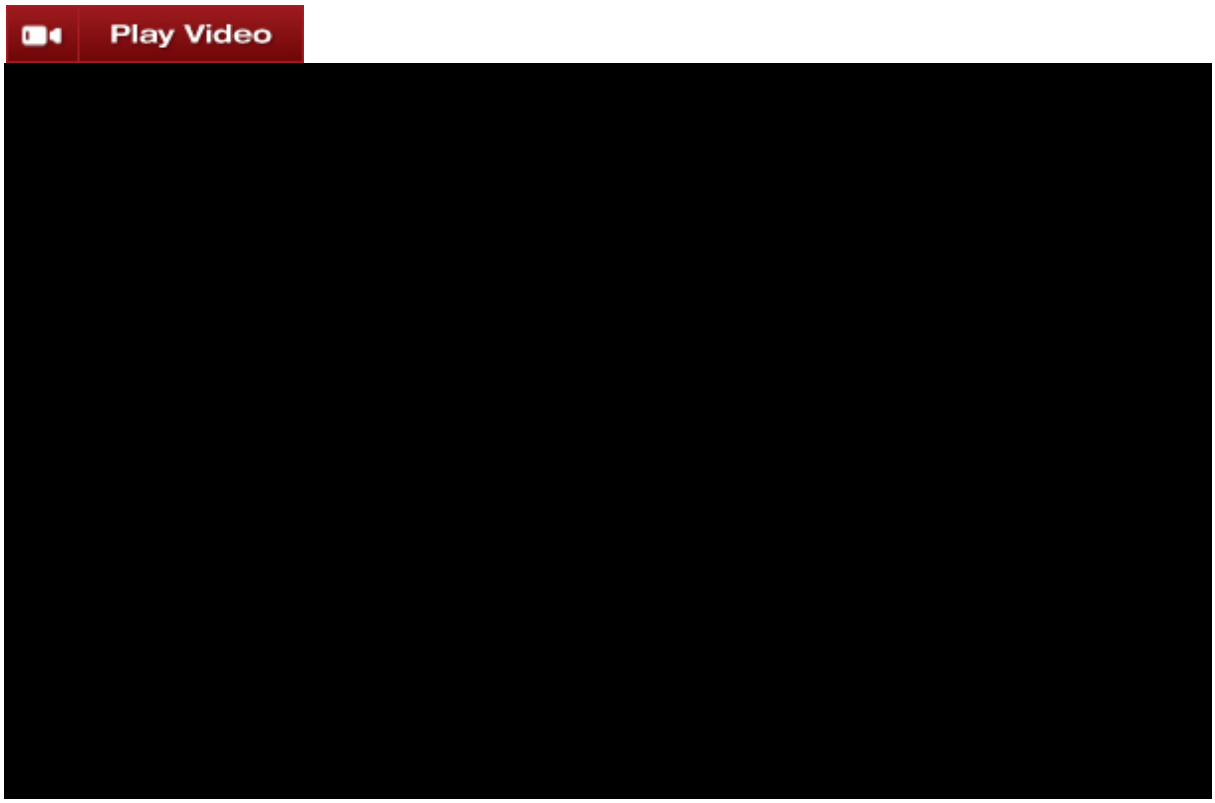
Notes:
Although the **Basic Entry**, **Chart Trader**, **Order Ticket**, and **SuperDOM** interfaces may be used to trade any of the NinjaTrader supported asset classes, the **Basic Entry** window is geared towards trading equities, the **SuperDOM** is geared towards trading futures, and the **FX Pro** and **FX Board** windows are used for FOREX and CFD instruments only.
To be able to fully utilize all order entry windows it is required to use a mouse with left, right, and middle buttons.

10.23.1 Attaching Orders To Indicators

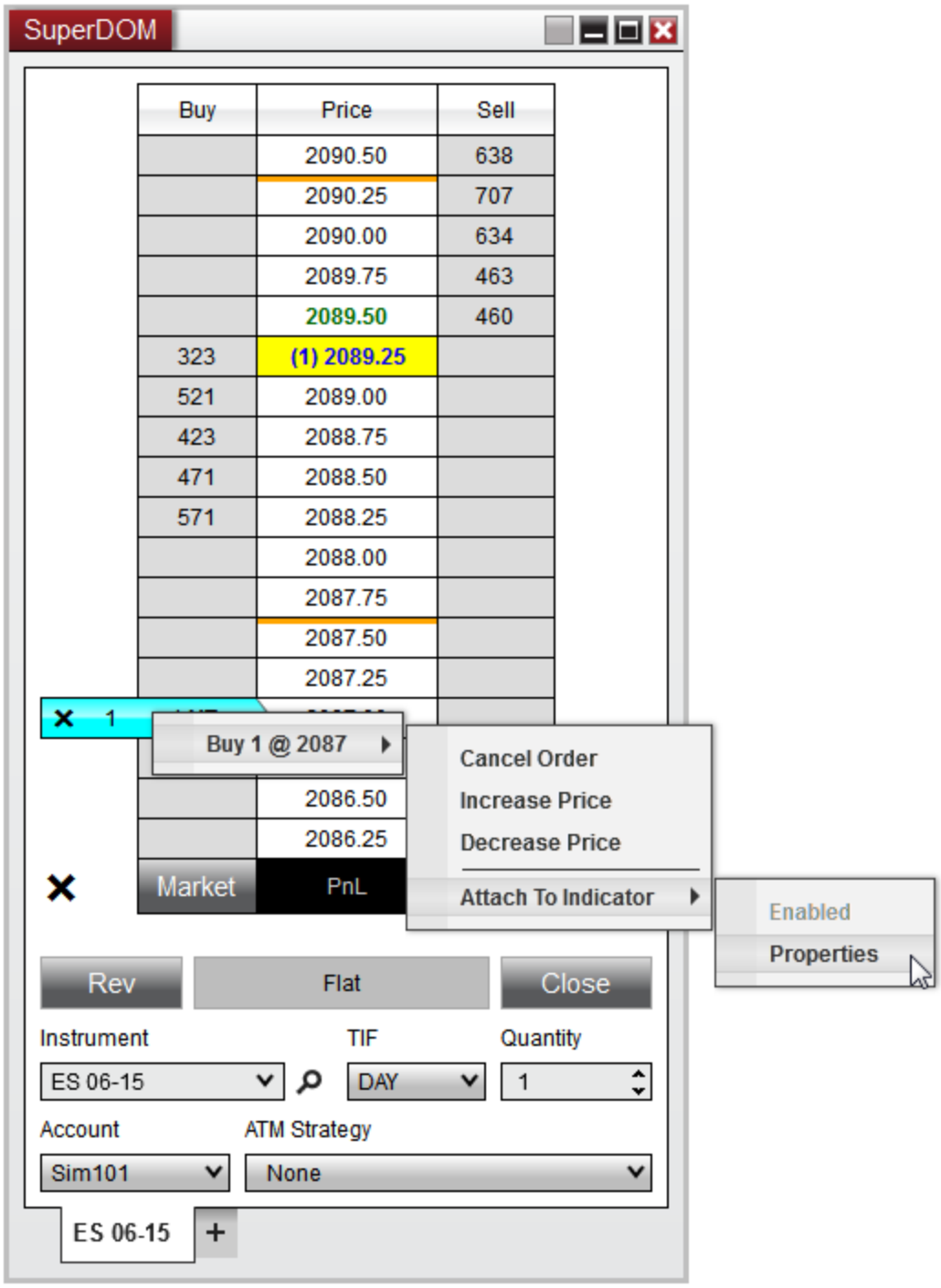
Indicator Tracking

Adding indicators to the **SuperDOM** or **Chart Trader** gives you the ability to "attach" a working order to the Indicator price level, which will automatically modify the price of the order as your indicator values change. The frequency of the modifications will depend on the Calculate settings of the indicator.

After you have configured an indicator to be displayed on the **SuperDOM** or **Chart Trader**, right clicking on a working order will now have a right click menu option called "**Attach to Indicator**". This feature will be available for both manually placed Entry/Exit orders as well as pre-configured ATM Strategy **Stop Loss** and **Profit Target** orders.



Note: **ATM Strategies** will only work with **Attach to Indicator** for stop orders which *do not* have a Stop Strategy configured. Enabling **Attach to Indicator** on an ATM Strategy which has an associated **Stop Strategy** will disable the **Stop Strategy** and will then be managed by the indicator instead.



Attaching an order to an indicator

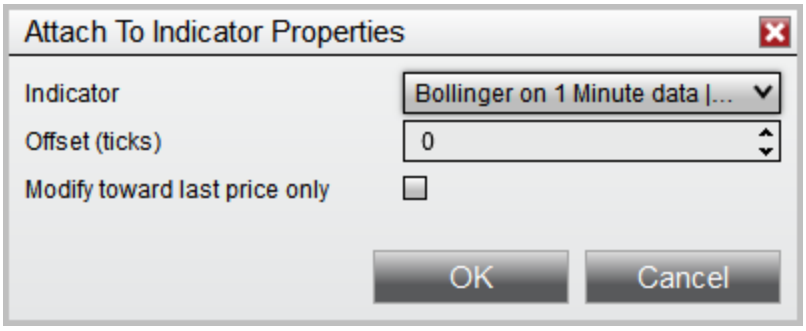
Configuring Attach to Indicator Properties

To setup the parameters to **Attach to Indicator**:

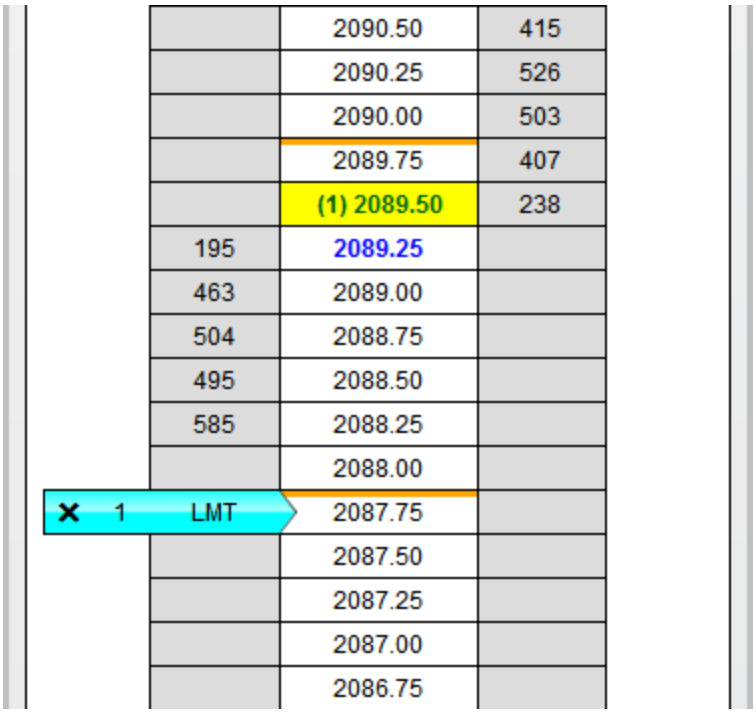
1. Right mouse click on a submitted order
2. Hover your mouse cursor over the order details
3. Navigate to the **Attach to Indicator** menu
4. Left mouse click on the **Attach to Indicator Properties** menu item

This will open an **Attach to Indicator Properties** window which will allow you to define the following properties:

Indicator	Selects the indicator plot* which is used for the indicator tracking
Offset (ticks)	The number of ticks (+/-) the order will follow the indicator value
Modify toward last price only	Enables / Disables the ability to hold the order price and only modify should the indicator change to a price closer to the last traded price. This prevents orders from modifying to price that would be worse than the previous value.



After you have configured your desired settings, pressing the **OK** button will automatically enabled **Attach to Indicator** on the order you configured which will immediately modify the order price if necessary.

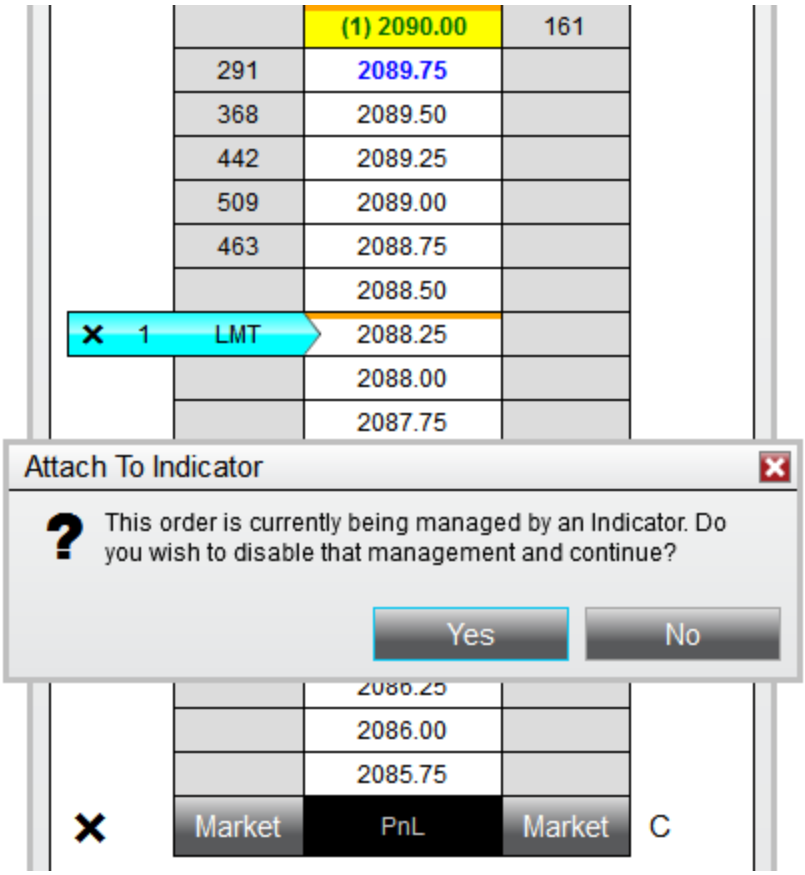


Note: Many indicators will have multiple plots. Please ensure you are selecting the correct plot and that the current values would be valid for the order you are attaching to the indicator to prevent unwanted fills or order rejections. For example, attaching a **Buy Limit** order to the **Upper Band** of a **Bollinger** indicator that is currently above the current market price will automatically modify that order to above the market price resulting in an immediate fill.

Disabling indicator tracking

Disabling Indicator Tracking

Orders that are attached to an indicator will be 100% managed by the **Attach to Indicator** feature. Should you attempt to manually modify an order, you will receive a prompt reminding you that this order is being managed by an Indicator. If you wish to manually modify the order, you can select "Yes" to on this prompt to disable the indicator management and allow you to change the order price if desired.



You can also disable **Attach to Indicator** through the right click menu on the order itself allowing you to re-configure your **Attach to Indicator Properties** if desired.

1. Right mouse click on a submitted order
2. Hover your mouse cursor over the order details
3. Navigate to the **Attach to Indicator** menu
4. Uncheck **Enabled** which will disable indicator tracking

As long as the order has not been filled/cancelled, you can always go back to this menu and re-check **enabled** to turn the **Attach to Indicator** feature back on.

10.23.2 Simulated Stop Orders

A Simulated Stop order (SS) is a conditional locally held (PC simulated) order type that allows you to execute either a market or a limit order once the market touches your order price and satisfies a user defined volume trigger. SS orders are very powerful and can be misused if not fully understood. Please take the time to review this section in it's entirety prior to using this order type.

Note: Simulated Stop (SS) orders are not supported in the NinjaTrader Direct Edition.

Understanding the benefits of the Simulated Stop

Benefits

- Hide your order from the market place
- Delay the triggering of a stop-market or stop-limit order (prevent having your position stopped out prematurely due to a quick drop and pop into your stop loss price level)
- Execute a limit order at an improved price from the stop price trigger (for example, you wish to trigger a buy limit order at 999 once the market price reaches 1000)

Understanding the risks of the Simulated Stop

Risks

- SS orders are held and simulated locally on your PC and are therefore subject to issues such as loss of internet connection and computer crashes
- SS orders require stable and reliable market data since they are simulated; if market data stops flowing the SS order stops simulating
- SS market order types can experience slippage during high volume periods and/or highly volatile markets

Warning: Since SS orders are held on your PC and submitted live when they trigger, it is possible that the order is rejected should you not have available margin to place this order. PLEASE BE AWARE OF YOUR ACCOUNT MARGIN LIMITATIONS.

Understanding the SS Volume Trigger

Volume Trigger

A SS order requires a Volume Trigger value to be set. This is the number of shares/contracts that represents a floor that once penetrated will trigger the SS order. SS orders trigger once the market price is trading at the SS order price and the Volume Trigger condition is breached. Volume Triggers for Stop Loss orders

are set as part of a Stop Strategy, Volume Triggers for all other stop orders are set via the properties dialog window of any order entry window.

SS orders are set to Initialized state (see Order State Definitions) and are color coded yellow in all of the NinjaTrader order display windows. Once triggered, either a limit order or a market order is submitted.

> **Note**: Simulated Stop Volume Triggers are **not** supported for Forex instruments. NinjaTrader will ignore this value when set for Forex instruments and instead will submit the stop order when price trades at the Simulated Stop price level.

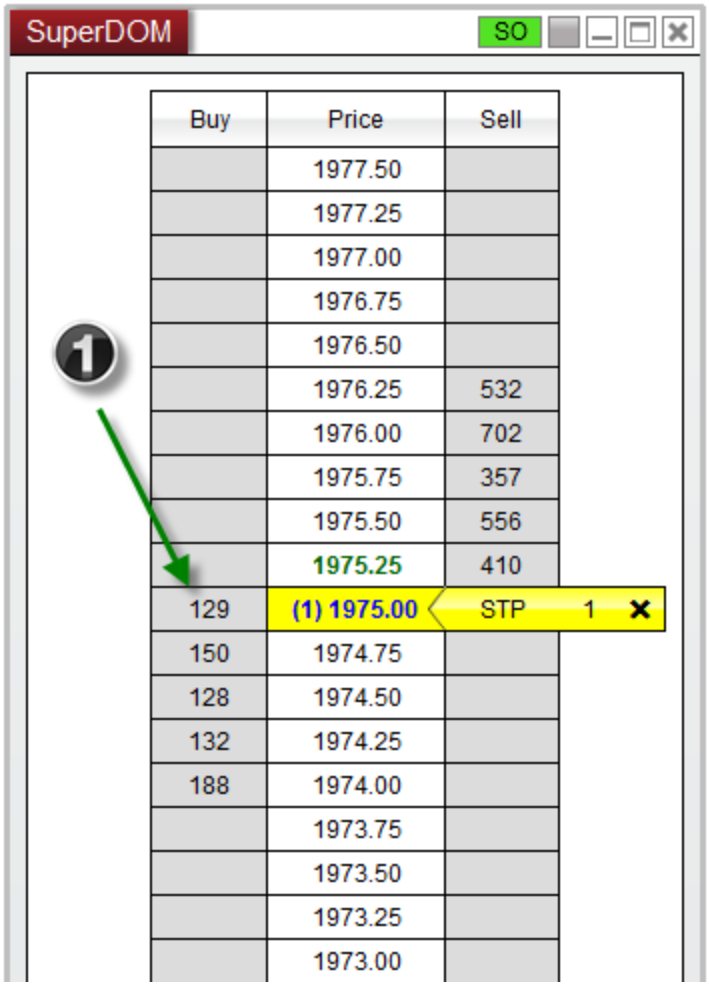
Simulated Stop examples

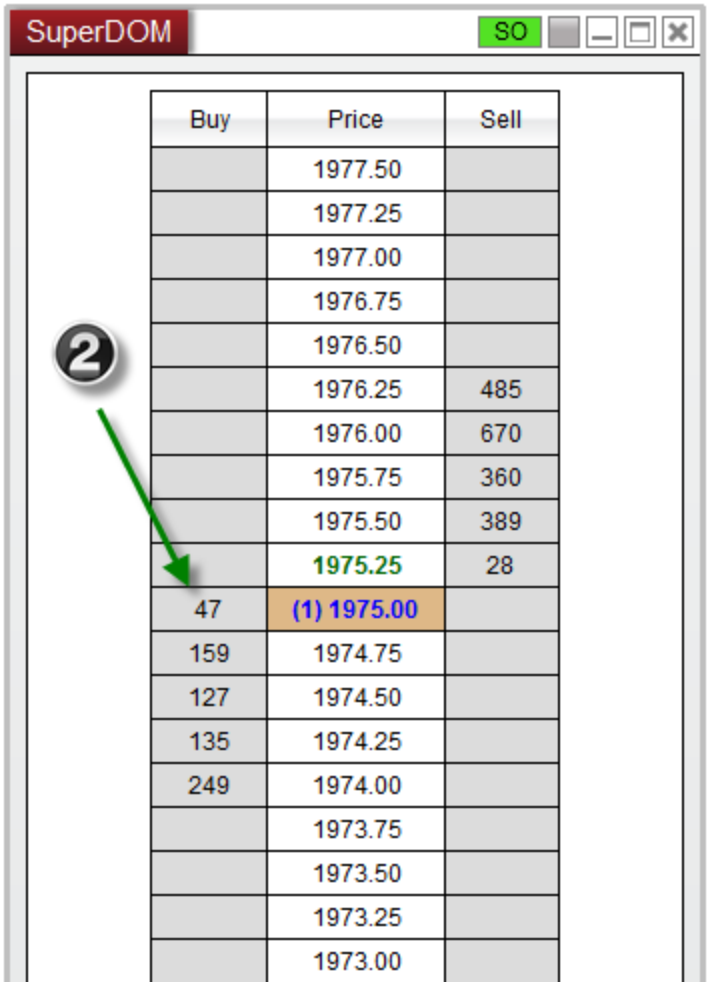
Sell Stop-Market Example

Order Type - Sell Stop-Market
Stop Price - 1975.00
Volume Trigger - 100

1. In the example below, even though the **Bid price** has reached our Sell Stop Price of 1975.00, the order has not executed because the **Bid volume** is still greater than our **Volume Trigger** of 100.

2. As soon as the **Bid volume** falls below the **Volume Trigger** at the Stop Price level, then a market order is issued, which takes us into position at the market price.



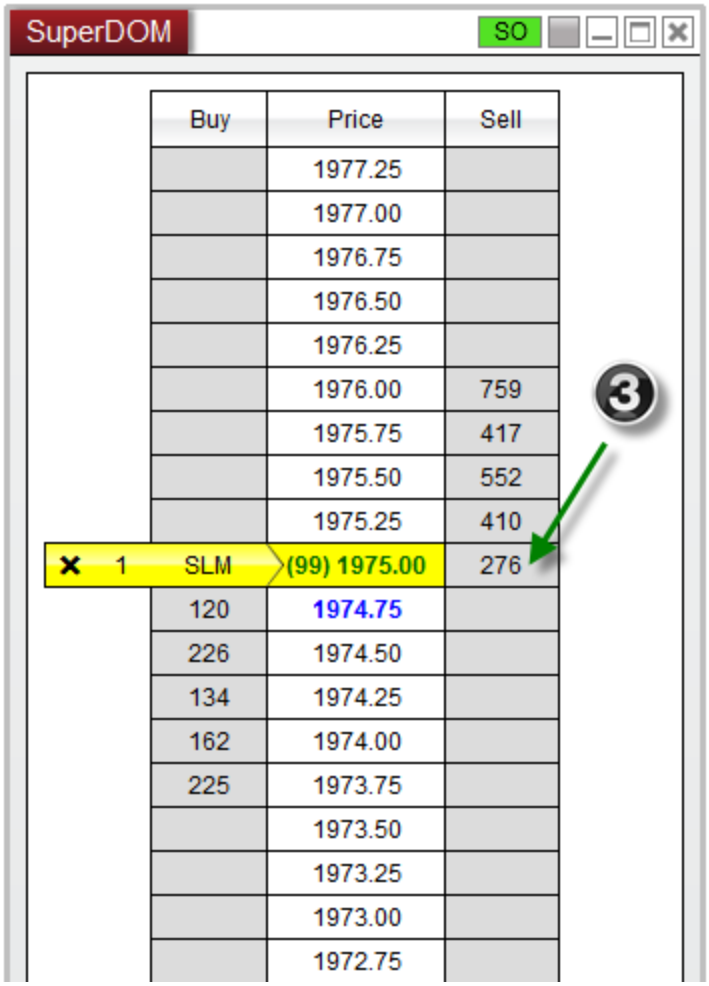


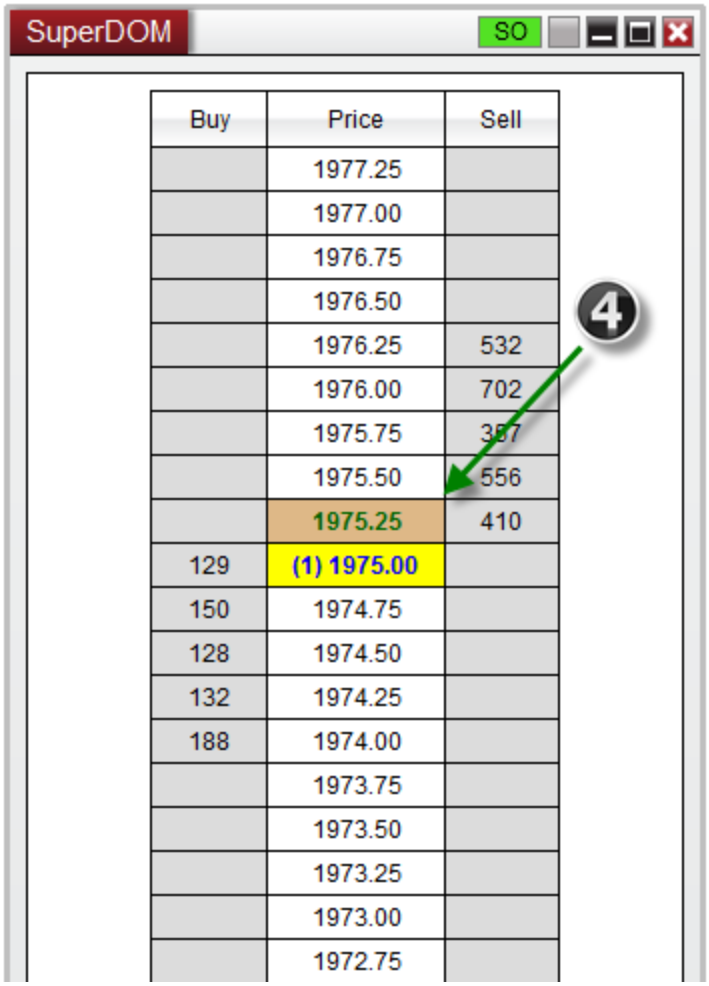
Note: If this was a Buy order, the **Ask volume** would be monitored.

Buy Stop-Limit Example

Order Type - Buy Stop-Limit
Stop Price - 1975.00
Limit Price - 1975.25 (1 tick offset)
Volume Trigger - 100

3. In this example, we again see that the **Ask price** has reached our Buy Stop Price of 1975.00, however the order has not executed because the **Ask volume** is still greater than our **Volume Trigger** of 100
4. As soon as the **Ask volume** falls below the **Volume Trigger** at our Stop Price, a Limit order is immediately submitted at the price of 1975.25, which takes us into position at our limit price or better.





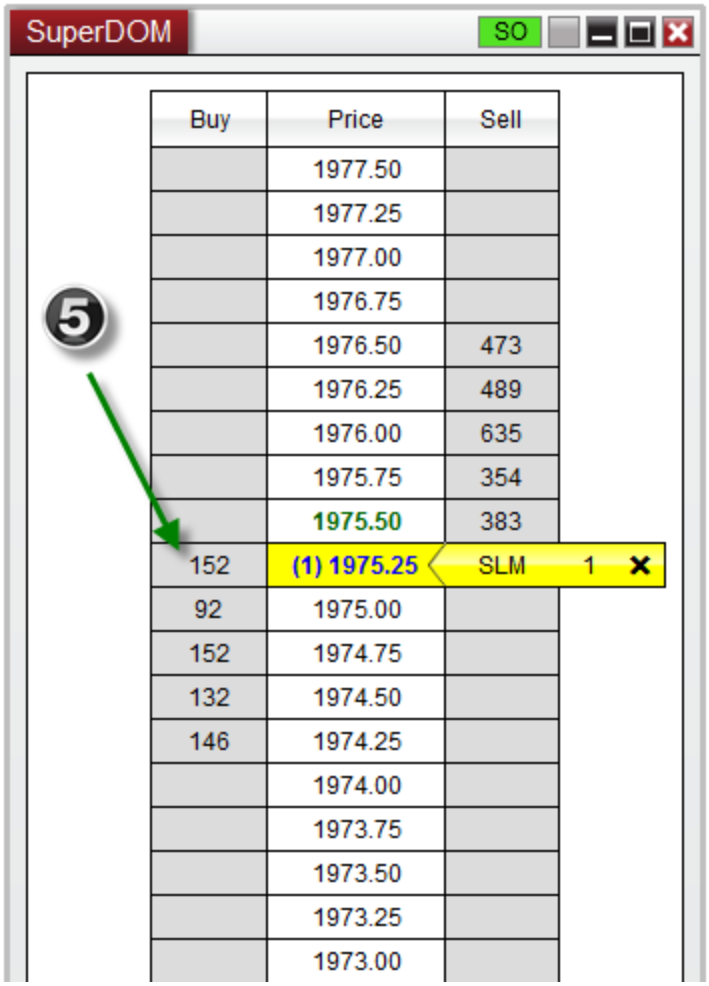
Note: If this was a Sell order, the **Bid volume** would be monitored.

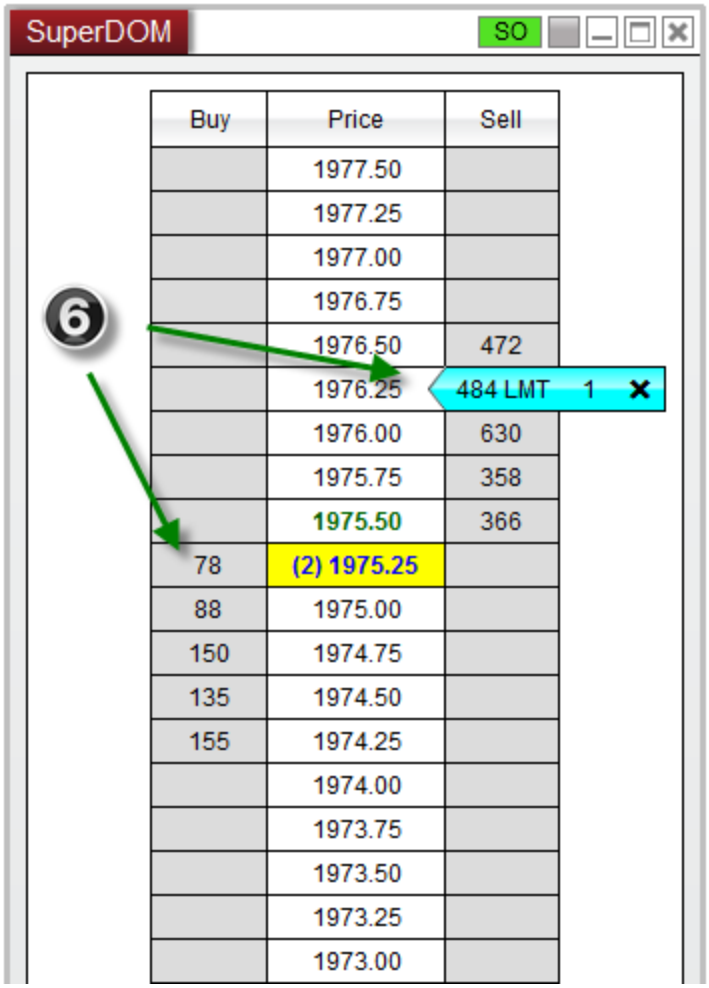
Sell Stop with Improved Limit Price Example

Order Type - Sell Stop-Limit
Stop Price - 1975.25
Limit Price - 1976.25 (- 4 tick offset)
Volume Trigger - 100

5. This example will once again trigger as the **Bid price** trades at our Stop Price of 1975.25 and the **Bid volume** is less than 100 contracts.
6. The interesting thing about this set up is that what we are doing is triggering a limit order at a higher price in order to try and get a better fill. This order strategy is not possible with standard order types and can only be done using NinjaTrader SS

technology. Once triggered, a limit order is submitted to sell at a price of 1976.25, 4 tics above our Stop Price.





Note: If this was a Buy order, the **Ask volume** would be monitored.

Understanding when to avoid using Simulated Stop orders

Avoid SS Orders

- During high volume and trade rate periods such as the first five minutes of trading
- During major economic events that can substantially affect volatility
- Markets that consistently trade with a large spread between the ask and bid price
- Markets that trade where the ask or bid price can consistently change by more than one tick

10.23.3 Order State Definitions

The table below describes the various order **States** your orders can be in as well as the color that represents this state in NinjaTrader. The colors can be seen when submitting, modifying or cancelling orders in the Order Entry windows as well as the Orders tab of the Control Center.

NinjaTrader Order State Definitions

Order State	Definition	Color Code
Initialized	Order information validated on local PC	Yellow
Submitted	Order submitted to the connectivity provider	Orange
Accepted	Order confirmation received by broker	Order type color
Working	Order confirmation received by exchange	Order type color
Suspended	Order held by broker and ready to be submitted when triggered	Order type color
Change submitted	Order modification submitted to the connectivity provider	Orange
Cancel pending	Order cancellation submitted to the connectivity provider/exchange	Orange
Cancelled	Order cancellation confirmed cancelled by exchange	No color
Rejected	Order rejected locally, by connectivity provider	No color

Order State	Definition	Color Code
	or exchange	
Partially filled	Order partially filled	No color
Filled	Order completely filled	No color
Trigger pending	Order held locally on PC and ready to be submitted to connectivity provider	Yellow

Notes: For orders in a *Accepted* or *Working* **Order State**, the order color will be reflective of the order type. For example, a Limit order would by Cyan (by default) when Working.
FXCM and Rithmic do not support GTD orders. These order will be submitted as GTC and the GTD functionality will be simulated on NinjaTrader. NinjaTrader must be connected for it to function

10.23.4 FIFO Optimization

All of the NinjaTrader order entry interfaces preserve FIFO (First In First Out) status with the exchanges when possible.

Why is FIFO important?

FIFO is important since getting your orders filled is dependant on a FIFO algorithm which basically means orders submitted ahead of yours at your order's price level will get filled ahead of you. Think of it like a long line at the grocery store. You are checked out only when those in line ahead of you have been checked out. NinjaTrader preserves your place in line when possible giving you the best possible advantage of getting your orders filled.

How does NinjaTrader preserve FIFO?

All NinjaTrader order entry interfaces simplify the visualization of orders. Let's say that you have a buy limit order for 1 contract, and then want to modify this order

from 1 contract to 2 contracts. Most other programs will simply change this order directly, but behind the scenes (at the broker's order servers) what is really happening is that the original order is cancelled (removed from the line) and then a new order for 2 contracts is submitted which then goes to the back of the line putting you at a disadvantage. Imagine waiting in the grocery store check out line for ten minutes. You forgot to get some bread, you leave the line in order to get the bread that you wanted (changing your order) and upon your return to the check out line, you have to start at the back of the line and wait all over again! With NinjaTrader, when you modify the order from 1 contracts to 2 contracts, an additional order for 1 contract is submitted. Now you have the original order for 1 contract waiting in the middle of the line somewhere and the new order for 1 contract at the back of the line. Your original order is not penalized and you maintain your position in line. The opposite is true for decreasing an order size. Although there are two working orders NinjaTrader consolidates the display so it looks as if there is only one order working. You decide to decrease the order size from 2 contracts back to 1. NinjaTrader will modify the newest orders first and the oldest orders last in order to preserve your FIFO status. Following our example, the second order that was placed would be cancelled and you would be left with the original order for 1 contract with its maintained position in the order queue.

Exceptions to FIFO

NinjaTrader will attempt to use **FIFO** when possible, but there are a few scenarios where this would not be possible.

Non-Aggregated Order Displays

Most of NinjaTrader's order interfaces (**Basic Entry**, **SuperDOM**, **Chart Trader**, etc) will have an aggregated order display to consolidate orders submitted/modified at the same price level and have also been designed for **FIFO** optimization. However any order feature that uses a non-aggregated order display, such as the Orders Tab of the **Control Center** or **Account Data** window, modifications will not be able to maintain **FIFO** optimization and will modify the order directly via a cancel/replace operation. If **FIFO** is important for your style of trading, you will want to consider making modifications from an aggregated order display feature, rather than directly from grid of the **Orders Tab.**

Quantity Modification for Stocks

NinjaTrader's features which support stocks will have an option to set how order modifications are handled when trading stocks. You will find this setting by right clicking on the Order Entry feature, selecting **Properties,** and will be called **Quantify modification for stocks** with the following options:

Increase quantity of preexisting order	When modifying quantity or price on an aggregated order NinjaTrader will modify the order with the least time in the market via a cancel/replace operation.
Submit new order for additional quantity	Orders will use **FIFO** optimization as normal

If the account you are trading with your stock brokerage charges you commission per individual share, you will want to consider using "**Increase quantity of preexisting order**" setting. While you will lose the **FIFO** optimization, order modifications will not incur an additional commission charge as a result of the additional orders that would otherwise be placed in response to your modification request.

Futures, Forex, and CFD's will always use **FIFO** optimization on aggregated orders.

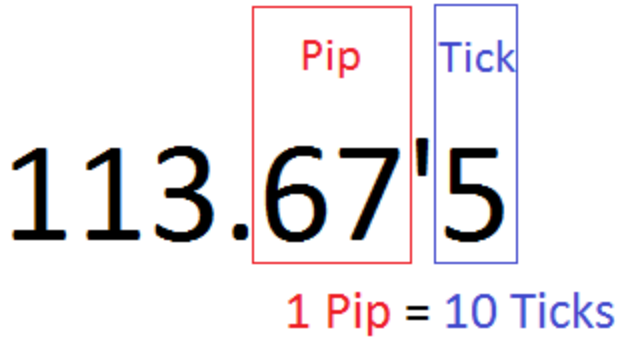
10.23.5 Working With Forex

NinjaTrader supports trading and viewing market data for spot forex pairs, in addition to other supported instrument types. Due to the unique nature of forex markets, there are a number of features throughout the platform tailored specifically to these instruments, and a few considerations to keep in mind when working with forex in NinjaTrader.

Pips Calculation Mode

Pips vs. Ticks

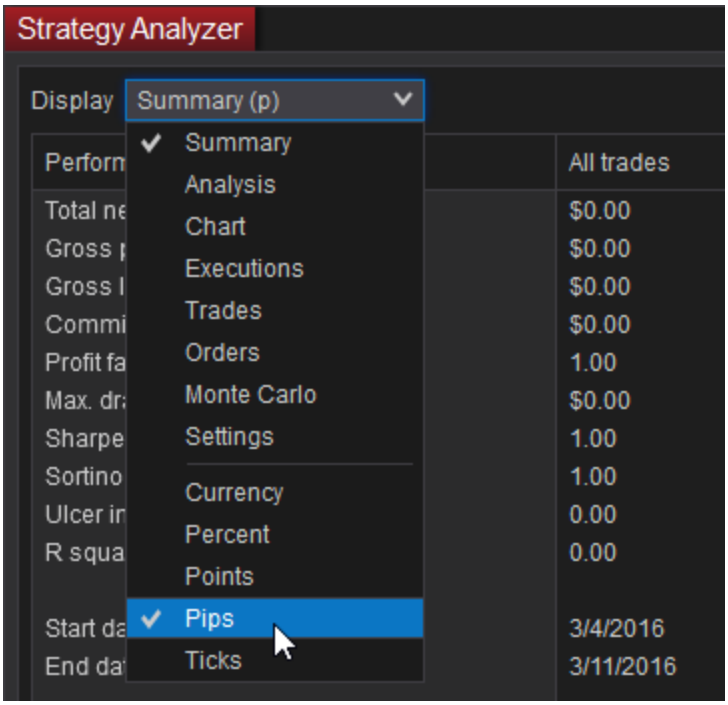
The "Pips" Calculation Mode can be used to calculate PnL and performance metrics throughout the platform. This mode allows you to tailor performance reporting specifically to your forex trades. Similar to the "Ticks" mode, "Pips" takes the lowest granularity of price movement for a forex instrument (called a tick in NinjaTrader), then divides it by 10 to arrive at the pip value for the instrument. For example, when viewing a USD/JPY quote of 113.67'5, the "7" would be the pip value, and the "5" would be the tick. Using the Pips Calculation Mode, the number of ticks in profit (the "5" in the example) will be divided by 10 to arrive at the number of pips of profit or loss.



Setting the Pips Calculation Mode

The Pips calculation mode can be used in realized/unrealized PnL fields in trading windows (Chart Trader, SuperDOM, Basic Entry, etc.), the Trade Performance window, and the Strategy Analyzer. In Trading Windows, the calculation mode can be changed by left-clicking within the PnL field, or by opening the window's Properties dialogue. For more information, see the relevant pages for each trading window.

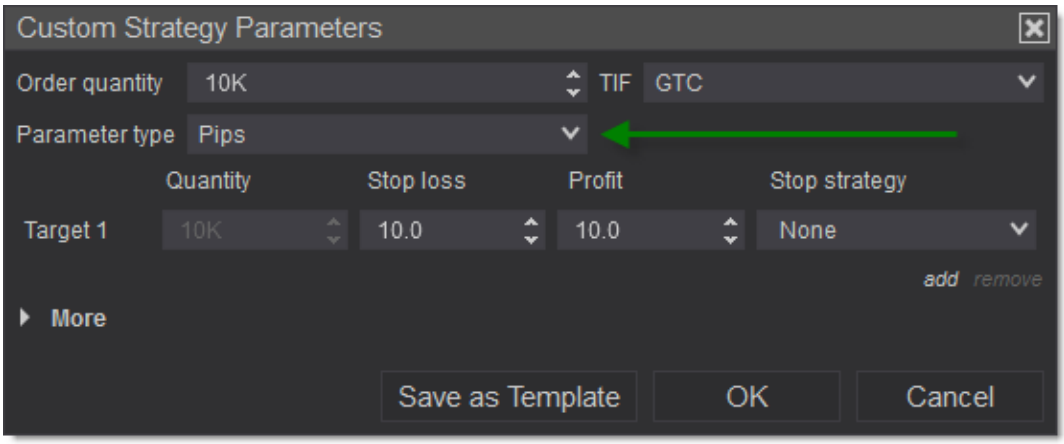
In the Trade Performance window and Strategy Analyzer, the calculation mode can be changed via the **Display** dropdown menu, which affects all relevant statistics.



Pips in ATM Strategies

ATM Strategy Parameters

The **Parameter Type** field within the ATM Strategy Parameters window can be changed to "Pips" to affect the way that stop loss and profit target prices are set by an ATM strategy. Just like the Pips PnL calculation mode, the Pips parameter type is based on a multiplicative factor of the Ticks parameter type (1 Pip = 10 Ticks). For example, rather than entering 200 ticks for your profit target (200 ticks = 20 pips), you can simply specify 20 pips.

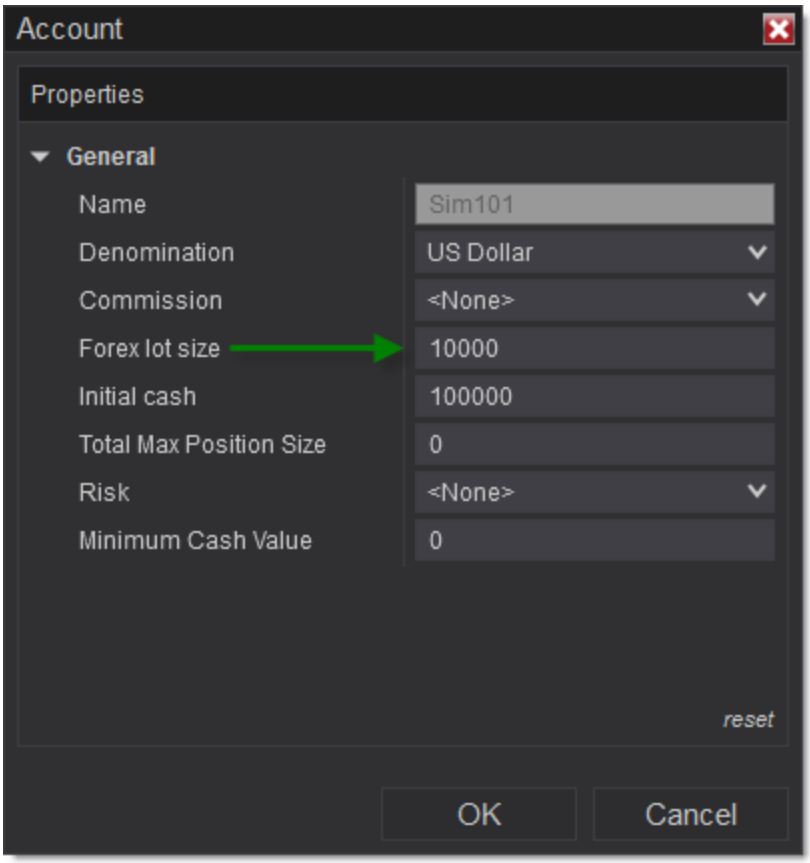


Note: If your forex data provider supports tenth-pip quotes, then you can also use the Ticks parameter type to set ATM orders with a sub-pip granularity.

Forex Lot Sizes

Setting Your FX Lot Size

A "Forex Lot Size" property can be set for accounts shown in the Accounts tab of the Control Center. This setting affects the default position size populated in trading windows when a forex instrument is selected. To access this property, first select the Accounts tab in the Control Center. Next, right click on the account you wish to edit, and select the **Edit Account** menu item. In the window that appears, set the Forex Lot Size property to your desired value. You can enter any amount here, whether or not it corresponds to a standard position size (Lot, Mini-Lot, Micro-Lot). For example, you could enter "102000" to automatically use a position size equal to one standard lot (100,000) plus two micro lots (2,000).



Notes:
- The Forex Lot Size property does not prevent you from entering or selecting different position sizes in trading windows, but only controls what is populated in the Quantity field by default.

Forex-Specific Trading Windows

FX Pro
The FX Pro window is laid out similarly to the Basic Entry window, with a few enhancements and modifications tailored specifically to forex instruments. For more information on using this window, see the FX Pro page.

FX Board
The FX Board is a unique forex trading window featuring a grid of two-sided tiles updated in real time, offering market data, spread info, and order management functionality for multiple pairs at once. FX Pro and FX Board windows can be linked together via Instrument Linking. When linked, you can simply click any tile in the FX Board, and the corresponding instrument will be selected in a linked FX Pro window. For more information on using this window, see the FX Board page.

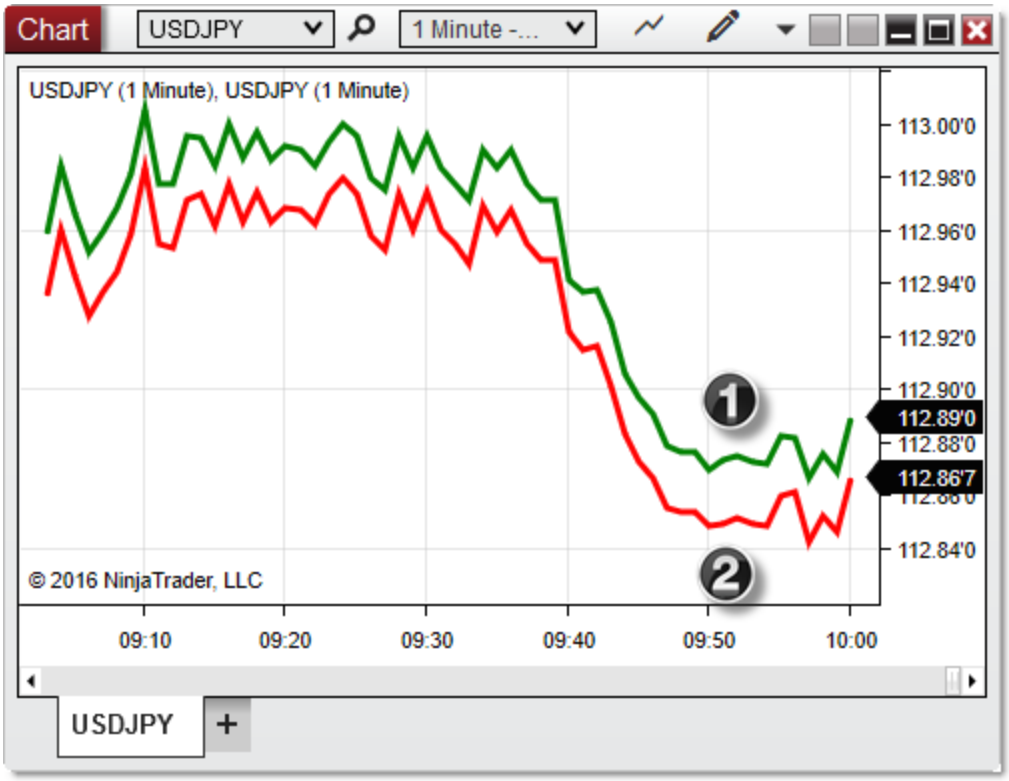
Other Windows
Forex instruments can be traded in other windows, as well, and are not limited to the two mentioned above. Forex-specific windows can also be linked to others via Instrument Linking. Other windows, such as Chart Trader or the Market Analyzer, do not include forex-specific features, but are capable of handling FX instruments just like any others.

How Bars Are Built and Orders Filled

Building Bars with "Last Price" Data Type
Forex price quotes do not use the concept of "Last Price" the same as other markets; only Bid and Ask quotes are available. Thus, when building bars using the default "Last" price type, the Bid price will be used instead. Using this price type, all bars on a chart will be built using Bid price updates, but you can choose to use the Ask price instead, if you wish. To change the price type used, first open the Data Series window on a chart, then toggle the value in the "Price Based On" field to your desired type.

Realtime Order Fills vs. Backtesting

Due to the absence of a last traded price quote in forex, all Buy orders in a live market are filled at the Ask price, and all Sell orders are filled at the Bid. However, when backtesting NinjaScript strategies, all simulated order fills will occur at the Bid price, regardless of whether they were Buy or Sells orders



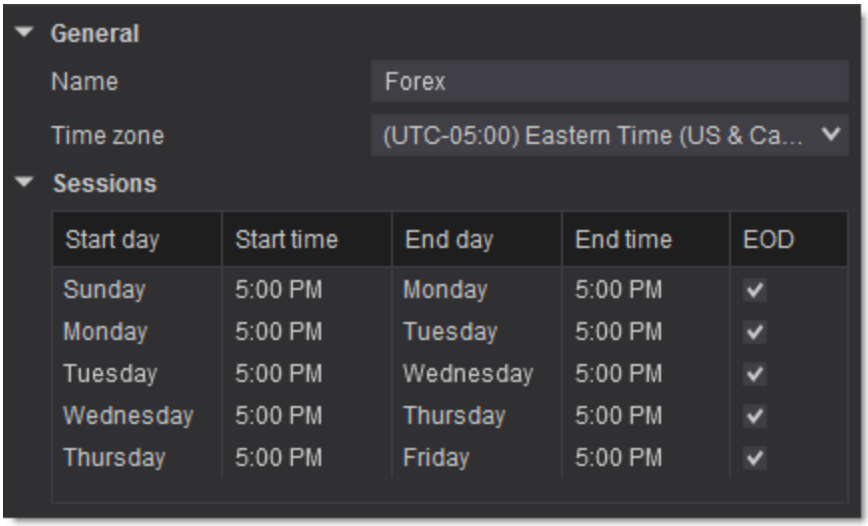
1. Ask Price: All realtime Buy orders are filled at the Ask
2. Bid Price: All realtime Sell orders and all backtest Buys and Sells are filled at the Bid

Note: In backtesting, a slippage value can be set to recreate the impact of the Bid/Ask spread on trade profit and loss. NinjaScript developers can calculate the spread in strategy logic, then dynamically set the Slippage property before entering orders. For non-programmers, an estimated slippage value can be applied to all trades via the Backtest/Optimization Properties section in the Strategy Analyzer.

Forex Trading Hours

Forex Trading Hours Template

All forex instruments are configured to use the pre-defined "Forex" Trading Hours template, which runs 24 hours per day from 5:00pm EST on Sunday to 5:00pm EST on Friday, with an End-of-Day session break at 5:00pm each day. This covers the full range of forex trading throughout the week, but other Trading Hours templates can be applied to restrict the data on your charts to be in line with any local market timing on which you may wish to focus. For more information, see the Trading Hours page.



Start day	Start time	End day	End time	EOD
Sunday	5:00 PM	Monday	5:00 PM	✓
Monday	5:00 PM	Tuesday	5:00 PM	✓
Tuesday	5:00 PM	Wednesday	5:00 PM	✓
Wednesday	5:00 PM	Thursday	5:00 PM	✓
Thursday	5:00 PM	Friday	5:00 PM	✓

Calculating Pip Value

How to Calculate the Pip Value for a Forex Pair

Multiplying the pip size of your currency pair by the lot size of your order will provide you the pip value. This will be in the quote/counter currency of the forex pair. The quote/counter currency is the second currency in the pair.

Example in USD for an USD Quote/Counter Currency

In the following example we will do this for a 10,000 lot on the EURUSD. The quote/counter currency is USD and the EURUSD's point size is 0.0001.
10,000 x 0.0001 = 1
This indicates that 1 pip would be $1 USD.

Example in USD for a GBP Base Currency

In the following example we will use the EURGBP. Let's say our account is in USD and we want to convert the pip value to USD. Again we will say we are trading a 10,000 lot size and the pip value for the EURGBP is 0.0001.

10,000 x 0.0001 = 1

This indicates that 1 pip would be £1 GBP. We would then multiply this by what the GBPUSD is trading at.

In this example the GBPUSD is trading at $1.26 (rounded).

1 x 1.26 = 1.26

This indicates that 1 pip would be $1.26 USD.

Note: If the conversion rate is not available the PnL information will be in the counter/quote currency of the pair. This would create a discrepancy in your

Trade Performance.

10.23.6 Where do your orders reside?

NinjaTrader

Orders in a state "Accepted" or "Working" or "Suspended" are at the brokerage or exchange. If the exchange does not support a specific order type, the order will be active on the NinjaTraders servers.
Most OCO (One Cancels Other) functionality is simulated on your local PC.

Continuum/CQG

Orders in a state "Accepted" or "Working" are at the brokerage or exchange. If the exchange does not support a specific order type, the order will be active on the Continuum servers.
Most OCO (One Cancels Other) functionality is natively supported on their servers, but please see the disclaimer section here for more information:
https://support.ninjatrader.com/s/article/Connecting-To-Your-NinjaTrader-Continuum-Feed

FOREX.com/City Index

Orders in a state "Accepted" or "Working" are on FOREX.com servers.
OCO (One Cancels Other) functionality is simulated on your local PC.

FXCM

Orders in a state "Accepted" or "Working" are on FXCM servers.
OCO (One Cancels Other) functionality is natively supported on their servers.

Interactive Brokers

Orders in a state "Accepted" or "Working" are at the brokerage or exchange. If the exchange does not support a specific order type, the order will be active on the Interactive Broker servers.
OCO (One Cancels Other) functionality is natively supported on their servers.

Rithmic

Orders in a state "Accepted" or "Working" are at the brokerage or exchange. If the exchange does not support a specific order type, the order will be active on the Rithmic servers.
OCO (One Cancels Other) functionality is simulated on your local PC.

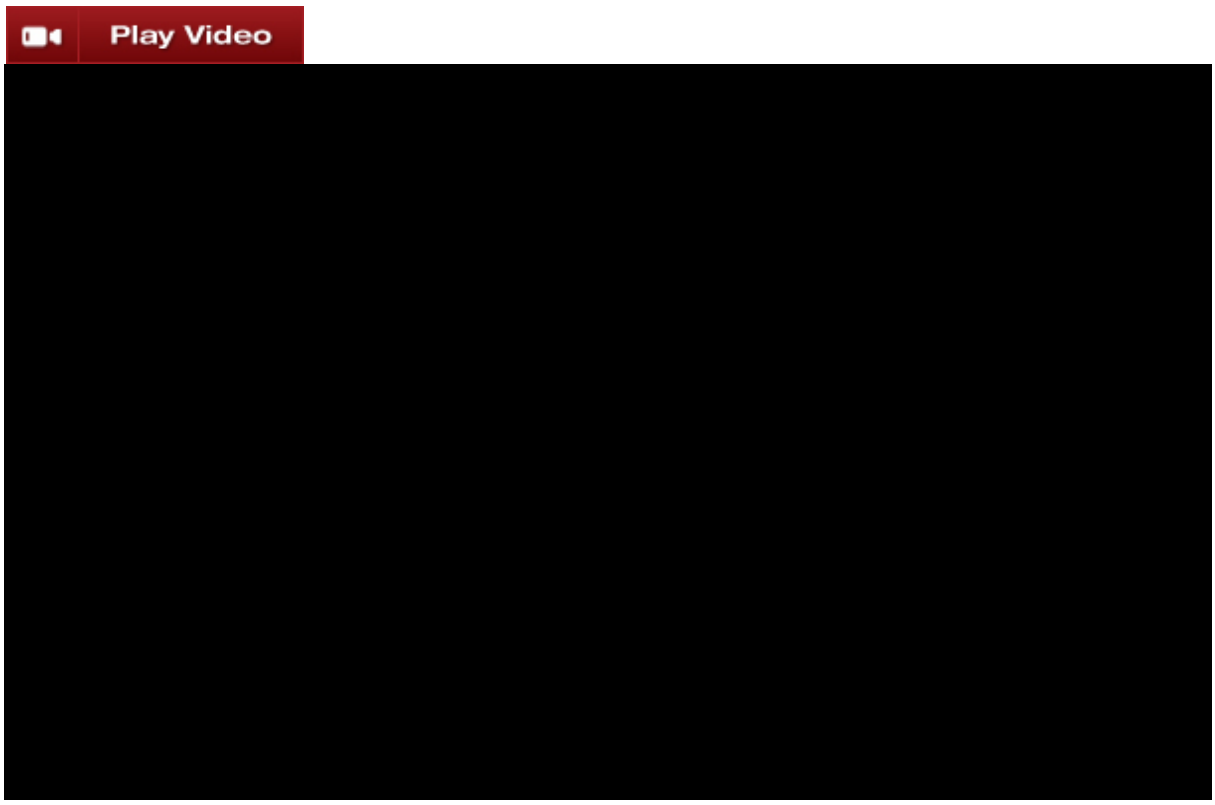
TD Ameritrade

Orders in a state "Accepted" or "Working" are at the brokerage or exchange. If the exchange does not support a specific order type, the order will be active on the TD Ameritrade servers.
OCO (One Cancels Other) functionality is natively supported on their servers.

Note: Please note that MIT orders and Simulated Orders, which are in TriggerPending state, could not be recovered by NinjaTrader, if the provider does not provide native support. The order state after connection recovery would then be 'Unknown'.

10.23.7 Trade Controls

Trade Controls are located in various Order Entry windows available throughout the product.



Trade Controls

Many of the NinjaTrader **Order Entry** features will have a number of shared controls design to aid you in setting various order parameters such as **Quantity**, **Price**, **TIF**, or used to display account/instrument **Position** information. These controls are designed to behave in the same manner no matter which order entry feature you're using.

10.23.7.1 Closing a Position or ATM Strategy Position

Closing a position or ATM Strategy with the Close button

In all NinjaTrader order entry windows there are **CLOSE** action buttons which will close a position or an ATM Strategy depending on which mouse button is pressed (the left mouse button will close the current position and cancel any working orders associated with the instrument/account combination and the middle mouse button (scroll wheel) will close the selected active **ATM Strategy** position only).

When a position or **ATM Strategy** position is closed, NinjaTrader goes through the following process: (Assume we are long on the S&P E-mini contract at an entry price of 1000.)

1. Identify the active account and instrument
2. Send cancellation requests for any open orders on the active instrument/account
3. Wait up to five seconds for cancellation confirmations from the connected brokerage
4. After receiving confirmation for all cancellation requests, send an offset order to close any open position (For example, a Buy order for 3 contracts would be sent to close an open short position of 3 contracts)

The exception are currencies (FOREX) where all working orders are cancelled and then a market order is submitted to close the position. (Does not apply to currencies with Interactive Brokers where the previously described process is used.)

Why not just submit a market order?

A lot of traders consume near 100% of their available margin. For example, you may have 1 contract position with one stop and target that consumes 90% of your margin. If you close a position by submitting a market order, the market order will get rejected due to insufficient funds and therefore cause grief if getting out of the position is critical.

The NinjaTrader approach offers several benefits:

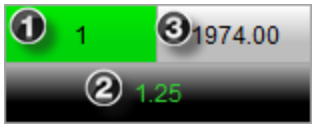
- Modifying existing limit orders avoids the potential problem of breaching your account margin limitations when closing a position
- With some brokers, modifying existing orders is more efficient than submitting new orders by up to 500 milliseconds
- Cancellation of non essential orders unlocks available margin that could potentially get consumed if a market order is required to close out any remaining contracts that are not covered by existing limit orders

This approach essentially guarantees the most efficient way to exit a position.

10.23.7.2 Position Display

The current selected account and instrument's position will be reflected directly on the **Order Entry** window with the following information:

1. Position Quantity and Direction Display
2. PnL (Profit and Loss) Display
3. Average Entry Price Display



The image above shows that we are in a 1 **Long position**, with an **Average Entry Price** of 1974.00, and that our current **open PnL** is 1.25 points.

Tip: If you are trading using multiple **ATM Strategies**, it is possible to reconfigure the position display to only display the position of the current selected **ATM Strategy**. Please see our Help Guide section on ATM Strategy Selection mode for more information.

▼ Understanding Position Quantity and Direction Display

Position Information

The current number of contracts in position will be displayed as a number in the position display. The direction of the position will be also represented in a highly visual manner:

- When not in a position, the text display will say "Flat" without a color
- Long positions will be reflected by a Green background color
- Short positions will be reelected by a Red background color

▼ Understanding Average Entry Price Display

Average Entry Price

When in a position, the **Average Entry Price Display** will show you the current price which is being used to calculate your open PnL. As you scale in and scale out of position at different prices, your **Average Entry Price** will be recalculated to reflect the new average price. The way this is calculated is set under the Trading category of NinjaTrader's general options menu.

> **Note**: The **SuperDOM's** average entry price will be displayed directly on the Price ladder display rather than a text field.

Understanding PnL Display

Profit and Loss

The **PnL Display** can be easily switched between *Currency, Percent, Points, Pips, Ticks* and *None* (hides your PnL). There are two ways to configure the **PnL Display**:

1. Single left clicking on the **PnL Display** itself will cycle between each display mode (with the exception of "none")
2. Right clicking on the order entry feature and selecting **Properties** will allow you to select the **PnL display unit** property directly

You can also configure the **PnL Display** to show you your selected account's **Realized PnL** for the current trading session when not in a position. To enable this feature, right click on the order entry feature, select **Properties**, and check **Show realized PnL when Flat.**

> **Tip**: **Show realized PnL when Flat** will show you your overall account PnL and not the selected instrument's PnL. If you'd like to see each individual instruments PnL, you can use the **Market Analyzer's** *Realized profit loss* and *Unrealized profit loss* Columns to display this information per instrument.

> **Note**: When viewed in Ticks, Points, or Pips the PnL will be calculated according to the average entry price for one unit. When viewed in Currency, PnL will be multiplied by the number of units in a position. When viewed in Percent, PnL will be percentage change of the Average Entry Price. This is

due to the fact that the Ticks, Points, and Pips display modes are intended to provide easily comparable measures of raw performance between trades, by eliminating the position size from the equation.

10.23.7.3 Price Selector

NinjaTrader order entry feature which have the ability to place custom orders for **Stop-Market**, **Stop-Limit**, **Limit** and **MIT** orders such as the Order Ticket, Basic Entry, FX Pro, and FX Board use a standard **Price Selector** which is used to specify the exact price used for these types of orders.

Setting and Adjusting Price

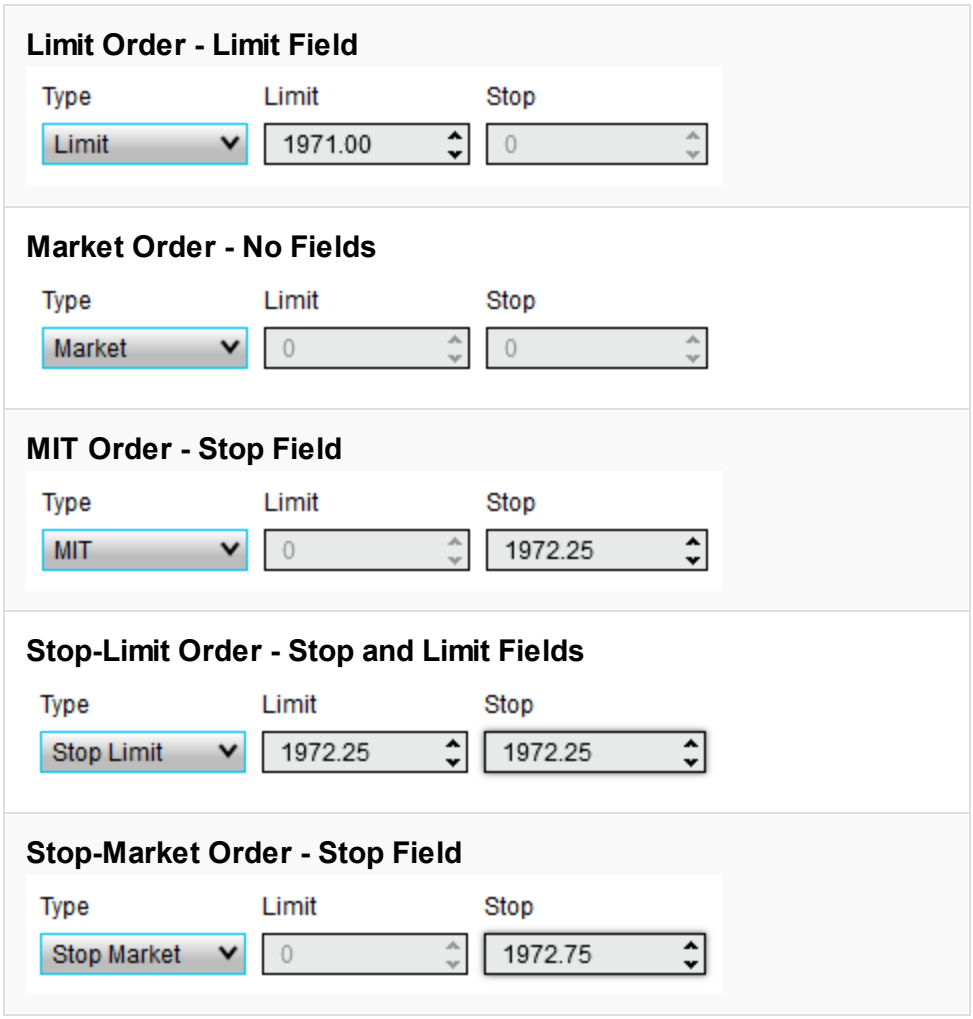
The price selector allows you type directly into the editor to specify a price, however you can also use a few shortcuts to obtain current market prices, as well as make quick adjustments to the selected price using your mouse. The table below shows the various shortcuts that can be used with the **Price Selector**:

Middle Mouse Click	Sets the current Last Price
Ctrl + Middle Mouse Click	Sets the current Ask Price
Alt + Middle Mouse Click	Sets the current Bid Price
Middle Scroll Up/Down	Adjusts the current price 1 tick
Ctrl + Middle Scroll Up/Down	Adjusts the current price 10 ticks

Order Types and Price Fields

Upon selecting an order type, the relevant fields specific to that type of order will be enabled to allow you to edit the order price before submitting the order. If a field is not relevant to an

order type, it will be disabled. Note that a **Stop-Limit** order has both fields enabled which implies that both fields must have a value in order to place this type of order.



10.23.7.4 Quantity Selector

The **Quantity Selector** is a standard control available from all order entry features which allows you to select the number of contracts that are prepared for an custom order.

Default Order Quantities

Minimum Quantity Size

The **Quantity Selector** is smart in that it will automatically fill in the minimum quantity value depending on the type of instrument that is selected. This is particularly useful when switching from one instrument type to another.

The table below will show the minimum quantity for each instrument type:

Instrument Type	Default Minimum Quantity
Future	1
Stock	100
CFD	1
Option	1
Forex	Forex lot size - 100K (Full), 10K (Mini), or 1K (Micro)

Note: Forex lot sizes are automatically determined by your Forex brokerage account connection. For Simulation account Forex lot size, see "*Managing simulation accounts"* section of the global Trading options

Increasing or Decreasing Quantity

Adjusting Quantity

The **Quantity Selector** allows you to type directly in the quantity field to specify an exact quantity with your keyboard.

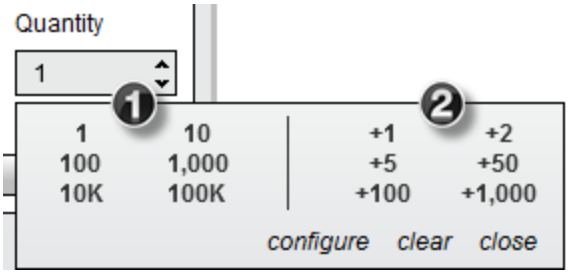
You can also control the quantity using the up/down arrows next to the quantity selector, or by using the scroll wheel on your mouse. These methods will change the quantity depending on the instrument type's minimum values described in the "*Default Order Quantities"* section above.

For example with a Stock selected, simply scrolling up with your mouse will change a quantity of 100 to 200. Holding the CTRL key on your keyboard and modifying the order quantity will increase or decreasing the value by 10. This means if you were to hold the CTRL key while scrolling on the **Quantity Selector** will increase Stock quantity from 100 to 1,100.

Preset Quantity Pad

Using Preset Quantities

Middle mouse clicking on the **Quantify Selector** will display a **Preset Quantity Pad** which will allow you to optionally predefine the number of contracts used as the quantity.



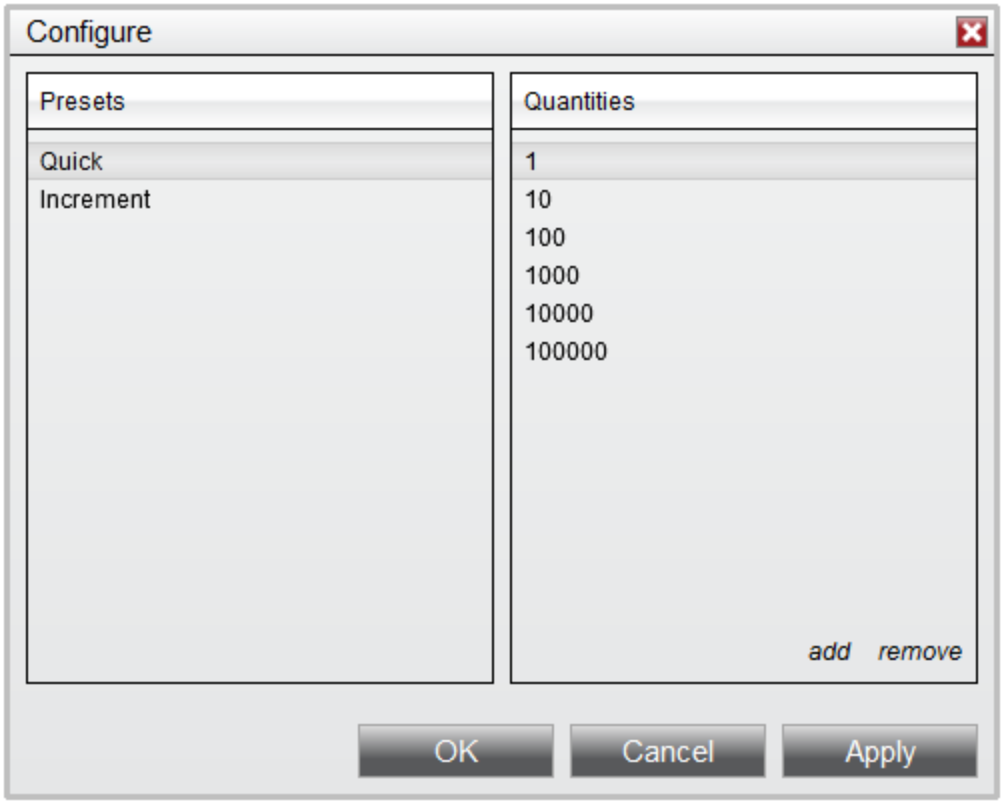
1. Quick	Sets the order quantity used to an exact pre-defined quantity
2. Increment	Increases the current order quantity value by pre-defined value

For example if your current quantity was set to a value of 1, and you wanted to quickly set the quantity to 10, you would simply select **10** from the left side of the **Preset Quantity Pad.** If you wanted to *increase* the current quantity by 2, you would select the **+2** from the right side of the **Preset Quantity Pad.** Doing so will increase the current value from 1 to 3. Selecting **+2** again would then change the quantity from 3 to 5.

Selecting the ***"clear"*** button will reset the current order quantity to the instrument's minimum order quantity size.

Customizing Preset Quantities

You can customize the preset quantity values that are displayed in these fields by selecting the "***configure"*** button from the **Preset Quantity Pad**.



Adding a Custom Preset

1. Select the desired Preset (Quick or Increment) from the left side **Presets** panel
2. Press the ***"add"*** button in the right panel
3. Set the desired Quantity value
4. Press **OK**

Removing a Custom Preset

1. Select the desired Quantity/Increment value from the right side **Quantities** panel
2. Select the ***"remove"*** button

Note: Order quantities will always be organized from low to high values

10.23.7.5 TIF Selector

The **TIF Selector** is a standard control available from all order entry features which allows you to set the **TIF** (Time-In-Force) to be submitted with a custom order. The selected **TIF** parameter is sent to your broker on order submission and will instruct how long you would like the order to be active before it is cancelled.

Note: An order's **TIF** is not managed by the NinjaTrader application and any cancellations are managed by the brokerage system.

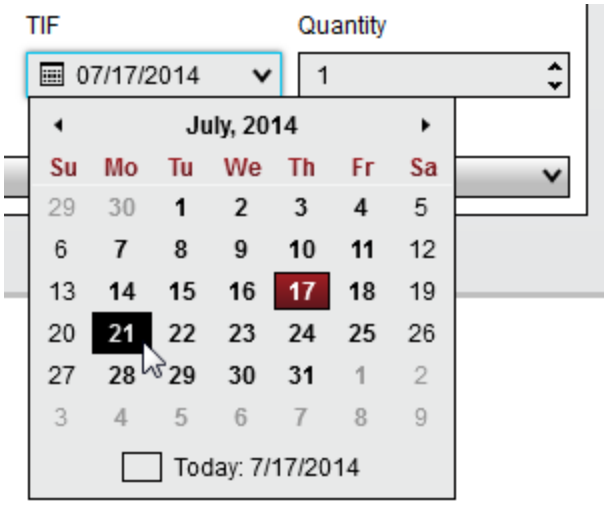
Available TIF Options

The available **TIF** options are determined by the connection technology of the current selected account. If a provider's connection technology does not support a certain **TIF**, it will not be listed to ensure a valid **TIF** is always been used. Possible **TIF** options are described in the table below:

DAY	Orders will remain active until the end of the trading session for the current day
GTC (Good 'Til Cancelled)	Orders will remain active until explicitly cancelled
GTD (Good 'Til Date)	Orders will remain active until the end of trading session on a user defined date

How to Submit an Order as GTD

When selecting **GTD** as the **TIF** for an order, you will be presented with a **Date Selector** to specify the date you would like the order to be cancelled.



In the image above, the current date is Thursday July, 17th. If you would like the prepared order to be live until end of session on the following Monday July 21st, you can simply select the 21st from the **Date Selector**.

10.23.8 Basic Entry

The **Basic Entry** window can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting the **Basic Entry** menu item.

Basic Entry Overview

The **Basic Entry** order entry window is comprised of several components: market data display, **Order Grid**, action buttons, as well as order entry and **ATM Strategy** management.

Display

› Display Overview

Misc

› Properties

Order Management

› Submitting Orders
› Modifying and Cancelling Orders
› Managing Positions

10.23.8.1 Display Overview

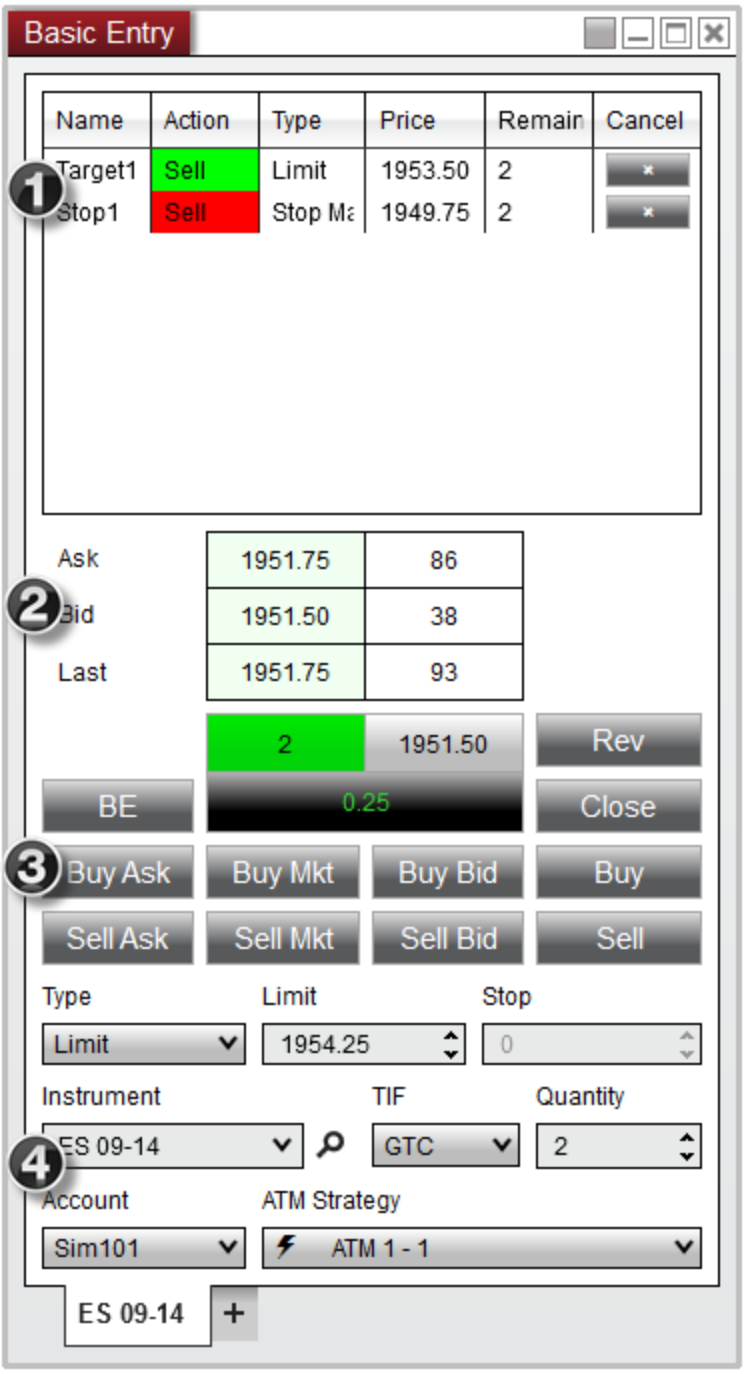
To open the **Basic Entry** Window, select the **New** menu from the NinjaTrader Control Center. Then left mouse click on the menu item **Basic Entry**.

Play Video

The image below shows each of the four sections in the **Basic Entry** window

1. Order Grid
2. Position and Level 1 (current inside market) display
3. Action Buttons
4. Order entry and **ATM Strategy** management

Note: Positions and orders will only display for the selected Account and Instrument.

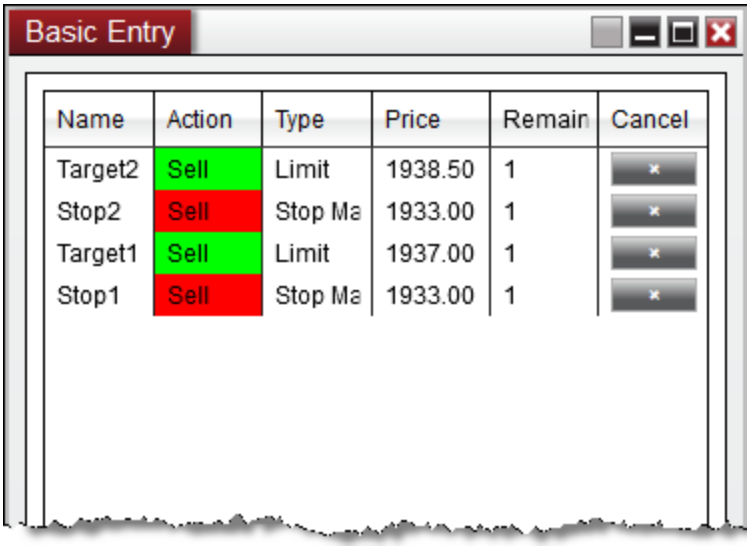


Please see the sections below for more information on each on: **Order Grid**, **Market Display**, **Action Buttons**, and **Order Control**.

Understanding the order grid section

Order Grid Display

The **Order Grid** displays active orders for the account and instrument selected in the **Basic Entry** window.



Column Definitions

Name	Order name such as Stop1 or Target1
Action	Buy or Sell
Type	Order type
Price	Order price
Remaining	Number of contracts/shares remaining to be filled
Cancel	Cancels the order(s)

Understanding the market display section

Market Data Display

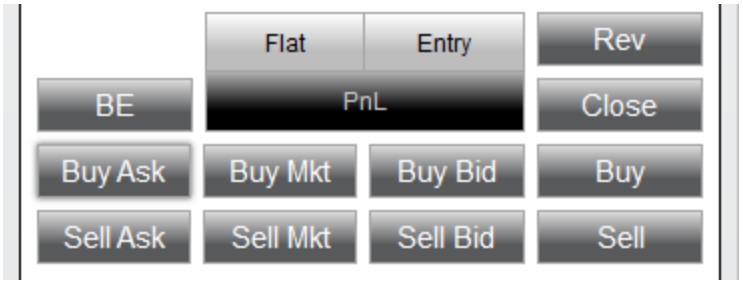
The market display section of the **Basic Entry** window is used to display market prices and position information. The market price displays will change colors when an uptick or a downtick has been detected.

1. Current best ask price and size
2. Current best bid price and size
3. Last traded price and size
4. Market position (FLAT or green background with position size for long, red background for short)
5. Position average entry price
6. Unrealized profit or loss for current position (Clicking on this cell with your left mouse button will change the display between points, ticks, currency, percent, and pips)

Understanding the action buttons section

Action Buttons

The **Basic Entry** has several buttons which are used to invoke a number of order related actions.



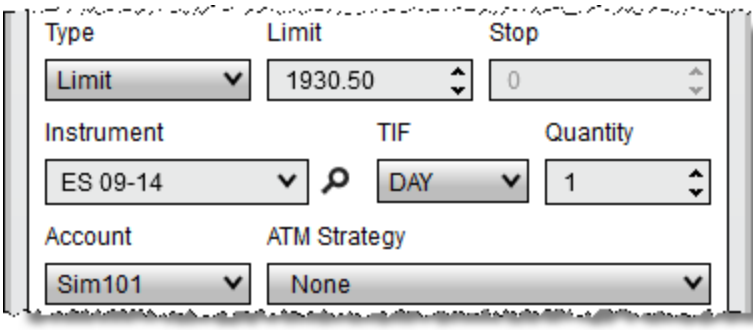
BE (Break-even)	Adjusts any open stop orders opposite of your position to the positions average entry price
Buy Ask	Submits a Buy Limit order at the current ask price
Sell Ask	Submits a Sell Limit order at the current ask price
Buy Mkt (Market)	Submits a Buy Market order at the current market price
Sell Mkt (Market)	Submits a Sell Market order at the current Market price
Buy Bid	Submits a Buy Limit order at the current bid price
Sell Bid	Submits a Sell Limit order at the current bid price
Rev	Closes the current open position and open a reverse position.
Close	Closes the current position and cancel any working orders associated to the instrument/account combination.
Buy	Submits a Buy order based on the current Order Controls configured
Sell	Submits a Sell order based on the current Order Controls configured

Please see the Submitting Orders section for more information on using these buttons.

Understanding the order control section

Order Controls

The **Order Control** section of the **Basic Entry** is used to specify several attributes for a pending order to be submitted.



Type	Selects the order Type to be submitted
Limit	Sets the order Limit price
Stop	Sets the order Stop Price
Instrument	Sets the Instrument
TIF	Sets the order Time in Force
Quantity	Sets the order Quantity
Account	Sets the Account
ATM Strategy	Selects the ATM Strategy

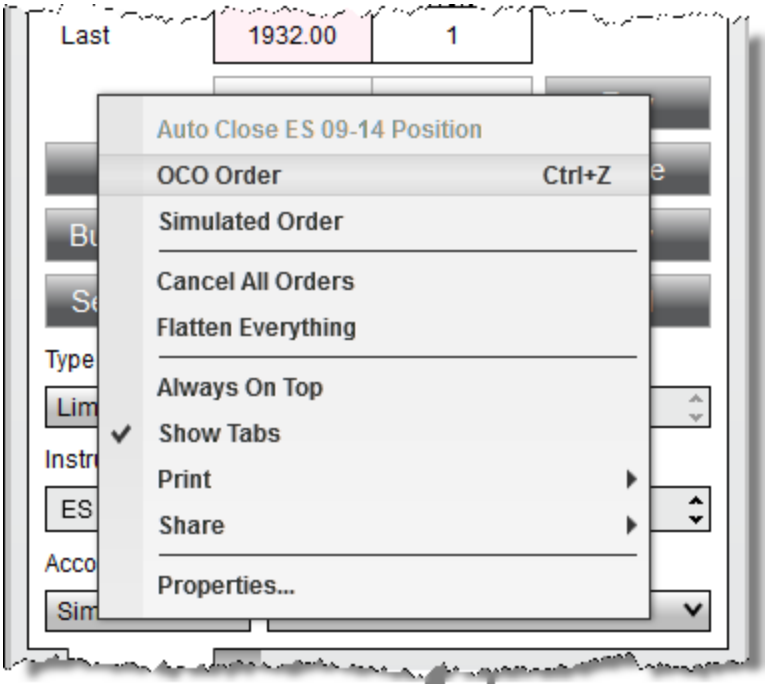
Understanding the right click menu

The **Basic Entry** window has two right click menus, depending on where you click:

- Right clicking on the **Basic Entry** window itself will bring up menu items specific to the **Basic Entry**
- Right clicking in the **Order Grid** will bring up menu items specific to orders

Basic Entry Control Right Click Menu

Right clicking on the **Basic Entry** window itself will bring up a number of menu items specific to the **Basic Entry**

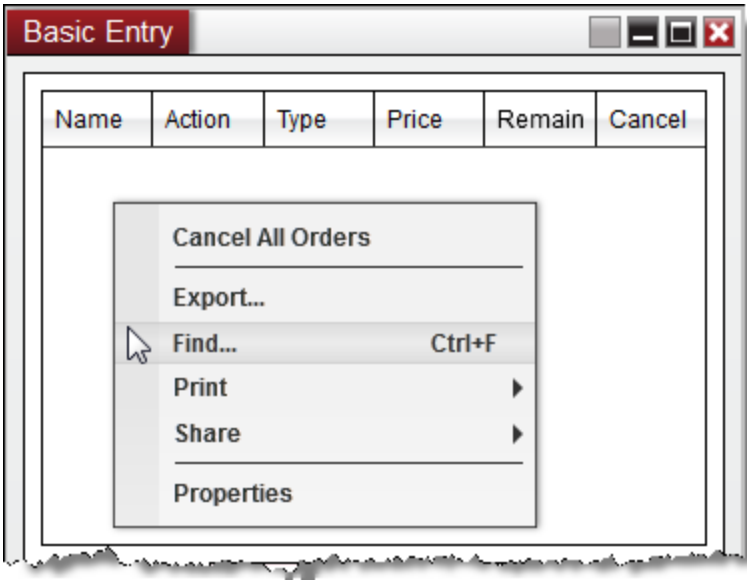


Auto Close Position	Automatically Closes the current instruments position at a specified time
OCO Order	Enables/Disables the OCO (one cancels other) function for a pending order
Simulated Order	Enables/Disables the Simulated Order functionality for a pending order
Cancel All Orders	Cancels all active orders on the current account
Flatten Everything	Closes all open positions and cancels all open orders on every account associated with NinjaTrader
Always On Top	Sets if the window should be always on top of other windows

Show Tabs	Sets if the window should allow for tabs
Print	Displays Print options
Share	Select to share via your share connections
Properties	Configure the Basic Entry window properties

Order Grid Control Right Click Menu

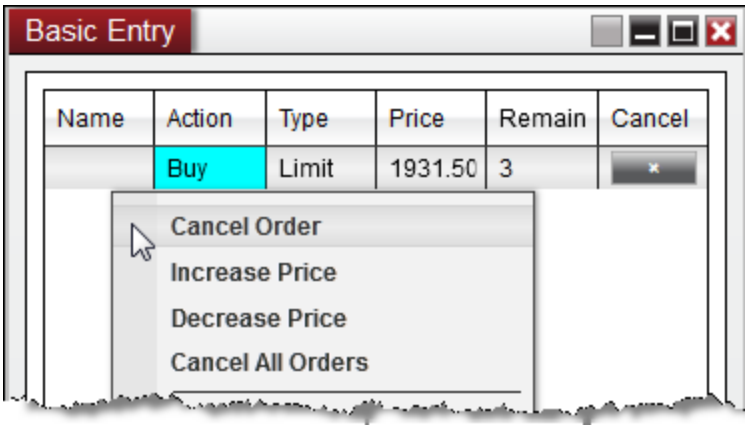
Right clicking in an empty grid will bring up a number of general menu items specific to the **Order Grid**



Cancel All Orders	Cancels all active orders on the current account
Export...	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Displays Print options

Share	Select to share via your share connections
Properties	Configure the Basic Entry window properties

By moving your mouse cursor over an order and pressing down on your right mouse button, you will see a context menu listing all individual orders consolidated at the corresponding price and any relevant actions that you can perform on those orders.



Cancel order	Cancels the individual order selected
Increase Price	Changes the price of the order +1 tick
Decrease Price	Changes the price of the order -1 tick
Cancel All Orders	Cancels all active orders on the current account

10.23.8.2 Submitting Orders

Submitting orders within the Basic Entry order entry window is both easy and efficient. In addition to entry and exit orders the Basic Entry window also offers access to NinjaTrader's ATM Strategies. For more information on ATM Strategies please see the ATM section of the user help guide or attend one of our free live training events.

Selecting instruments and account

How to Select an Instrument

There are multiple ways to select an Instrument in the **Basic Entry** window.

- Select the **Instrument Selector** to open a list of recently used instruments or instruments contained in a predefined list
- With the **Basic Entry** window selected begin typing the instrument symbol directly on the keyboard. Typing will trigger the **Overlay Instrument Selector**.

For more Information on instrument selection and management please see Instruments section of the Help Guide.

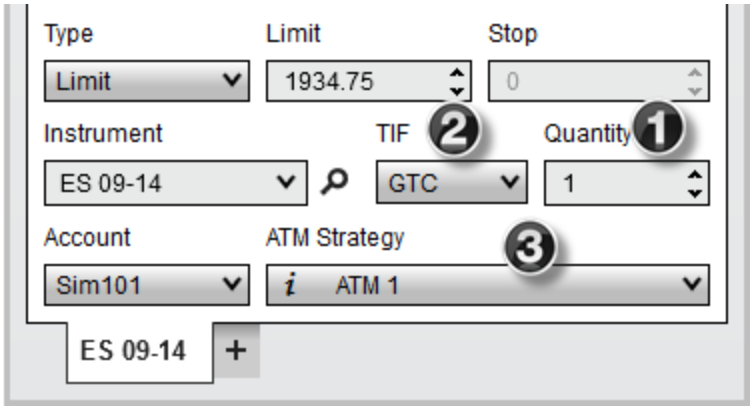
How to Select an Account

A list of all connected accounts will be listed in the "**Account**" drop down list. To change the account select the account you wish to trade through via this drop down list.

Understanding order settings

To submit an Order

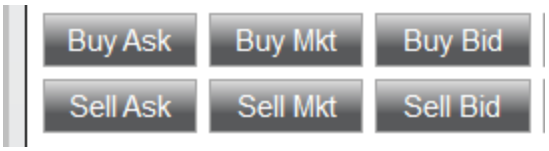
1. Set the order "**Quantity**" field (info)
2. Set the **"TIF"** (Time in Force) field (info)
3. Set the **"ATM Strategy**" (info)
4. Enter an order with any of the methods described below



How to submit orders with quick buttons

Quick Buttons

You can enter orders rapidly by pressing on any one of the quick order buttons.



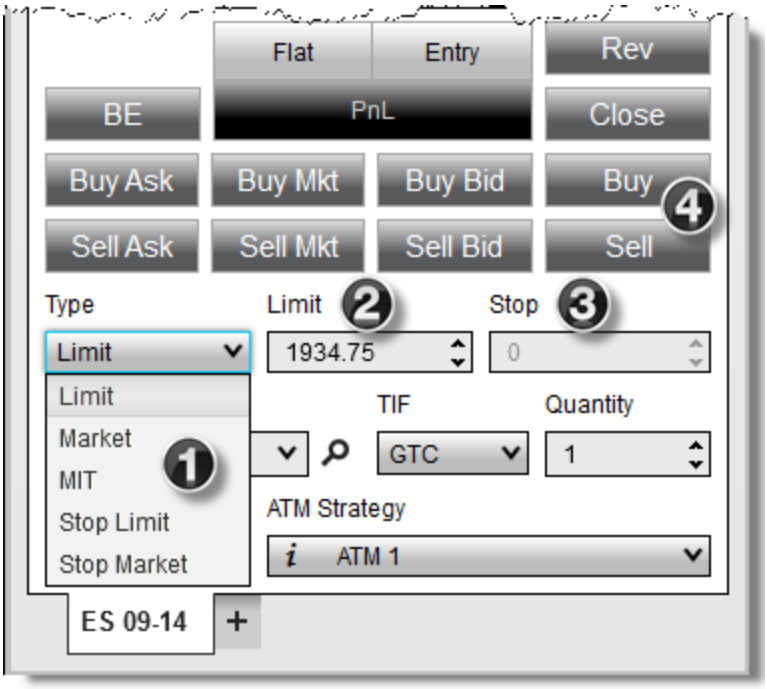
Buy Ask	Submits a Buy Limit order at the current ask price
Sell Ask	Submits a Sell Limit order at the current ask price
Buy Mkt (Market)	Submits a Buy Market order at the current market price
Sell Mkt (Market)	Submits a Sell Market order at the current Market price
Buy Bid	Submits a Buy Limit order at the current bid price
Sell Bid	Submits a Sell Limit order at the current bid price

How to submit custom orders

Custom Orders

You can place a custom order by setting order parameters.

1. Select the order **Type**
2. Set the **Limit** price if applicable
3. Set the **Stop** price if applicable
4. Left mouse click either the **BUY** or **SELL** button



Tips

1. You can quickly retrieve the current last, bid, or ask price in the Limit and Stop price fields using the following commands:

- Middle click in the field to retrieve the last traded price,
- CTRL + middle click in the filed to retrieve the best ask price
- ALT + middle click in the field to retrieve the best bid price

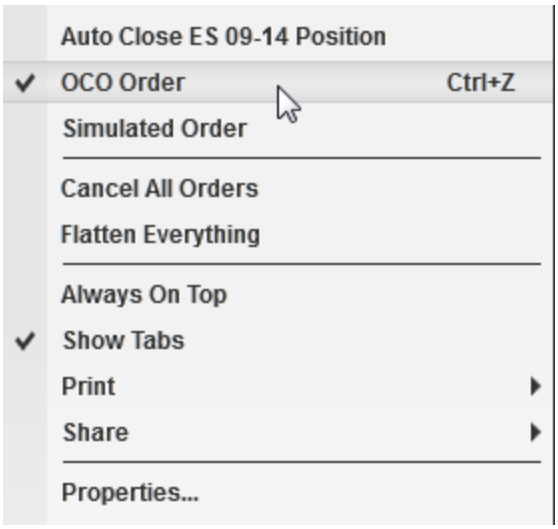
2. Hold down the CTRL key when increasing/decreasing limit/stop prices to change the price in steps of 10 tick increments.

Understanding the OCO (One Cancel Other) function

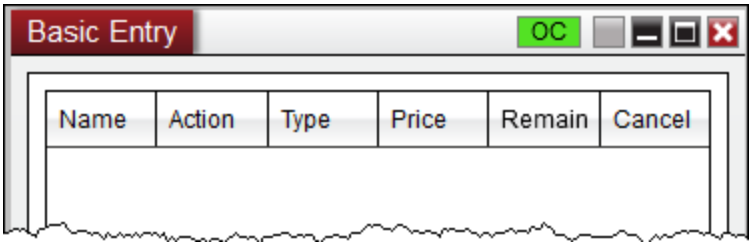
OCO Orders (One Cancels Other)

Stop Loss and **Profit Target** orders (submitted automatically via an ATM Strategy) are always sent as **OCO**, however, you can submit entry or exit orders as **OCO** orders as well. Why? The market may be trading in a channel and you wish to sell at resistance or buy at support, whichever comes first by placing two limit orders at either end of the channel.

To place **OCO** orders, press down on your right mouse button inside the **Basic Entry** window and select the menu name "**OCO Order** or use the short cut key CTRL+Z.



The "**OC**" (OCO indicator) will light up green at the top of the **Basic Entry** window. All orders placed while this indicator is lit will be part of the same **OCO** group. Once any order of this group is either filled or cancelled, all other orders that belong to this group will be cancelled.



If you want each **OCO** order to create it's own set of **Stop Loss** and **Profit Target** orders ensure that the ATM Strategy control list is set to either **<Custom>** or a strategy template name before you submit each OCO order.

After you have placed your orders, it is advised to disable the **OCO** function via the right click menu, or use the short cut key CTRL+Z.

Warning: If an order which was part of an **OCO** group has already been filled or cancelled, you will need to submit the pending order with a new **OCO ID** otherwise the pending order will be rejected.

To reset an **OCO** ID, simply disable the **OCO** function, and re-enable. This will generate a new **OCO ID** and allow you to place new orders.

Break Out/Fade Entry Example

One of the great features of NinjaTrader is its ability to submit two entry orders, one of which will cancel if the other is filled.

You can accomplish a breakout/breakdown approach by:

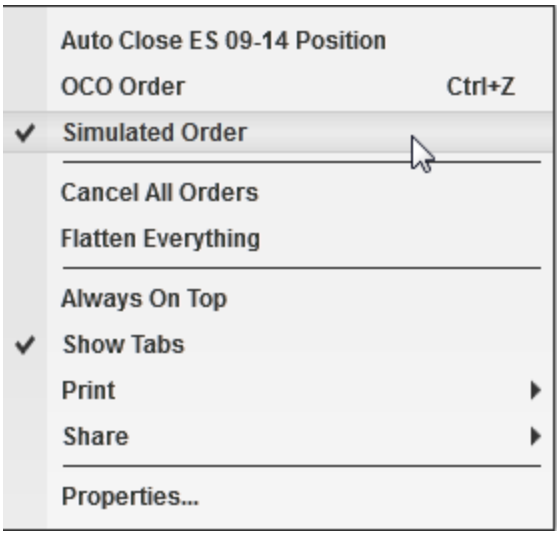
- Right clicking in the **Basic Entry** window and selecting the menu item "**OCO Order**" to enable the OCO function
- For your first order, select the desired option from the "**ATM Strategy**" drop down list
- Submit your stop order to buy above the market
- For your second order, select the desired option from the "**ATM Strategy**" drop down list
- Submit your stop order to sell below the market
- *CRITICAL*: Right click in the **Basic Entry** window and select the menu item "**OCO Order**" to disable OCO for future orders.

For a market fade approach just substitute limit orders for stop orders.

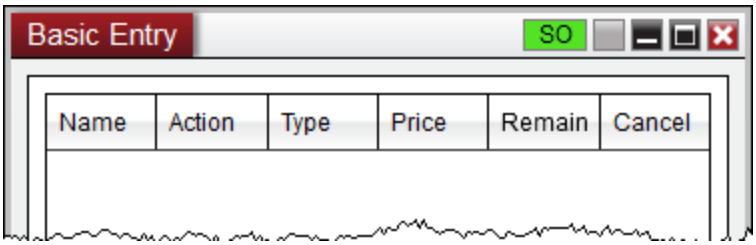
How to submit Simulated Stop Orders (Simulated Order)

Simulated Stop Orders (Simulated Order)

To submit a **Simulated Stop Order** (entry and exit NOT Stop Loss; simulated Stop Loss orders are enabled via an ATM stop strategy) you must enable **Simulated Order** mode via the right mouse click context menu by selecting the **Simulated Order** menu item..



The "**SO**" (Simulated Order indicator) will light up green at the top of the **Basic Entry** window. All stop orders placed while this indicator is lit will be submitted as a Simulated Stop Orders.



10.23.8.3 Modifying and Cancelling Orders

You can modify an existing order's quantity, price, or cancel an order entirely from **Order Grid** display of the **Basic Entry** window.

▼ Modifying existing orders

Changing the Price of an Order

1. You can increase the price of an order in one tick increments by right mouse clicking on the order in the order grid and selecting "**Increase Price**".

2. You can decrease the price of an order in one tick increments by right mouse clicking on the order in the Order Grid and selecting "**Decrease Price**".

3. Double clicking on the Price field will enable the **price editor** which will allow you to type in a new price manually, or use the scroll wheel on your mouse to select a relative price.

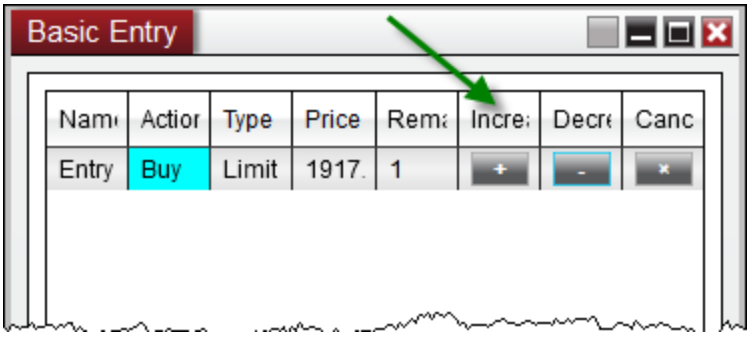
Tip:
Hold down the CTRL key when scrolling in the **price editor** to change the price in steps of 10 tick increments.

Enabling Increase and Decrease Columns

You can optionally enable columns on the **Order Grid** display which will allow you to increase or decrease the price of an order using a button click.

To enable these columns:

1. Right click on the **Basic Entry** Window and select Properties
2. Expand the "**Columns**" section
3. Check the "**Increase**" and/or the "**Decrease**" options
4. Press "**OK**"



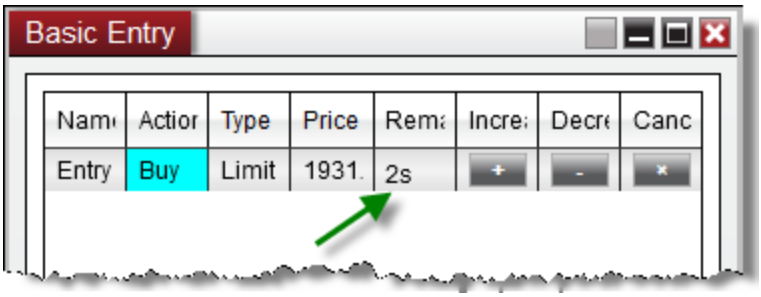
1. You can increase the price of an order in one tick increments by left mouse clicking on the "**+**" button. Holding the CTRL key down while pressing the "**+**" button will modify the order by 5 tick increments, and holding the ALT key will modify the order by 10 tick increments.

2. You can decrease the price of an order in one tick increments by left mouse clicking on the "**-**" button. Holding the CTRL key down while pressing the "**-**" button will modify the order by 5 tick increments, and holding the ALT key will modify the order by 10 tick increments.

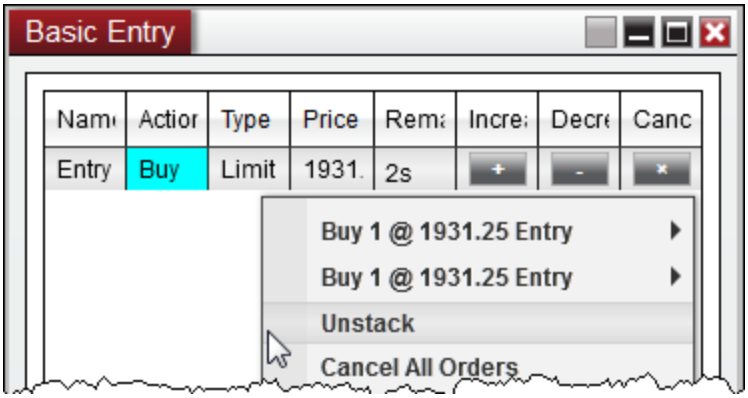
Changing the Quantity of an Order

You can change the size of an order by left clicking in the "**Remaining**" column, typing in a new quantity value and pressing the "Enter" key on your keyboard.

Changing the quantity of an existing order will submit a new order the same price to preserve your place in queue. Your orders will now show up as "stacked" indicated by the small letter "**s**" next to the order.



If you would like to break up these orders to manage individually, you can right click on the Order and select "**Unstack**"



Cancelling orders

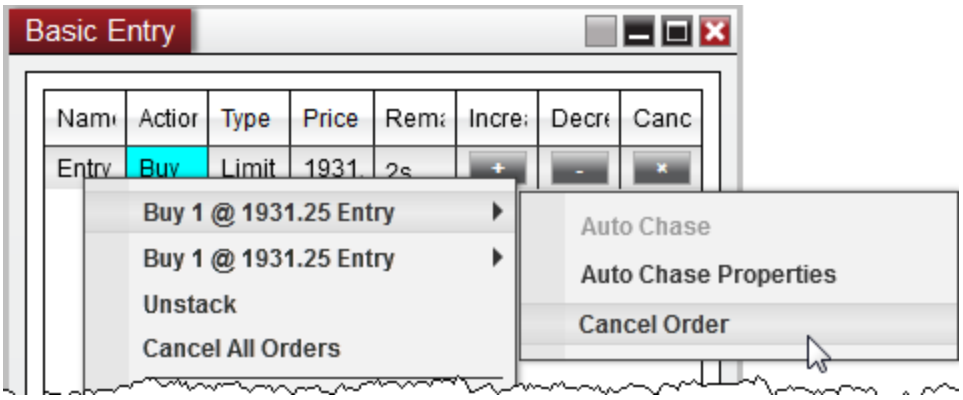
Cancelling an Individual Order

1. You can cancel an order by left mouse clicking on the "**X**" button.
2. You can also right click on the order itself and press the "**Cancel Order**" menu item

Cancelling Stacked Orders

If you have stacked orders, indicated by the small letter "**s**" in the Remaining column, you can cancel one of the orders, and leave the other remaining using the steps below:

1. Right click on the stacked order row
2. Move your mouse over the order individual order
3. Select "**Cancel Order**"



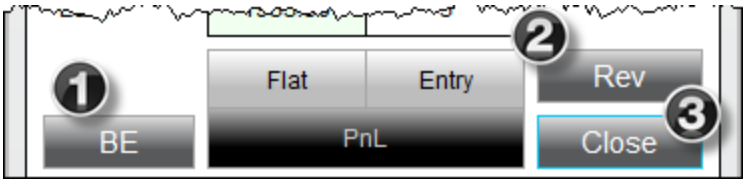
10.23.8.4 Managing Positions

How to Manage Open Positions

1. Clicking on the "**BE**" (break-even) button with your left mouse button will adjust any stop orders in the opposite direction of your open position (if position is long it will adjust stop sell orders) to the positions average entry price. Clicking on this button with your middle mouse button (scroll wheel) will only adjust any Stop Loss orders associated to the selected active **ATM Strategy** in the strategy drop down list. Orders resting at a better price than the average entry price will NOT be modified.

2. Clicking on the "**Rev**" button will close the current open position and open a reverse position.

3. Clicking on the "**Close**" button with your left mouse button will close the current position and cancel any working orders associated to the instrument/account combination. Clicking on this button with your middle mouse button (scroll wheel) will close the selected active **ATM Strategy** only. This means that the position size of the **ATM Strategy** will be closed and any working orders associated to that ATM Strategy will be cancelled.



Please see the help topic on Closing a Position or ATM Strategy for more information on the mechanics behind closing various types of positions.

10.23.8.5 Properties

The Basic Entry order entry window is highly efficient by design but can also be customized to your preferences through the Basic Entry Properties menu.

How to access the Basic Entry properties window

You can access the Basic Entry properties dialog window by clicking on your right mouse button within the Basic Entry window and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the **Basic Entry** properties window:

Property Definitions

General	
ATM strategy selection mode	Sets the behavior mode of the price ladder display and strategy selector (see more)
Grid font	Sets the font options
PnL display unit	Sets the display unit for profit and loss
Scale quantity	Sets the scale order quantity amount.
Show realized PnL when flat	Displays realized profit and loss for the selected account when flat
Simulated order volume trigger	Sets the value for a simulated order volume trigger (for entry and exit orders and NOT used for stop loss)
Tab name	Sets the tab name
Quantity modification for stocks	Sets if modifying an existing stock order quantity changes the the order, or submits a new order
Colors	
Action button	Sets the color for action buttons (CLOSE, BE etc...)
Buy button	Sets the color for all the buy buttons

Downtick background	Sets the background color of the market data display when a downtick is detected
Downtick foreground	Sets the foreground (text) color of the market data display when a downtick is detected
Order - limit	Sets the color used for limit orders
Order - MIT	Sets the color used for MIT orders
Order - profit target	Sets the color used for profit target orders
Order - stop limit	Sets the color used for stop-limit orders
Order - stop loss	Sets the color used for stop loss orders
Order - stop-market	Sets the color used for stop-market orders
Sell button	Sets the color used for all the sell buttons
Uptick background	Sets the background color of the market data display when an uptick is detected
Uptick foreground	Sets the foreground (text) color of the market data display when an uptick is detected
Columns	

Order Columns (...)	Expanding this section will allow you to disable / enable any of the columns in the Orders Grid display
Window	
Always on top	Sets if the window will be always on top of other windows.
Show tabs	Sets if the window should allow for tabs

How to set the default properties

Once you have your properties set to your preference, you can left mouse click on the "**preset**" text located in the bottom right of the properties dialog. Selecting the option "**save**" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "**preset**" text and select the option to "**restore**" to return to the original settings.

Using Tab Name Variables

Tab Name Variables
A number of pre-defined variables can be used in the "Tab Name" field of the **Basic Entry Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.23.9 Chart Trader

Chart Trader can be enabled within any chart window via the chart properties dialog window or by left mouse clicking on the **Chart Trader** icon in the chart toolbar.

Play Video

Chart Trader Overview

When enabled, the **Chart Trader** panel will be visible on the right side of the chart window. You will be able to submit, modify and cancel orders directly from within the chart.

10.23.9.1 Order & Position Display

Chart Trader allows for the placement of orders, and the management of orders and positions, directly from a chart. Orders and positions within **Chart Trader** are displayed in a visual manner, allowing you to quickly compare them with current market movements while modifying orders in real-time. **Chart Trader** contains two primary components: the **Chart Trader** panel, which is used to place, modify, or cancel/close orders and positions, and the chart panel, on which **Chart Trader** draws visual representations of resting orders and open positions.

▼ Understanding order display

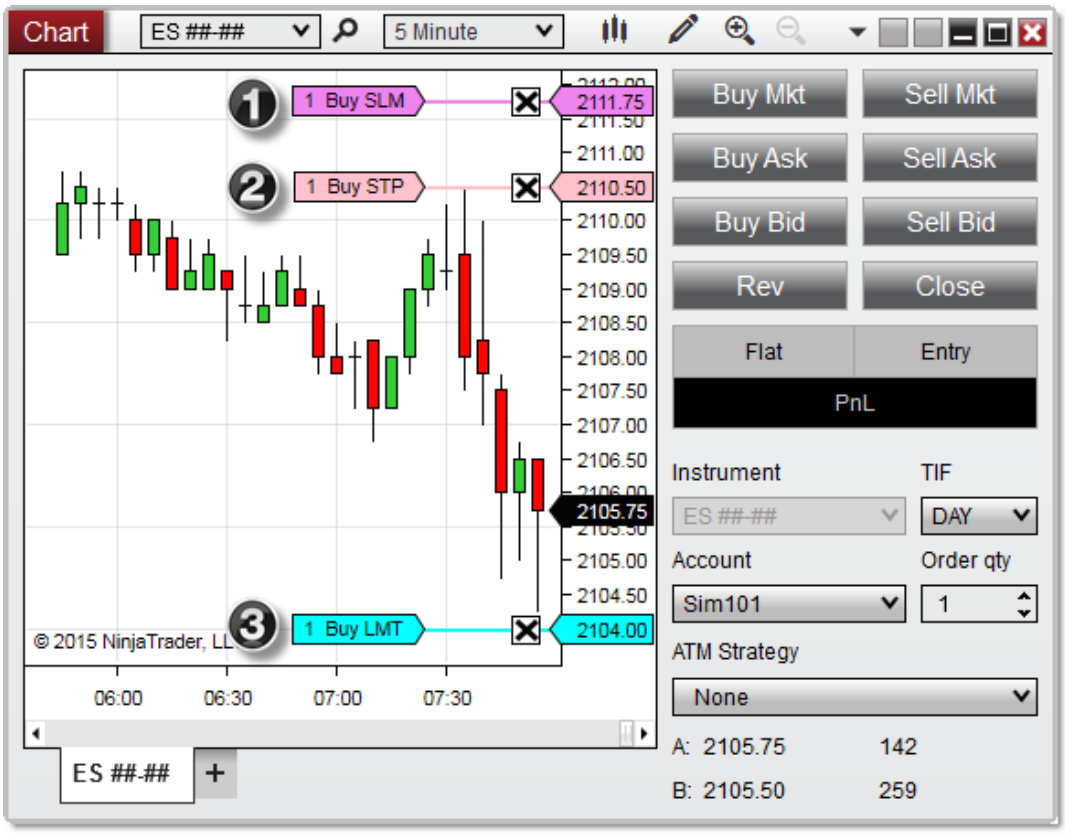
Order Display

A resting order is displayed on the chart as a color-coded line connecting an order price label in the right margin of the chart with a second label displaying the order quantity and type.

Limit Order	Default color is cyan with the text "LMT"
Stop-Market Order	Default color is pink with text "STP"
Stop-Limit Order	Default color is violet with text "SLM"
Market Order	Market orders are not displayed (see **Control Center** Orders Tab for more information)
Market if Touched (MIT) Order	Default color is spring green with the text "MIT"
Stop Loss Order	Default color is red
Profit Target Order	Default color is lime

Chart trader color properties can be set via the **Chart Trader** properties window.

The image below displays how orders are visualized in a NinjaTrader chart with **Chart Trader** enabled.



1. Buy stop-limit order for 1 contract at a price of X
2. Buy stop-market order for 1 contract at a price of X
3. Buy limit order for 1 contract at a price of X

Note: Orders will only display for the selected Account and Instrument.

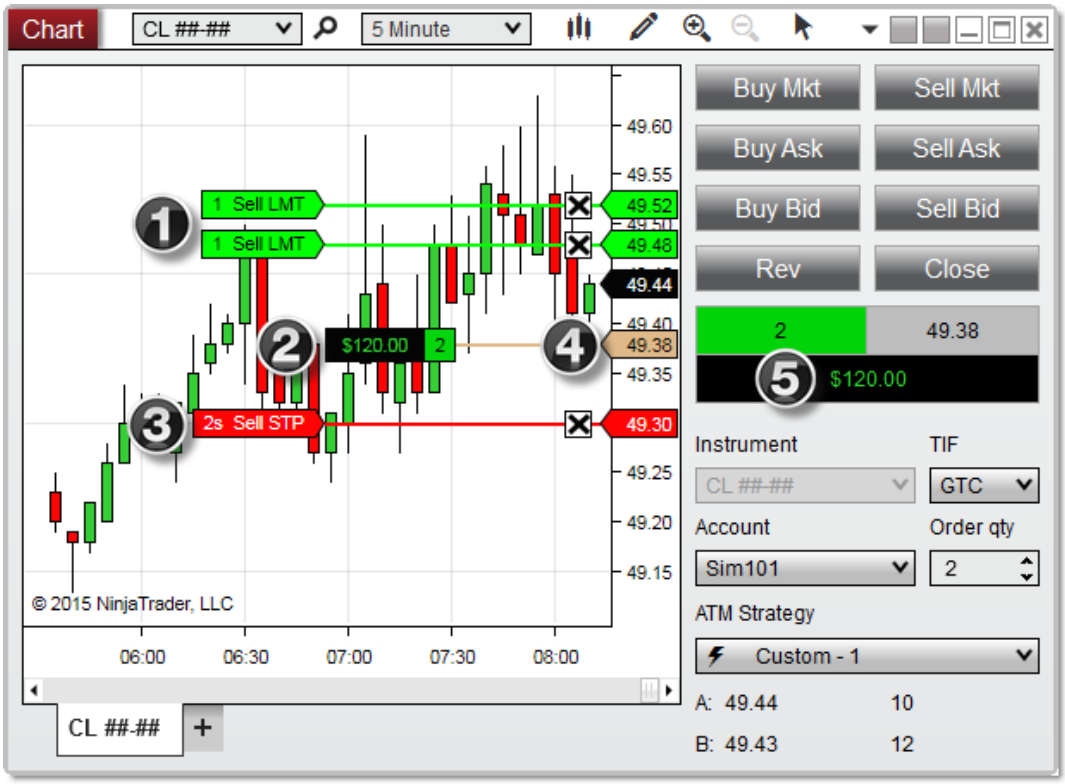
Understanding Position Display

Position Display

An open position is displayed slightly differently. A position is displayed on the chart as a colored line connecting an entry price label in the right margin of the chart with a second label displaying the position size and current unrealized profit or loss. The text displaying the profit and loss is color coded, with green representing profit and red representing loss. The quantity displayed in the left-hand label is color coded, as well, with green representing a long position and red representing a short position.

Note: The display of unrealized PnL in **Chart Trader** can be switched between points, currency, pips, ticks, and percent by either left mouse clicking in the PnL field in the **Chart Trader** panel, or via the **Chart Trader** Properties window.

The image below displays the chart with an active position managed by an Advanced Trade Management strategy.



1. Two profit target orders
2. Position size and PnL flag for 2 contracts long
3. Two stop loss orders*
4. Average entry price
5. PnL in **Chart Trader** panel

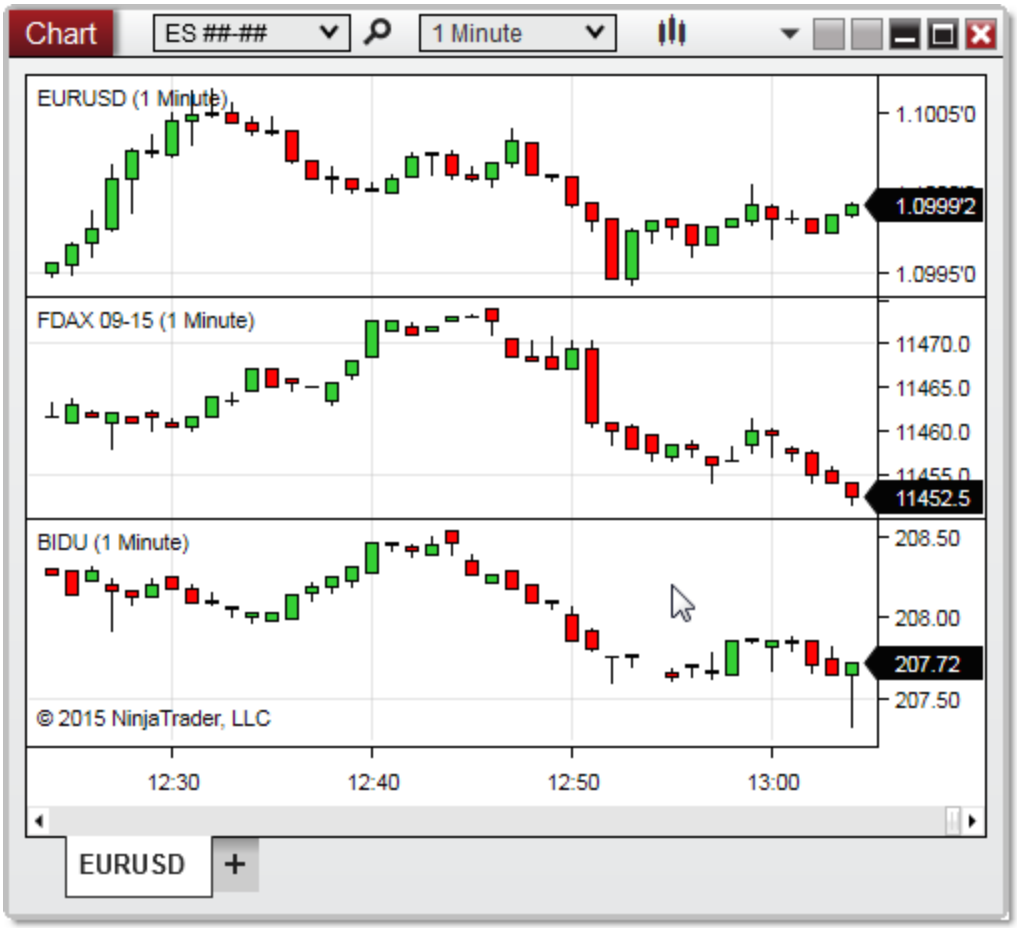
* The stop loss line and flag represents two orders, as indicated by the letter "s" next to the qty number "2."

Note: Positions will only display for the selected Account and Instrument.

Working with multiple instruments

Multi-Series Charts

NinjaTrader charts include the ability to plot multiple instruments within a single chart window, and each individual instrument on a chart can be selected and worked with separately using **Chart Trader**. For more information on how to manage instruments on a chart, see the Working with Multiple Data Series page.



In the image above, we have applied a EURUSD instrument, an FDAX ##-## instrument, and a BIDU instrument in three separate panels of the same chart.

Selecting Data Series

With more than one instrument applied to a chart, you can change the instrument upon which **Chart Trader** will act via the Instrument dropdown menu. This menu will list all of the **Data Series** currently applied to your chart. When an instrument is selected, only orders and positions for that particular instrument will be displayed in the chart panel, and any quick buttons used or order parameters set in the **Chart Trader** panel will apply to that instrument.

1. With the EURUSD selected in the Instruments dropdown menu, we can only that instrument's orders and positions.

2. With BIDU selected, we see a different set of orders and positions.

10.23.9.2 Hidden View

Chart Trader can be displayed fully, disabled fully, or collapsed. When the collapsed view is enabled, the order and position display functionality of **Chart Trader** is still enabled, and orders can still be placed and managed directly from the chart panel, but the **Chart Trader** panel itself will be hidden. This serves to maximize screen space for charts while maintaining much of the important functionality of **Chart Trader**.

Collapsing and restoring the Chart Trader panel

There are two ways to collapse the **Chart Trader** panel. You can either click the **Chart Trader** icon on the chart toolbar, then select the "Chart Trader (Hidden)" menu item, as seen in the image below, or you can edit the "Chart Trader" property within the chart Properties window. When you wish to view the **Chart Trader** panel once more, you can use the same methods to select the "Chart Trader" menu item.

In the image above, we see an open position and a modifiable resting order on the chart, even though the **Chart Trader** panel is hidden.

10.23.9.3 Submitting Orders

There are several methods that can be used to submit orders directly from a chart using **Chart Trader**.

▼ How to submit an order

Submitting Orders

To submit an order via the **Chart Trader** panel:

1. If you have more than one instrument applied to the chart, select an instrument on which to place the order via the **Instrument** dropdown menu

2. Select the order time-in-force via the **TIF** dropdown menu

3. Select an account via the **Account** dropdown menu

4. Enter an order quantity in the **Order Qty** field

5. If you would like to use an Advanced Trade Management (ATM) strategy with the order, set the **ATM Strategy** options via the **ATM Strategy** dropdown menu. Options in this menu include:
   - **a)** None: Orders are submitted without an attached **ATM Strategy**
   - **b)** Custom: This will open the **Custom Strategy Parameters** window, in which you can create and save a new **ATM Strategy.**
   - **c)** <Strategy Name> - X: A strategy template name followed by a number represents an active instance of an **ATM Strategy** on the chart.
   - **d)** User Defined Strategy Template: Stops and targets are submitted from a predefined user template

A more thorough explanation of these concepts can be found on the ATM Strategy Parameters page.

With these parameters set, you can then enter an order with any of the methods described in the sections below.

▼ Understanding order options in the right click menu

Order Options

Order options will appear in the right click menu when **Chart Trader** is enabled. These options provide the ability to select pre-defined order types and prices based on the location of your mouse cursor. After right mouse clicking in the chart panel to view the right click menu, left mouse click on the desired order option to submit an order. After an order has been submitted in this way, it can be moved or canceled at will before it is filled.

Note: Available order types in the right click menu will be limited to those which will be accepted by a brokerage, based on the side of the market on which you right click. For example, in the image above, Buy Limit is not an option, since the right mouse button was clicked above the market price, and Buy Limit

orders cannot be placed above the market price. Sell Stop Market and Sell Stop Limit are missing in the image above, as well, for the same reason.

Stop-Limit Offset

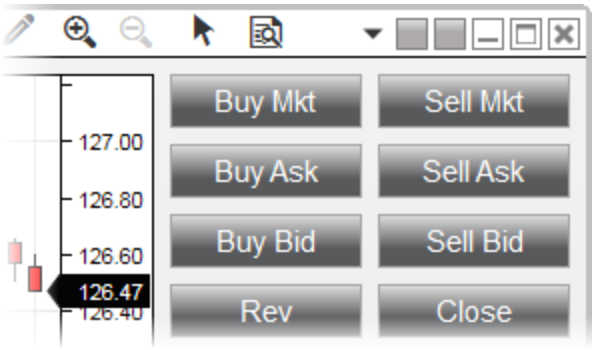
When submitting a stop-limit order, a numeric field will appear, allowing you to set the limit offset of the order (the number of ticks away at which you wish to place the Limit price of the Stop-Limit order). Either by using your mouse scroll wheel or clicking on the up/down arrows in the numeric field, set the number of ticks and press the checkmark button to complete the order submission. For example, if you intend to place an order with a buy Stop price of 1000 and a Limit price of 1001 (4 ticks for the S&P E-mini contract), you would set the numeric field value to 4. Following the same example and submitting a sell stop-limit order, setting the numeric field value to 4 would result in a stop price of 1000 and a limit price of 999. Pressing the "**X**" button will cancel the order submission operation.

The numeric field also supports negative values. When a negative value is entered, a **Simulated Stop** order will be place (see the "*Understanding Simulated Stop Orders*" section below).

Understanding the Quick Buttons

Quick Buttons

You can quickly submit orders via the **Chart Trader** panel's Quick Buttons.



Button Actions

Buy Mkt - Submits Buy order at market

Sell Mkt - Submits Sell order at market

Buy Ask - Submits a Buy Limit order at the Ask price

Sell Ask - Submits a Sell Limit order at the Ask price

Buy Bid - Submits a Buy Limit order at the Bid price

Sell Bid - Submits a Sell Limit order at the Bid price

How to scale in or out of an active ATM strategy

Scaling In or Out of an Active ATM Strategy

When you have an active strategy selected in the **ATM Strategy** dropdown menu, any new orders submitted will scale into or out of that active strategy instance. Once filled or partially filled, existing stop loss and profit target orders will be modified to reflect the new position strategy size. You can preset a default scale in or out quantity via the "Scale Quantity" property accessible via the **Chart Trader** properties window. As an example, your initial strategy may call for opening a position of four contracts, but you want subsequent scale orders to be only one contract. If the **Chart Trader's** "Scale Quantity" property is set to a value of 1, the **Qty** (Quantity) field will be set to a value of 1 automatically when an active strategy is selected in the list.

> **Note**: For a complete understanding of order submission and subsequent actions that you can have NinjaTrader automate, see the ATM Strategy Parameters page.

Understanding "One Cancels Other" (OCO) orders

OCO Orders (One Cancels Other)

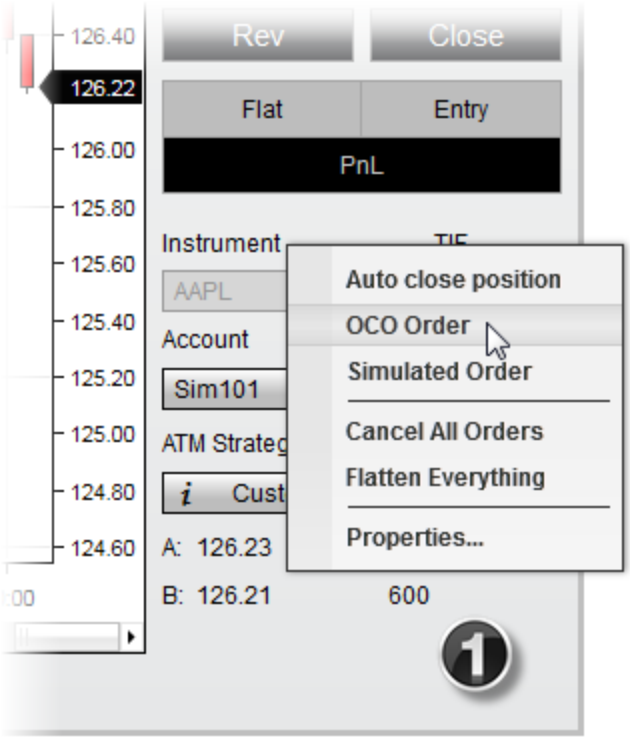
One Cancels Other functionality ties two resting order together, so that when one is canceled or filled, the other will be canceled automatically. This can be ideal for manually placing bracket entry orders, or placing Stop Loss and Profit Target orders. Stop loss and profit target orders submitted automatically via an ATM strategy are always sent as OCO; however, you can submit entry or exit orders as OCO orders as well. Why? The market may be trading in a channel and you wish to sell at resistance or buy at support, whichever comes first by placing two limit orders at either end of the channel.

To place OCO orders, first right mouse click within the **Chart Trader** panel, then select the menu item **OCO Order** or use the default Hot Key **CTRL+Z.**

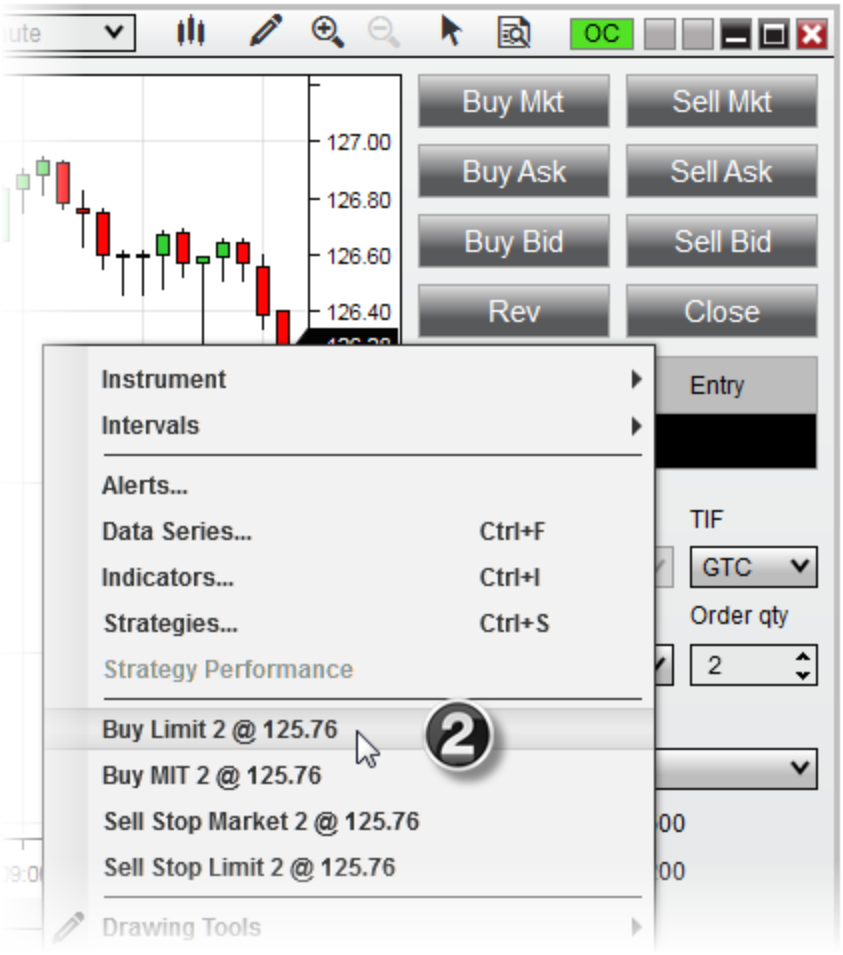
All orders placed while OCO is enabled will be part of the same OCO group. Once any order of this group is filled or cancelled, all other orders that belong to this group will be cancelled. If you want each OCO order to create it's own set of stop loss and profit target brackets, ensure that the ATM Strategy drop down menu is set to either **Custom** or an **ATM Strategy** template before you submit each OCO order.

Warning: After placing two orders within the same OCO group, it is important to disable OCO functionality before submitting any other orders. If you wish to place another set of OCO orders immediately after placing an initial set, first disable, then re-enable OCO before placing the second set of orders. This will generate a new **OCO ID**, rather than adding the new orders to an existing OCO group.

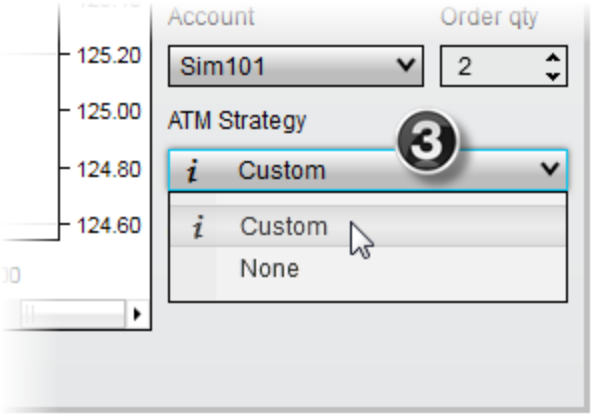
Example:
Below are steps for submitting a Sell Limit and a Buy Limit order via OCO.



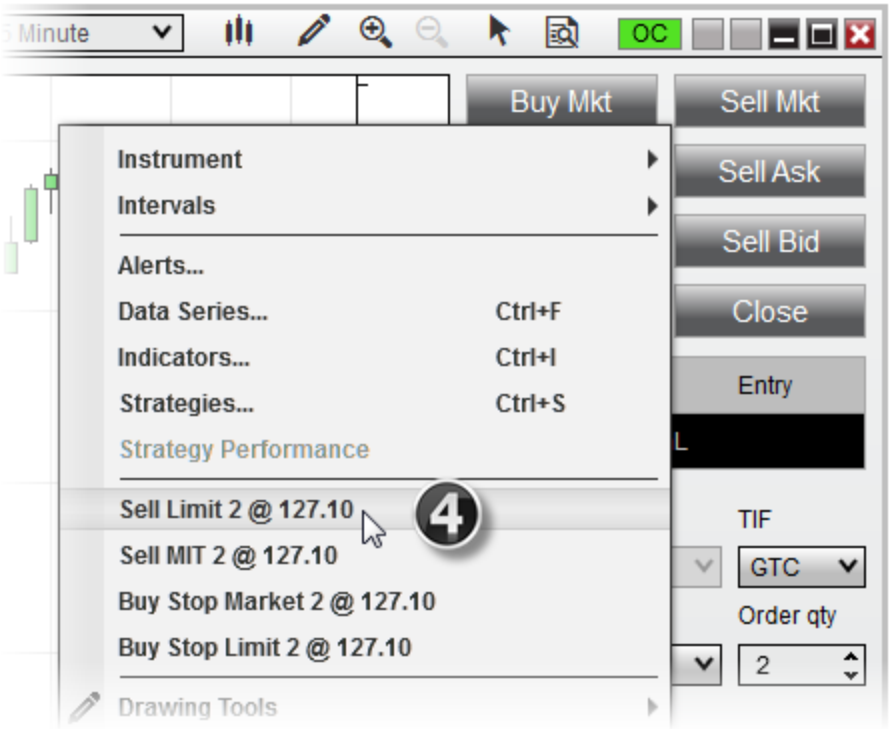
1. Enable OCO



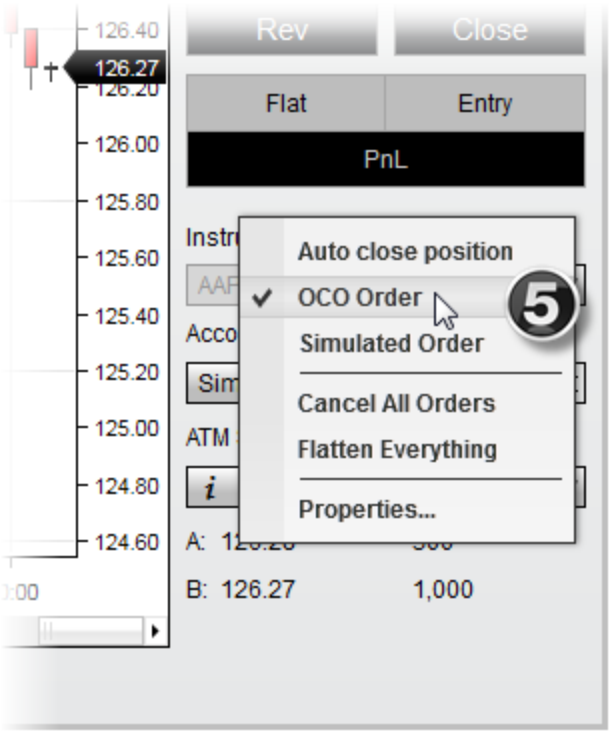
2. Notice that there is a green **OC** at the top right of the chart, indicating OCO is enabled. Place a Buy Limit order



3. If you are initiating two orders to enter a new strategy, then re-select the strategy in the strategy selection list



4. Place a Sell Limit order



5. Disable OCO (it is critical that you disable OCO before submitting another OCO group)

Understanding simulated stop orders

Simulated Stop Orders (Simulated Order)

Simulated Stop Orders allow you to place orders that trigger at a specific price but try to fill at a better price. To submit a **Simulated Stop** order, you must enable Simulated Orders via the right click menu and select the **Simulated Order** menu item, or use a user-defined Hot Key. All stop orders placed while this setting is enabled will be submitted as a **Simulated Stop** order.

Notes:

- This applies to entry and exit orders specifically, **NOT** Stop Loss orders; simulated Stop Loss orders are enabled via a stop strategy
- When a **Simulated Stop** order is displayed in the chart panel via **Chart Trader**, it's line and type/quantity label will be colored yellow by default to differentiate it from a Stop Limit order

One of the powerful features of **Simulated Stop** orders is that you can submit a "negative limit stop-limit" order. This means that you can place an order in which the Limit price is better than the Stop price. As an example, you may want to buy on strength indicated by a move up to a particular price. Once that occurs, you want to enter at a better price using a limit order several ticks below the stop price. Any stop-limit order submitted with a negative limit offset automatically becomes a simulated order and will be held on your PC until the stop is triggered or canceled.

10.23.9.4 Modifying and Cancelling Orders

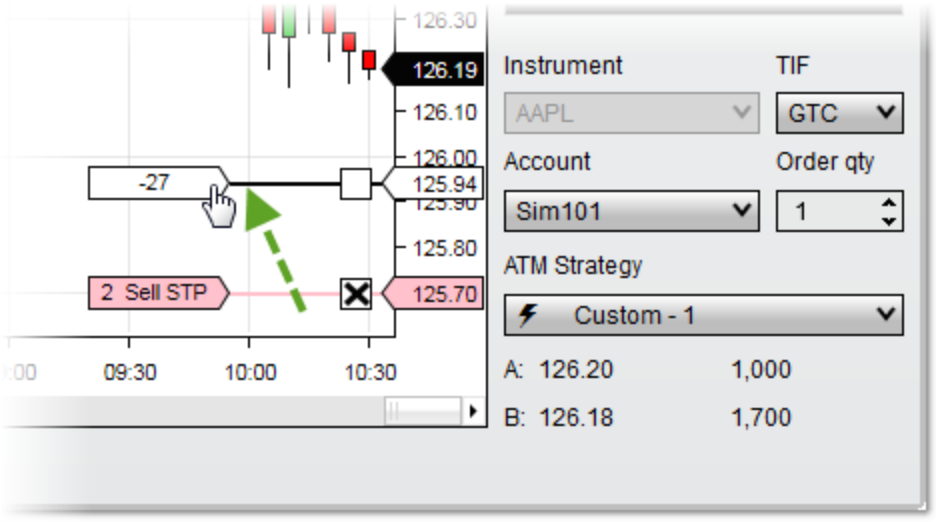
An order can be modified or canceled within a chart when **Chart Trader** is enabled.

▼ How to modify an order price

Modifying Order Price

To modify the price of an order, left mouse click on the label at the left side of the order line. A ghost order line will appear and display the number of ticks you are away from the inside market.

When the ghost order line is above the Ask price, the label will display a positive value. When it is below the Bid price, the label will display a negative value. At the Ask the label will display "@Ask," and at the Bid the label will display "@Bid."



Once you have the ghost order at the price you desire, left click with your mouse to complete the modification. This is a click and click method, rather than a click and drag method.

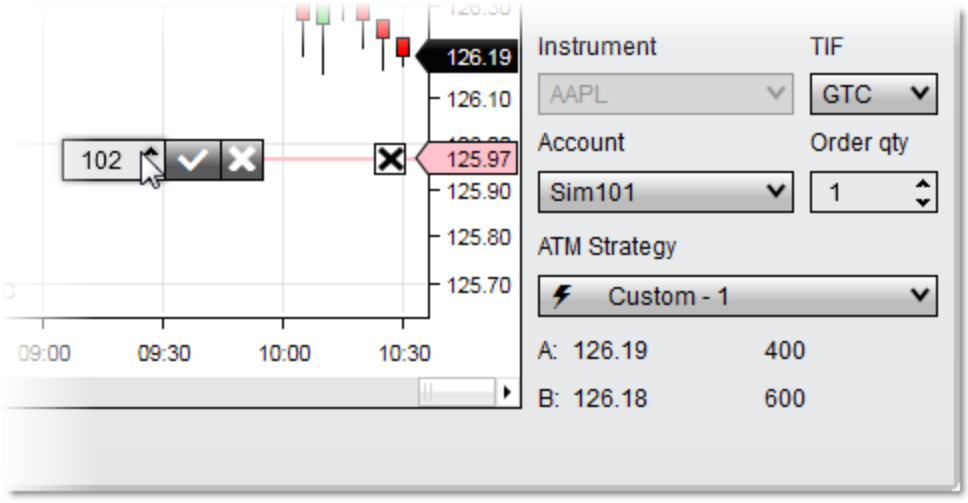
To cancel a pending order price modification, press the "Esc" key on your keyboard.

How to modify an order size

Modifying Order Size

To modify the size of an order:

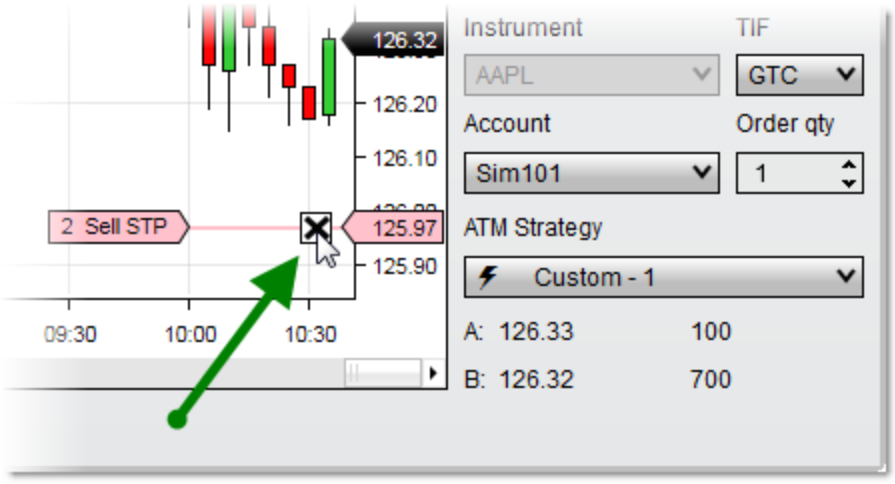
Left mouse click on the area of the order label that displays the order quantity. An order size modification control will appear. Modify the quantity in the quantity field by using either the up/down arrows, the mouse scroll wheel, or by typing the desired quantity. When done, click the check mark to accept the modification, or press the **X** to cancel the modification.



How to cancel an order

Cancelling an Order

To cancel an order, left mouse click on the red **X** on the order marker. Remember that if you cancel an order that is part of an **OCO** group, any other orders in that group will be canceled, as well. For more information on using **OCO** orders with **Chart Trader**, see the "*Understanding 'One Cancels Other' (OCO) Orders*" section of the Submitting Orders page.



10.23.9.5 Attach to Indicator

Chart Trader provides the ability to attach an order to an indicator plot, automatically moving the order in lockstep with the indicator as its plot value changes. This feature can be used for entries as well as resting exit orders such as Stop Losses and Profit Targets.

Play Video

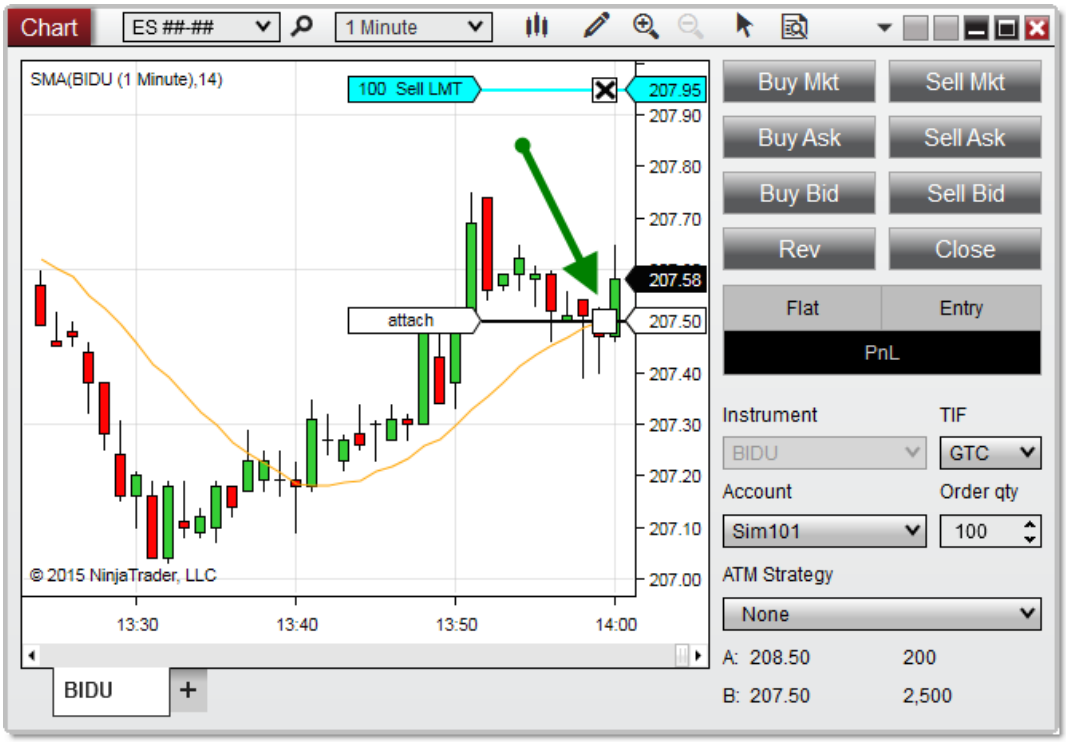
Attach to Indicator

How to attach an order to an indicator

Attaching an Order to an Indicator

Use the following steps to attach an order to an indicator:

1. Apply at least one indicator to your chart, in the same panel as the **Data Series** to which you wish to place the order (See the Working with Indicators page for more information).
2. Apply a resting order to the chart (See the Submitting Orders page for more information).
3. Left mouse click the order label at the left side of the order line.
4. Hold the Ctrl key on your keyboard. You will see a ghost order line matching up with an indicator plot on the chart, with a label that reads "attach."
5. Left mouse click anywhere within the chart while continuing to hold the Ctrl key.
6. The **Attach to Indicator Properties** window will appear, in which you can select the specific indicator to which to attach the order (if more than one is applied to the chart).
7. Change any properties as needed (see the "*Attach to Indicator Properties*" section below), then click the **OK** button.



In the image above, the Ctrl key is held down on the keyboard after left mouse clicking the order label. The next left mouse click will bring up the **Attach to**

Indicator Properties window, which allows us to attach the order to the SMA plot on the chart.

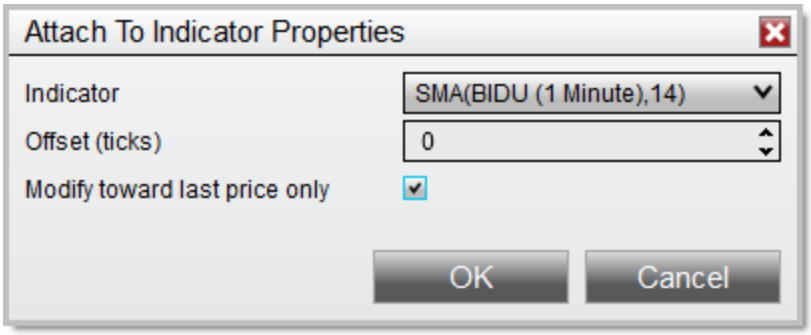
Attach to Indicator properties

Attach to Indicator Properties

The **Attach to Indicator Properties** window can be accessed in one of two ways. Using the process outlined above to attach an order to an indicator will bring up this window automatically, allowing you to set parameters for the indicator tracking before attaching an order. Alternatively, you can use the process outlined below:

1. Right mouse click the label connected to the order line for an order on the chart
2. Hover your cursor over the order listed in the right click menu that appears
3. Select the **Attach to Indicator** menu item
4. Select the **Properties** menu item

The **Attach to Indicator Properties** window allows you to set the following properties:



Indicator	Allows the selection of a specific indicator to which to attach an order
Offset	Sets an offset value to allow an order to trail above or below an indicator plot
Modify toward last price only	Restricts an order to only move towards the last traded price as it follows an indicator plot, never further away

10.23.9.6 Chart Trader Properties

Many of the visual display settings of **Chart Trader** can be customized using the **Chart Trader Properties** window.

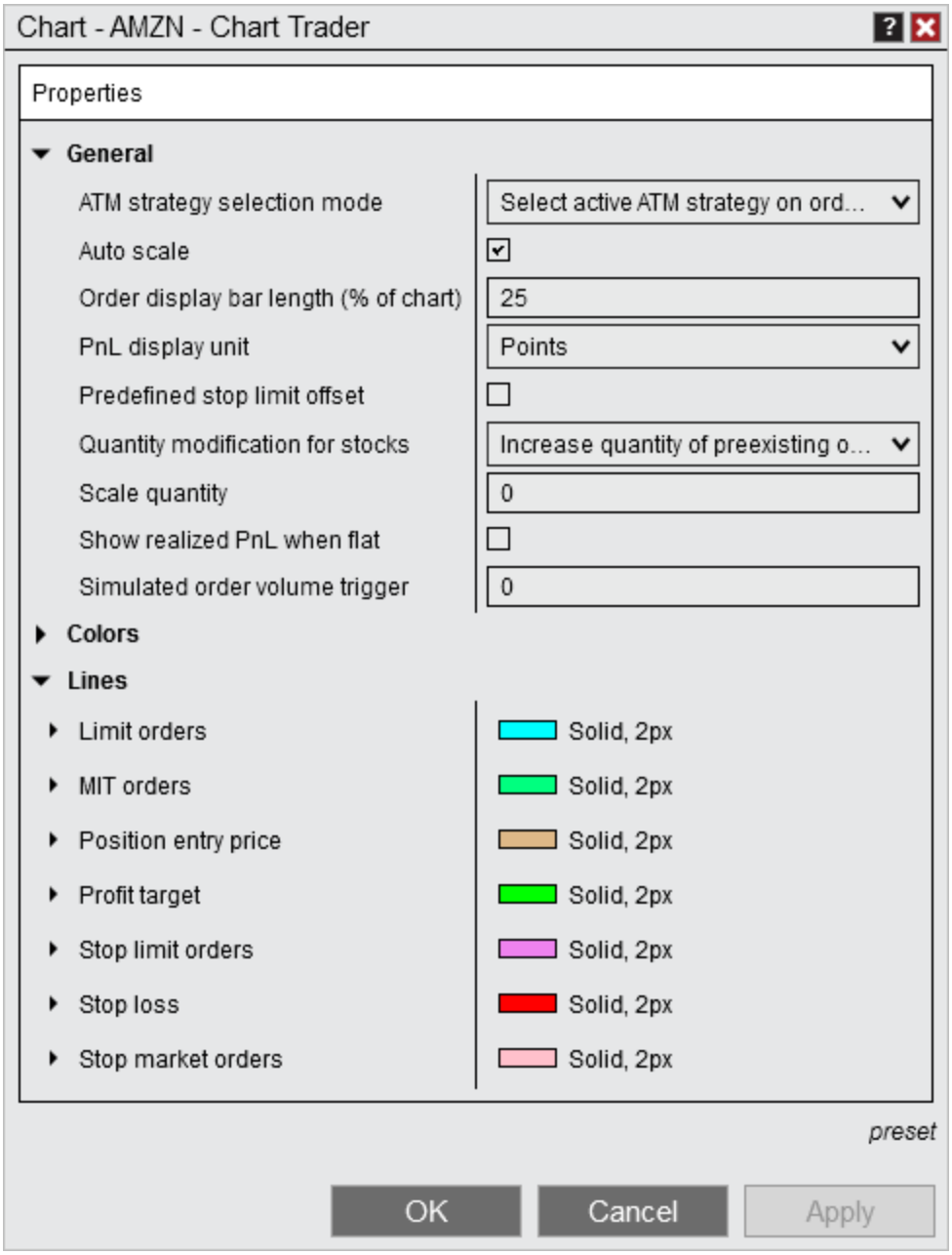
How to access the Chart Trader Properties window

To access the **Chart Trader Properties** window:

1. Right mouse click within the **Chart Trader** panel
2. Select the **Properties** menu item

Available properties and definitions

The following **Chart Trader** properties are available for configuration within the Chart Trader Properties window:

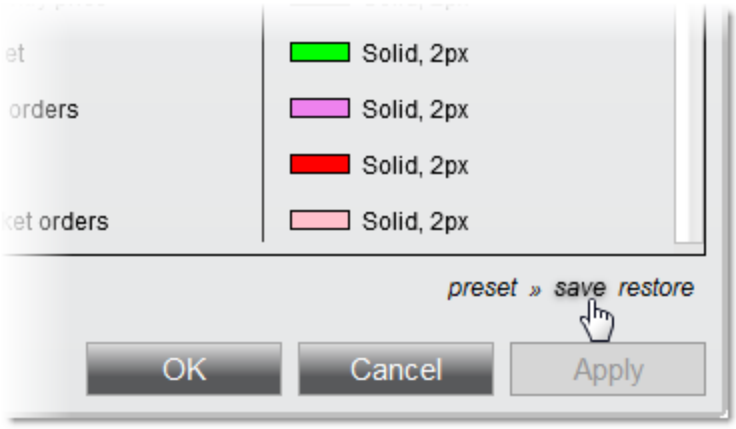


ATM Strategy selection mode	Sets the behavior of the **ATM Strategy** dropdown menu (see the ATM Strategy Selection Mode page for more information)
Auto scale	Enables or disables the inclusion of orders and positions in the chart's auto scaling

Order display bar length (% of chart)	Sets the length an order bar is displayed horizontally across the chart as a percentage
PnL display unit	Sets the display unit for profit loss in currency, percent, ticks, pips, or points
Predefined stop limit offset	Enables or disables the to set the offset the limit price is away from the stop price for entry/exit stop-limit orders. When enabled another property called **Predefined stop limit offset value** will appear to set the offset value.
Quantity modification for stocks	Set whether new orders submitted to price levels where an orders already exists will increase the original orders size or be submitted as a separate order (only applies to stocks)
Scale quantity	Sets the scale order quantity amount
Show realized PnL when flat	Displays realized profit and loss for the selected account in the PnL field when flat
Simulated order volume trigger	Sets the value for a simulated order volume trigger (for entry and exit orders, **NOT** Stop Loss orders)
Colors	Sets the colors to be used for various Action buttons
Lines	Sets the color, dash style, and width of lines used to represent specific order types

▼ How to set the default properties

Once you have your **Chart Trader** properties set to your liking, you can left mouse click on the **preset** text, then click the **save** option to save these properties as default.



If you change your settings and later wish to go back to the original settings, you can left mouse click on the **preset** text, then click the **restore** option.

10.23.1 FX Pro

The **FX Pro** window can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting the **FX Pro** menu item.

FX Pro Overview

The FX Pro order entry window is comprised of several components: the Order Grid, the Level II panel (optional), position and level 1 display, as well as order entry and ATM Strategy management.

Display

› Display Overview

Misc

› Properties

Order and Position Management

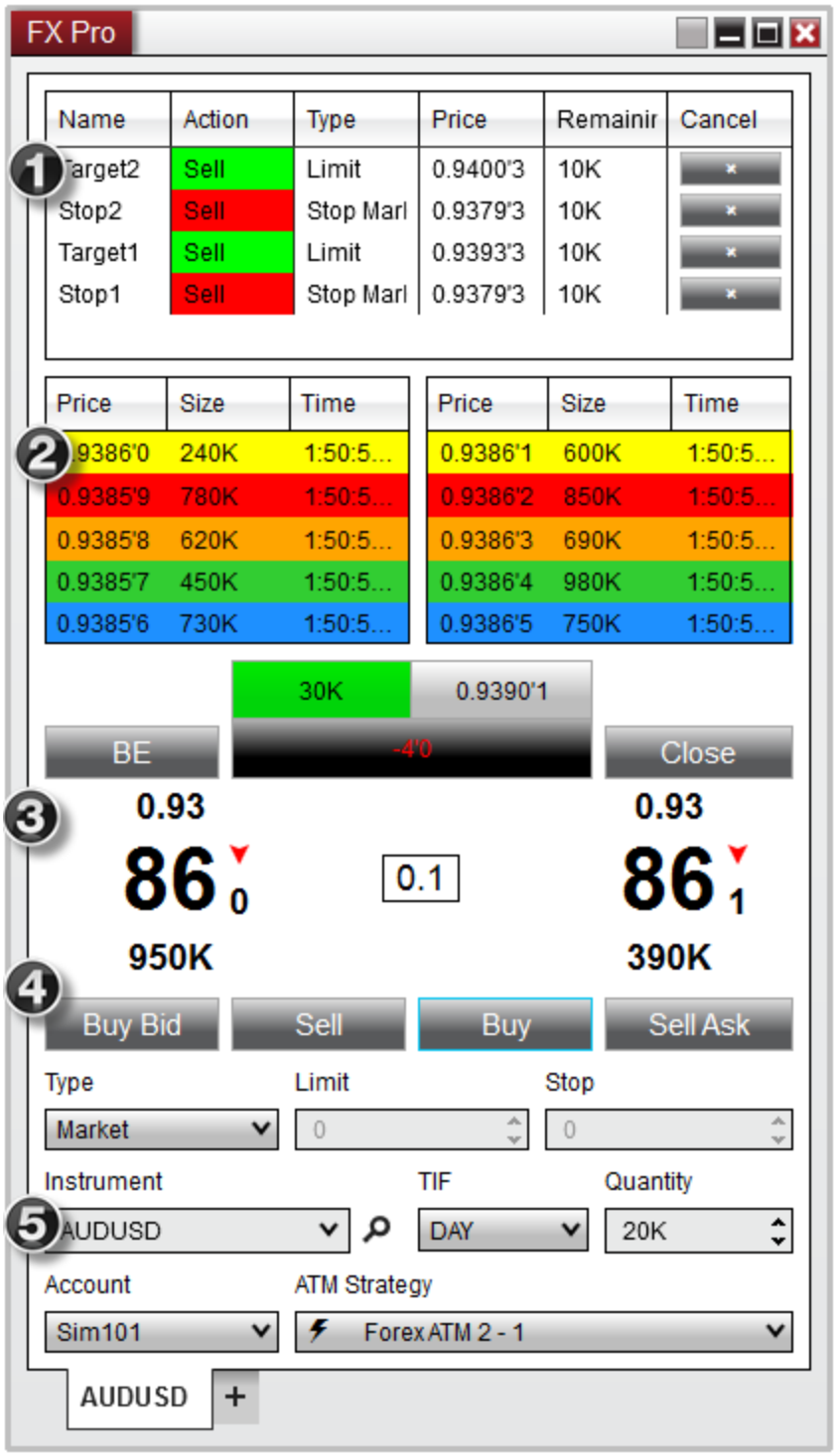
› Submitting Orders
› Modifying and Cancelling Orders
› Managing Positions

10.23.10.1 Display Overview

To open the **FX Pro** window, select the **New** menu from the NinjaTrader Control Center. Then left mouse click on the menu item **FX Pro**.

The image below shows each of the 5 sections in the **FX Pro** window

1. Order Grid
2. Optional Level II panel
3. Position and Level 1 (current inside market) display
4. Action Buttons
5. Order entry and **ATM Strategy** management



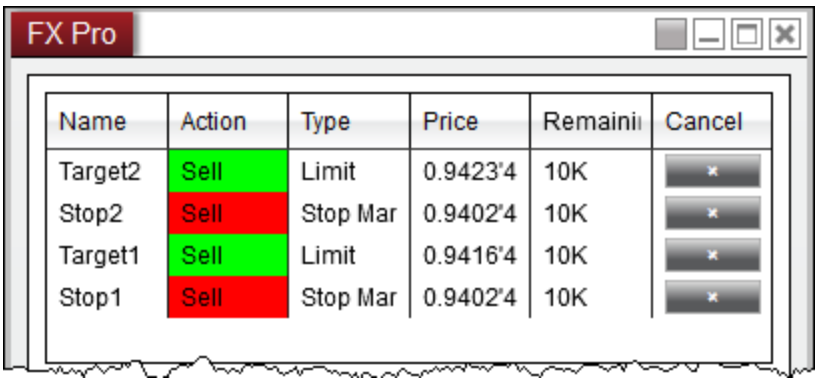
Please see the sections below for more information on each of these sections.

Note: Positions and orders will only display for the selected Account and Instrument.

Understanding the order grid section

Order Grid Display

The **Order Grid** displays active orders for the account and instrument selected in the **Basic Entry** window.



Column Definitions

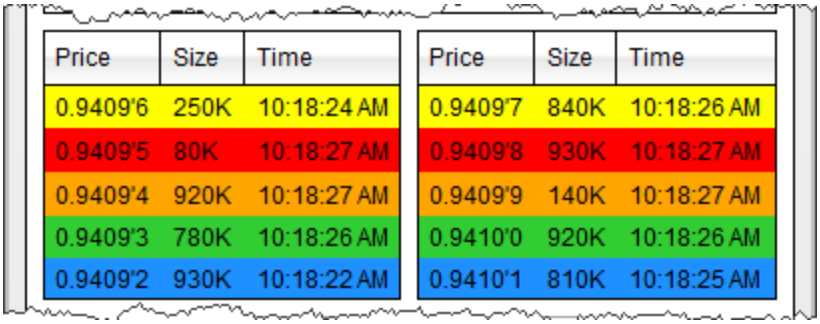
Name	Order name such as Stop1 or Target1
Action	Buy or Sell
Type	Order type
Price	Order price
Remaining	Number of contracts/shares remaining to be filled
Cancel	Cancels the order(s)

Understanding the level II (market depth) section

Level II (Market Depth) Display

The Level II panel displays bid and ask market depth data color coded by price. The Level II display can be enabled/disabled by right mouse clicking inside the **Level II** display and selecting the menu item **Show Level II**.

Note: the Level II panel that is displayed only for brokerages that support ECN style FX trading. If your brokerage does not support ECN FX then you will not see this panel on your FX Pro window.



Price	Size	Time	Price	Size	Time
0.9409'6	250K	10:18:24 AM	0.9409'7	840K	10:18:26 AM
0.9409'5	80K	10:18:27 AM	0.9409'8	930K	10:18:27 AM
0.9409'4	920K	10:18:27 AM	0.9409'9	140K	10:18:27 AM
0.9409'3	780K	10:18:26 AM	0.9410'0	920K	10:18:26 AM
0.9409'2	930K	10:18:22 AM	0.9410'1	810K	10:18:25 AM

Column Definitions

Price	The bid or ask price. The bid data is shown in the left section and the ask in the right.
Size	The number of lots at that price level available to buy or sell (represented in short hand notation where K represents 1000)
Time	The last time the bid/ask was refreshed

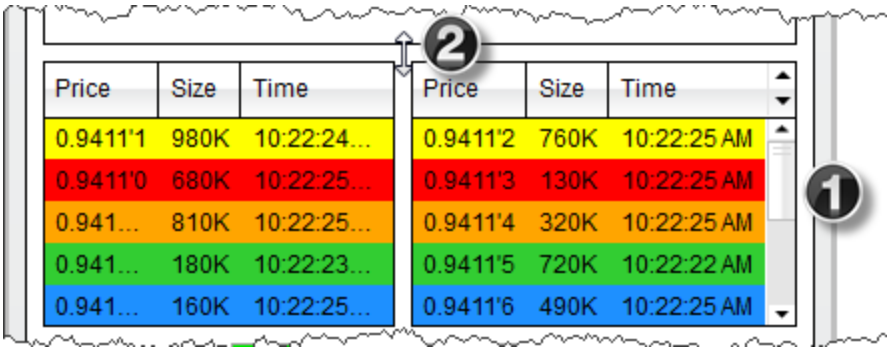
If a price is at a sub pip level, the sub-pip value is displayed as a value after an apostrophe, as in the following example in which the sub-pip value is highlighted in red.

Example: 1.4115'**5** (The price is at 1.4115 pips plus 5 half pips)

Customizing the Number of Price Levels Displayed

By default the **Level II** display in the **FX Pro** window will display five rows of market depth, however you can configure the **Level II** display to show additional rows by following the steps below:

1. Right mouse click inside the **FX Pro** window
2. Select **Properties**
3. Input the desire number of rows in the **Number of price levels** field
4. Press **OK**



Tip: Depending on the size of your **FX Pro** window, the additional rows can potentially extend below the viewable range of the Level 2 display, at which point a **scroll bar** (1) will appear to allow you to access those levels. You may optionally resize the Level 2 display by clicking on and dragging on the **section splitter** (2) between the **Order Grid** and **Level 2** display.

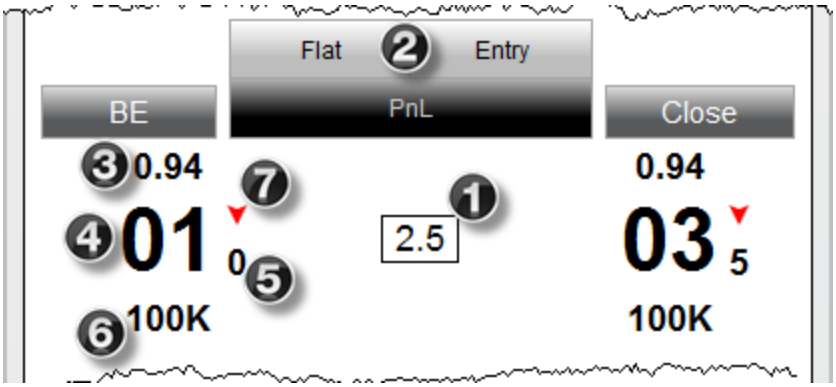
Understanding the market display section

Market Display

The Market Display panel shows the inside bid and ask along with current position information.

Market Display Definitions

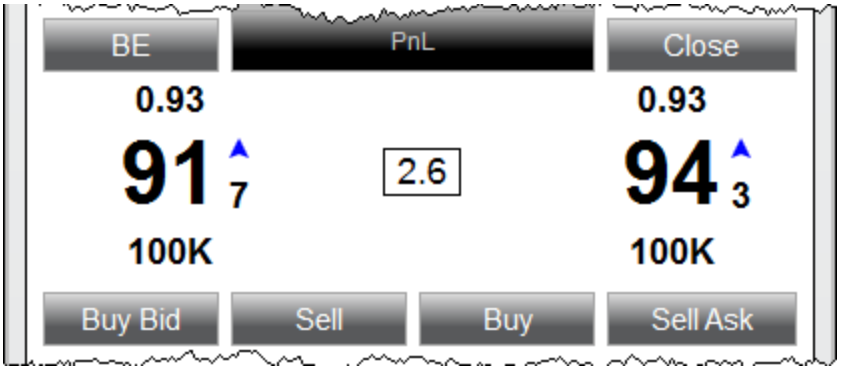
1. The current spread between the best bid and best ask (the image below is showing a spread of 2.5 pips)
2. Position information
3. The handle of the current bid (current ask is on the right side of the spread)
4. The current bid
5. Tenth-pip value. In the image below, the current bid is 0.9401 and 0/10 of a pip displayed as 0.9401'0
6. Current volume (displayed as 100K when volume is not available)
7. The direction of the last tick (blue up arrow for an uptick, red down arrow for down tick)



Note: FX brokerage technologies that do not support an ECN model will **NOT** display sub pips, nor will bid/ask volume be displayed.

Understanding the action buttons section

The **FX Pro** has several buttons which are used to invoke a number of order related actions.



BE (Break-even)	Adjusts any open stop orders opposite of your position to the positions average entry price
Close	Closes the current position and cancel any working orders associated to the instrument/account combination.

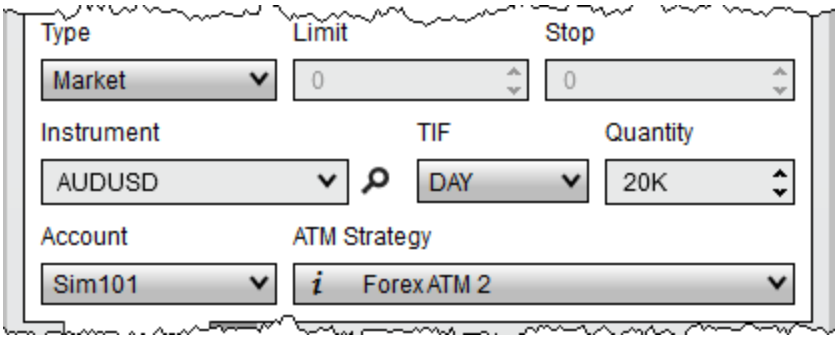
Buy Bid	Submits a Buy Limit order at the best bid price
Sell	Submits a Sell order based on the current Order Controls configured
Buy	Submits a Buy order based on the current Order Controls configured
Sell Ask	Submits a Sell Limit order at the best ask price

Please see the Submitting Orders section for more information on using these buttons.

Understanding the order control section

Order Entry Controls

The **Order Control** section of the **FX Pro** is used to specify several attributes for a pending order to be submitted.



Type	Selects the order Type to be submitted
Limit	Sets the order Limit price, if applicable
Stop	Sets the order Stop Price, if applicable
Instrument	Sets the Instrument

TIF	Sets the order Time in Force
Quantity	Sets the order Quantity
Account	Sets the Account
ATM Strategy	Selects the ATM Strategy

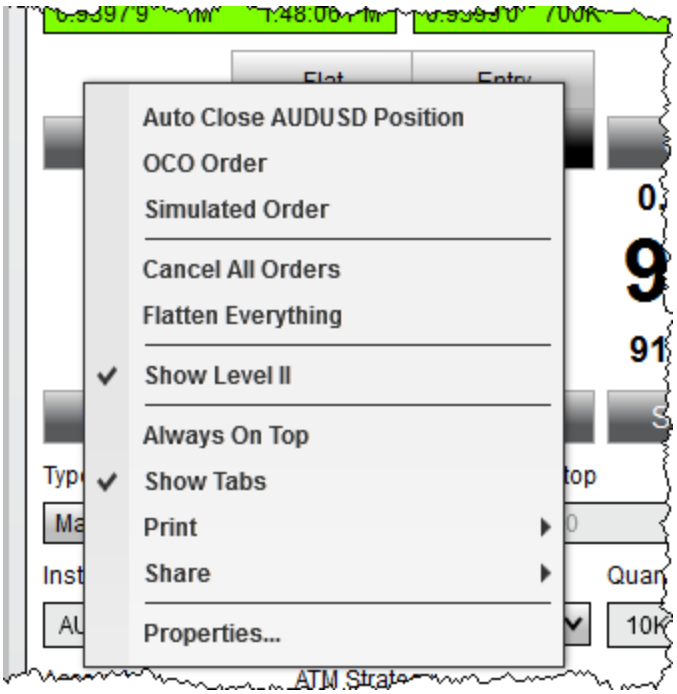
Understanding the right click menu

The **FX Pro** window has two right click menus, depending on where you click:

- Right clicking on the **FX Pro** window itself will bring up menu items specific to the **FX Pro**
- Right clicking in the **Order Grid** will bring up menu items specific to orders

FX Pro Control Right Click Menu

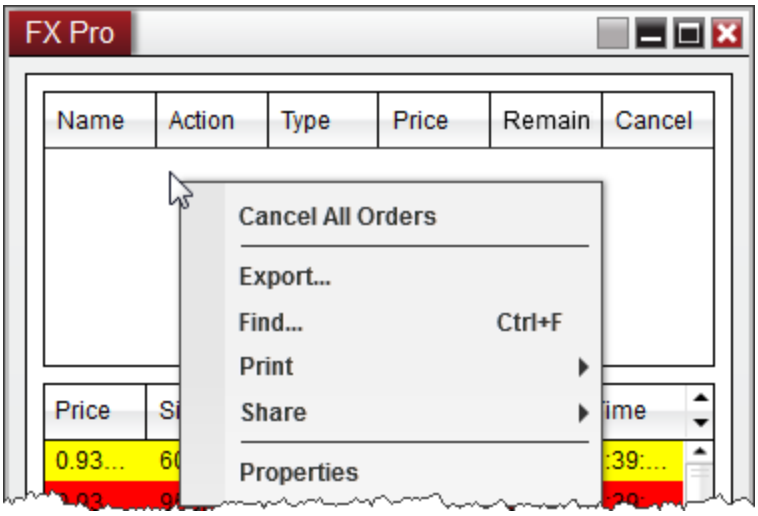
Right clicking on the **FX Pro** window itself will bring up a number of menu items specific to the **FX Pro**



Auto Close Position	Automatically Closes the current instruments position at a specified time
OCO Order	Enables/Disables the OCO (one cancels other) function for a pending order
Simulated Order	Enables/Disables the Simulated Order functionality for a pending order
Cancel All Orders	Cancels all active orders on the current account
Flatten Everything	Closes all open positions and cancels all open orders on every account associated with NinjaTrader
Show Level II	Enables/Disables the Level II display panel
Always On Top	Sets if the window should be always on top of other windows
Show Tabs	Sets if the window should allow for tabs
Print	Displays Print options
Share	Select to share via your share connections
Properties	Configure the FX Pro window properties

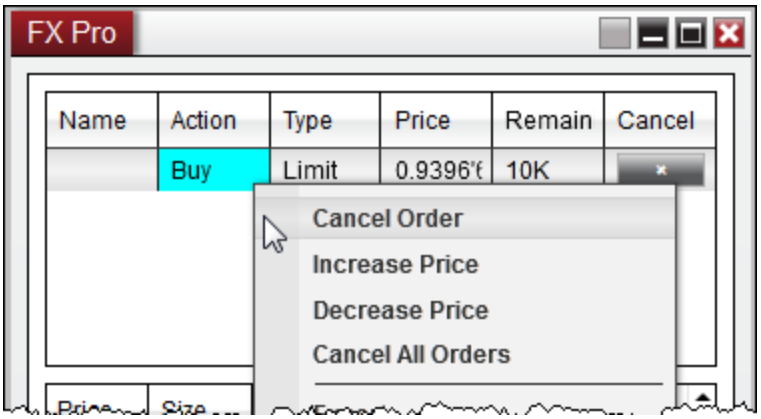
Order Grid Control Right Click Menu

Right clicking in an empty grid will bring up a number of general menu items specific to the **Order Grid**



Cancel All Orders	Cancels all active orders on the current account
Export...	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Displays Print options
Share	Select to share via your share connections
Properties	Configure the FX Pro Window's properties

By moving your mouse cursor over an order and pressing down on your right mouse button, you will see a context menu listing all individual orders consolidated at the corresponding price and any relevant actions that you can perform on those orders.



Cancel order	Cancels the individual order selected
Increase Price	Changes the price of the order +1 tick
Decrease Price	Changes the price of the order -1 tick
Cancel All Orders	Cancels all active orders on the current account

10.23.10.2 Submitting Orders

The FX Pro window is designed for efficient order-entry. In addition to entry and exit orders, the FX Pro window also offers access to NinjaTrader's ATM Strategies. For more information on ATM Strategies, please see the Advanced Trade Management page or attend one of our free live training events.

▼ Selecting instruments and account

How to Select an Instrument

There are multiple ways to select an Instrument in the **FX Pro** window.

- Select the **Instrument Selector** to open a list of recently used instruments or instruments contained in a predefined list
- With the **FX Pro** window selected begin typing the instrument symbol directly on the keyboard. Typing will trigger the **Overlay Instrument Selector**.

For more Information on instrument selection and management please see Instruments section of the Help Guide.

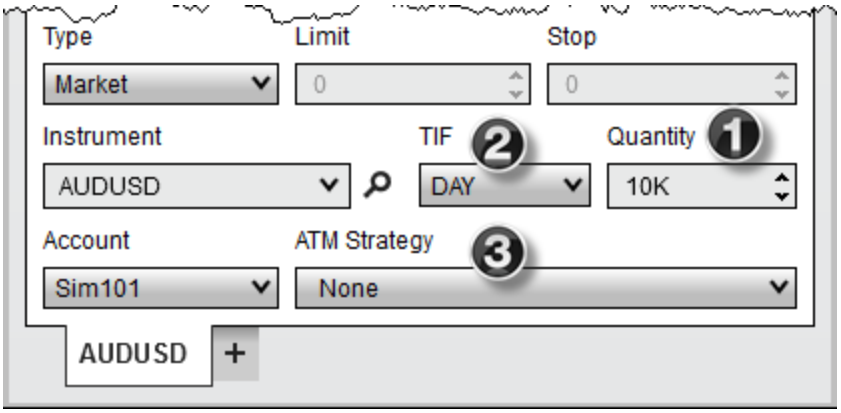
How to Select an Account

A list of all connected accounts will be listed in the "**Account**" drop down list. To change the account select the account you wish to trade through via this drop down list.

Understanding order settings

To Submit an Order

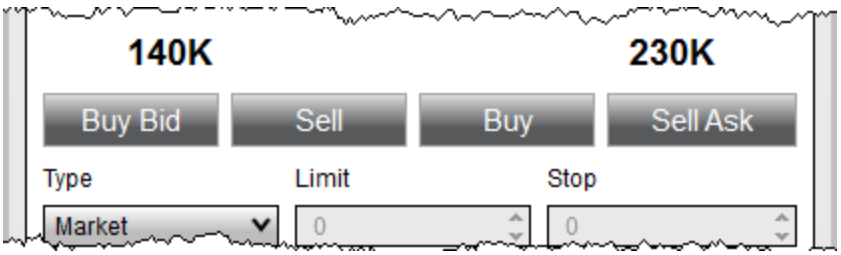
1. Set the order "**Quantity**" field (info)
2. Set the **"TIF"** (Time in Force) field (info)
3. Set the **"ATM Strategy**" (info)
4. Enter an order with any of the methods described below



How to submit orders with quick buttons

Quick Buttons

You can enter orders rapidly by pressing on any one of the quick order buttons.



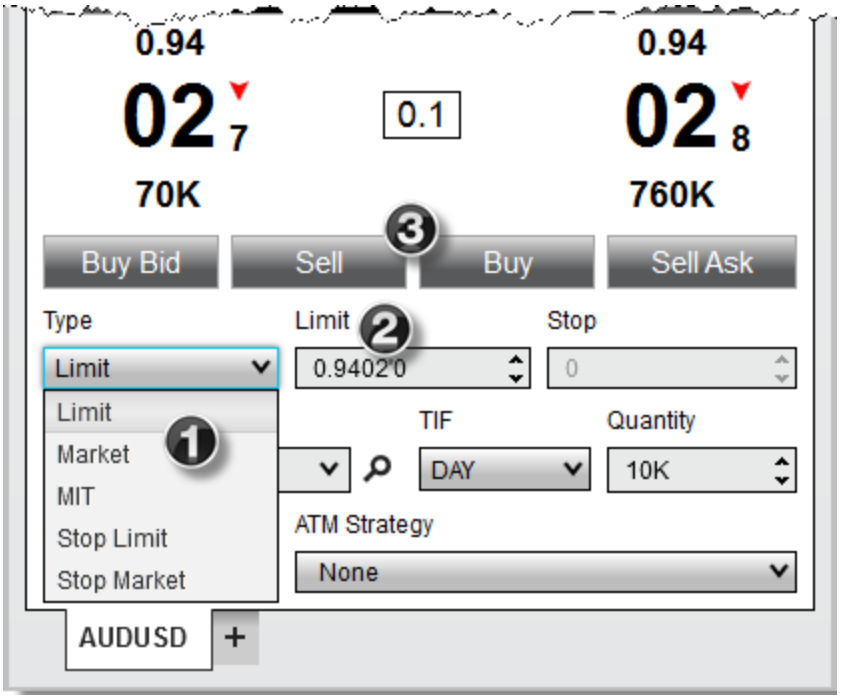
Buy Bid	Submits a Buy Limit order at the best bid price
Sell	Submits a Sell order based on the current **Order Controls** configured
Buy	Submits a Buy order based on the current **Order Controls** configured
Sell Ask	Submits a Sell Limit order at the best Ask price

How to submit custom orders

Custom Orders

You can place a custom order by setting order parameters.

1. Select the order **Type**
2. Set the **Limit** price if applicable
3. Set the **Stop** price if applicable
4. Left mouse click either the **BUY** or **SELL** button



Tips:
1. You can quickly retrieve the current bid or ask price in the Limit and Stop price fields using the following commands:

   - CTRL + middle click in the filed to retrieve the best ask price
   - ALT + middle click in the field to retrieve the best bid price

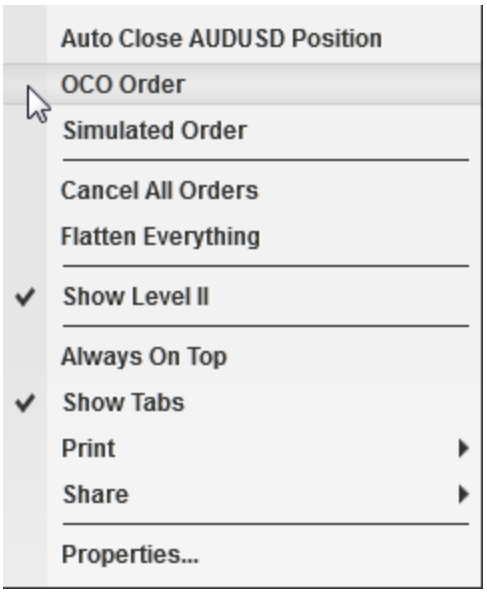
2. Hold down the CTRL key when increasing/decreasing limit/stop prices to change the price in steps of one-pip increments (10 tenth-pips)
3. Left clicking on a price in the Level II panel will load that price into the limit and stop price fields automatically.

Understanding the OCO (One Cancel Other) function

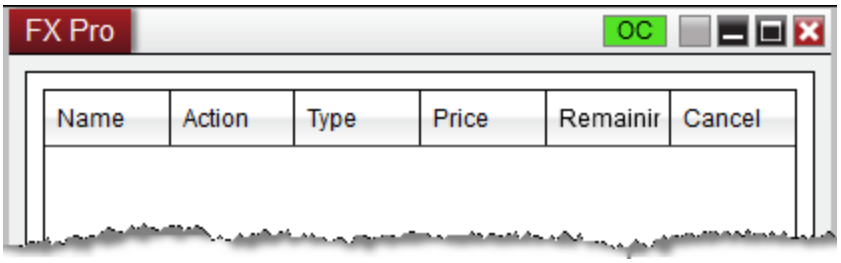
OCO Orders (One Cancels Other)

Stop Loss and **Profit Target** orders (submitted automatically via an ATM Strategy) are always sent as **OCO**, however, you can submit entry or exit orders as **OCO** orders as well. Why? The market may be trading in a channel and you wish to sell at resistance or buy at support, whichever comes first by placing two limit orders at either end of the channel.

To place **OCO** orders, press down on your right mouse button inside the **FX Pro** window and select the menu name "**OCO Order** or use the short cut key CTRL+Z.



The "**OC**" (OCO indicator) will light up green at the top of the **FX Pro** window. All orders placed while this indicator is lit will be part of the same **OCO** group. Once any order of this group is either filled or cancelled, all other orders that belong to this group will be cancelled.



If you want each **OCO** order to create it's own set of **Stop Loss** and **Profit Target** orders ensure that the ATM Strategy control list is set to either **<Custom>** or a strategy template name before you submit each OCO order.

After you have placed your orders, it is advised to disable the **OCO** function via the right click menu, or use the short cut key CTRL+Z.

Warning: If an order which was part of an **OCO** group has already been filled or cancelled, you will need to submit the pending order with a new **OCO ID** otherwise the pending order will be rejected.

To reset an **OCO** ID, simply disable the **OCO** function, and re-enable. This will generate a new **OCO ID** and allow you to place new orders.

Break Out/Fade Entry Example

One of the great features of NinjaTrader is its ability to submit two entry orders, one of which will cancel if the other is filled.

You can accomplish a breakout/breakdown approach by:

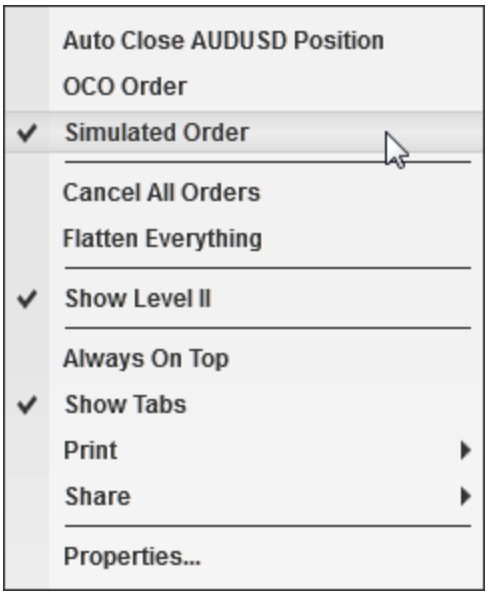
- Right clicking in the **FX Pro** window and selecting the menu item "**OCO Order**" to enable the OCO function
- For your first order, select the desired option from the "**ATM Strategy**" drop down list
- Submit your stop order to buy above the market
- For your second order, select the desired option from the "**ATM Strategy**" drop down list
- Submit your stop order to sell below the market
- *CRITICAL*: Right click in the **FX Pro** window and select the menu item "**OCO Order**" to disable OCO for future orders.

For a market fade approach just substitute limit orders for stop orders.

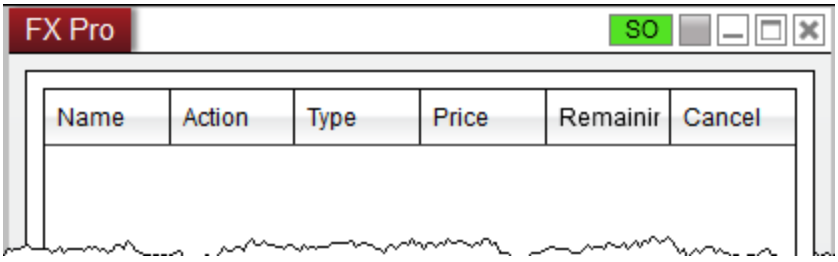
How to submit Simulated Stop Orders (Simulated Order)

Simulated Stop Orders (Simulated Order)

To submit a **Simulated Stop Order** (entry and exit NOT Stop Loss; simulated Stop Loss orders are enabled via an ATM stop strategy) you must enable **Simulated Order** mode via the right mouse click context menu by selecting the **Simulated Order** menu item..



The "**SO**" (Simulated Order indicator) will light up green at the top of the **FX Pro** window. All stop orders placed while this indicator is lit will be submitted as a Simulated Stop Orders.



10.23.10.3 Modifying and Cancelling Orders

You can modify an existing order's quantity, price, or cancel an order entirely from **Order Grid** display of the **FX Pro** window.

Modifying existing orders

Changing the Price of an Order

1. You can increase the price of an order in tenth-pip increments by right mouse clicking on the order in the order grid and selecting "**Increase Price**".

2. You can decrease the price of an order in tenth-pip increments by right mouse clicking on the order in the Order Grid and selecting "**Decrease Price**".

3. Double clicking on the Price field will enable the **price editor** which will allow you to type in a new price manually, or use the scroll wheel on your mouse to select a relative price.

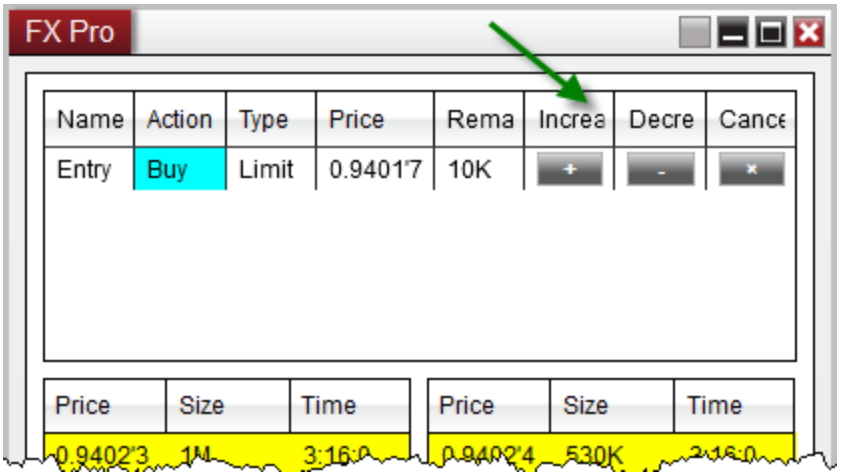
Tip:
Hold down the CTRL key when scrolling in the **price editor** to change the price in half-pip increments.

Enabling Increase and Decrease Columns

You can optionally enable columns on the **Order Grid** display which will allow you to increase or decrease the price of an order using a button click.

To enable these columns:

1. Right click on the **FX Pro** Window and select Properties
2. Expand the "**Columns**" section
3. Check the "**Increase**" and/or the "**Decrease**" options
4. Press "**OK**"



1. You can increase the price of an order in tenth-pip increments by left mouse clicking on the "**+**" button. Holding the CTRL key down while pressing the "**+**" button will modify the order by half-pip increments, and holding the ALT key will modify the order by one full pip.

2. You can decrease the price of an order in tenth-pip increments by left mouse clicking on the "**-**" button. Holding the CTRL key down while pressing the "**-**" button will modify the order by half-pip increments, and holding the ALT key will modify the order by one full pip.

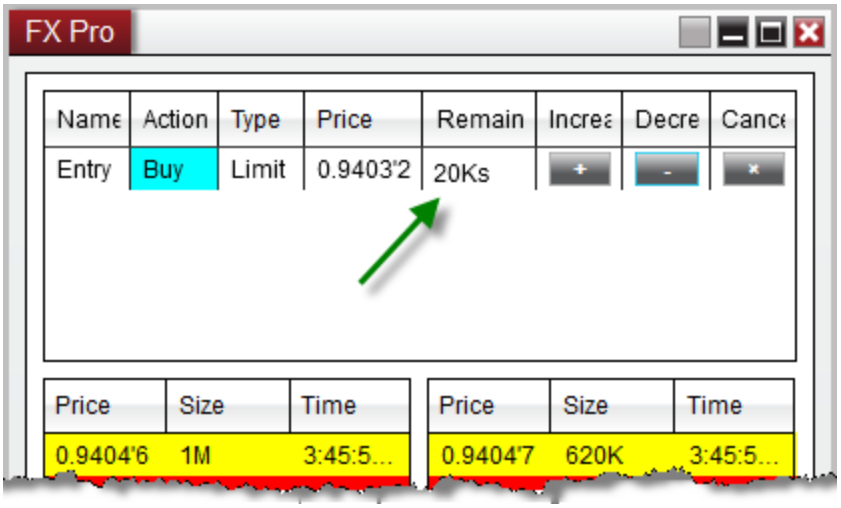
Changing the Quantity of an Order

You can change the size of an order by double left clicking in the "**Remaining**" column, typing in a new quantity value and pressing the "Enter" key on your keyboard.

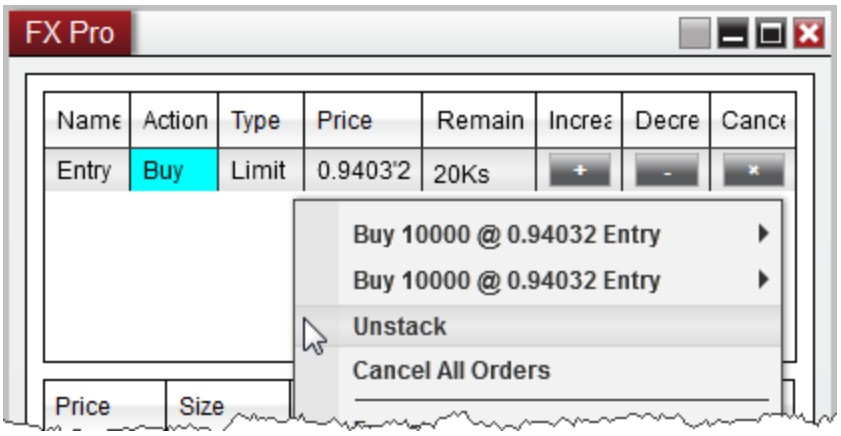
You can also use the scroll wheel on your mouse, or left mouse click on the up/down arrows in the remaining field using the up/down arrows to scroll to a new size by 1K (1000).

> **Tip:** Holding down the CTRL key and scrolling will change the FX order size by 100K (100,000)

Changing the quantity of an existing order will submit a new order the same price to preserve your place in queue. Your orders will now show up as "stacked" indicated by the small letter "**s**" next to the order.



If you would like to break up these orders to manage individually, you can right click on the Order and select "**Unstack**"



Cancelling orders

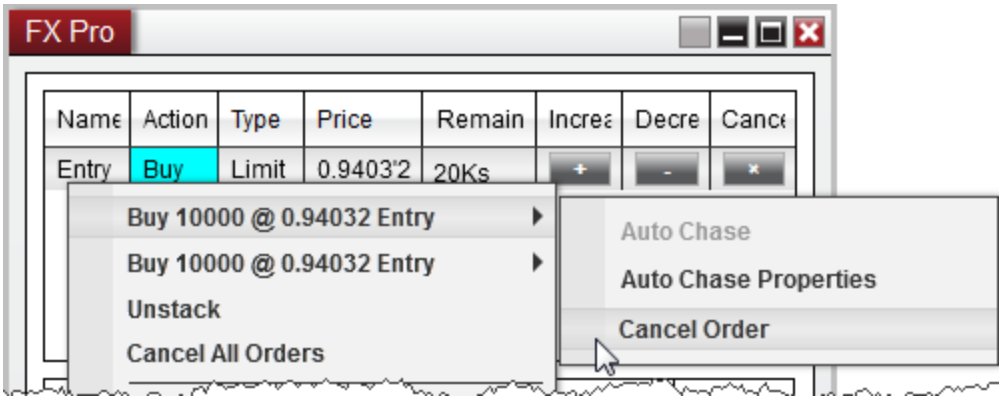
Cancelling an Individual Order

1. You can cancel an order by left mouse clicking on the "**X**" button.
2. You can also right click on the order itself and press the "**Cancel Order**" menu item

Cancelling Stacked Orders

If you have stacked orders, indicated by the small letter "**s**" in the **Remaining** column, you can cancel one of the orders, and leave the other remaining using the steps below:

1. Right click on the stacked order row
2. Move your mouse over the order individual order
3. Select "**Cancel Order**"

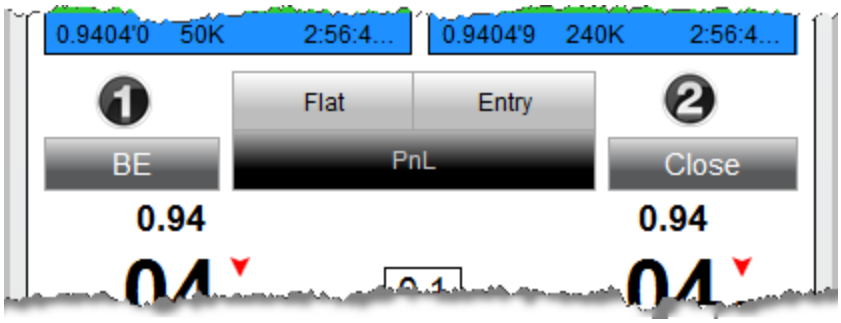


10.23.10.4 Managing Positions

How to Manage Open Positions

1. Clicking on the "**BE**" (break-even) button with your left mouse button will adjust any stop orders in the opposite direction of your open position (if position is long it will adjust stop sell orders) to the position's average entry price. Clicking on this button with your middle mouse button (scroll wheel) will only adjust any Stop Loss orders associated to the selected active **ATM Strategy** in the strategy drop down list. Orders resting at a better price than the average entry price will NOT be modified.

2. Clicking on the "**Close**" button with your left mouse button will close the current position and cancel any working orders associated to the instrument/account combination. Clicking on this button with your middle mouse button (scroll wheel) will close the selected active **ATM Strategy** only. This means that the position size of the **ATM Strategy** will be closed and any working orders associated to that ATM Strategy will be cancelled.



Please see the help topic on Closing a Position or ATM Strategy for more information on the mechanics behind closing various types of positions.

10.23.10.5 Properties

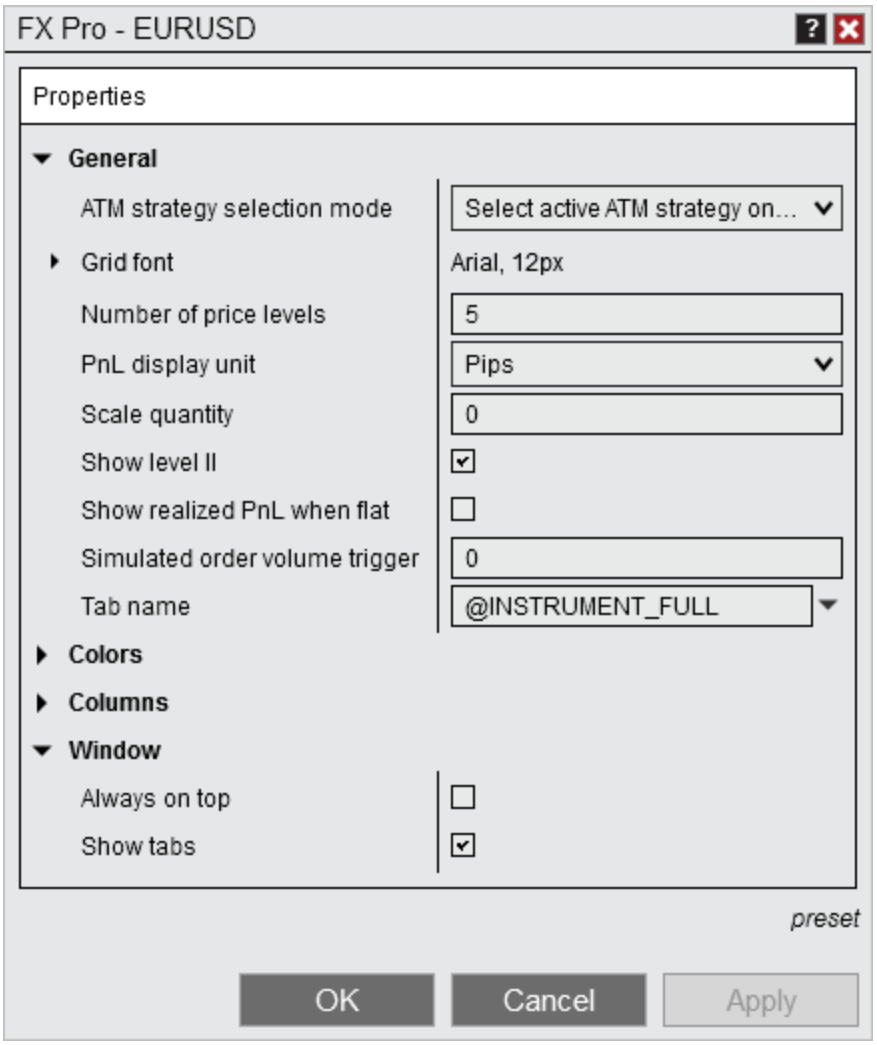
The FX Pro order entry window is highly efficient by design but can also be customized to your preferences through the FX Pro Properties menu.

How to access the FX Pro properties window

You can access the FX Pro properties dialog window by clicking on your right mouse button within the FX Prowindow and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the **FX Pro** Properties window:



Property Definitions

General	
ATM strategy selection mode	Sets the behavior mode of the price ladder display and strategy selector (see more)
Grid font	Sets the font options
Number of price levels	Sets the number of rows shown in the Level II display panel

PnL display unit	Sets the display unit for profit and loss
Scale quantity	Sets the scale order quantity amount.
Show level II	Enables / Disables the Levell II display panel
Show realized PnL when flat	Displays realized profit and loss for the selected account when flat
Simulated order volume trigger	Sets the value for a simulated order volume trigger (for entry and exit orders and NOT used for stop loss)
Tab name	Sets the tab name
Quantity modification for stocks	Sets if modifying an existing stock order quantity changes the the order, or submits a new order
Colors	
Action button	Sets the color for action buttons (CLOSE, BE etc...)
Buy button	Sets the color for all the buy buttons
Order - limit	Sets the color used for limit orders
Order - MIT	Sets the color used for MIT orders
Order - profit target	Sets the color used for profit target orders
Order - stop limit	Sets the color used for stop-limit orders
Order - stop loss	Sets the color used for stop loss orders

Order - stop market	Sets the color used for stop-market orders
Price level (...)	Sets the color used for each individual price color (rows 1 through 10)
Sell button	Sets the color used for all the sell buttons
Columns	
Order Columns (...)	Expanding this section will allow you to disable / enable any of the columns in the Orders Grid display
Window	
Always on top	Sets if the window will be always on top of other windows.
Show tabs	Sets if the window should allow for tabs

How to set the default properties

Once you have your properties set to your preference, you can left mouse click on the "**preset**" text located in the bottom right of the properties dialog. Selecting the option "**save**" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "**preset**" text and select the option to "**restore**" to return to the original settings.

Using Tab Name Variables

Tab Name Variables

A number of pre-defined variables can be used in the "Tab Name" field of the **FX Pro Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.23.1 FX Board

The **FX Board** window can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting the **FX Board** menu item.

FX Board Overview

The **FX Board** is a real-time dealing rates interface which can be used to execute live custom orders for any number of Forex and CFD instruments.

Display

- Display Overview
- Working with Instrument Tiles

Misc

- Properties

Order and Position Management

- Submitting Orders
- Modifying and Cancelling Orders
- Managing Positions

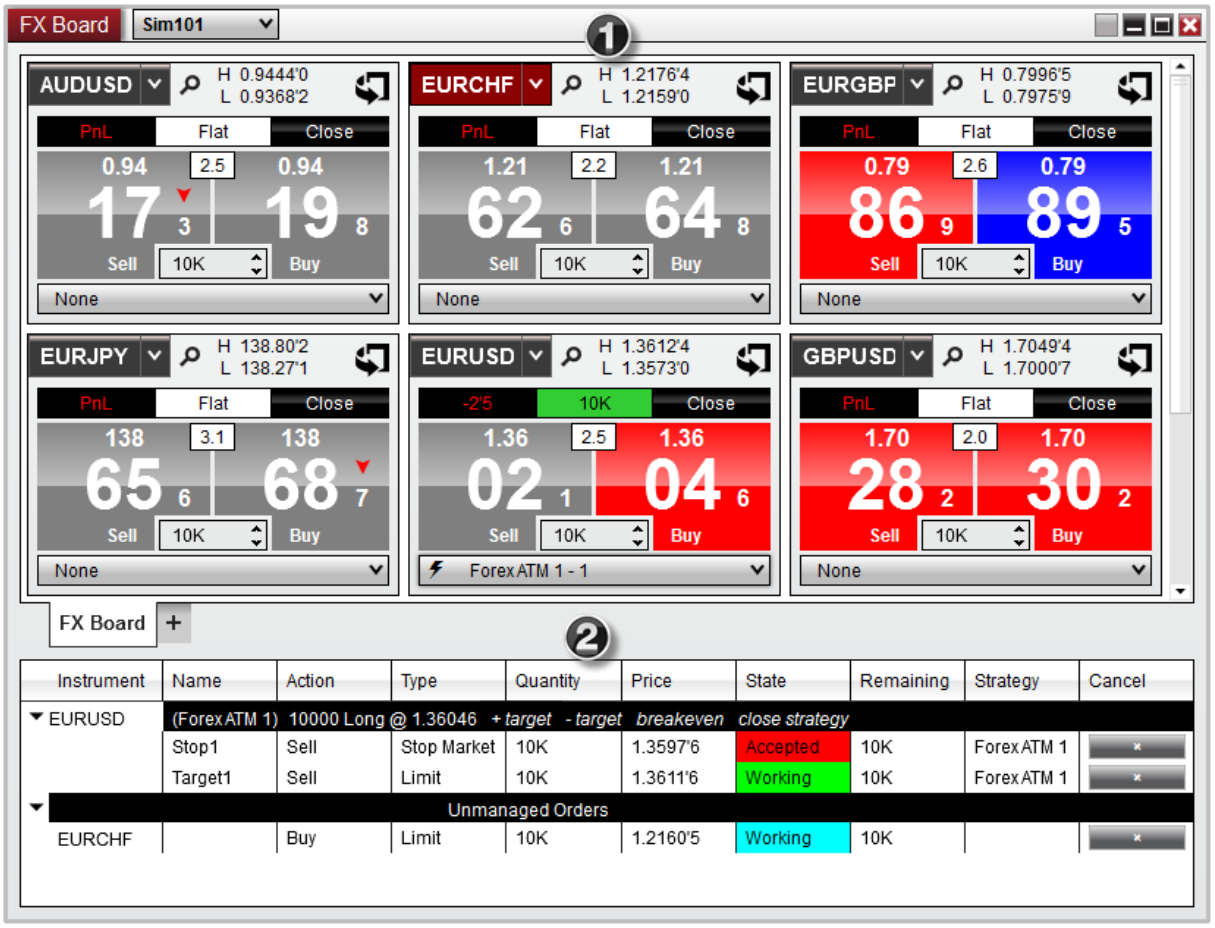
10.23.11. Display Overview

Play Video

To open the **FX Board** window, select the **New** menu from the NinjaTrader **Control Center**. Then left mouse click on the menu item **FX Board**.

The **FX Board** is divided into two sections:

1. Market Data Display
2. Order Grid Display (optional)



Please see the sections below for more information on: **Market Display** and **Order Grid**

Note: Positions and orders will only display for the selected Account and Instruments.

Understanding the market data section

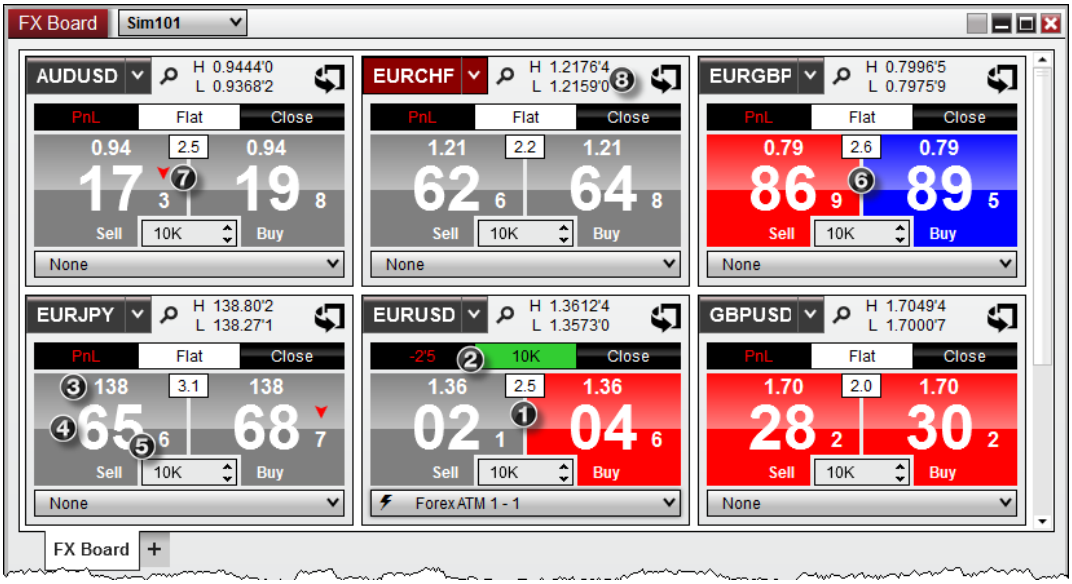
Market Display

The Market Display panel shows the inside bid and ask along with current position information. Each tile will have two panels, representing the bid and ask price, where each respective panel will flash to represent when a tick has been received and the direction of the tick.

Market Display Definitions

1. The current spread between the best bid and best ask (the image below is showing a spread of 2.5 pips)
2. Position information

3. The handle of the current bid (current ask is on the right side of the spread)
4. The current bid
5. Sub pip value
6. The direction of the current tick (blue panel for an up tick, red panel for down tick)
7. The direction of the last tick received (blue arrow for an up tick, red arrow for down tick)
8. Current day high/low values

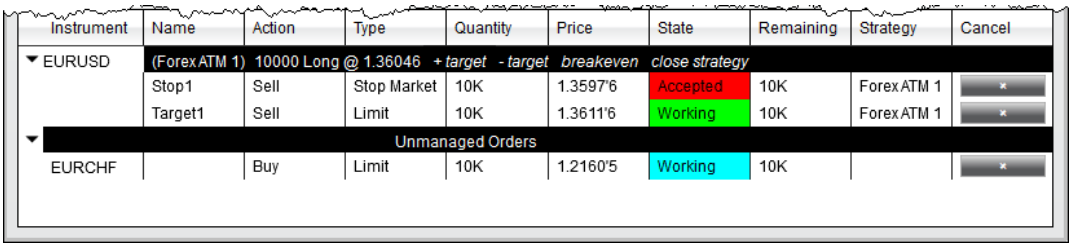


Instrument tiles also give you the ability to quickly place orders, or can be "flipped" to place custom orders. Please see the section on Submitting Orders for more information.

Understanding the order grid section

Order Grid Display

The **Order Grid** displays active orders for the account and instrument tiles selected in the **FX Board** window.



Column Definitions

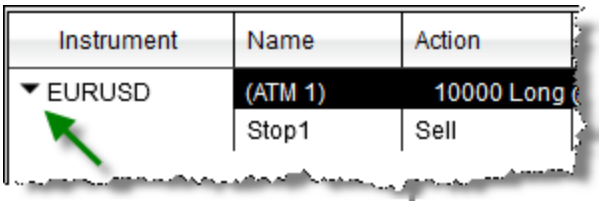
Instrument	Name of the instrument
Name	Order name such as Stop1 or Target1
Action	Buy or Sell
Type	Order type
Quantity	Number of contracts submitted
Price	Order price
State	State of the order
Remaining	Quantity remaining to be filled
Strategy	Name of ATM Strategy associated with the order
Cancel	Cancels the order(s)

Order Grouping

Orders that are submitted to the same instrument will be grouped together in the **Order Grid** and displayed in an aggregated view to consolidate these orders together.

1. Orders that are part of an **ATM Strategy** will be further aggregated by the **ATM Strategy** *template name*, as well as control specific to **ATM Strategies** (see the section on Managing Positions for more information on these controls)
2. Orders that are *not* part of an **ATM Strategy** will be aggregated under the row heading name "*Unmanaged Orders*" indicating that there is no **ATM Strategy** associated with that particular order

Tip: You can collapse orders under an instrument header by selecting the down arrow next to the instrument name.



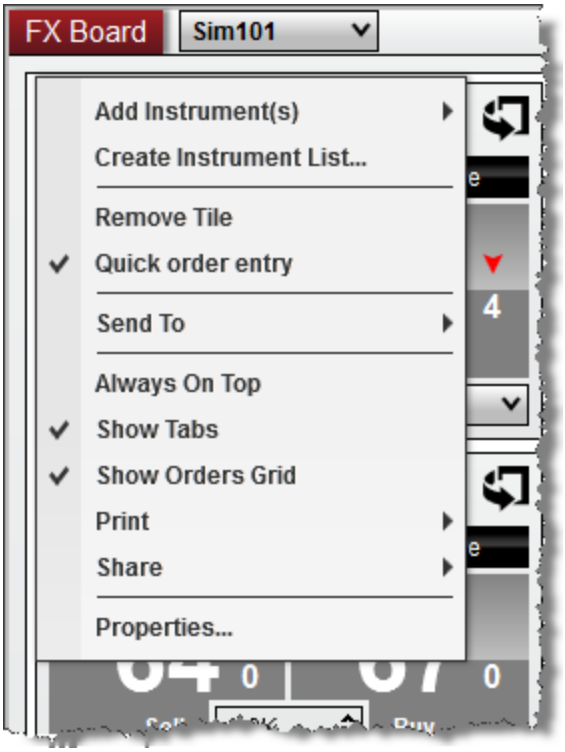
Understanding the right click menu

The **FX Board** window has two right click menus, depending on where you click:

- Right clicking on the **FX Board** window itself will bring up menu items specific to the **FX Board**
- Right clicking in the **Order Grid** will bring up menu items specific to orders

FX Board Control Right Click Menu

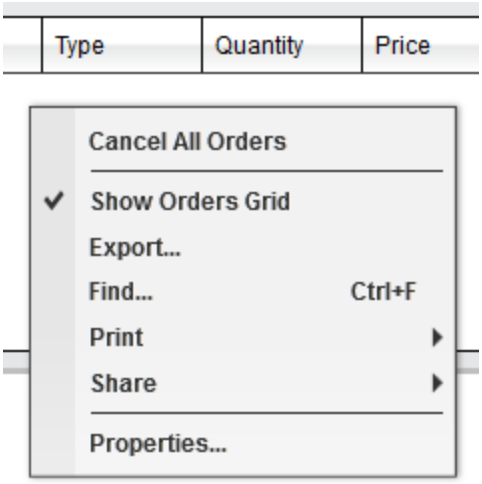
Right clicking on the **FX Board** window itself will bring up a number of menu items specific to the **FX Board**



Add Instrument(s)	Adds an individual instrument or list of instruments to the **FX Board** window
Create Instrument List...	Dynamically creates a list of all the current instruments in the **FX Board** window which can be accessed later
Remove Tile	Removes the current selected instrument tile
Quick order entry	Enables/Disables the ability to place quick orders from the Instrument tiles
Send To	Loads the selected instrument into another NinjaTrader window
Always On Top	Sets the **FX Board** window to always be on top of other windows
Show Tabs	Sets if the window should allow for tabs
Show Orders Grid	Enables/Disables the **Orders Grid** panel display
Print	Displays Print options
Share	Select to share via your share connections
Properties	Configure the **FX Board** window properties

Order Grid Control Right Click Menu

Right clicking in an empty grid will bring up a number of general menu items specific to the Order Grid



Cancel All Orders	Cancels all active orders on the current account
Show Orders Grid	Enables/Disables the **Orders Grid** panel display
Export	Exports the grid contents to "CSV" or "Excel" file format
Find...	Search for a term in the grid
Print	Displays Print options
Share	Select to share via your share connections
Properties	Configure the **FX Board** Window's properties

By moving your mouse cursor over an order and pressing down on your right mouse button, you will see a context menu listing all individual orders consolidated at the corresponding price and any relevant actions that you can perform on those orders.

Cancel Order	Cancels the individual order selected
Increase Price	Increases the price of the order by one tenth-pip
Decrease Price	Decreases the price of the order by one tenth-pip
Cancel All Orders	Cancels all active orders on the current account

10.23.11.1 Working with Instrument Tiles

The **FX Board** can be setup for use with an unlimited number of Forex or CFD **instrument tiles** which are used to display current market data, as well as used for order entry.

▼ Managing instrument tiles

Adding an Individual Instrument

You can add as many individual Forex or CFD **instrument tiles** to your **FX Board** window as you would like.

- Press down on your right mouse button in the **FX Board** window and select the menu item **Add Instrument(s)**. Through the **Instrument Selector** menu, you can navigate through various instrument lists to locate the instrument you desire, and left click on the instrument to add the individual instrument to the **FX Board.**

Adding a List of Instruments

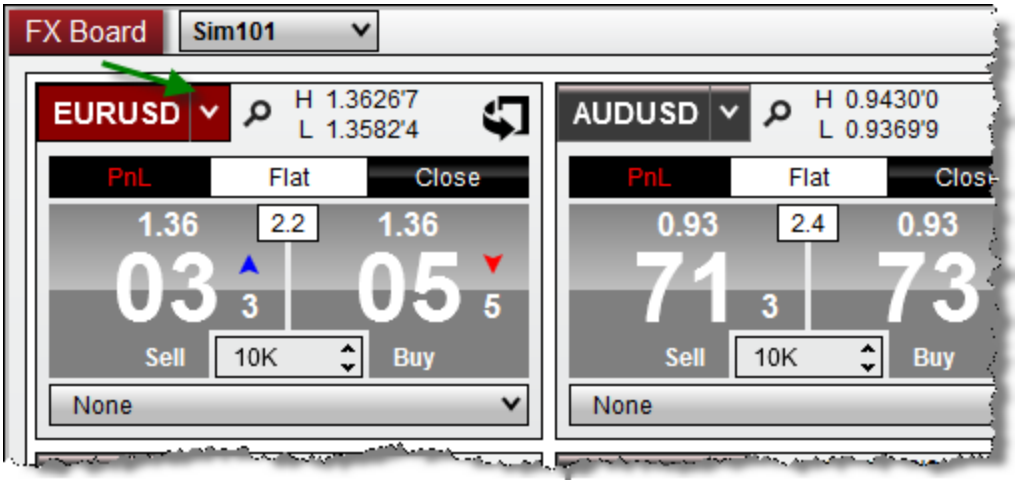
You can also rapidly add an entire list of predefined instruments to the **FX Board** window.

- Press down on your right mouse button in the **FX Board** window and select the menu item **Add Instrument List**. Then select the instrument list you would like to add to the **FX Board**.

Please see the Instrument Lists section of the user help guide for additional information on creating, editing, and deleting instrument lists.

Changing Instruments

Once an **instrument tile** has been added to the **FX Board** display, you can quickly change the instrument by using the **Instrument Selector**.



Arranging Tiles

You can customize the arrangement in which each **instrument tile** is displayed by left clicking and dragging the **instrument tile** to the desired location.

Removing Instruments

To remove an instrument tile, simply right click on the desired tile and select **Remove Tile.**

Creating instrument lists from the FX Board

Creating an Instrument List

If you have a **FX Board** setup with a number of different instruments you would like to save for later, you can quickly add the entire display of instruments into an **Instrument List** for quick access.

- Press down on your right mouse button in the **FX Board** window and select the menu item **Create Instrument List**, then give the **Instrument List** a unique name and press **OK**.

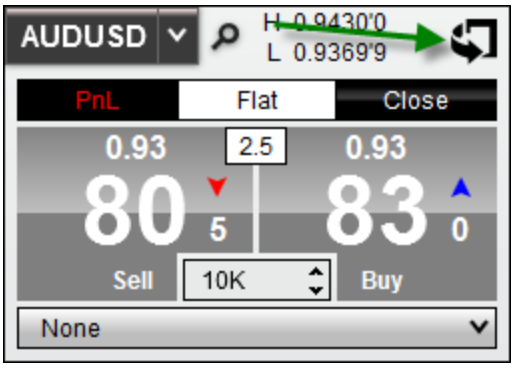
You will now be able to access this list from other features of NinjaTrader using the Instrument Selector. You can further edit this list by using the Instrument Lists window

Flipping tiles for custom order entry

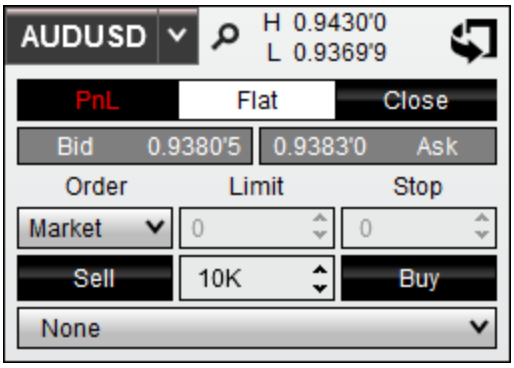
Flipping Tiles

Each **instrument tile** located on the **FX Board** display has a reverse side which can be flipped over allowing for custom order entry.

To flip a tile, simply click the  icon on the top right corner of the **instrument tile.**

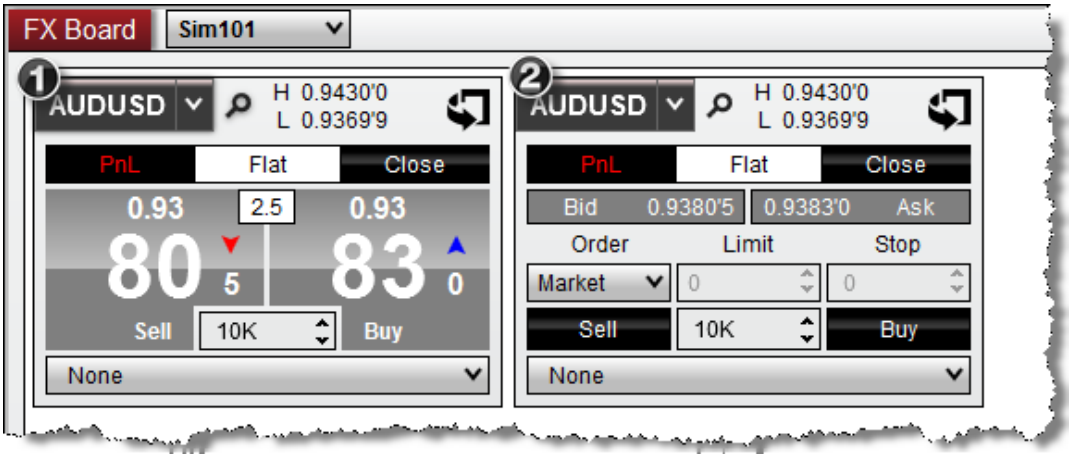


Once the tile has been flipped, you will see a number of additional controls which can be used to define and submit custom orders.



Please see the section on Submitting Orders for more information on how to use these controls.

Tip: It is possible to add two tiles of the same instrument, allowing you to display the ❶ **Market Display** and ❷ **Custom Order Entry** sides simultaneously.



Customizing tiles

Tile size

You can optionally reduce, or increase the size of the **instrument tiles** displayed in the **FX Board** window by choosing one of 3 preset options.

To change the size of the instrument tiles:

- Right click on the **FX Board** window, select **Properties**, locate the **Tile size** property and select: **Small**, **Medium**, or **Large**

Disabling Quick Order Entry

By default, **Sell** and **Buy** buttons of each **instrument tile** will act as a order entry feature to quickly submit and execute market orders. However, this feature can be disabled which will allow you to only use the front of the **instrument tile** for market data and position display.

To disable **Quick Order Entry**, right click on the **FX Board** and uncheck **Quick Order Entry.**

With his configuration, you will still have the ability to flip the **instrument tile** and use the custom controls to place orders.

Highlight Duration

When a new tick is received on an instrument, the **Sell** or **Buy** button of an **instrument tile** will highlight to represent the direction of the current tick. The default duration for this highlight is 1000ms, or 1 second. The **FX Board** will allow you increase, or decrease the amount of time the tile will remain highlighted to your preferences.

To configure this property:

- Right click on the **FX Board** window, select **Properties**, locate the **Highlight duration (ms)** property and input a custom value (in milliseconds).

The higher the value used, the longer the tile will remained highlighted. Setting the **highlight duration** to a value of "0" (zero) will disable highlighting all together.

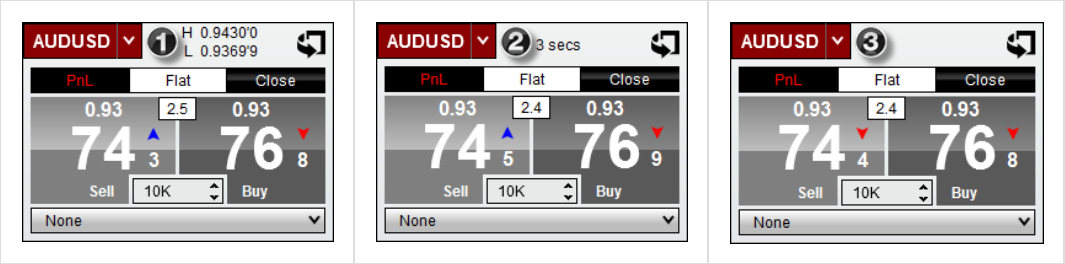
Market Data Display Text

Instrument tiles will display the current **daily high/low** value for the selected instrument as determined by your data provider. However, this text can be changed to display the amount of **time since last tick**, or the text can disabled completely.

To configure this setting:

- Right click on the **FX Board** window, select **Properties**, locate the **Display** property, and choose from one of the following display options in the table below

Daily High low	Time last tick	None



Note: Customizations will apply to the entire **FX Board** window, and will *not* be applied on a per **instrument tile** basis.

For further customization of the **FX Board** window, please see the FX Board Properties topic.

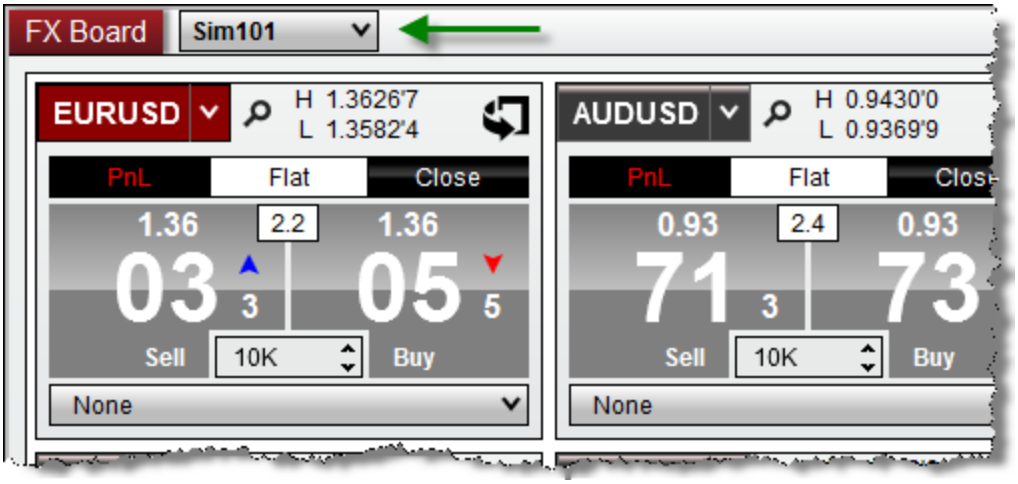
10.23.11.3 Submitting Orders

Submitting orders within the **FX Board** order entry window is both easy and efficient. In addition to entry and exit orders the **FX Board** window also offers access to NinjaTrader **ATM Strategies**. For more information on **ATM Strategies** please see the ATM section of the user help guide or attend one of our free live training events.

Selecting accounts

How to Select an Account

A list of all connected accounts will be listed in the **Account selector** located at the top of the **FX Board** window.

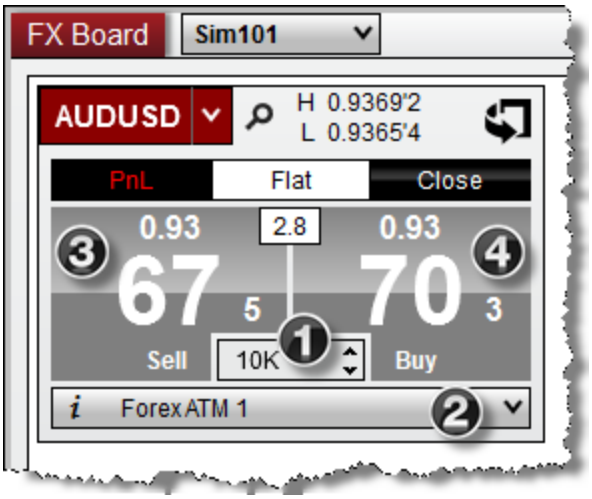


To change the account, select the account name or number you wish to trade through via the **account selector**. The account selected will be the account used for all instruments in the **FX Board** window.

How to submit orders with quick buttons

Quick Market Orders

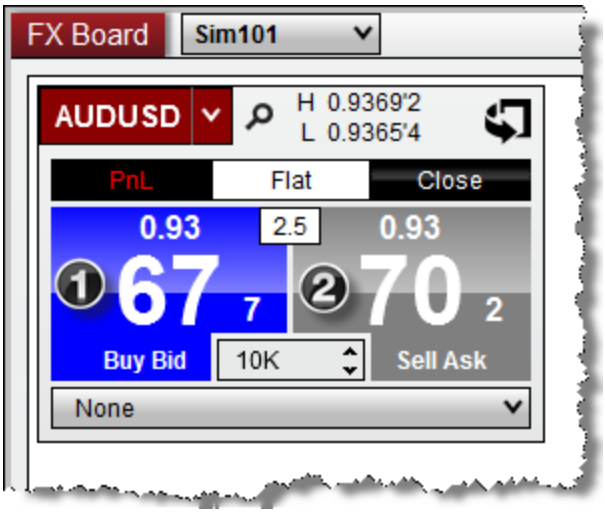
You can rapidly execute market orders directly from an **instrument tile**.



1. Set the order **Quantity** field (info)
2. Set the **ATM Strategy** option (info)
3. Pressing the **Sell button** on the left will execute a Sell Market order
4. Pressing the **Buy button** on the right will execute a Buy Market order

Quick Limit Orders

Holding down the CTRL key will switch the **quick entry button** to submit an order at the bid or ask.



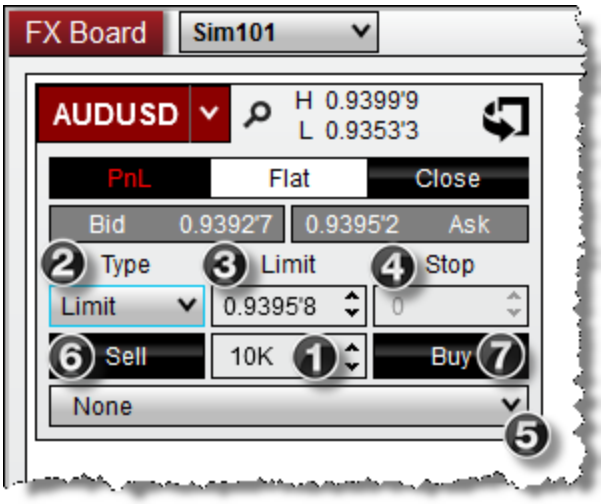
1. Holding down CTRL on your keyboard and pressing the **Buy Bid button** on the left will submit a Buy Limit order at the best bid price
2. Holding down CTRL on your keyboard and pressing the **Sell Bid button** on the right will submit a Sell Limit order at the best ask price.

How to submit custom orders

Custom Orders

You can place a custom order by setting order parameters.

1. Set the order **Quantity** field (info)
2. Select the order **Type**
3. Set the **Limit** price if applicable
4. Set the **Stop** price if applicable
5. Set the **ATM Strategy** option (info)
6. Left mouse click on the **Sell** button to submit a SELL order
   - or -
7. Left mouse click on the **Buy** button to submit a BUY order



Tips

1. You can quickly retrieve the current last, bid, or ask price to be used in the **Limit price field** or **Stop price field** using the following commands:

- Middle click in the price field to retrieve the last traded price
- Left click on the **Bid** button to retrieve the best ask price
- Left click on the **Ask** button to retrieve the best bid price

2. Hold down the CTRL key when increasing/decreasing limit/stop prices to change the price in steps of 10 tick increments.

10.23.11.4 Modifying and Cancelling Orders

You can modify an existing order's quantity, price, or cancel an order entirely from **Order Grid** display of the **FX Board** window.

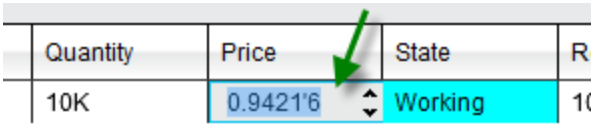
Modifying existing orders

Changing the Price of an Order

1. You can increase the price of an order in tenth-pip increments by right mouse clicking on the order in the **Order Grid** and selecting "**Increase Price**".

2. You can decrease the price of an order in tenth-pip increments by right mouse clicking on the order in the **Order Grid** and selecting "**Decrease Price**".

3. Double clicking in the **price field** will enable the **price editor** which will allow you to type in a new price manually, or use the scroll wheel on your mouse to select a relative price.



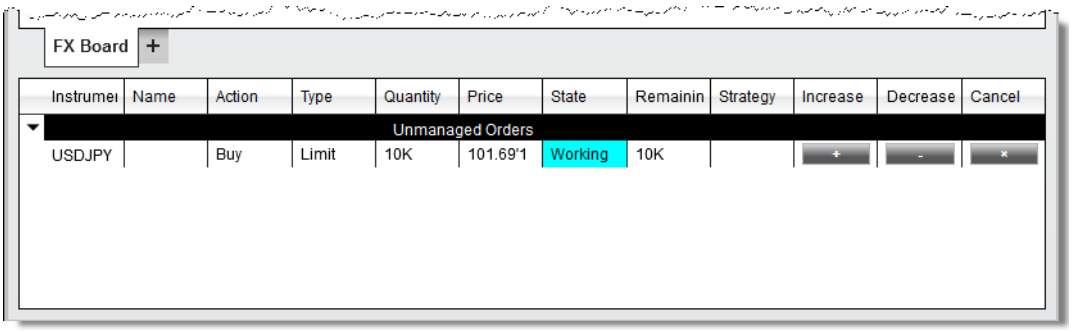
Tip: Hold down the CTRL key when scrolling in the **price editor** to change the price in steps of half-pip increments.

Enabling Increase and Decrease Columns

You can optionally enable columns on the **Order Grid** display which will allow you to increase or decrease the price of an order using a button click.

To enable these columns:

1. Right click on the **FX Board** window and select **Properties**
2. Expand the **Columns - Orders** section
3. Check the **Increase** and/or the **Decrease** options
4. Press **OK**



1. You can increase the price of an order in tenth-pip increments by left mouse clicking on the "**+**" button
2. You can decrease the price of an order in tenth-pip increments by left mouse clicking on the "**-**" button

Tips:
1. Holding the CTRL key down while pressing the "**+**" or "**-**" button will modify the order by half-pip increments

2. Holding the ALT key down while pressing the "**+**" or "**-**" button will modify the order by one pip increments

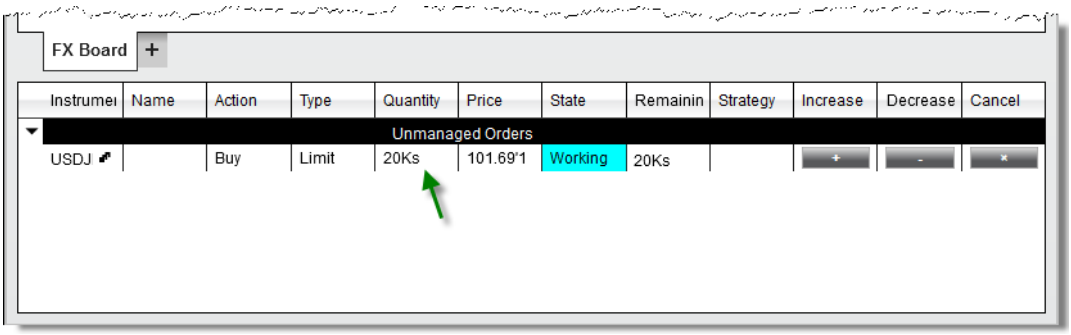
Changing the Quantity of an Order

You can change the size of an order by double left clicking in the either the **Quantity** or **Remaining** column, typing in a new quantity value, and pressing the "Enter" key on your keyboard.

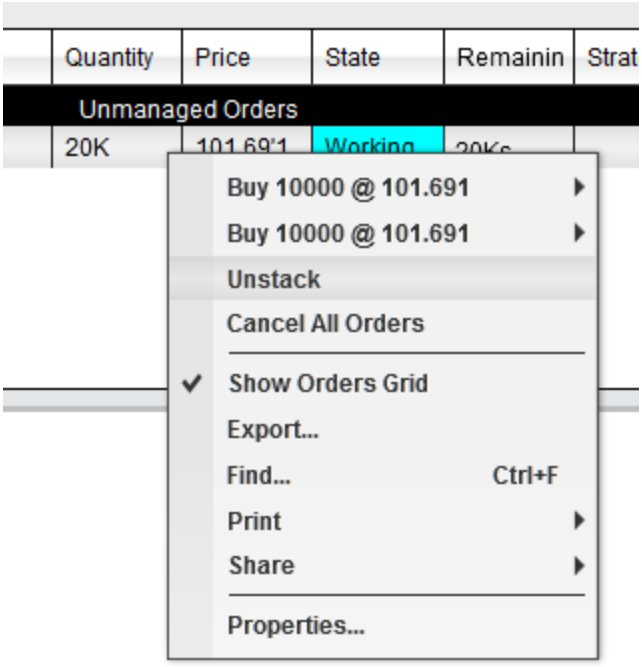
You can also use the scroll wheel on your mouse, or left mouse click on the up/down arrows in the remaining field using the up/down arrows to scroll to a new size by 1K (1000).

Tip: Holding down the CTRL key and scrolling will change the FX order size by 100K (100,000)

Note: Changing the quantity of an existing order will submit a new order the same price to preserve your place in queue. Your orders will now show up as "stacked" indicated by the small letter "**s**" next to the order.



If you would like to break up these orders to manage individually, you can right click on the order row and select **Unstack**



Cancelling orders

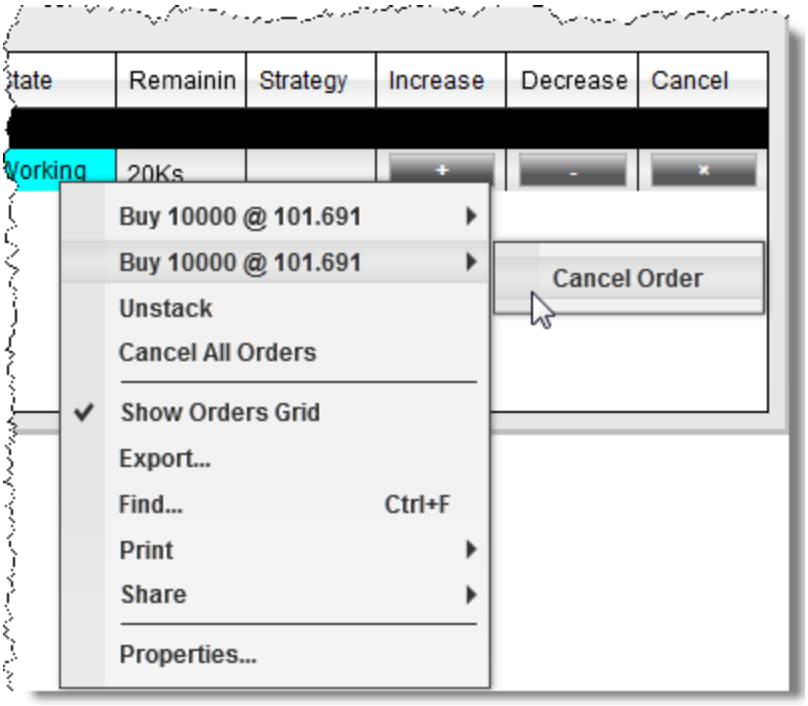
Cancelling an Individual Order

1. You can cancel an order by left mouse clicking on the "**X**" button.
2. You can also right click on the order itself and press the "**Cancel Order**" menu item

Cancelling Stacked Orders

If you have stacked orders, indicated by the small letter "**s**" in the **Quantity** column, you can cancel one of the orders, and leave the other(s) remaining using the steps below:

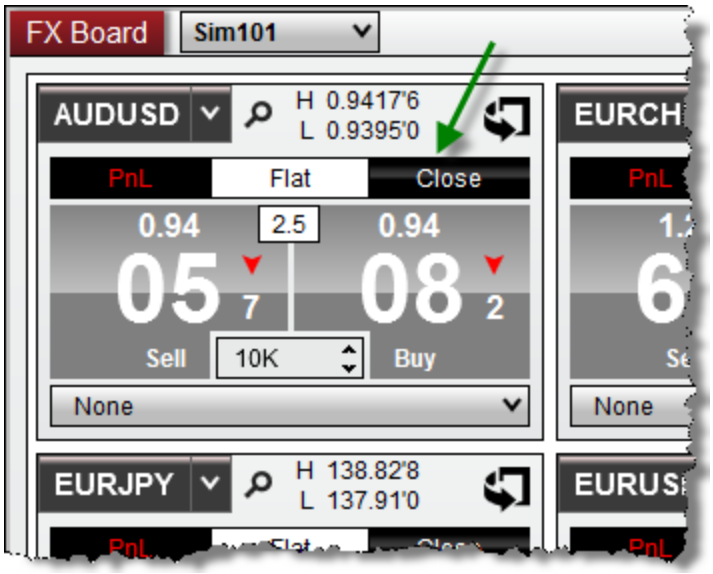
1. Right click on the stacked order row
2. Move your mouse over the order individual order
3. Select "**Cancel Order**"



10.23.11.4 Managing Positions

Closing Positions

Clicking on the "**Close**" button with your left mouse button will close the current position, as well as cancel any working orders associated to the instrument/account combination.



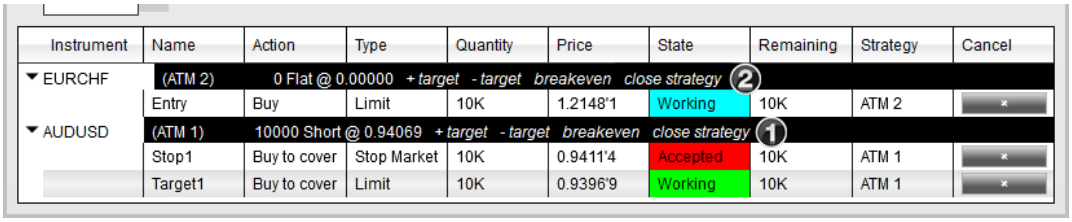
Clicking on this button with your middle mouse button (scroll wheel) will close the selected active **ATM Strategy** only. This means that the position size of the **ATM Strategy** will be closed and any working orders associated to that **ATM Strategy** will be cancelled.

Please see the help topic on Closing a Position or ATM Strategy for more information on the mechanics behind closing various types of positions.

Understanding the FX Board's ATM Strategy position controls

Closing an ATM Strategy

If you have an open position protected by an active **ATM Strategy**, or have submitted an entry order with an active **ATM Strategy**, you can close the strategy by selecting the "***close strategy***" button on the **order grid** of the **FX Board.**



1. Closing an active **ATM Strategy** in position will close the instrument position and cancel the working profit target and stop loss.
2. Closing an active **ATM Strategy** entry order will cancel the order and terminate the associated **ATM Strategy**.

Breakeven

Clicking on the "***breakeven***" button with your left mouse button will adjust any **ATM Strategy** stop orders in the opposite direction of your open position (if position is long it will adjust stop sell orders) to the positions average entry price.

Adding or Removing ATM Strategy Targets

If you have an active **ATM Strategy** displayed in the **FX Board** window, you can add or remove targets. For example, you may have a 2 contract position with 1 Stop Loss and Profit Target for 2 contracts each. You may decide to split this target (add target) so that you can exit the final contract at a higher price. Pressing the "**+ *target***" button on an active **ATM Strategy** will add an addition target, while pressing the "**- *target***" button will remove a target.

How to Scale in or out of an Active ATM Strategy

When you have an active **ATM Strategy** selected in the strategy control list, orders submitted scale into or out of the strategy. Once filled or partially filled, the

existing stop loss and profit target orders are modified to reflect the new position strategy size.

10.23.11.(Properties

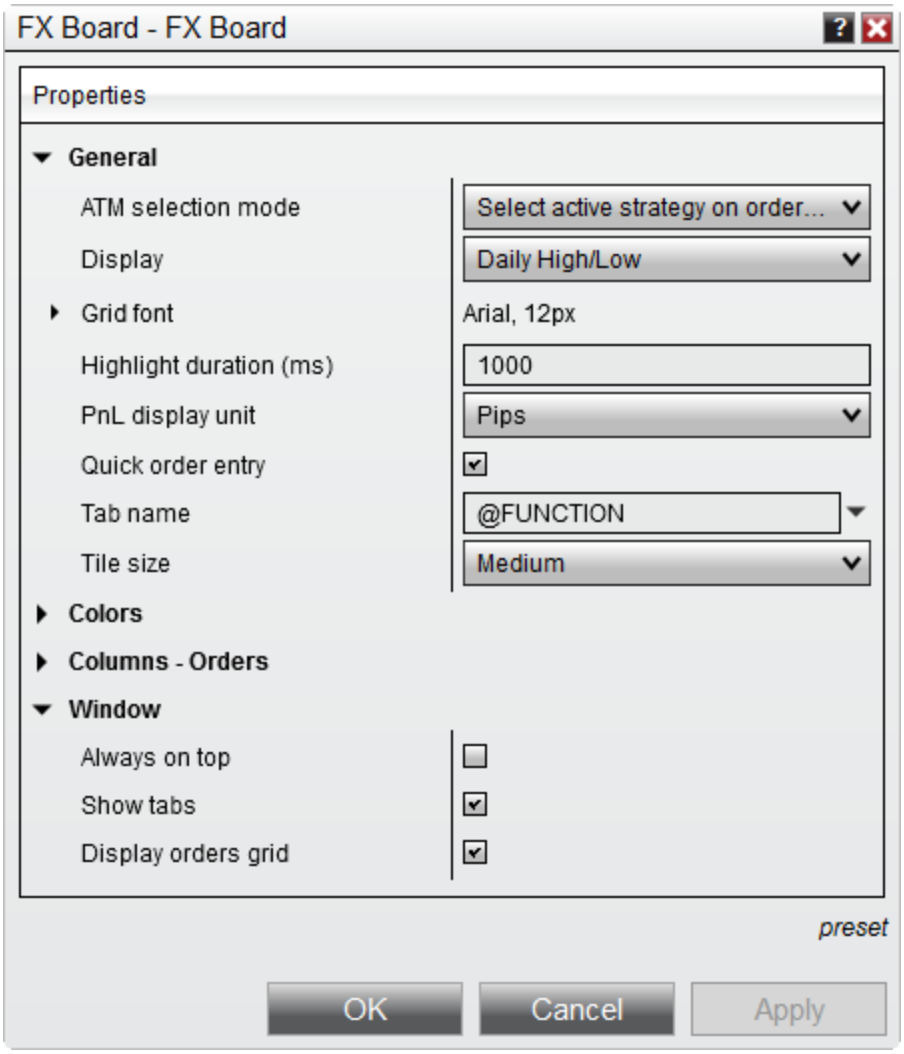
The FX Board order entry window is highly efficient by design but can also be customized to your preferences through the FX Board **Properties** menu.

How to access the FX Board properties window

You can access the FX Board properties dialog window by clicking on your right mouse button within the FX Board window and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the **FX Board** properties window:



Property Definitions

General	
ATM strategy selection mode	Sets the behavior mode of the price ladder display and strategy selector (see more)
Display	Selects the Market Data Display Text
Grid font	Sets the font options

Highlight duration (ms)	Sets the amount of time a tile will remain highlighted after a new tick **Note**: The lowest value which will take effect is 500 (ms)
PnL display unit	Sets the display unit for profit and loss
Quick order entry	Enables / Disables the quick order functionality on the front of an instrument tile
Tab name	Sets the tab name
Tile size	Select the size used for all tiles in the FX Board display
Colors	
Button background	Sets the color for all buttons
Button foreground	Sets the text color for all buttons
Downtick background	Sets the color used for highlighting when a down tick is detected
Downtick foreground	Sets the text color used for highlighting when a down tick is detected
Order - limit	Sets the color used for limit orders
Order - MIT	Sets the color used for MIT orders
Order - profit target	Sets the color used for profit target orders
Order - stop limit	Sets the color used for stop-limit orders

Order - stop loss	Sets the color used for stop loss orders
Order - stop market	Sets the color used for stop-market orders
Uptick background	Sets the color used for highlighting when a down tick is detected
Uptick foreground	Sets the text color used for highlighting when a down tick is detected
Columns - Orders	
Columns (...)	Expanding this section will allow you to disable / enable any of the columns in the Orders Grid display
Window	
Always on top	Sets if the window will be always on top of other windows.
Show tabs	Sets if the window should allow for tabs
Display orders grid	Enables / Disables the Order Grid display

How to set the default properties

Once you have your properties set to your preference, you can left mouse click on the "**preset**" text located in the bottom right of the properties dialog. Selecting the option "**save**" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "**preset**" text and select the option to "**restore**" to return to the original settings.

Using Tab Name Variables

Tab Name Variables

A number of pre-defined variables can be used in the "Tab Name" field of the **FX Board Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.23.11 Order Ticket

The **Order Ticket** window can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting the **Order Ticket** menu item.

Order Ticket Overview

The Order Ticket order entry window allows you to view market data and submit orders.

Display

› Display Overview

Misc

› Properties

Order Management

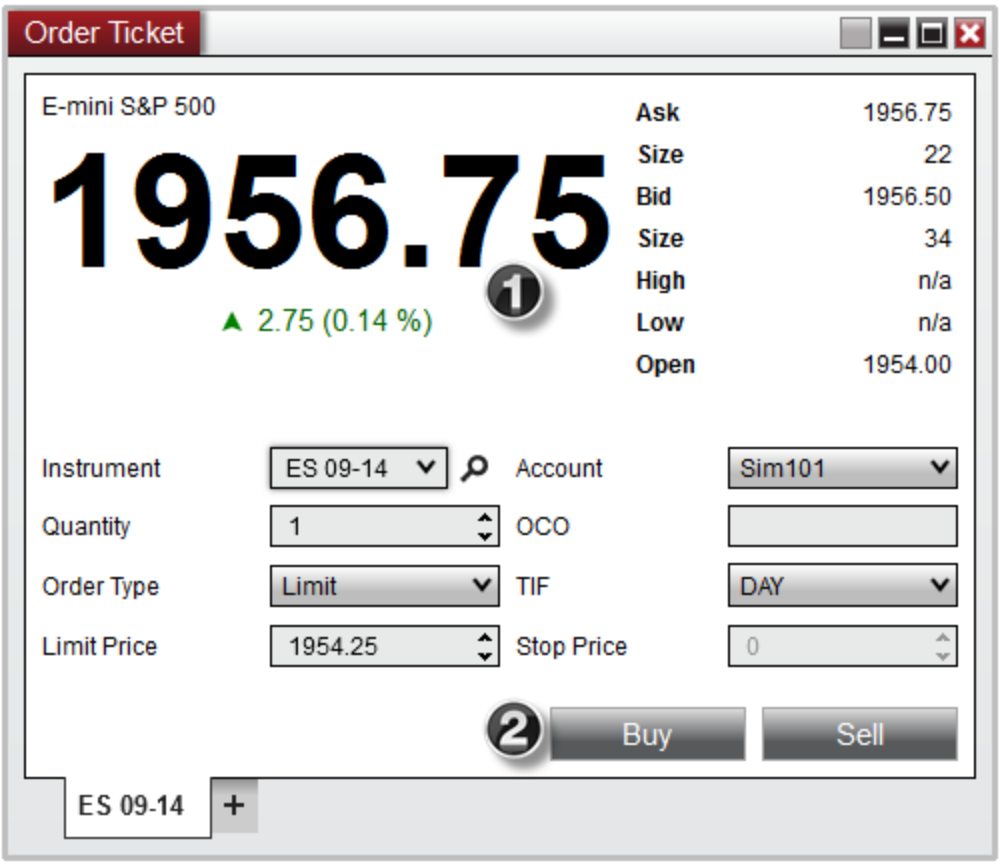
› Submitting Orders

10.23.12. Display Overview

To open the **Order Ticket** Window, select the **New** menu from the NinjaTrader Control Center. Then left mouse click on the menu item **Order Ticket**.

The image below shows the two sections in the of the **Order Ticket** window:

1. **Market Display**
2. **Order Entry**



Note: Positions and orders will only display for the selected Account and Instrument.

Understanding the market display section

Market Display

The **Order Ticket** will display current market data information for the selected instrument.

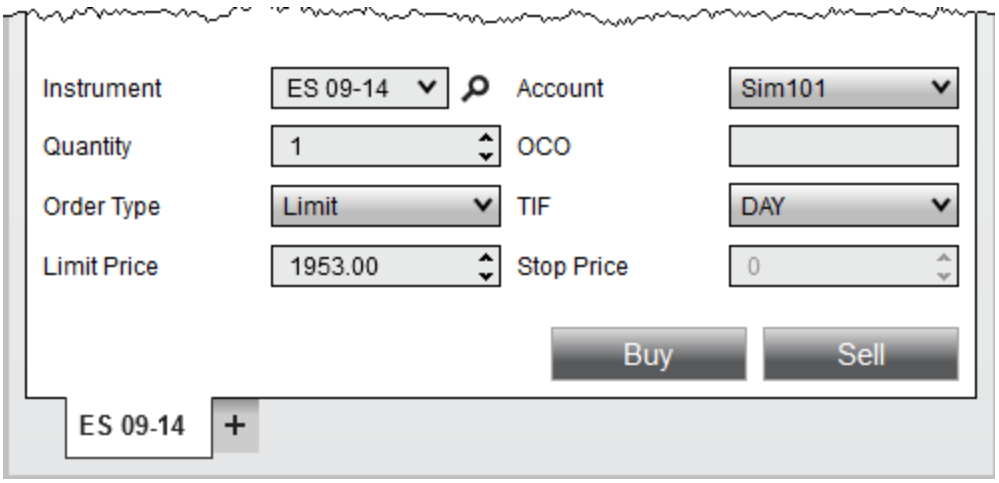
Market Display Definitions

1. Instrument Description
2. Last Price
3. Current day net change
4. Best ask price and ask size
5. Best bid price and bid size
6. Current day high, low, and open

Understanding the order control section

Order Entry Controls

The Order Control region of the **Order Ticket** is used to specify several attributes for a pending order to be submitted.



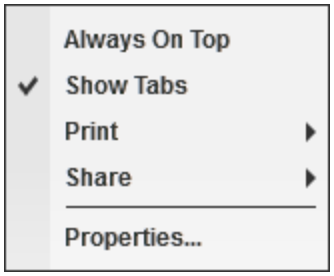
Instrument	Sets the Instrument
Account	Sets the Account
Quantity	Sets the order Quantity

OCO	Sets a user defined OCO ID
Order Type	Sets the order type to be submitted
TIF	Sets the Time in Force
Limit Price	Sets the order Limit price
Stop Price	Sets the order Stop price
Buy	Submits an order to buy
Sell	Submits an order to sell

Understanding the right click menu

Right Click Menu

Right mouse click on the **Order Ticket** window to access the right click menu.



Always On Top	Sets if the window should be always on top of other windows
Show Tabs	Sets if the window should allow for tabs
Print	Displays Print options

Share	Displays Share options
Properties	Sets the Order Ticket properties

10.23.12.2 Submitting Orders

The **Order Ticket** window is used to quickly define and submit custom orders. This interface does *not* display any type of position or trade management features. For those purposes, you would want to consider one of the other order management features: Basic Entry, Chart Trader, FX Pro, FX Board, or SuperDOM.

Selecting instruments and account

How to Select an Instrument

There are multiple ways to select an Instrument in the **Order Ticket** window.

- Select the **Instrument Selector** to open a list of recently used instruments or instruments contained in a predefined list
- With the **Order Ticket** window selected begin typing the instrument symbol directly on the keyboard. Typing will trigger the Overlay Instrument Selector.

For more Information on instrument selection and management please see Instruments section of the Help Guide.

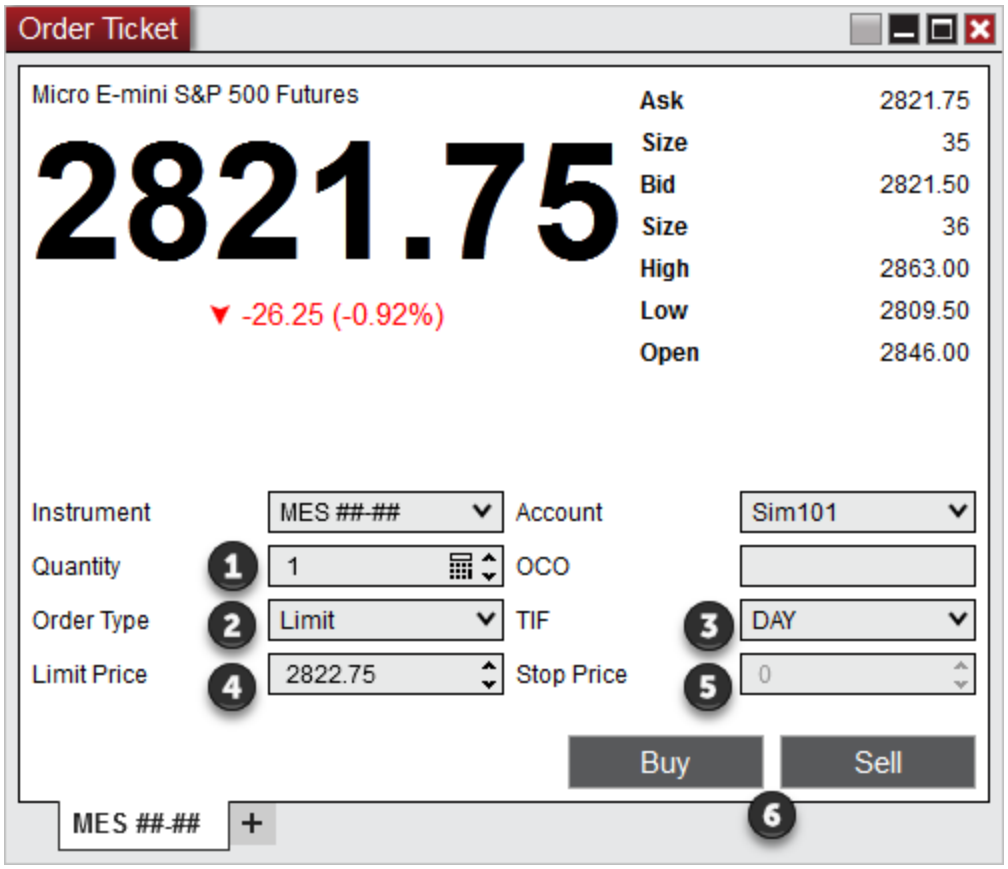
How to Select an Account

A list of all connected accounts will be listed in the **Account Selector**. To change the account select the account you wish to trade through via this drop down list.

How to submit custom orders

To Submit an Order

1. Set the order **Quantity** field (info)
2. Select the order **Type**
3. Set the **TIF** (Time in Force) field (info)
4. Set the **Limit** price if applicable
5. Set the **Stop** price if applicable
6. Left mouse click either the **BUY** or **SELL** button



Tips

1. You can quickly retrieve the current last, bid, or ask price in the Limit and Stop price fields using the following commands:

- Middle click in the field to retrieve the last traded price,
- CTRL + middle click in the filed to retrieve the best ask price
- ALT + middle click in the field to retrieve the best bid price

2. Hold down the CTRL key when increasing/decreasing limit/stop prices to change the price in steps of 10 tick increments.

Close Tab on Order Submission

The **Order Ticket** window can optionally be configured to automatically close the current tab or window after the order has been submitted. This was designed to help discard unwanted **Order Ticket** displays when trading multiple instruments.

To enable this functionality:

1. Right click on the **Order Ticket** tab or window
2. Select **Properties...**
3. Check **Close tab on order submission**
4. Press **OK**

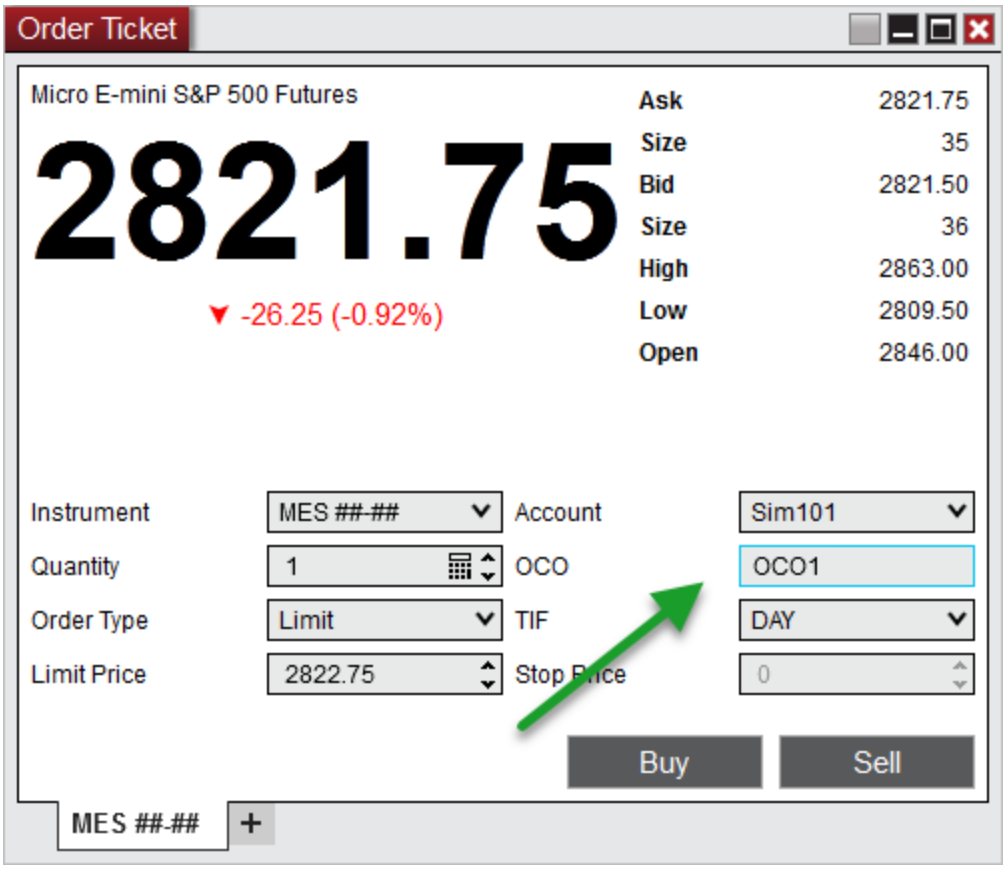
After the order has been submitted, it can be referenced from the Orders Tab of the **Account Data** or **Control Center** window.

Understanding the OCO (One Cancel Other) function

OCO Orders (One Cancels Other)

The **Order Ticket** window allows you to specify a custom user defined **OCO** ID using any combination of numbers and letters in the **OCO** field.

The image below shows the **Order Ticket** window configured a pending order in which we've input an **OCO** ID of "*OCO1*".



In the example above, any additional orders placed with the **OCO** ID set to "*OCO1*" will be tied together in an order group as long as these orders are active. If one order in the group is either filled, cancelled or rejected, all orders in the group with the same **OCO** id will be cancelled.

Warning: If an order which was part of an **OCO** group has already been filled or cancelled, you will need to submit the pending order with a new **OCO ID** otherwise the pending order will be rejected.

10.23.12.3 Properties

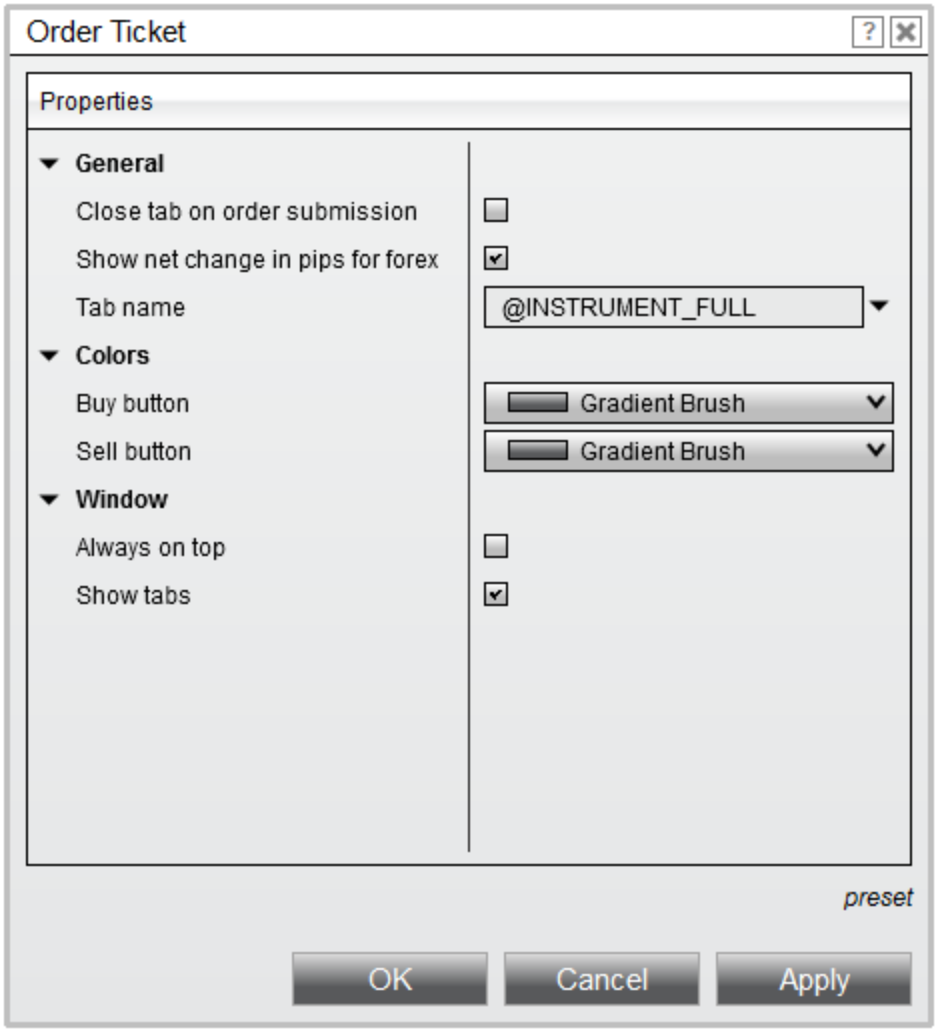
The Order Ticket order entry window is highly efficient by design but can also be customized to your preferences through the Order Ticket Properties menu.

How to access the Order Ticket properties window

You can access the Order Ticket properties dialog window by clicking on your right mouse button within the Order Ticketwindow and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the **Order Ticket** Properties window:



Property Definitions

General	
Close tab on order submission	Enables/Disables the feature which closes the current tab after an order is submitted
Show net change in pips for forex	Enables/Disables forex daily net change to reflect pips. Otherwise, uses points
Tab name	Sets the tab name

Colors	
Buy button	Sets the color for all the buy buttons
Sell button	Sets the color used for all the sell buttons
Window	
Always on top	Sets if the window will be always on top of other windows.
Show tabs	Sets if the window should allow for tabs

How to set the default properties

Once you have your properties set to your preference, you can left mouse click on the "**preset**" text located in the bottom right of the properties dialog. Selecting the option "**save**" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "**preset**" text and select the option to "**restore**" to return to the original settings.

Using Tab Name Variables

Tab Name Variables
A number of pre-defined variables can be used in the "Tab Name" field of the **Order Ticket Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.23.13 SuperDOM

The **SuperDOM** window can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting either the **Static SuperDOM** or **Dynamic SuperDOM** menu items.

SuperDOM Overview

The **SuperDOM** provides complete functionality for the management of orders, positions, and discretionary exit and stop strategies in a highly visual and efficient manner. The DOM at the end of **SuperDOM** stands for Depth of Market which you can see displayed in the Buy and Sell columns of the NinjaTrader SuperDOM.

Display

- Price Ladder Display
- Static Vs Dynamic
- Order Display

Misc

- Using SuperDOM Columns
- Working with Indicators
- Properties

Order and Position Management

- Submitting Orders
- Modifying and Cancelling Orders
- Managing Positions

10.23.13. Price Ladder Display

The SuperDOM is designed to allow the trader to view market prices, market depth, current inside market, indicator price levels, PnL, current positions, and pending orders at a glance. The unique display of each item within the SuperDOM Price Ladder display makes managing open orders and positions easy and efficient.

To open the **SuperDOM** Window, select the **New** menu from the NinjaTrader Control Center. Then left mouse click on the menu item **SuperDOM (Static)** or **SuperDOM (Dynamic)** (Please see Static vs Dynamic Price Ladder Display for more information)

Understanding the function of each column in the Price Ladder display

The price ladder is broken down into three functional columns by default, and can be extended to display any number of additional custom columns.

Buy Column

The left column is the **Buy** ❶ column which is used to:

- Submit buy orders

- Modify buy orders
- Display the total contracts on the bid at their respective prices (also known as market depth)

Price Column

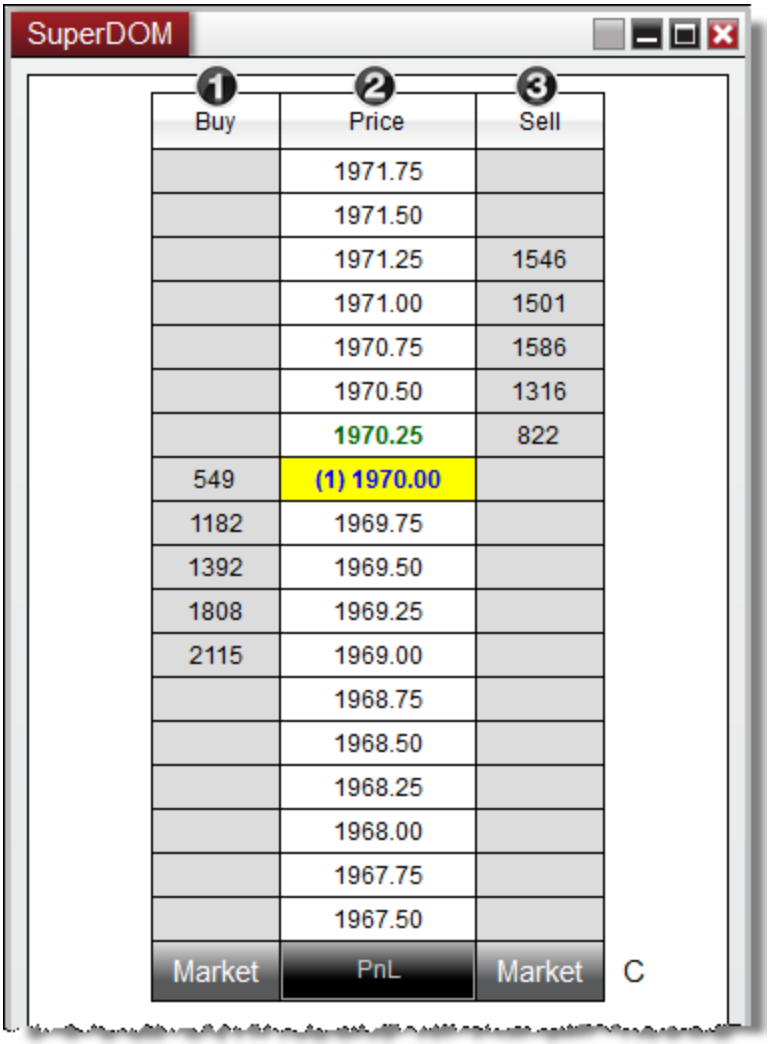
The center column, known as the **Price** ② column, is used to:

- Modify stop loss and profit target orders
- Display market prices
- Display the current bid, ask, and last traded prices
- Display indicator price levels

Sell Column

The right column is the **Sell** ③ column which is used to:

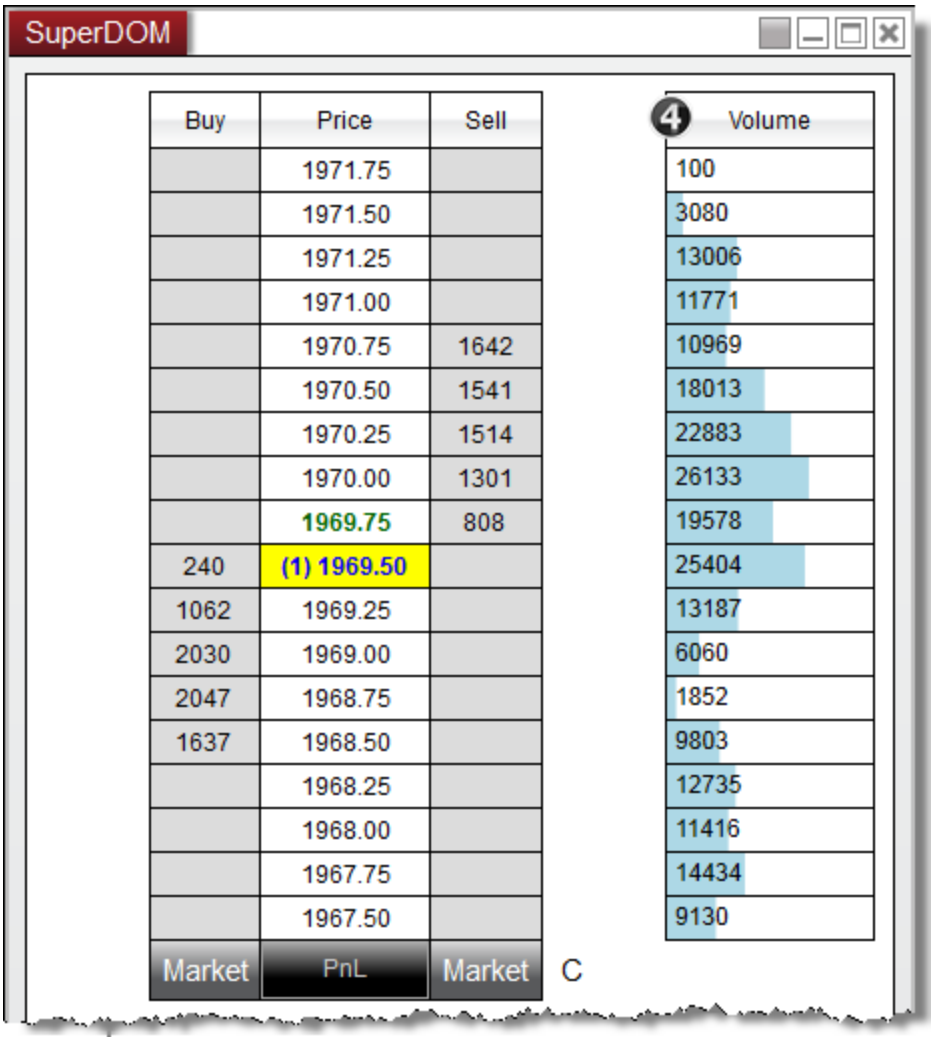
- Submit sell orders
- Modify sell orders
- Display the total contracts on the ask at their respective prices (also known as market depth)



Note: To view market depth for equities the Level II window must be used.

Columns

You can optionally configure additional **Columns** 4 to display other points of interest relative to the ladder display.



By default, NinjaTrader will come pre-loaded with the following columns:

- APQ (Approximate Position in Queue)
- Notes
- PnL
- Volume

We also support custom NinjaScript column development, meaning that programmers and vendors can create custom columns which can be installed to extend functionality of the **SuperDOM** display.

Please see our Help Topic on Understanding SuperDOM Columns for more information.

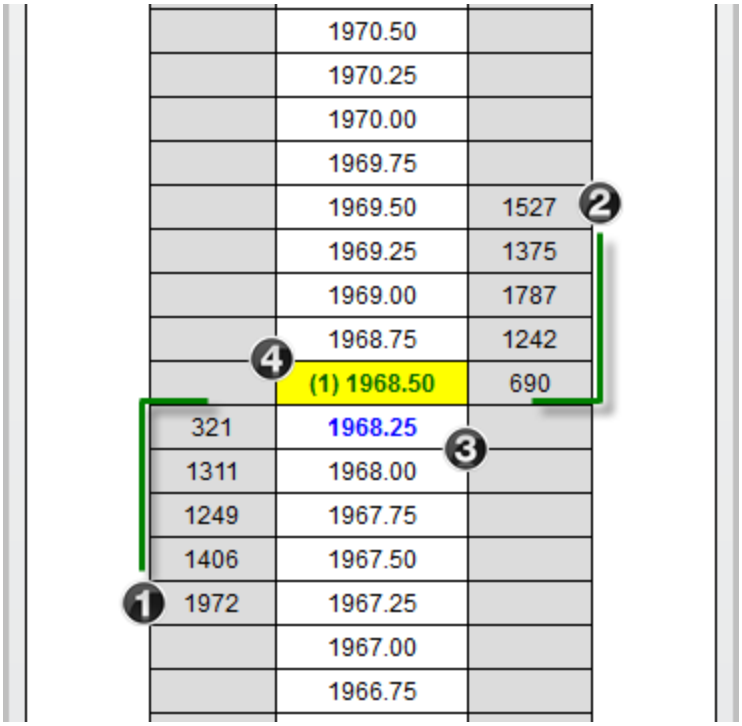
Understanding how market data is displayed

The Price Ladder display section of the SuperDOM displays the current inside market and market depth. Various aspects of this display can be user defined in the SuperDOM Properties window.

The following market depth items can be displayed:

1. Bid Depth
2. Ask Depth
3. Best Bid
4. Best Ask

Last traded price and size (yellow cell in the image below)



How to use the quick buttons at the bottom of the Price Ladder display

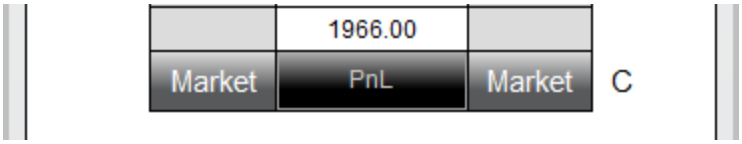
The bottom row of the price ladder contains three functions: Buy Market, PnL, and Sell **Market**.

Market (left cell)

Submits buy market, limit at ask, or limit at bid orders

PnL (center cell)
Displays unrealized profit or loss for the current position

Market (right cell)
Submits sell market, limit at ask, or limit at bid orders



You can change the type of order the MARKET cells submit by holding down the SHIFT key to place limit orders at the ask, or by holding down the CTRL key to place limit orders at the bid. Clicking with your left mouse button on the PnL cell will change the display between points, ticks, currency, percent and pips.

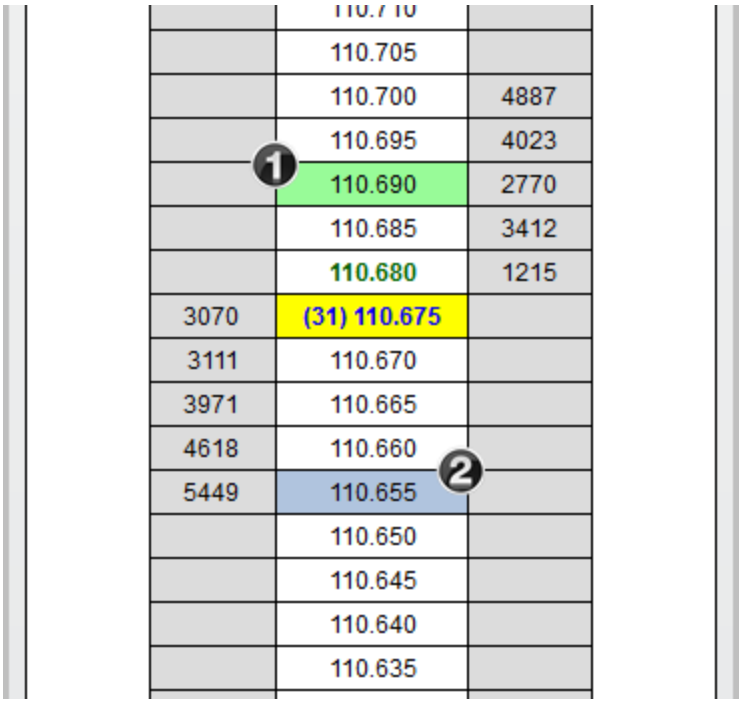
How to display the daily high and low prices

Daily High and Low

The market's daily high ❶ and daily low ❷ can be optionally displayed. To enable this feature:

- Right click on the **SuperDOM**, select **Properties**, check **Show daily high/low markers**

You can further customize the color of the markers in the SuperDOM Properties dialog window.



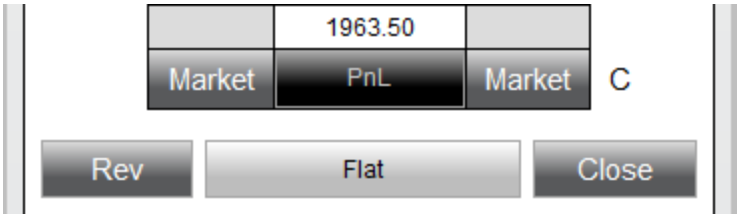
Note: Daily High and Low values are not calculated by NinjaTrader and are sent from your data provider. Not all data providers provide this information for all instrument types.

Understanding how position and profit & loss information is displayed

PnL Display

The PnL field in the bottom of the Price column will show the current unrealized profit/loss for your current open position, and read as PnL when you are FLAT.

You can optionally enable "Show PnL when flat" in the SuperDOM Properties to view your daily account PnL when FLAT



Current Position Display

The cell between the CLOSE button and the REV button will tell you your current position.

When long the field will show as green and list the number of contracts, and when short field will show as red and list the number of contracts. When you do not have an open position the field will say FLAT.

How to adjust the Price Ladder display

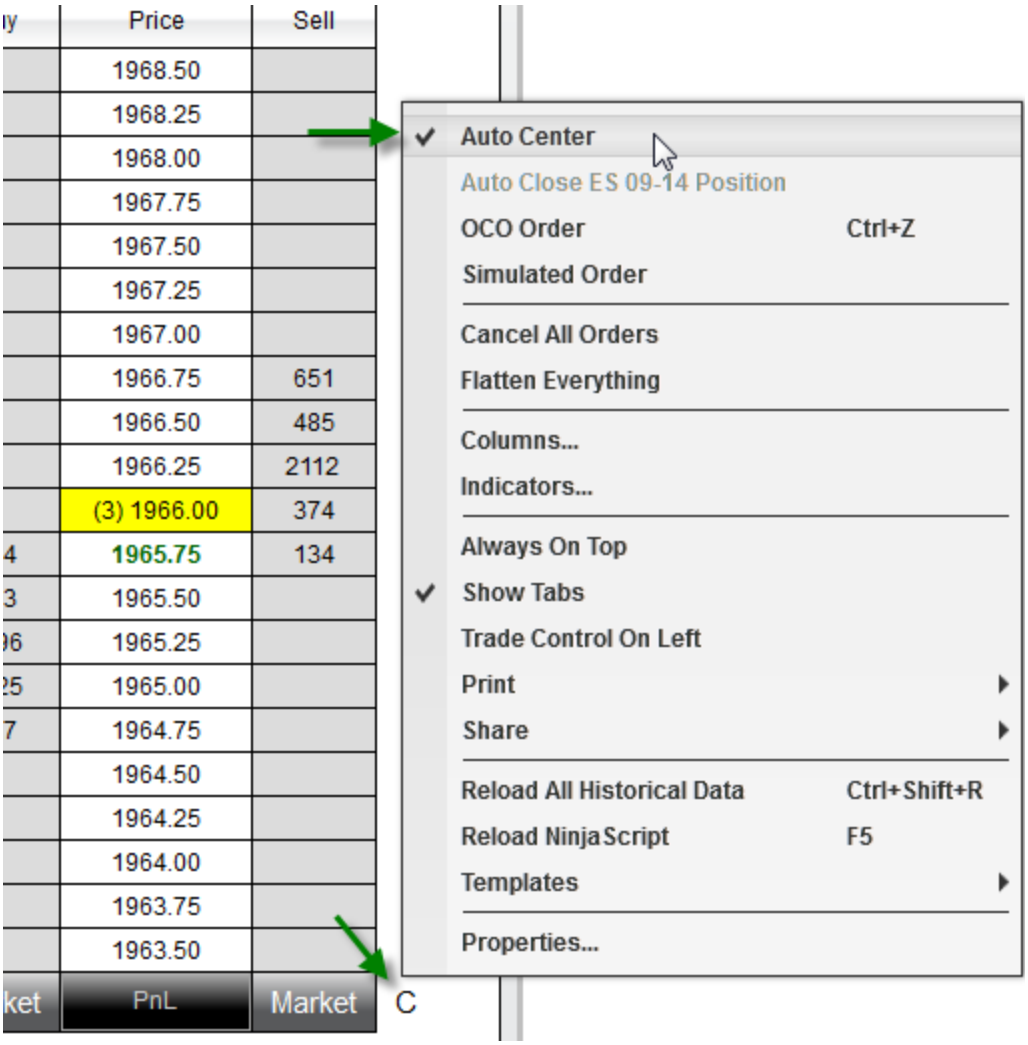
Play Video

Adjusting the Price Ladder display

Move your cursor into the Price Ladder region and use your mouse scroll wheel to adjust market prices up or down.

You can also left mouse click on the "C" button at any time to center the inside market.

Optionally, the Auto Center property will automatically center the inside market price should the last traded price trade outside the visible range on the Price Ladder. You can enable or disable Auto Center at any time by clicking on your right mouse button in the border of the SuperDOM and selecting the menu name **Auto Center**.



Number of Visible Price Rows

The number of price rows will be dynamically adjusted by vertically re-sizing the **SuperDOM** window. The larger the size of the **SuperDOM** window, the more price levels will be added in automatically.

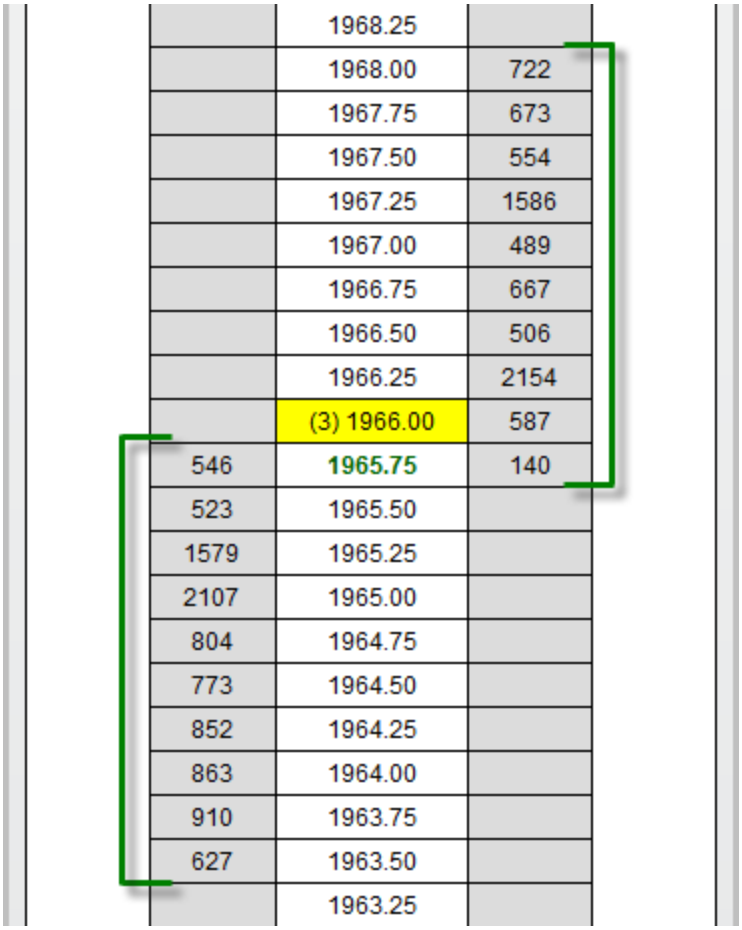
This achieved in the same manner you would use to resize any other sort of general application window, by moving your mouse cursor to the edge of the window and clicking and dragging until you have reached your desired size.

Increasing / Decreasing the Number of Market Depth Levels

By default, the **SuperDOM** will display 10 levels of market depth. However you can configure additional levels of depth to be displayed and is only limited by the

number of levels provided by the exchange/data provider combination you are using.

For example, an exchange might provide 20 levels of market depth for an instrument you are trading. If you would like to view all 20 of these levels on the **SuperDOM**, simply right click on the **SuperDOM**, select **Properties**, and set the **# of market depth levels property** value to 20 and the press **OK**.



10.23.13.2 Static vs Dynamic Price Ladder Display

Play Video

You may have the option of using either a static (original SuperDOM) or dynamic price ladder display depending on your FCM or broker. The difference between these options is how the inside market is displayed in the Price Ladder.

Understanding the Static Price Ladder display

Static
The inside market (ask/bid and last price) climb up and down the Price Ladder in response to a change in market price.

The price rows are static (do not change).

Understanding the Dynamic Price Ladder display

Dynamic
The inside market (ask/bid and last price) is in a fixed location in the Price Ladder display.

The price rows are dynamic in that each row's price changes in response to a change in market price.

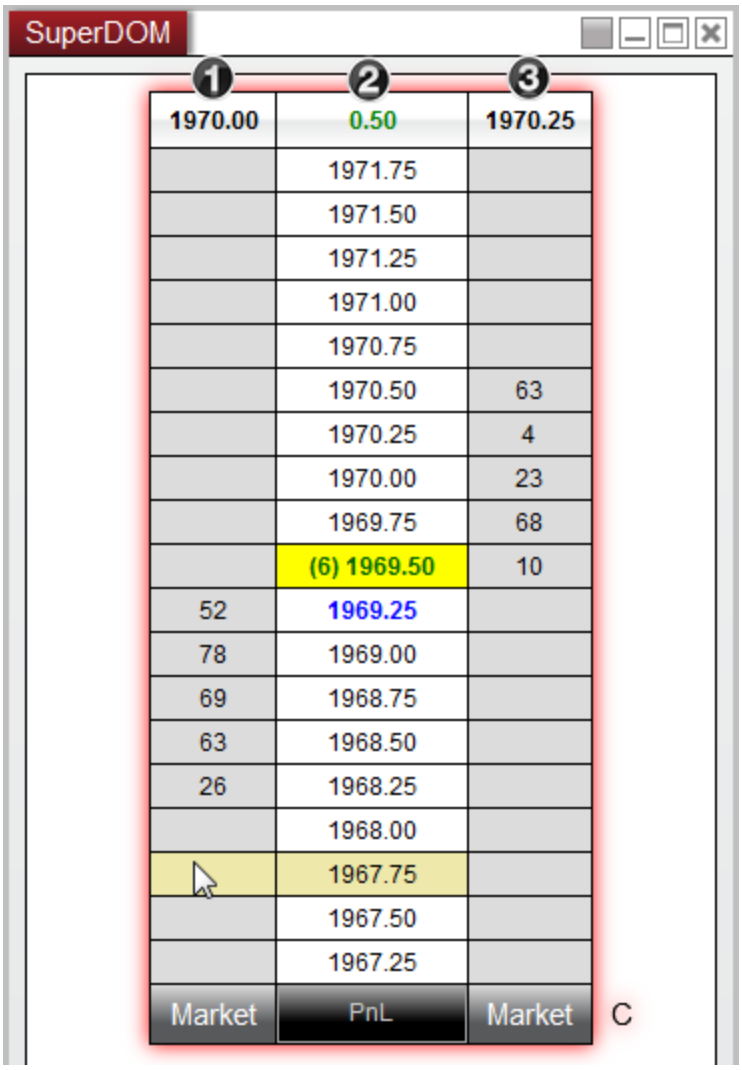
Suspending the Dynamic Price Ladder

To assist with submitting and modifying orders in the Dynamic Price Ladder display during volatile market activity you can choose to suspend (freeze) the Price Ladder display simply by moving your mouse cursor over the **Price Ladder**.

Once suspended, the ladder will highlight red in color and you can now safely submit or modify an order without the price of the underlying row changing. In addition, the inside market will be displayed in the top row of the SuperDOM while the price ladder is frozen.

In the image below, we can easily tell that the **Price Ladder** has been has been suspended as the ladder has been highlighted in red. We can also identify the following information displayed in the top row:

1. Best bid
2. Net change (last traded price) from the time the display was suspended
3. Best ask



The **Price Ladder** will remain suspended until you remove your cursor from the price ladder display, at which point all rows will begin updating dynamically once again.

10.23.13. Order Display

Play Video

Orders are displayed in a highly visual manner. Different order types and order objectives (stop loss or profit target orders) are uniquely color coded.

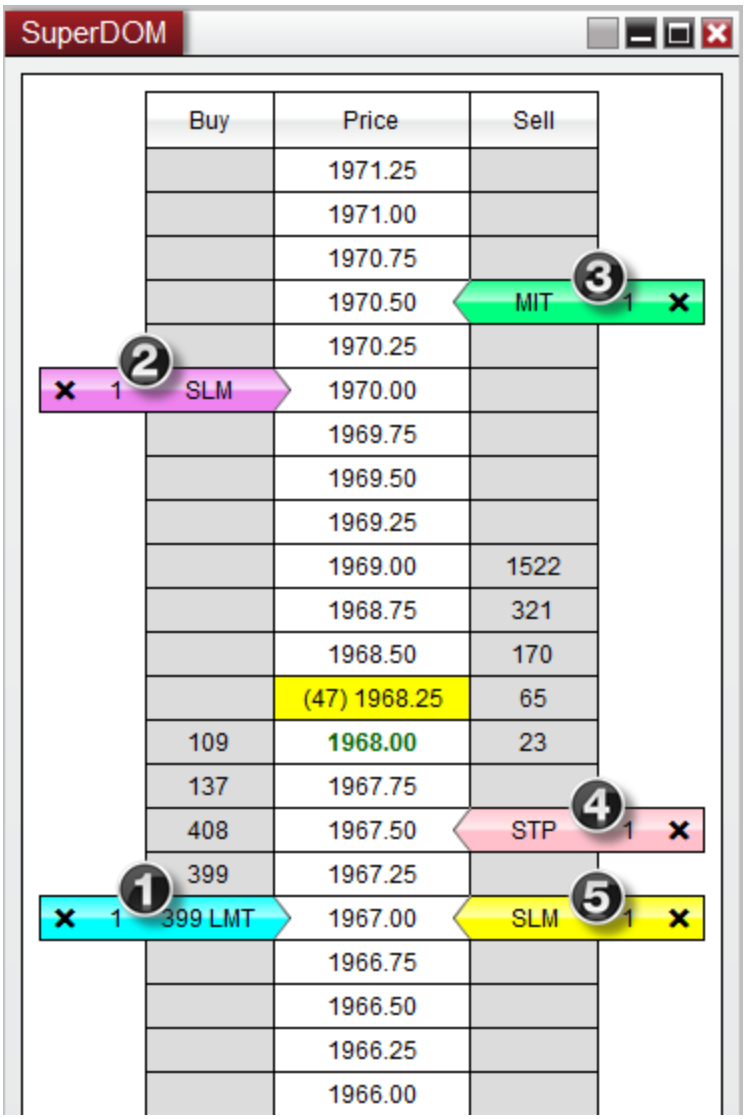
Understanding how orders are displayed

Order Display

All orders are displayed by coloring a cell or group of cells within the Price Ladder

1) Limit Order	Default color is cyan with the text "LMT"
2) Stop-Limit Order	Default color is violet with text "SLM"
3) MIT Order	Default color is spring green with text "MIT"
4) Stop-Market Order	Default color is pink with text "STP"
5) Simulated Stop Order	Default color is yellow with text "SLM" or "STP"

The image below shows a working limit, stop-market, and stop-limit order for one contract each.

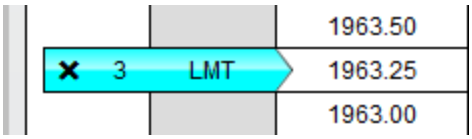


Note: Orders will only display for the selected Account and Instrument.

Understanding how the quantity of an order are displayed

Size Marker

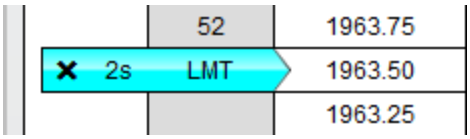
There is also an associated Size Marker which displays the remaining contracts to be filled for the order(s) at the corresponding price. In the image blow, three contracts are remaining to be filled and are working at a price of 1963.25



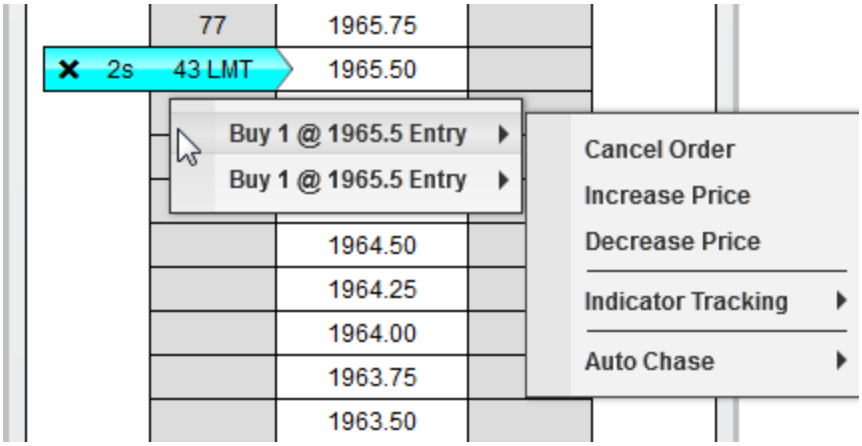
Understanding how multiple orders at the same price are displayed

Consolidated Order Display

The SuperDOM will consolidate the display of all orders resting at the same price and mark an "s" within the Size Marker display to indicate that there are multiple orders stacked at that price. The Size Marker then indicates the cumulative remaining contracts for all orders resting at that price. The image blow depicts a consolidated display of two limit orders for 1 contract each.



By moving your mouse cursor over the order (cyan colored cell) and pressing down on your right mouse button, you will see a context menu listing all individual orders consolidated at the corresponding price and any relevant actions that you can perform on those orders.

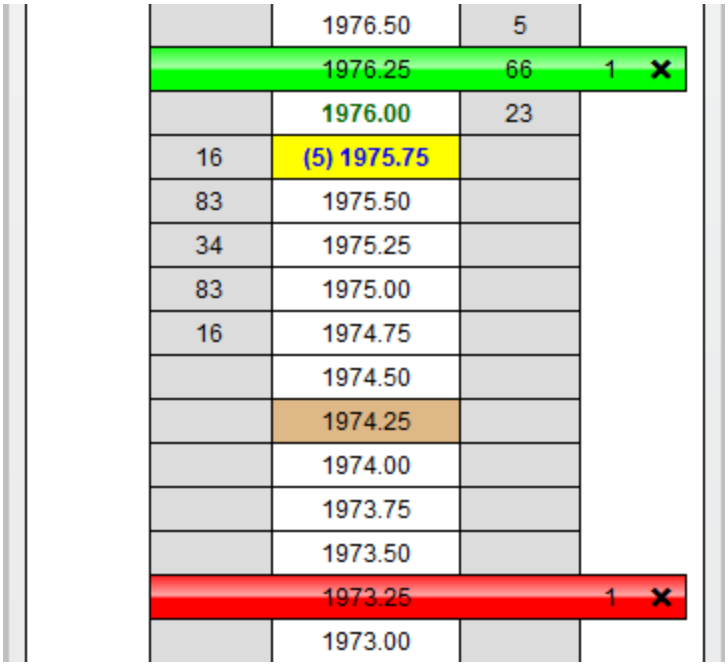


Understanding how Stop Loss and Profit Target orders are displayed

Stop Loss and Profit Target display

Orders submitted as Stop Loss and Profit Target orders are uniquely displayed by coloring all three cells in the price row where the order(s) are working. This makes it very easy to visualize your stop and profit objectives relative to the current market. All other orders are displayed by coloring a single cell in either the BUY or SELL column.

The two below displays an image of a Stop Loss and Profit Target pair, notice that the Size Marker displays the number of contracts remaining to be filled, and that they are sell orders since they are displayed on the sell side of the Price Ladder. Also note the brown colored cell at price level 1974.25, this represents the average entry price for the open position.



▼ How to view out of range Stop Loss and Profit Target orders

Displaying Stop Loss and Profit Target orders outside the visible range

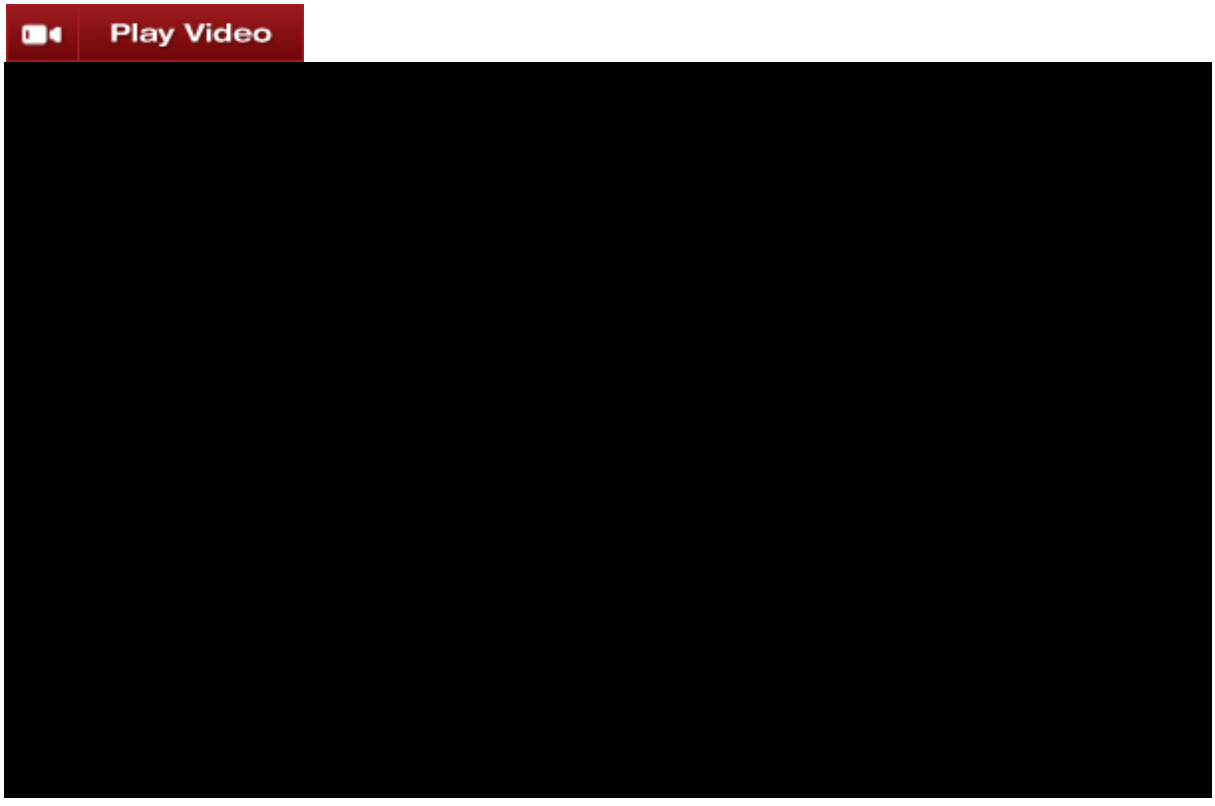
There may be times when your Stop Loss or Profit Target orders are outside of the visible price range of the SuperDOM price ladder. You can easily bring these orders in range by first disabling **Auto Center** from the **SuperDOM** right click menu, and then clicking with your middle mouse button in the Price column.

As long as **Auto Center** is disabled, clicking on the bid or above with your middle mouse button will bring into visible range the first stop loss or profit target order above the highest displayed price of the price ladder. Clicking below the bid with your middle mouse button will bring into visible range the first stop loss or profit target order below the lowest displayed price of the Price Ladder.

You can then quickly navigate back to the last traded price by either re-enabling **Auto Center**, or pressing the **C** button to manually re-center the price.

> **Note**: This function only works if "Single Click Order Modification" is set to False in the SuperDOM Properties window. If set to true, middle click will instantly modify your Stop Loss or Profit Target orders. Please see the help topic on Modifying and Cancelling Orders for more information on that feature

10.23.13.4 Submitting Orders



Orders are submitted in the NinjaTrader SuperDOM using different combinations of mouse clicks and keyboard keys. **Limit**, **stop-market**, **stop-limit**, and **MIT** orders are placed with the following conventions:

Left Mouse Button

- **Limit** orders are placed with the left mouse button
- **MIT** orders are placed with the Ctrl key + left mouse button

You can optionally reconfigure these default settings to allow for the left click to submit **MIT** orders from the SuperDOM Properties:

- Check "**Left** mouse button is MIT"

With this configuration, the left mouse button will now submit **MIT** orders, and Ctrl + Left mouse button will submit **Limit** orders

Middle Mouse Button (Scroll Wheel)

- **Stop-market** orders are placed with the Ctrl key and middle mouse button
- **Stop-limit** orders are placed with the middle mouse button

Or you configure the Middle mouse button to submit **stop-market** orders from the SuperDOM Properties:

- Check "Middle mouse button is stop market"

With this configuration, the middle mouse button will now submit **stop-market** orders, and Ctrl + middle mouse button will submit **stop-limit** orders

Note: It is highly recommended that you review the Advanced Trade Management (ATM) section for a complete understanding of order submission and subsequent actions that you can have NinjaTrader automate.

Selecting instruments and account

How to Select an Instrument

There are multiple ways to select an Instrument in the **SuperDOM** window.

- Select the **Instrument Selector** to open a list of recently used instruments or instruments contained in a predefined list

- With the **SuperDOM** window selected begin typing the instrument symbol directly on the keyboard. Typing will trigger the **Overlay Instrument Selector**.

For more Information on instrument selection and management please see Instruments section of the Help Guide.

How to Select an Account

A list of all connected accounts will be listed in the **Account Selector** drop down list. To change the account select the account you wish to trade through via this drop down list.

Understanding order settings

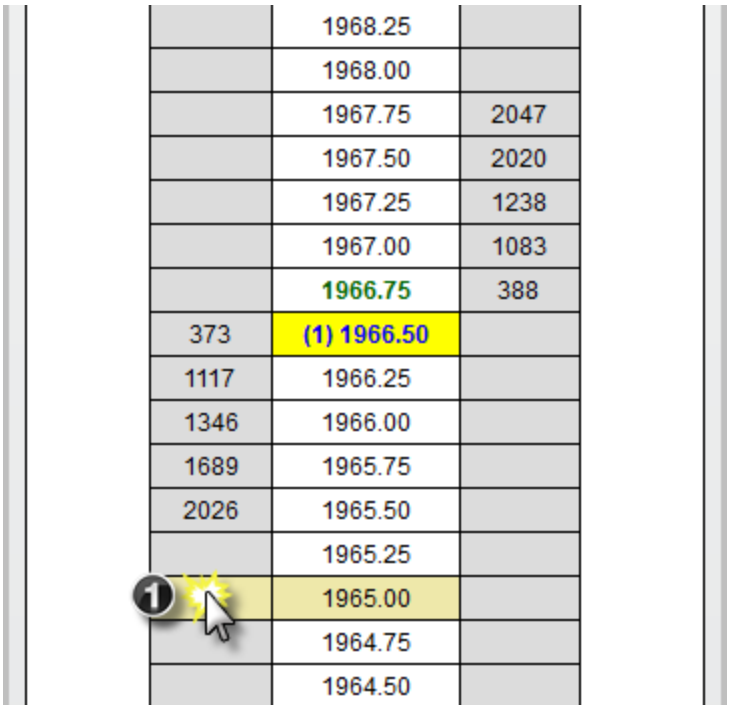
To submit an Order

1. Set the order "**Quantity**" field (info)
2. Set the "**TIF**" (Time in Force) field (info)
3. Set the "**ATM Strategy**" option (info)
4. Enter an order with any of the methods described below

How to submit a limit order

Limit Orders

To submit a **limit** order, select either the **Buy** column for buy orders or the **Sell** column for sell orders and press down on your  left mouse button in the cell that corresponds to the price you wish the **limit** order to be submitted at.

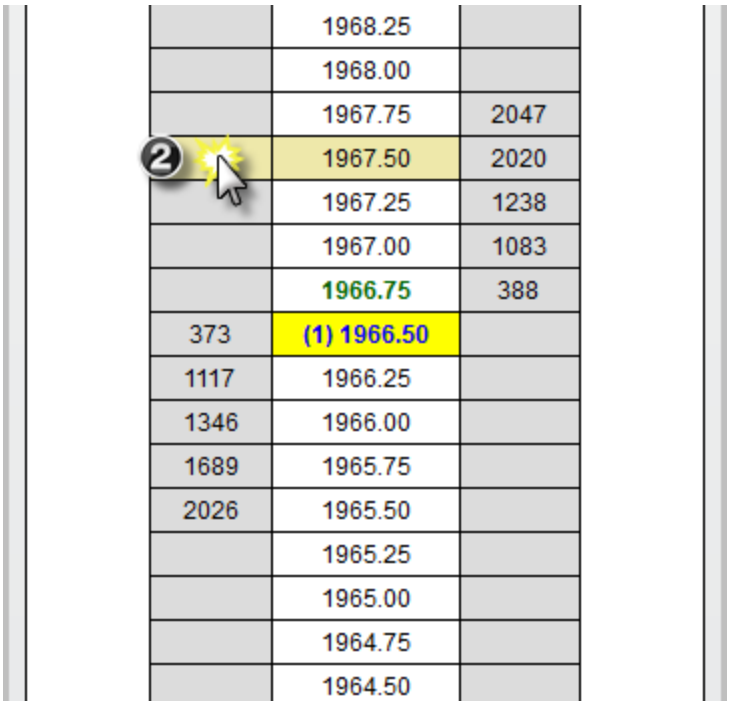


Clicking at the location marked in the image above would submit a buy **limit** order at the price 1965.00.

How to submit a stop-market order

Stop-Market

To submit a **stop-market** order, select either the **Buy** column for buy orders or the **Sell** column for sell orders and press down on your ❷ middle mouse button (scroll wheel) while holding the CTRL key down in the cell that corresponds to the price you wish the **stop-market** order to be submitted at.



In the image above, holding down the CTRL key on your keyboard and middle mouse clicking on the price point would enter a buy **stop-market** order at 1967.50.

How to submit a stop-limit order

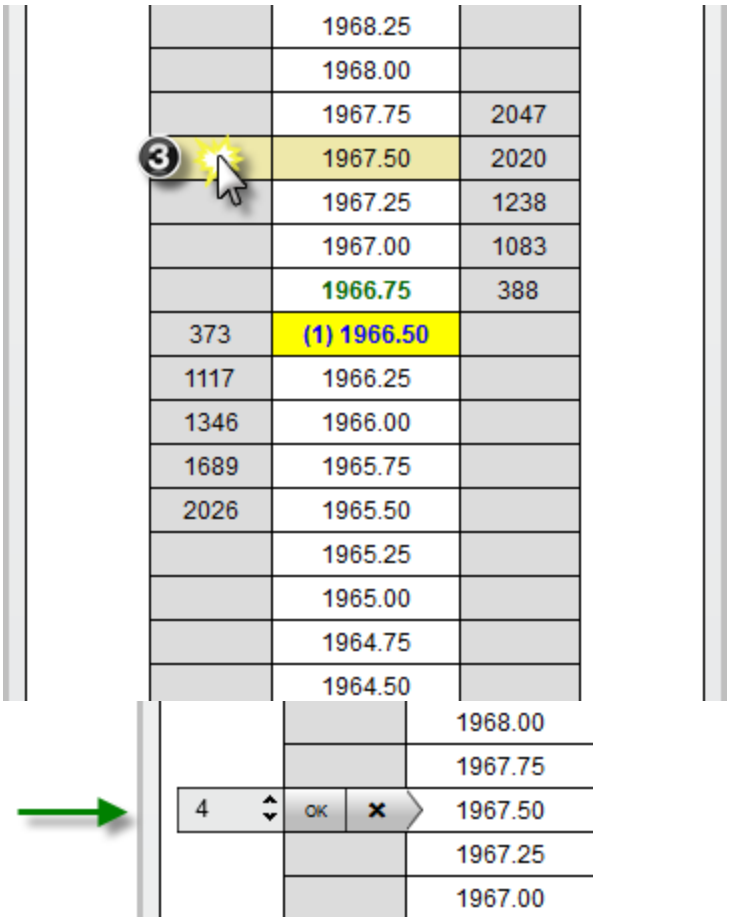
Stop-Limit Order

To submit a **stop-limit** order, select either the **Buy** column for buy orders or the **Sell** column for sell orders and press down on your 3 middle mouse button (scroll wheel) in the cell that corresponds to the price you wish the **stop limit** order to be submitted at.

A numeric field (image lower right) will appear that represents the number of ticks away you wish the limit price of the **stop-limit** order to be placed at. Either by using your mouse scroll wheel or clicking on the up/down arrows in the numeric field, set the number of ticks and press the "check mark" button to complete the order submission. Pressing the "x" button will cancel the order submission operation.

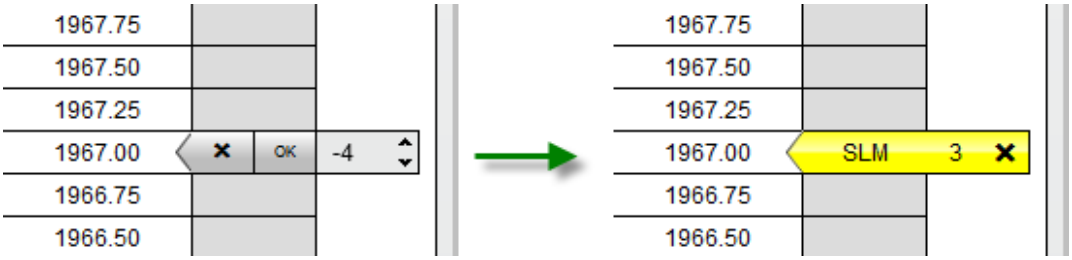
For example, if you intend to have an order with a stop price of 1967.50 and a limit price of 1968.50 (4 ticks spread for the SP Emini contract) you would set the numeric field value to 4. Following the same example submitting a sell **stop-limit**,

setting the numeric field value to 1 would result in a stop price of 1967.50 and a limit price of 1967.25.



Negative Stop-Limit Offset

You will notice that there are also negative values. By selecting a negative value, you automatically submit a Simulated Stop order, which is indicated by a **yellow order flag**. This allows you to place orders that trigger at a break out price but try to fill you at a better price.



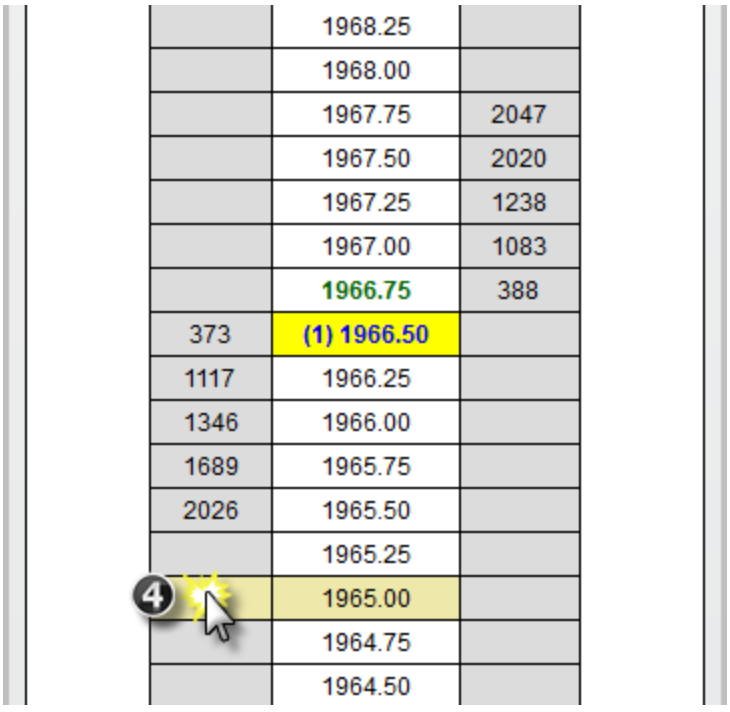
Single Click Stop-Limit Orders

If you generally place **stop-limit** orders using the same offset between limit and stop price, you can enable single click submission of **stop-limit** orders by setting the "Stop-limit offset" property to an integer value via the SuperDOM properties. By default, this setting is set to "Off" which forces the numeric field (image above right) to display. Setting this property to a value of 1 would instantly place a **stop-limit** order with a stop price of X and a limit price of X + 1 for buy orders or X - 1 for sell orders.

How to submit an MIT (Market If Touched) order

MIT Orders

To submit a **MIT** order, select either the **Buy** column for buy orders or the **Sell** column for sell orders and press down on your 4 left mouse button while holding the CTRL key down in the cell that corresponds to the price you wish the **MIT** to be submitted at.

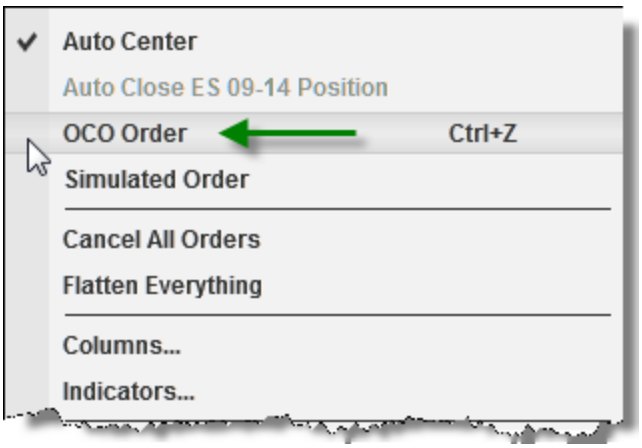


In the image above, holding down the CTRL key on your keyboard and left mouse clicking on the price point would enter a buy **MIT** order at 1965.00

Understanding the OCO order (one cancels other) function

OCO Orders (One Cancels Other)

Stop loss and profit target orders (submitted automatically via an ATM strategy) are always sent as OCO, however, you can submit entry or exit orders as OCO orders as well. Why? The market may be trading in a channel and you wish to sell at resistance or buy at support, whichever comes first by placing two **limit** orders at either end of the channel. To place OCO orders, via the right mouse click context menu select the menu name "**OCO Order**" or use the shortcut key Ctrl + Z.



The "oc" (OCO indicator) will light up green. All orders placed while this indicator is lit will be part of the same OCO group. Once any order of this group is either filled or cancelled, all other orders that belong to this group will be cancelled.

If you want each **OCO** order to create it's own set of **Stop Loss** and **Profit Target** orders ensure that the ATM Strategy control list is set to either **<Custom>** or a strategy template name before you submit each OCO order.

After you have placed your orders, it is advised to disable the **OCO** function via the right click menu, or use the short cut key CTRL+Z.

Warning: If an order which was part of an **OCO** group has already been filled or cancelled, you will need to submit the pending order with a new **OCO ID** otherwise the pending order will be rejected.

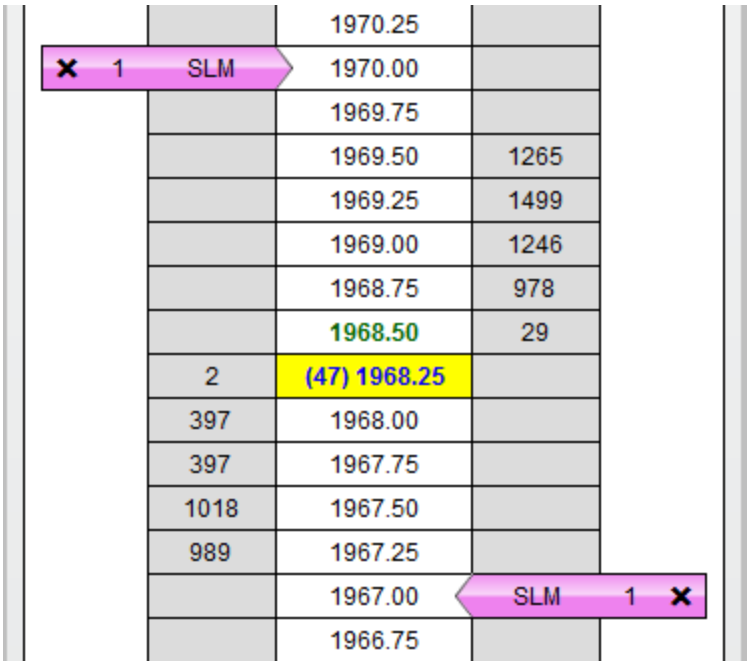
To reset an **OCO** ID, simply disable the **OCO** function, and re-enable. This will generate a new **OCO ID** and allow you to place new orders.

Break Out/Fade Entry Example

One of the great features of NinjaTrader is its ability to submit two entry orders, one of which will cancel if the other is filled.

You can accomplish a breakout/breakdown approach by:

- Right click in the SuperDOM and select the menu item "**OCO Order**" to enable the OCO function
- For your first order, select the desired option from the "ATM Strategy" drop down list
- Submit your stop order to buy above the market
- For your second order, select the desired option from the "ATM Strategy" drop down list
- Submit your stop order to sell below the market
- CRITICAL: Right click in the **SuperDOM** and select the menu item **OCO Order** to disable OCO from being applied to subsequent orders.

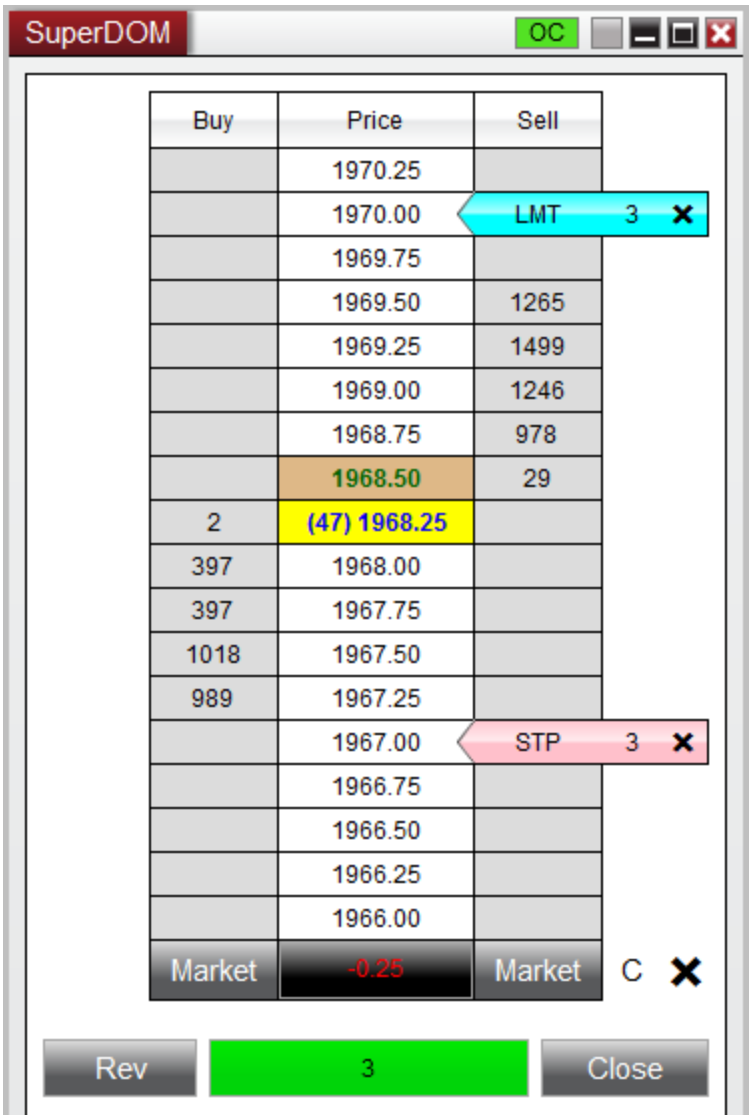


For a market fade approach just substitute **limit** orders for stop orders.

Using the OCO Function to Bracket an Open Position

If you have an open position without an ATM strategy attached, and you wish to add limit and stop orders to protect the position follow these steps:

- Set the ATM strategy in the ATM Strategy selection drop down box to a value of **<None>**
- Right click in the SuperDOM and enable OCO order placement by selecting the menu name "**OCO Order**"
- Then place a limit order where you want to exit at a profit
- Then place a stop order where you want to exit at a loss
- Lastly, right click again and select the menu item "**OCO Order**" to disable the OCO order placement



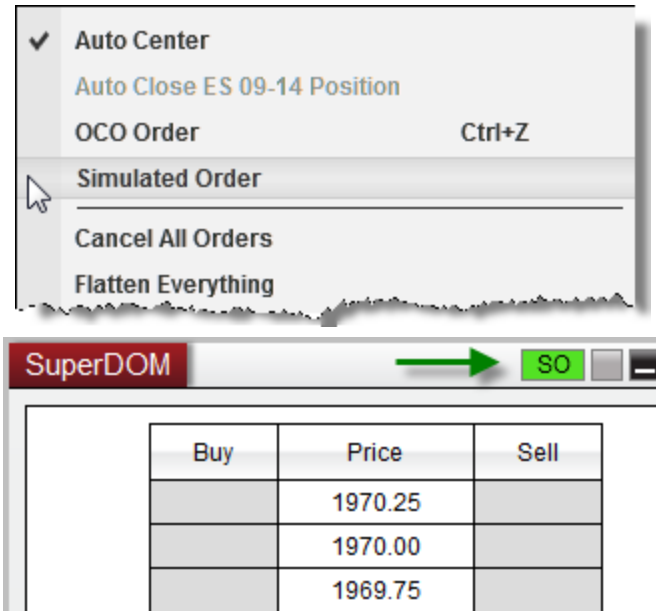
Now you have a target and a stop placed protecting your open position, and when one of these orders is filled the other will be cancelled automatically.

How to submit simulated stop orders (Simulated Order)

Simulated Stop Orders

To submit a Simulated Stop Order (entry and exit NOT stop loss; simulated stop loss orders are enabled via an ATM stop strategy) you must enable Simulated Order mode via the right mouse click context menu by selecting the "**Simulated Order**" menu item or use the shortcut key Ctrl + A. The "so" (Simulated Order

indicator) will light up green. All stop orders placed while this indicator is lit will be submitted as a Simulated Stop Orders.



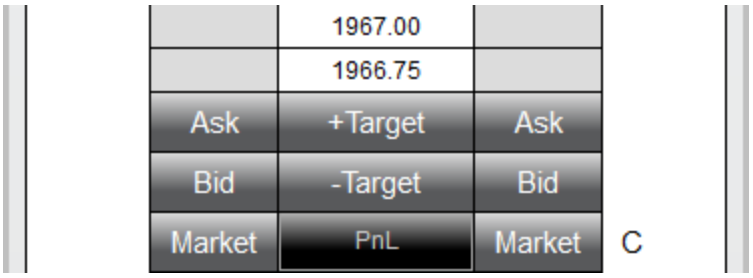
One of the powerful features of Simulated Stop Orders is that you can submit a "negative limit stop-limit" order. This means that you can place an order where the limit price is better than the stop price. As an example, you may want to buy on strength indicated by a move up to a particular price. Once that occurs, you want to enter at a better price using a limit order several ticks below (if you are buying) the stop price.

For more information please visit the Simulated Stop Orders section of the user Help Guide.

How to submit orders with the Quick Buttons

Quick Buttons

Setting “Show Quick Buttons” to true in the SuperDOM Properties enables: Ask and Bid buttons in the Buy and Sell Columns, a **+Target** (add target) button, and a **-**Target (remove target) button.

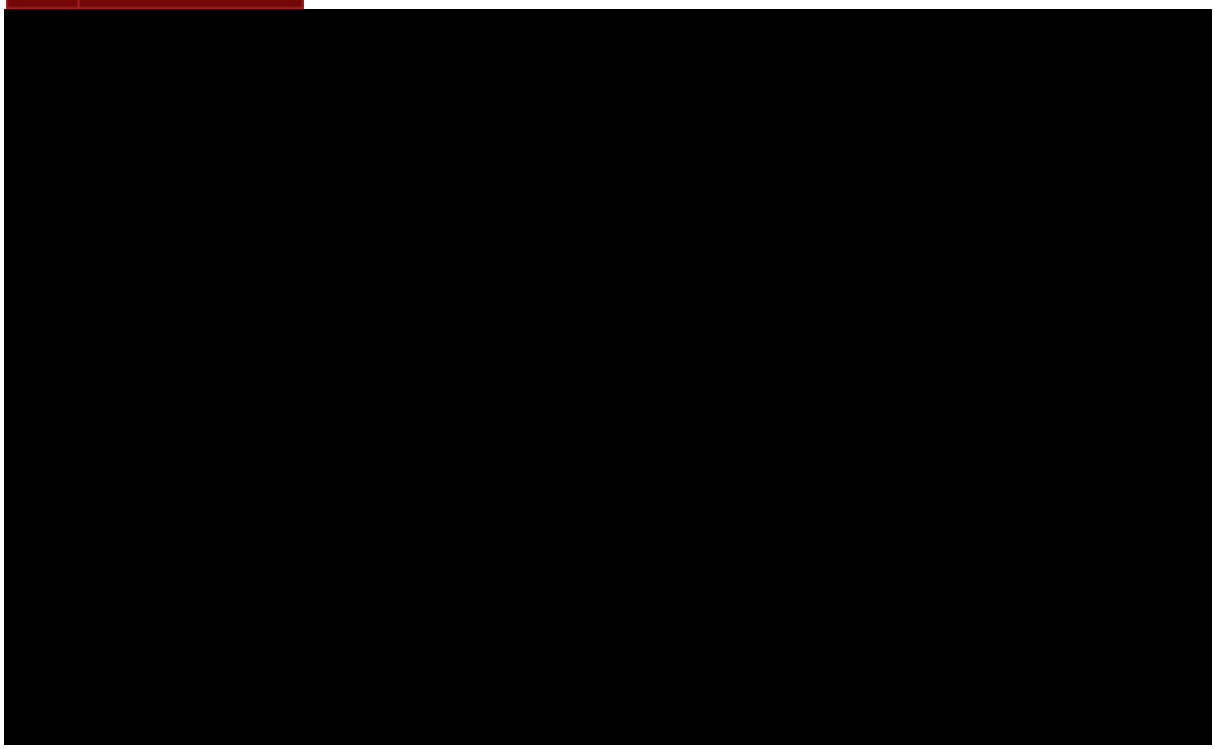


When enabled, pressing an "Ask" button with the left mouse button will submit a **limit** order at the ask price, pressing a "Bid" button will submit a **limit** order at the bid price.

For more information on adding and removing targets please view the Managing Positions section of the user help guide.

10.23.13.4 Modifying and Cancelling Orders

Play Video



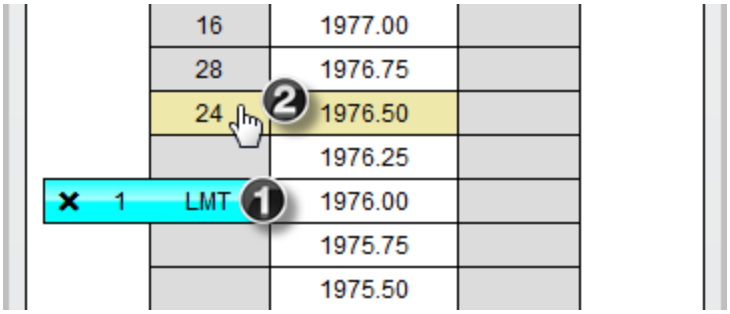
Orders are modified within the SuperDOM by selecting the order and clicking on the new price cell. Optionally you can also enable Single Click Order Modification of your Profit Target and Stop Loss orders within the SuperDOM Properties.

How to modify the price of entry and exit orders

Modifying entry and exit orders

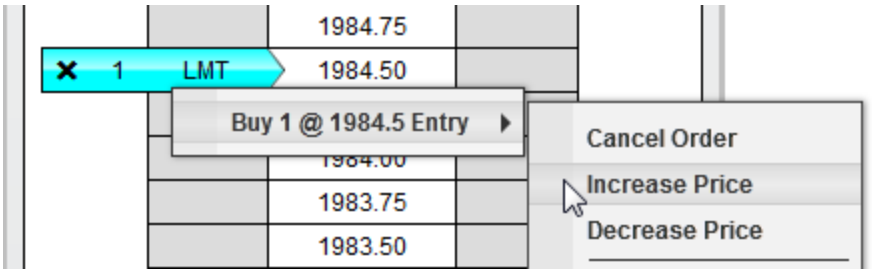
Pending orders in NinjaTrader may be modified by clicking to select the order and clicking once more at the new price point. This approach is more effective than drag and drop because it eliminates the potential errors made by accidentally letting go of your mouse button and dropping an order on the wrong price.

1. Click using your left mouse button on the order you wish to modify.
2. Once selected, you will see the cursor change to a hand from an arrow, then choose the price you are modifying the order to and click using your left mouse button to complete the modify process.



The left mouse button is used to modify the price of **limit**, **stop-market**, **stop-limit**, and **MIT** orders. You can cancel out of a price modification (remove the hand cursor) by pressing the ESC key.

You can also increase or decrease the price of an order by pressing down on the right mouse button with the mouse cursor hovering over the order, which will display all orders consolidated at that price. You can then select any individual order to **increase price** or **decrease price** in one tick increments



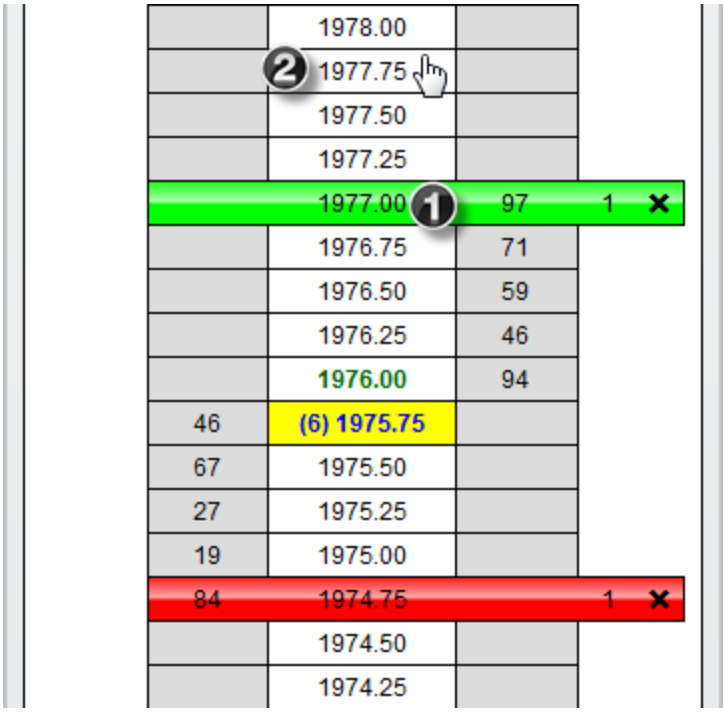
How to modify the price of Stop Loss and Profit Target orders

Play Video

Modifying Stop Loss and Profit Target orders

1. Click with your left mouse button in the center column on the Stop Loss or Profit Target order you want to modify.

2. Once selected, you will see the cursor change to a hand from an arrow, then choose the price you are modifying the order to and click using your left mouse button to complete the modify process.

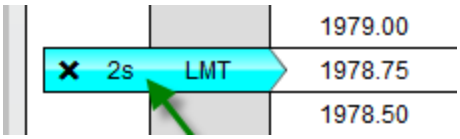


Note: If there are multiple orders consolidated at a price level, modifying the price will modify all orders at that price level.

▼ How to modify the size of an order

Modifying the size of an order

To modify the size of an order, click on the Size Marker (marked by the green arrow in the image below) with your left mouse button



The quantity field will appear which allows you to set the new order quantity by either entering a new quantity or using the mouse wheel to scroll the value higher or lower. Either press the "OK" button to submit the change or the "X" button to cancel the operation.

Order size changes are handled according to NinjaTrader's advanced FIFO optimization capabilities.

Tips:
1. Holding the Ctrl key + scrolling will increment order quantities by a value of 10
2. Middle clicking on the order quantity will bring up the Quantity Selector

How to modify Stop Loss and Profit Target orders with a single click

Single Click Order Modification

You have the option of enabling Single Click Order Modification for ATM Stop Loss and Profit Target orders via the SuperDOM Properties dialog window accessible by right mouse click context menu. This is an advanced feature that can provide you with the clear advantage of efficiently modifying orders in fast moving markets. If you are a scalper then this option is for you.

Once enabled, to modify Stop Loss and Profit Target orders click in the center/PRICE column. Clicking in the PRICE column on the BID or above when long will adjust your Profit Target order prices, below the BID will adjust Stop Loss order prices. Clicking in the PRICE column on the ASK or below when short will adjust your Profit Target order prices, above the ASK will adjust your Stop Loss order prices.

Left Mouse Click	Modifies the closest Stop Loss or Profit Target order
Middle Mouse Click	Modifies the second closest Stop Loss or Profit Target order
Middle Mouse Click + CTRL Key	Modifies the third closest Stop Loss or Profit Target order

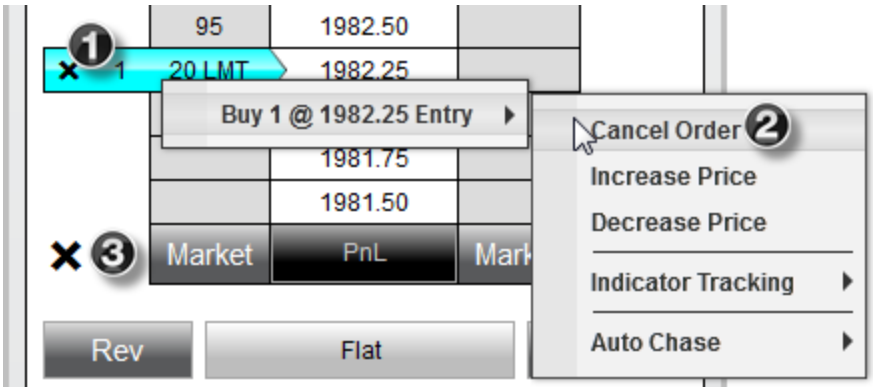
Notes:
1. Single Click order Modification for Stops and Targets are limited to only the first 3 nearest stops and targets
2. If you have more than one active strategy working in the market, single click modification will be applied to the stops and targets associated to the selected strategy as indicated in the strategy control list (drop down list) in the lower portion of the SuperDOM window.
3. This advanced mode DOES NOT provide single click access to working orders (Entry/Exit) that reside in either the BUY or SELL columns.

How to cancel orders

Cancelling Orders

There are several options for cancelling orders within the NinjaTrader SuperDOM.

1. Pressing down on the left mouse button on the black "**X**" will cancel all orders consolidated at the corresponding price level.
2. Pressing down on the right mouse button with the mouse cursor hovering over the order will display all orders consolidated at that price. You can then select any individual order for cancellation.
3. Pressing on the large "**X**" will cancel all orders on either the "BUY" side (in this example) or the sell side.

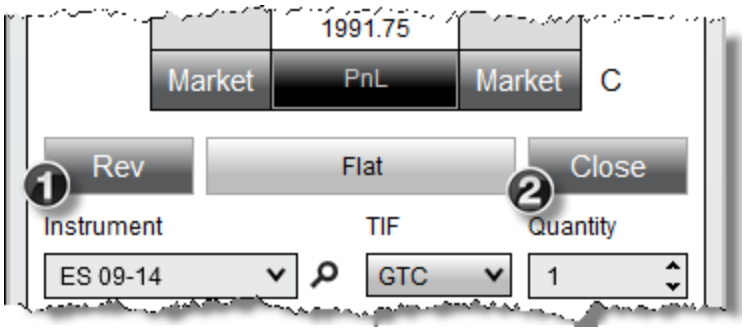


You can also cancel "ALL" orders by right mouse clicking inside the **SuperDOM** and selecting the menu item **Cancel All Orders**.

10.23.13. Managing Positions

Play Video

The SuperDOM has action buttons that allow you to quickly: close open positions, reverse positions, or even add/remove targets to your ATM Strategy

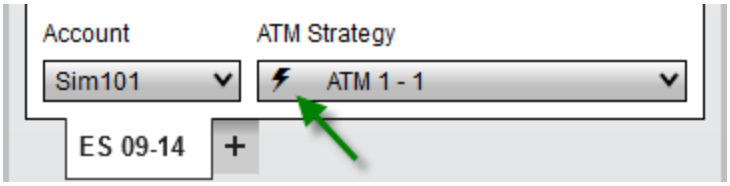


1. Left mouse clicking on the "Rev" will close the current open position and open a reverse position.
2. Left mouse clicking on the "**Close**" will close the current position and cancel any working orders associated with the instrument/account combination. Clicking on this button with your middle mouse button (scroll wheel) will close the selected active strategy only. This means that the position size of the strategy will be closed and any working orders associated to that strategy will be cancelled.

Note: Positions will only display for the selected Account and Instrument.

How to scale in or out of an active ATM strategy

When you have an active strategy selected in the strategy control list indicated by the lightning bolt icon (see image below), orders submitted scale into or out of the selected strategy. Once filled or partially filled, existing stop loss and profit target orders are modified to reflect the new position strategy size. You can preset a default scale in or out quantity via the "Scale quantity" property accessible via the SuperDOM properties window.



As an example, your initial strategy may call for opening a position of 4 contracts but you want subsequent scale orders to be only 1 contracts. If the SuperDOM "Scale quantity" property is set to a value of 1, when an active strategy is selected in the strategy control list, the SuperDOM "Order qty" field will be set to a value of 1 automatically.

Adding or Removing Targets

How to Add or Remove Targets

If you have an active ATM strategy displayed in the SuperDOM, you can add or remove targets. For example, you may have a 2 contract position with 1 Stop Loss and Profit Target for 2 contracts each. You may decide to split this target (add target) so you can exit the final contract at a higher price.

It is important to understand the following logic:

- If you have 1 target and you remove a target, you will be left with a stop loss order only
- New targets are added 4 ticks from your current outside target for futures, $0.20 for stocks

Two Methods for Adding and Removing Targets

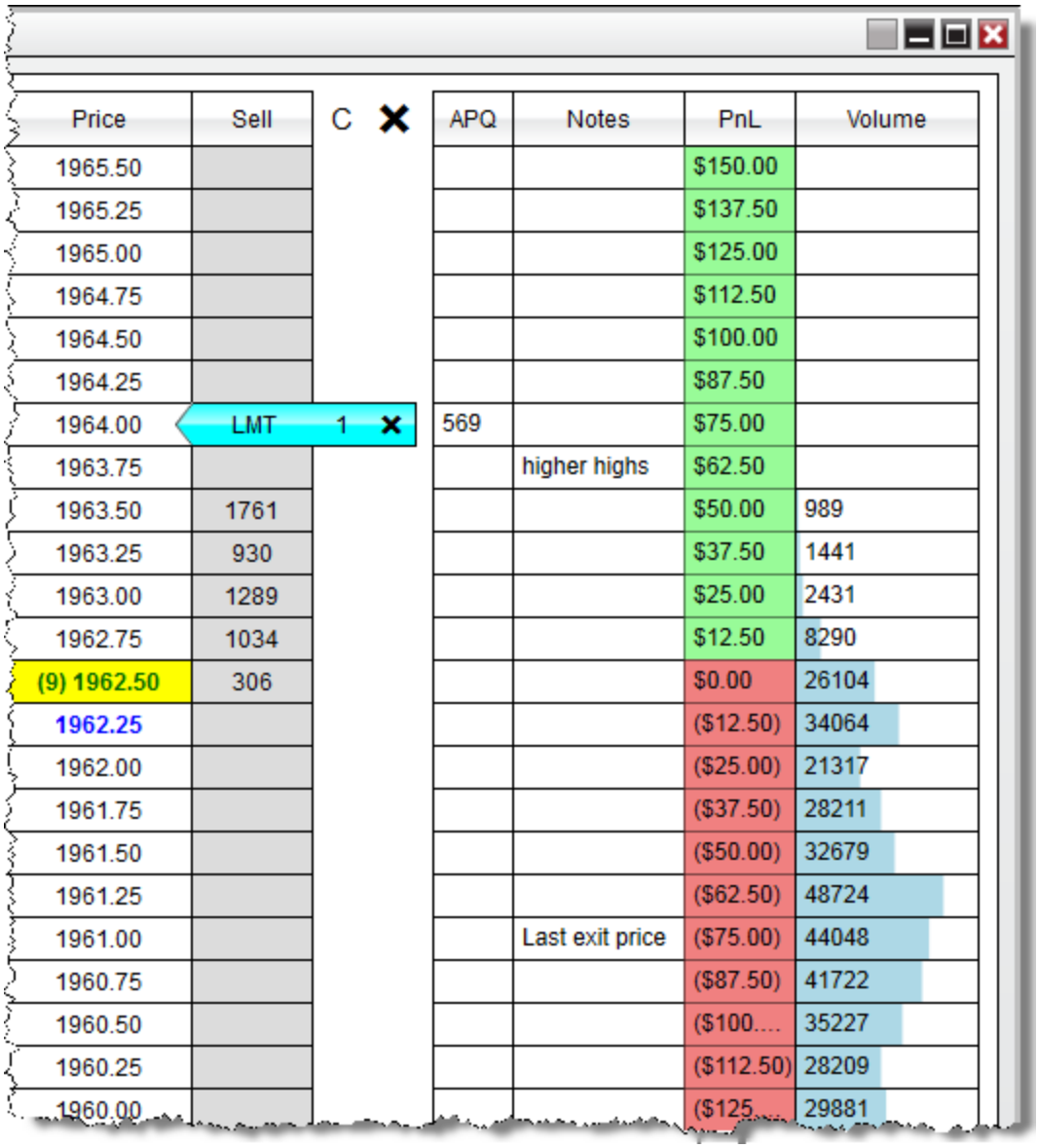
There are two locations within the SuperDOM where you can add or remove a target.

1. Pressing down on the Left mouse button on the "+ TARGET" (to add) or "- TARGET" (to remove) buttons when "Show Quick Buttons" is set to True in the SuperDOM properties dialog window
2. Right mouse click context menu and select **Add Target** or **Remove Target**

10.23.13.7 Using SuperDOM Columns

Play Video

In addition to the standard Price Column used to display bid/ask data, the NinjaTrader **SuperDOM** has the ability to add additional columns for even further analysis for real-time market prices. NinjaTrader comes with 4 pre-built system columns (displayed in the image below), with many more which can be downloaded to extend functionality.



Understanding the Columns window

The Columns window is used to add, remove and edit all columns within a **SuperDOM**

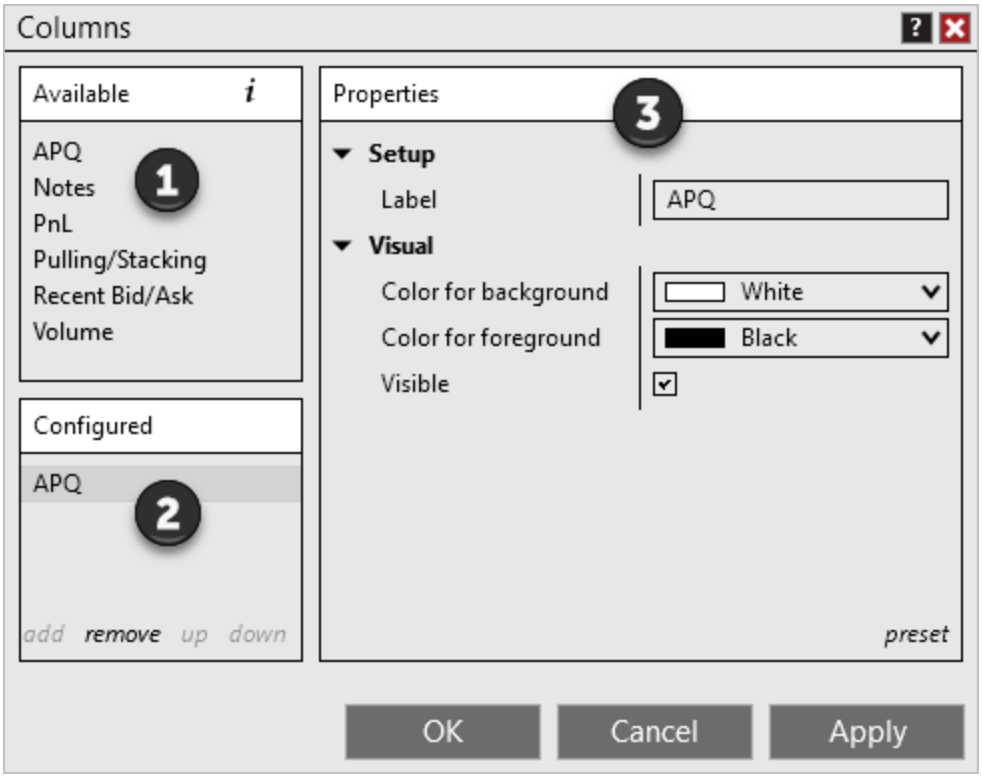
Accessing the Columns Window

Right mouse click on the **SuperDOM** window and select the menu **Columns**

Sections of the Columns Window

The image below displays the three sections of the **Columns** window:

1. List of **Available** columns (a description of the selected column can be viewed by clicking on the *i* symbol, see the green arrow in the image below)
2. Current columns **Configured** on the **SuperDOM**
3. Selected columns **Properties**

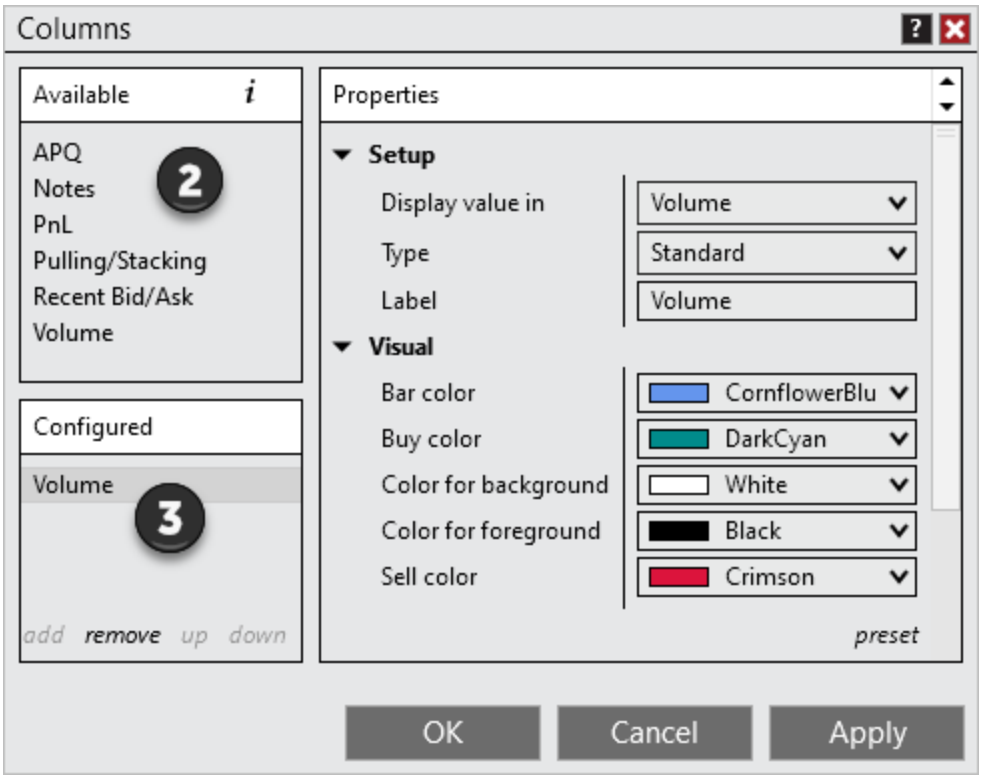


How to add columns

Adding a Column

To add an column to a **SuperDOM**:

1. Open the **Columns** window (see the "*Understanding the columns window*" section above)
2. Left mouse click on the **Available** column you want to add and press the Add button or simply double click on it
3. The column will now be visible in the list of **Configured** columns
4. The column's parameters will now be editable on the right side of the columns window (see the "*How to edit a column's parameters"* section below)

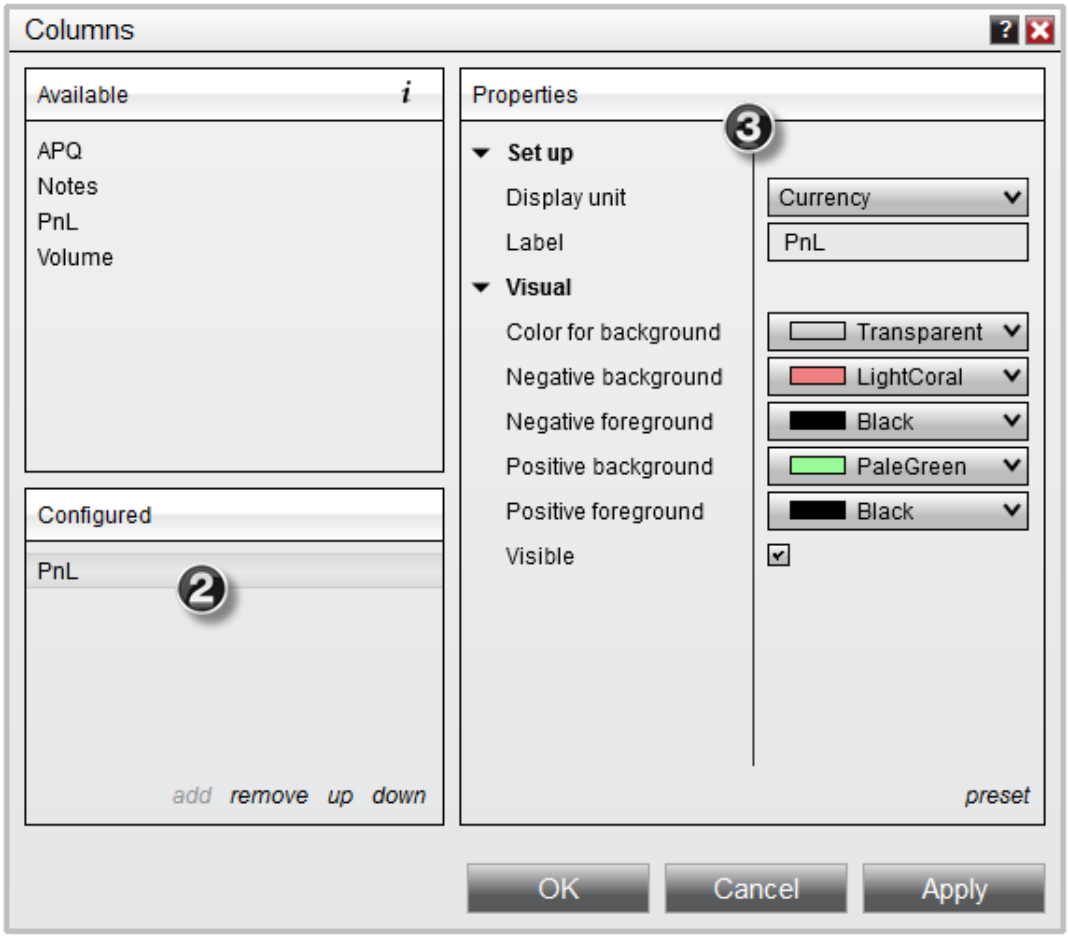


How to edit a column's parameters

Editing a Column

You can customize any column from the Columns window:

1. Open the columns window (see the "*Understanding the columns window*" section above)
2. Highlight the column you would like to edit from the list of applied columns (as shown in the image below).
3. Once highlighted this column's parameters will be available to edit on the right hand side.



Column Parameters

The following parameters are common to most columns:

Setup	
Label	Sets the text used for the column header
Visual	
Color for background	Sets the color used for the column cells
Color for foreground	Sets the color used for the column text

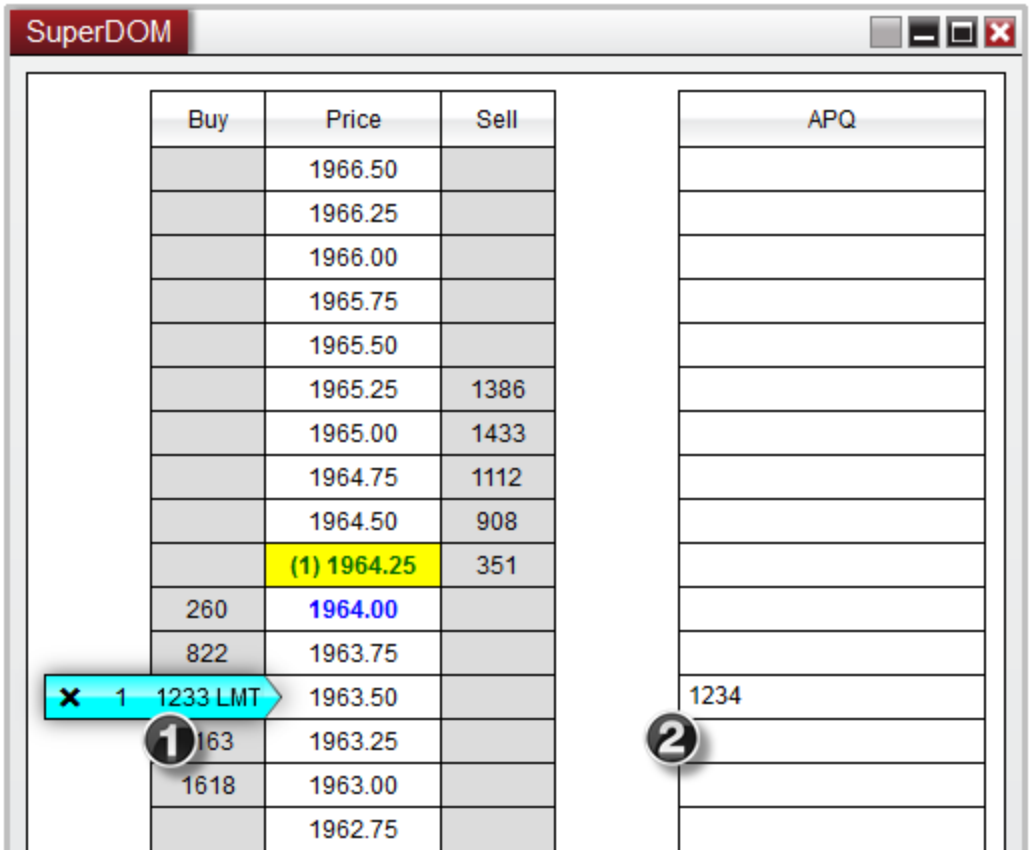
Visible	Enables / Disables if the column should be displayed on the SuperDOM
Time Frame	
Trading Hours	Sets the hours used for historical bar calculations

Each column will have its own set of parameters specific to that column. Please see the "*Understanding the default systems columns*" section below for more information on each of NinjaTrader's pre-built columns. For any custom columns that have been downloaded, please refer to the column's developer for more information on settings specific to their custom column.

Understanding the APQ (Approximate Position in Queue) Column

APQ (Approximate Position in Queue) Column

The **APQ** column will calculate the number of contract resting ahead of your **Limit** orders based on the number of contracts that were advertised at the time the order was submitted, in other words - it will give you the worst possible position in the queue for your order - so you know conservatively how many contracts need to be filled before it's your orders turn.



1. Let's say you place a Buy limit order at a price of 1963.50, and at the time the order was confirmed as working from the exchange, there were 1233 contracts working at this level ahead of you.
2. **APQ** will assume that your order has a queue position of 1234, and will continue to monitor the number of contracts that are advertised at this level, and give you the number of contracts that are remaining based off the volume updates that occur at that price level.

Notes:

- The value displayed in the **APQ** is a calculation based on the level II volume from your data provider. For simulated orders, there is no way to accurately track your order against the live orders that are being sent from the data provider and filled at a live exchange, and as a result, the estimate will have little to no value to your simulation orders. An order placed on a live account would be more accurately reflected, however it should be noted that this calculation is a client side calculated theoretical value.
- Stop orders do not have an **APQ** as it has no queue effect. A stop order will trigger once the market trades at the stop price.

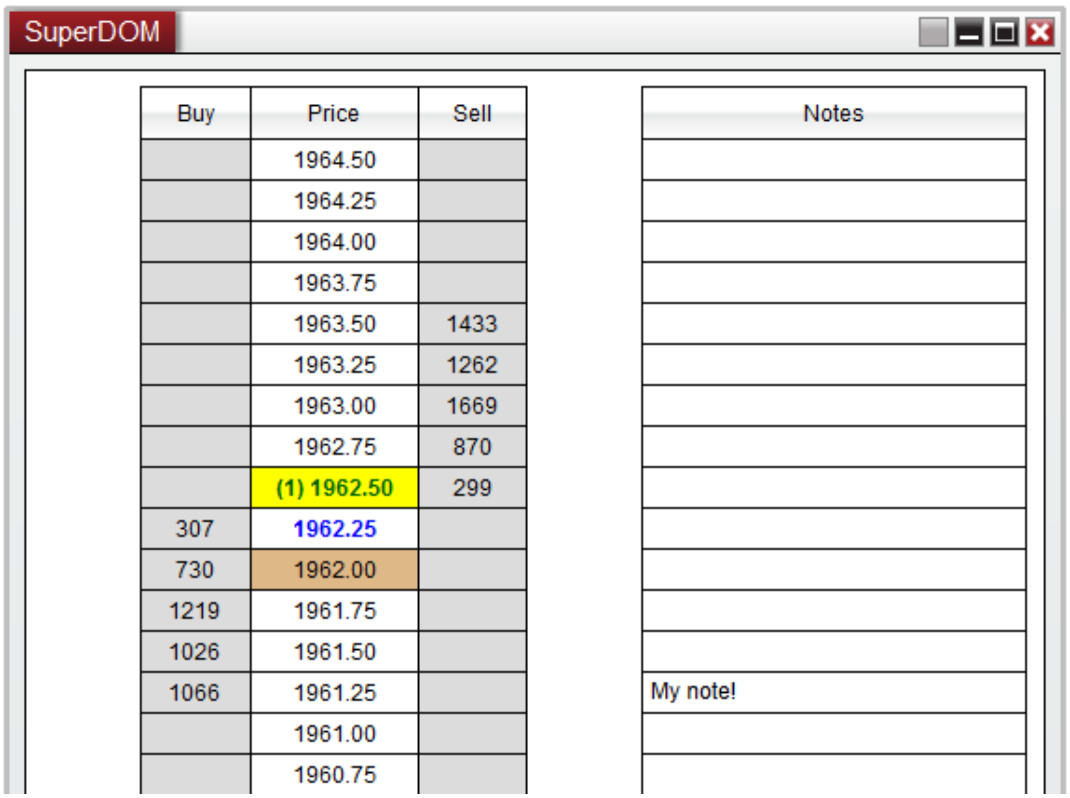
Understanding the Notes Column

Notes Columns

The **Notes** column will give you the ability to record custom user-defined text at any price row on the **SuperDOM**. This will allow you to monitor and track individual price levels with any text you may find useful.

To record a note:

1. Double click on the corresponding price row in the Notes column to enter the text-edit mode
2. Using your keyboard, type in the text you wish to display
3. Press Enter on your Keyboard accept the text.



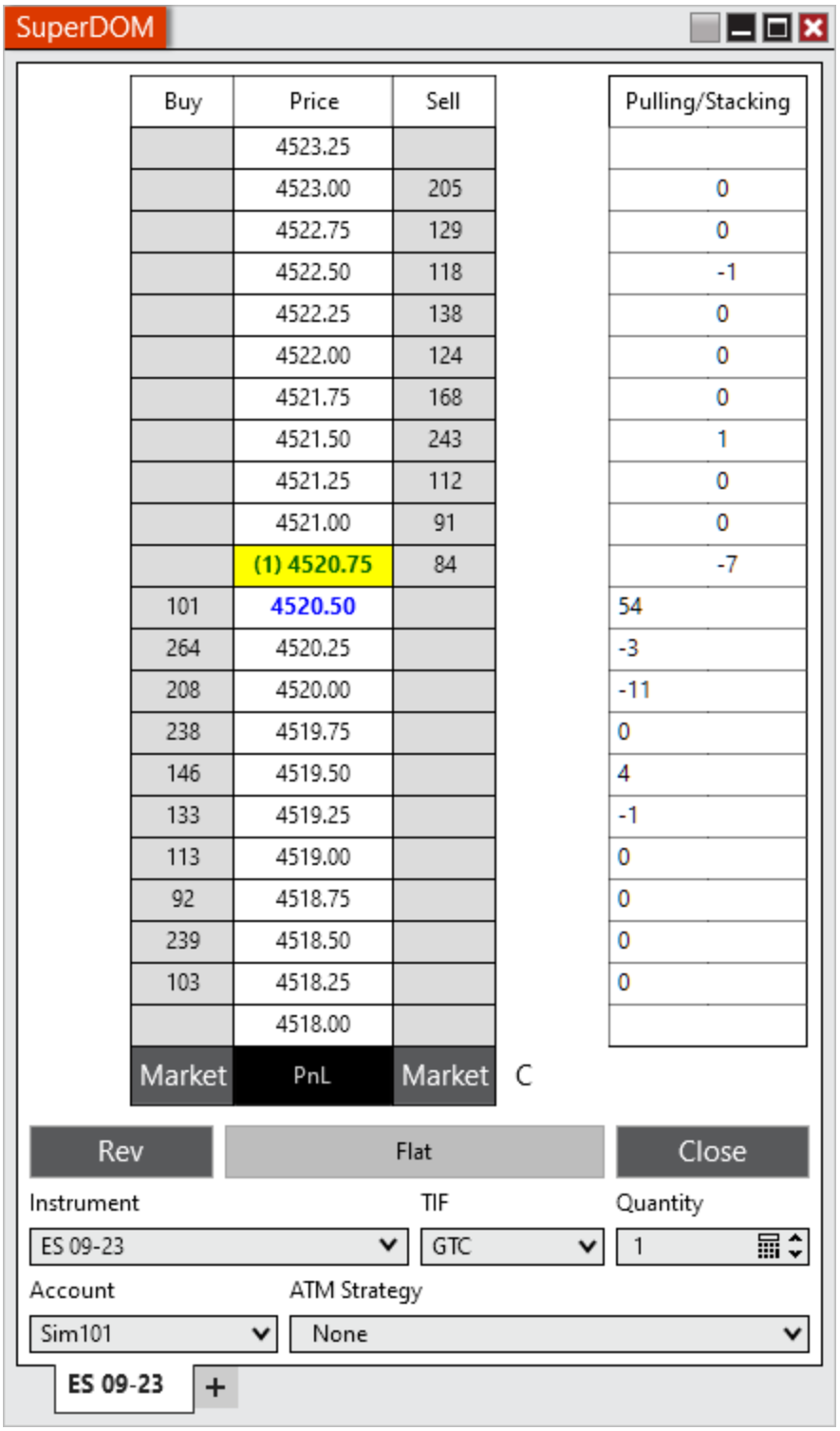
Your custom note will now be synchronized with the price corresponding price row and will remain at that price level as your scroll up or down on the **SuperDOM**.

To remove a note, simply double click on the note row to re-enter the text-edit mode which will allow you to erase the text using your backspace or delete key on your keyboard.

Understanding the Pulling/Stacking Column

Pulling/Stacking Columns

The **Pulling/Stack** column is a customizable display that indicates the changes in the market depth based on user settings or if a reset notification is received on the **SuperDOM**.



Example: In the screenshot above the sell depth at 4520.75 was initially at 91, but dropped to 84 resulting in a display of -7.

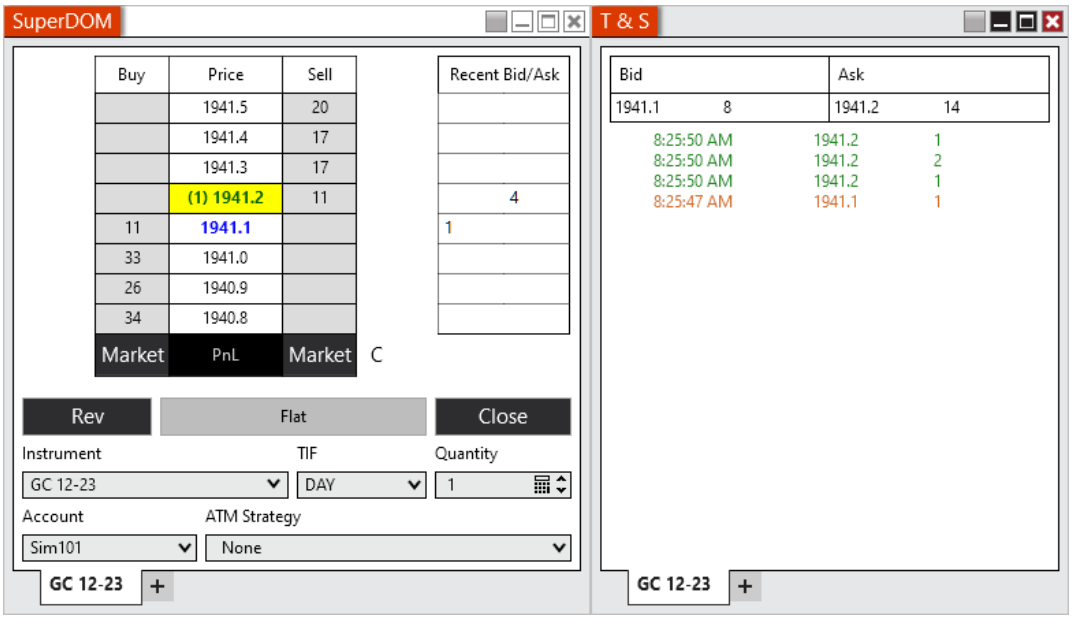
Column Properties

Display	Sets the ability to display values for Ask, Bid, or Ask & Bid
Reset when	Bid/Ask change: It will start tracking changes as soon as a price starts receiving real-time depth data and continue to accumulate the changes so long as the associated bid/ask price is consistent. If the associated bid/ask price moves, the values reset. It will only track values within the “# of market depth levels”. No longer receiving depth data: It will start tracking changes as soon as a price starts receiving real-time depth data and continue to accumulate the changes so long as real-time depth data is being received at that price level. If the price then moves and there is no longer depth data at that price level since it is outside of the “# of market depth levels”, it would then reset.
Reset Tolerance	This is a conditional item for Reset When so that if the change occurs, but returns back within the specified milliseconds, it will not rest.

Understanding the Recent Bid/Ask Column

Recent Bid/Ask Columns

The **Recent Bid/Ask** column is a customizable display that indicates the recent volume that occurred at the bid or ask prices on the **SuperDOM**.



Example: in the above screenshot we can see a volume of 1 occurred at the bid price and a volume of 4 occurred at the ask.

Column Properties

Display	Sets the ability to display values for Ask, Bid, or Ask & Bid
Reset when	Price returns: The accumulated volume will stay displayed until the associated bid/ask price leaves the price then later returns to that price level. Bid/Ask change: The accumulated volume will stay displayed so long as the bid/ask price stays the same. When the bid/ask price change, the value will reset.
Reset Tolerance	This is a conditional item for Reset When so that if the change occurs, but returns back within the specified milliseconds, it will not rest.

Understanding the PnL Column

PnL Column

The **PnL** column will display the amount of Profit or Loss for each price row based on your average entry price. This column has a setup property to display the number of units in Currency, Percent, Pips, Points, or Ticks (please see "*How to edit a column's parameters"* section above)

Once there is a position opened on the selected instrument, the **PnL** column will then calculate what you can expect your PnL to be at each price row on the **SuperDOM** based on the current position size, entry price and the tick size / point value of the instrument that is being traded.

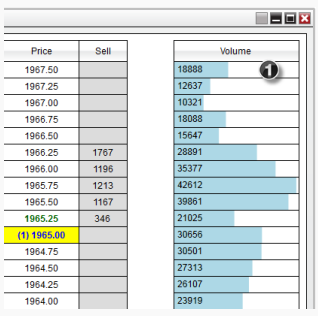
Understanding the Volume Column

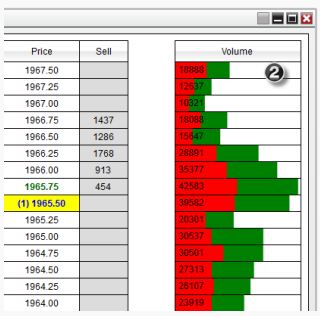
Volume Column

The **Volume Column** will display the number of contracts that have traded in the current session. This column has two Setup Properties to determine how the volume information is displayed. You data feed provider must support historical tick data and is using the **Volume Column** in Buy/Sell mode must also support Historical Bid/Ask tick data.

> **Note**: The **SuperDOM Volume Column** will reset as the first tick of the next session comes in. If you open a fresh **SuperDOM Volume Column** outside of the instruments trading hours you will not see any Volume until the next sessions opening tick.

Display value in	
Volume	Displays the actual number of contracts executed at each price level
Percent	Displays a value percentage based off of the total number of contracts traded in the session
Type	
1. Standard	Trades are represented as the cumulative number of contracts that have been executed at each price level
2. BuySell	Trades are categorized as a buy (at the ask or above) or as a sell (at the bid or below) and then color coded based on the color parameters used in the Visual section (see *"How to edit a column's parameters"* section above





How to remove columns

Removing a Column

To remove a column from your NinjaTrader **SuperDOM**:

- Open the **Columns** window (see the "*Understanding the Columns window*" section above), select a column from the **Configured** columns list, press the **Remove** button, and then press the **OK** button to exit the **Columns** window.

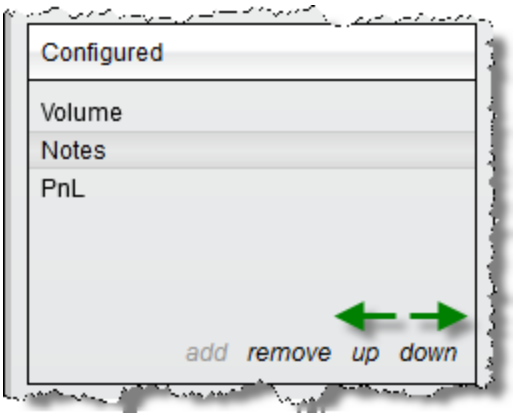
Customize the display of columns

Moving/Resizing Columns

Each column added to the **SuperDOM** can be individually resized or moved.

To move the order of columns in the **SuperDOM** window

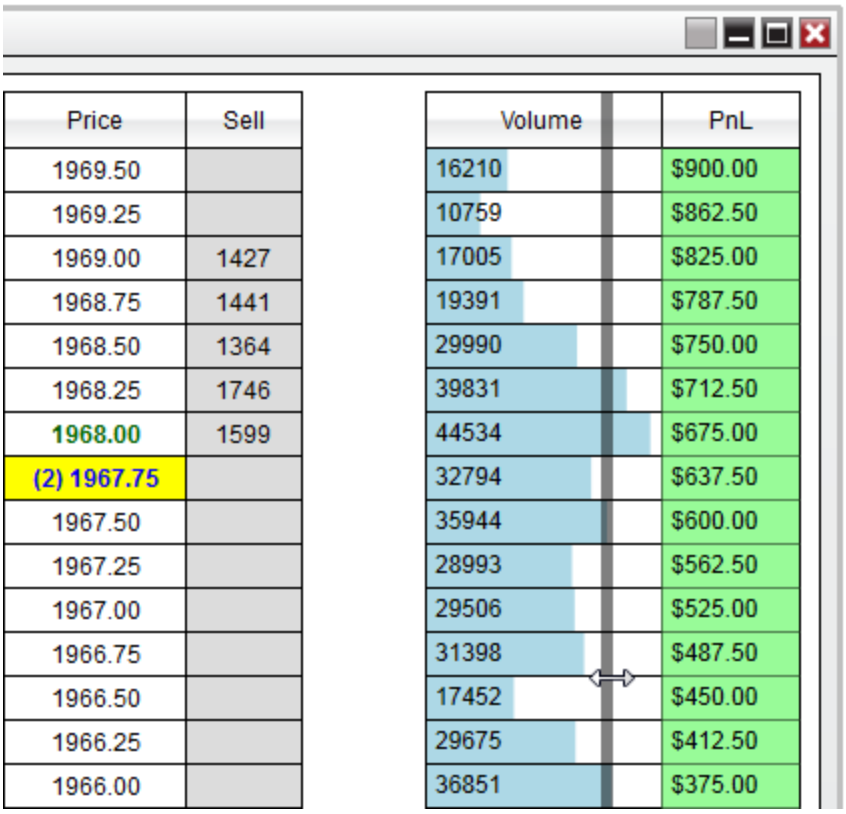
- Right click on the **SuperDOM** and select **Columns**.
- From the Columns window you can use "up" or "down" in the **Configured** columns section.
- Left mouse click "up" to move the selected applied column left in the SuperDOM window
- Left mouse click "down" to move the selected applied column right in the SuperDOM window



To resize the width of a column:

- Move your cursor to the edge of the column you wish to resize, where your cursor will turn into a left and right facing arrow

- Left mouse click and drag to meet the width you desire

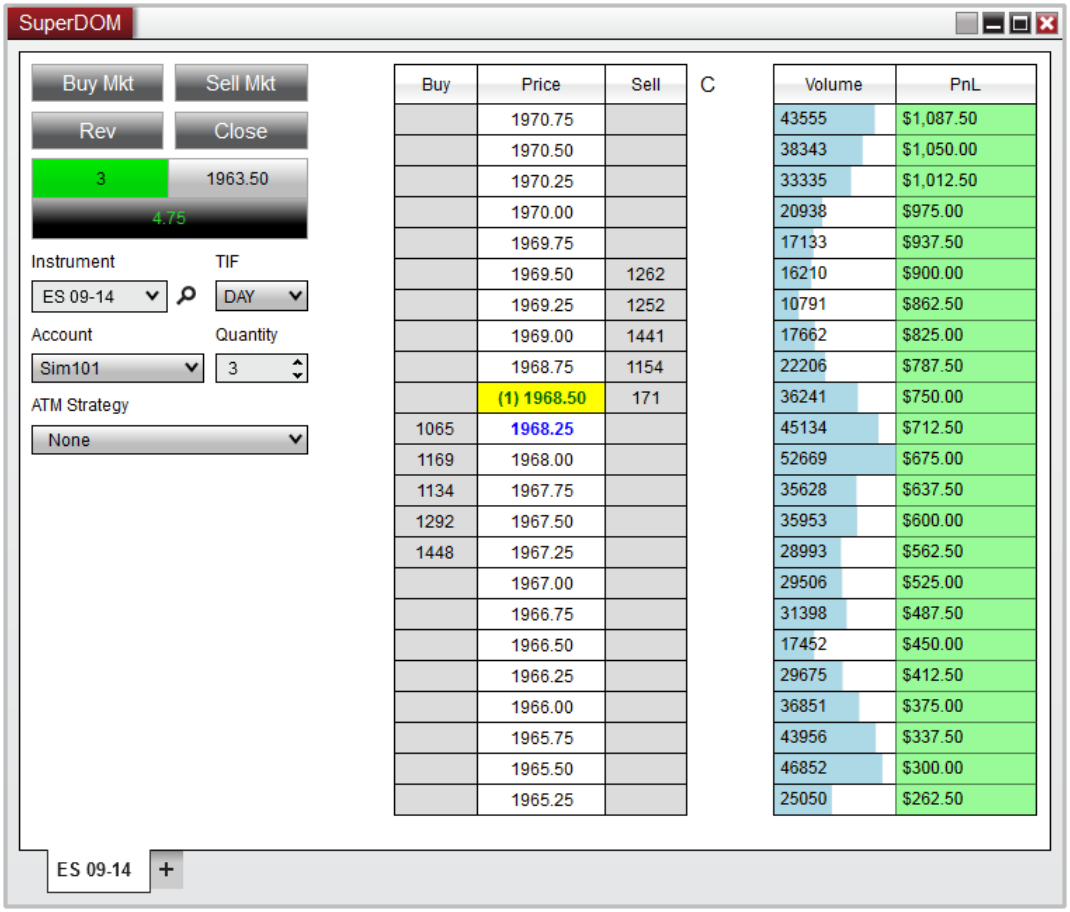


Price	Sell
1969.50	
1969.25	
1969.00	1427
1968.75	1441
1968.50	1364
1968.25	1746
1968.00	1599
(2) 1967.75	
1967.50	
1967.25	
1967.00	
1966.75	
1966.50	
1966.25	
1966.00	

Volume	PnL
16210	$900.00
10759	$862.50
17005	$825.00
19391	$787.50
29990	$750.00
39831	$712.50
44534	$675.00
32794	$637.50
35944	$600.00
28993	$562.50
29506	$525.00
31398	$487.50
17452	$450.00
29675	$412.50
36851	$375.00

Trade Control On Left

By default, the **Trade Control** will be displayed on the bottom of the **SuperDOM**. However you can optionally set the Trade Control to be displayed on the left of the **SuperDOM** Price Ladder for a more compacted view which has been optimized for using multiple columns on the **SuperDOM**. To enable this display, simply right click on the **SuperDOM** window and select the **Trade Control On Left** menu item.



Custom column development

In addition to the 4 system columns that come pre-built with the NinjaTrader application, you also have the ability to create custom columns of your own. For example, you could create your own custom volume column to apply to your NinjaTrader SuperDOMs.

For more information on using NinjaScript to build custom SuperDOM Columns please see the NinjaScript section of the user help guide.

The option to hire a https://ninjatraderecosystem.com/search-results/?fwp_category=programming-services to build your custom indicators is also available.

10.23.13.8 SuperDOM Templates

SuperDOM templates allow you to save a variety of visual and functional properties for the **SuperDOM**, allowing you to quickly recall these settings at a later time.

How to save a SuperDOM Template

A **SuperDOM Template** can be applied to a new or previously opened **SuperDOM** to load customized settings, including any additional columns saved as part of the template.

Saving a SuperDOM Template

To save a **SuperDOM Template:**

1. Once you have a **SuperDOM** set up to your liking, right mouse click within the window and select the menu item **Templates**, followed by **Save As**
2. The **Save As** window will appear. Enter a name for your template and press the **save** button.

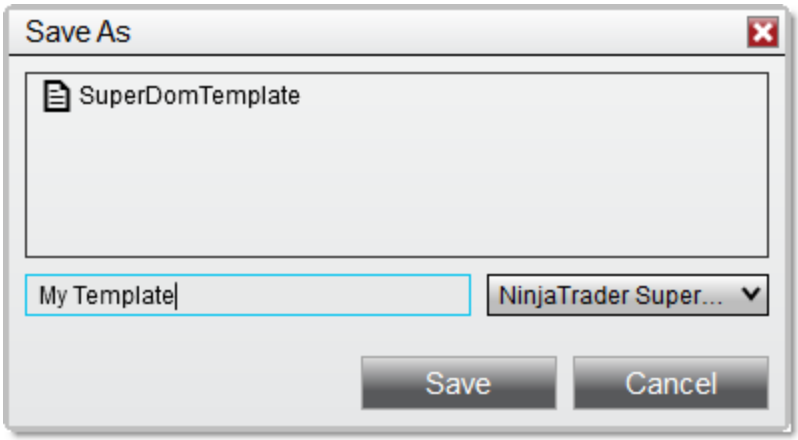
Changing the Default SuperDOM Template

A **SuperDOM Template** can be saved as the default used for all new **SuperDOM** windows. Once saved, the default template will determine the properties of each new **SuperDOM** opened, unless you specify a different template.

To save a **SuperDOM Template** as default:

1. Right mouse click within an open chart and select the **Templates** menu
2. Select the menu item **Save as Default**

In the image below, we are saving a new chart template named "My Template."



How to load, remove, or rename a SuperDOM Template

Loading a SuperDOM Template

A **SuperDOM Template** that was previously saved can be loaded on any **SuperDOM** window.

To load a **SuperDOM Template**:

1. Right mouse click and select the menu item **Templates** followed by the **Load** menu item
2. The **Load** window will appear. Select the template to load from the list of templates, then press the **Load** button.

Removing a SuperDOM Template

To remove a **SuperDOM Template** from the list of saved templates:

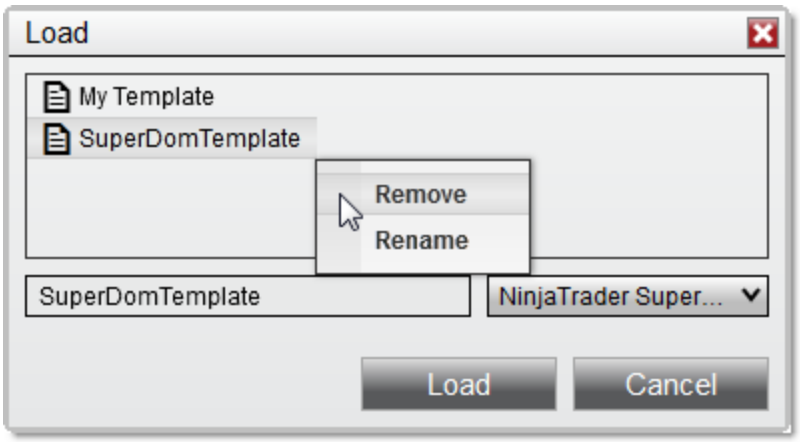
1. Right mouse click within a chart and select the menu item **Templates** followed by either the **Save As** or **Load** menu items
2. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the template for removal from the list of templates, then select the **Remove** menu item.

Renaming a SuperDOM Template

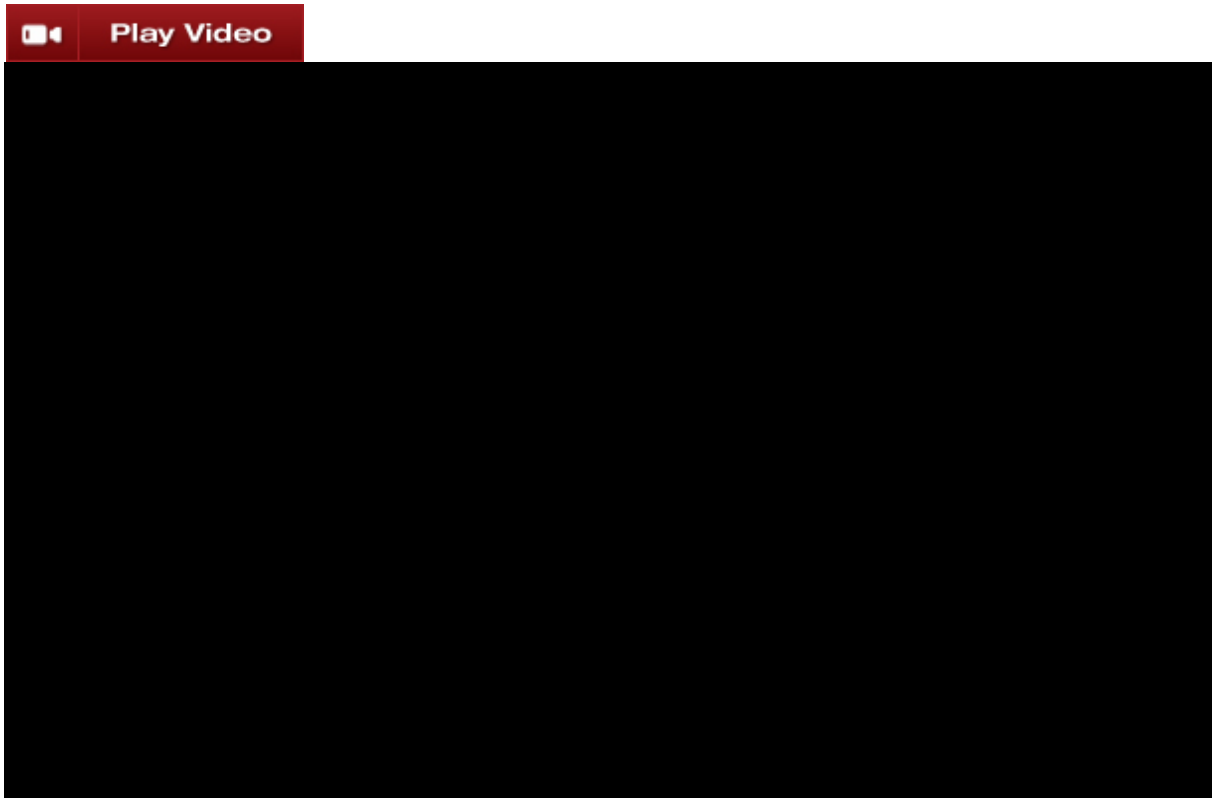
To rename an existing **SuperDOM Template** from the list of saved templates:

3. Right mouse click within a chart and select the menu item **Templates** followed by either the **Save As** or **Load** menu items
4. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the template from the list of templates, then select the **Rename** menu item.

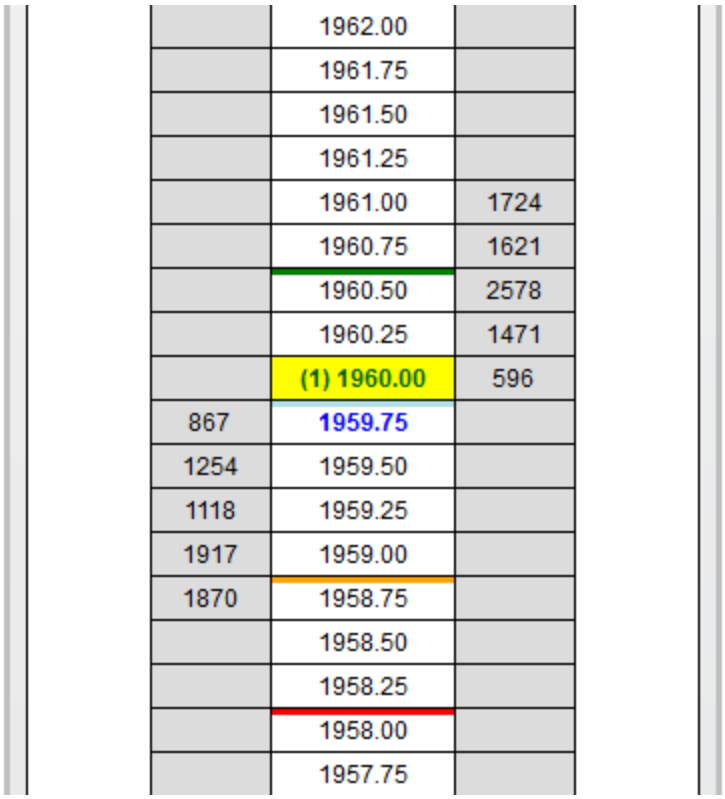
In the image below, we can either remove or rename the selected **SuperDOM Template**.



10.23.13.9 Working with Indicators



The **SuperDOM's** Price Ladder display has the ability to add any number of price action indicators which can be used to visualize and analyze indicator values in relation to the **SuperDOM** display, as well as attaching working orders to the indicator price level for a hand-free trade management system.



NinjaTrader comes with over 30 pre-built indicators which can be added the **SuperDOM**. Indicators can be added, removed and edited via the Indicators window.

Understanding the Indicators window

The Indicators window is used to add, remove and edit all indicators within a **SuperDOM**.

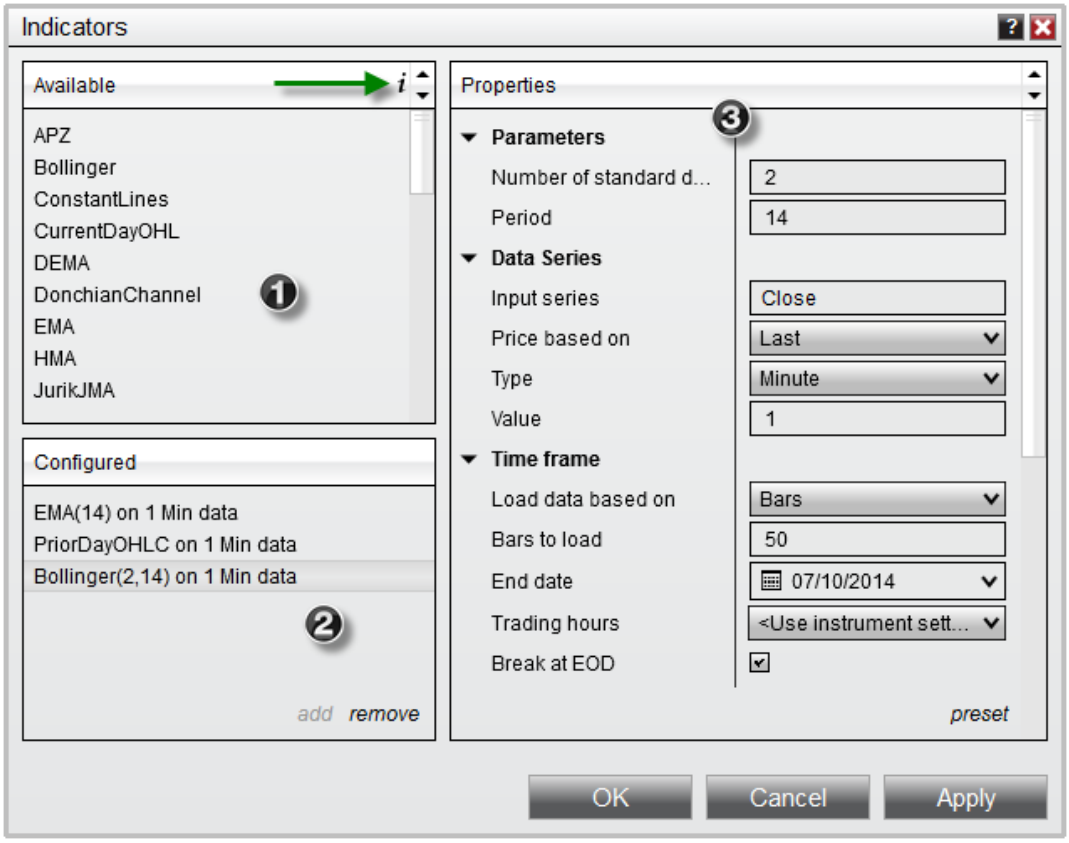
Accessing the Indicators Window

- Right mouse click in the **SuperDOM** select the menu **Indicators**

Sections of the Indicators Window

The image below displays the three sections of the Indicators window.

1. List of **Available** indicators (a description of the selected indicator can be viewed by clicking on the *i* symbol, see the green arrow in the image below)
2. Current indicators **Configured** on the **SuperDOM**
3. Selected indicator's **Properties**

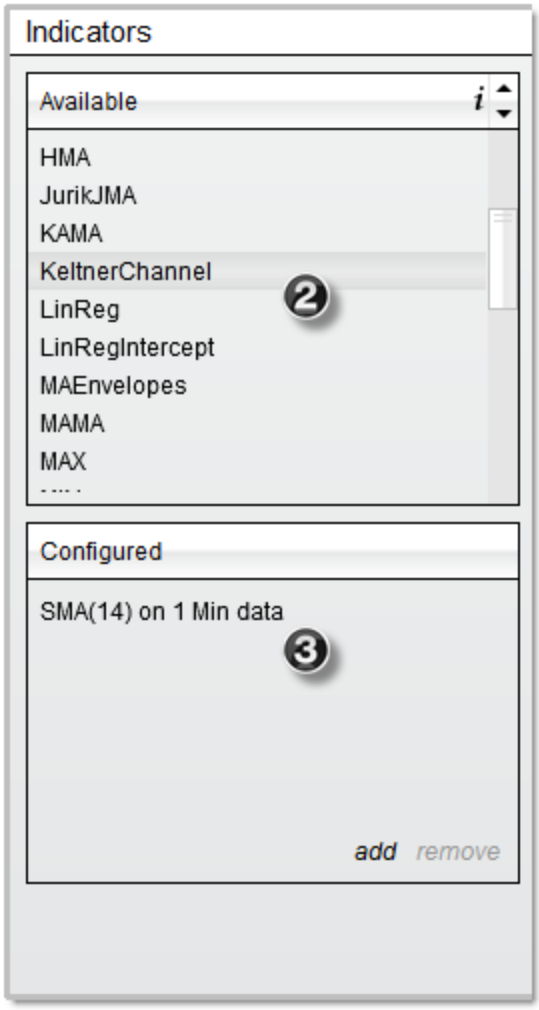


How to add an indicator

Adding an Indicator

To add an indicator to a **SuperDOM**:

1. Open the Indicators window *(see the "Understanding the Indicators window" section above)*
2. Left mouse click on the **Available** indicator you want to add and press the **Add** button or simply double click on it
3. The indicator will now be visible in the list of **Configured** indicators
4. The indicator's parameters will now be editable on the right side of the Indicators window *(see the "How to edit an indicator" section below)*

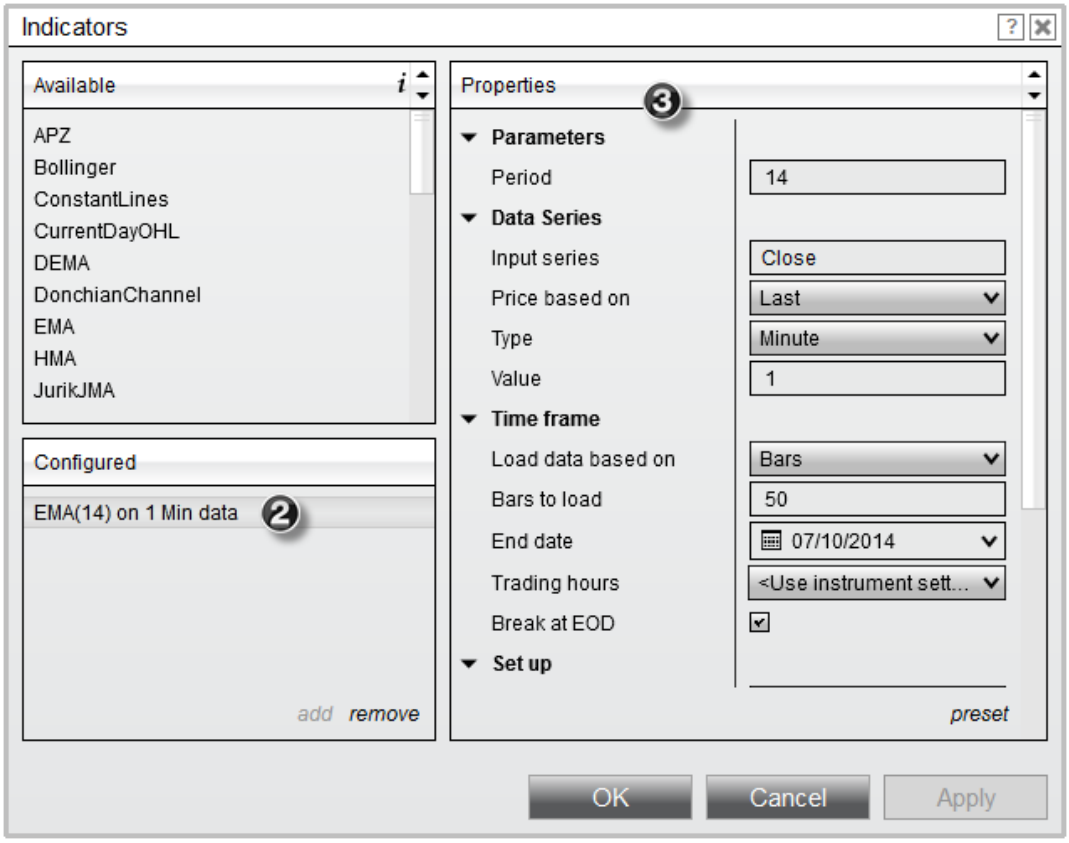


How to edit an indicator's parameters

Editing an Indicator

You can customize any indicator from the Indicators window:

1. Open the Indicators window (see the *"Understanding the Indicators window"* section above)
2. Highlight the indicator you would like to edit from the list of applied indicators (as shown in the image below).
3. Once highlighted this indicator's parameters will be available to edit on the right hand side.



Indicator Parameters

The following parameters are common to all indicators:

Data Series	
Input Series	Please see the Input series section for further information.
Price based on	Sets the type of market data used to drive the Data Series. (Last, Ask, Bid)
Type	Sets the bar type of the Data Series. (See the Bar Types section of the Help Guide for more information)
Value	Sets the Data Series value.

Time frame	
Load data based on	Determines how much data is loaded based on number of bars, number of days, or a custom date range.
Bars to load	Sets the number of bars or days to load data.
End date	Sets the end date of the data used in indicator's calculation
Trading hours	Sets the Trading hours used for the Data Series. (See the Trading Hours section of the Help Guide for more information)
Break at EOD	Enables or disables the bars being reset at EOD (End Of Day). (See the "Understanding Historical Data" section of the Help Guide for more information)
Set up	
Calculate	Sets the frequency that the indicator calculates. **On bar close** will slow down the calculation until the close of a bar; **On price change** will calculate on when there has been a change in price; **On each tick** calculate the indicator's value which each incoming tick.
Maximum bars look back	Max number of bars used for calculating an indicator's value. The TwoHundredFiftySix setting is the most memory friendly.
Visual	
Visible	Sets if the indicator plot is visualized on the display

Plots (...)	Allows you to customize the appearance of the indicator by changing the Color or Thickness

Saving an Indicator's Parameters as Default

You can optionally save your customized indicator's parameters as a default preset. Doing so will recall your customized settings the next time you add this specific indicator to a **SuperDOM**.

Once you have your indictor's properties set to your preference, you can left mouse click on the "**preset**" text located in the bottom right of the properties dialog. Selecting the option "**save**" will save these settings as the default settings used every time you open a new window/tab.

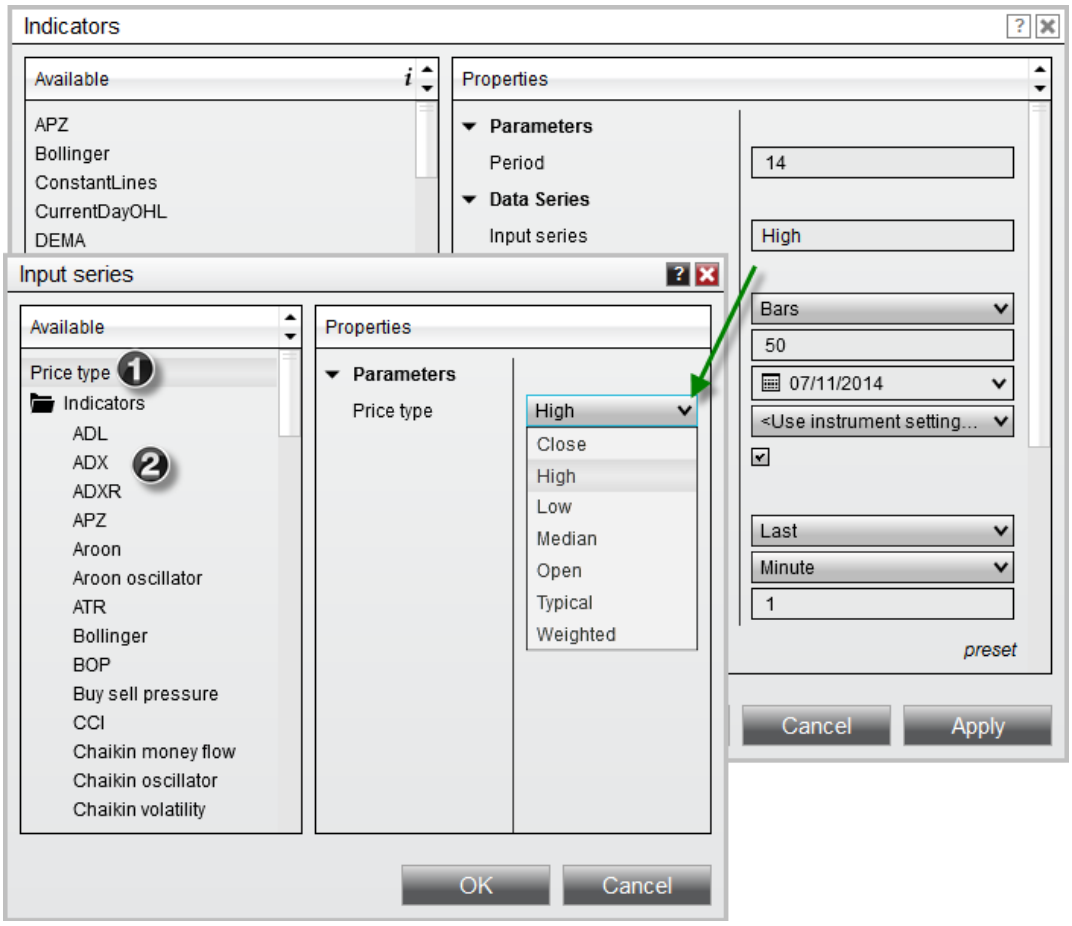
If you change your settings and later wish to go back to the original settings, you can left mouse click on the "**preset**" text and select the option to "**restore**" to return to the original settings.

Indicator Input Series

The indicator Input Series window allows you to select the input series for your indicator's calculations. This allows you to configure different data types, such as the High, or Open price, or even calculate your indicators based off of multiple nested indicators.

To access this window, move your mouse over the **Input Series** field, which will change to an "**Edit input...**" button.

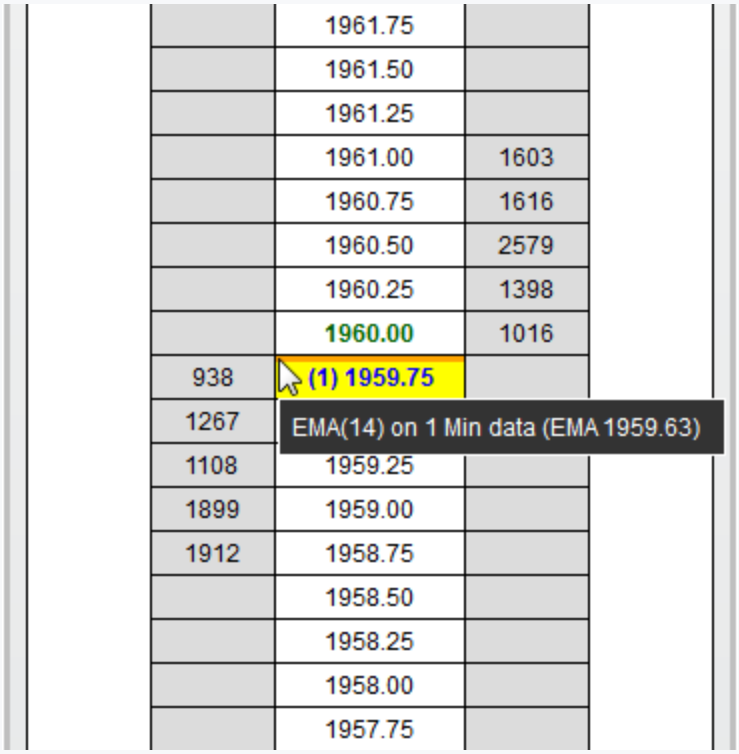
1. You can then select the Close, High, Low, Median, Open, Typical, or Weighted value of any Data Series within a **SuperDOM**.
2. Additionally, you can also choose another indicator as the input series. When you select another indicator as the input series, you can define the properties used in the input series for the second indicator. Once you have selected the input series of your choice left mouse click the OK button to exit the Input Series window.



Understanding how indicators are displayed

Indicator Display

Once an indicator has been configured and applied to the **SuperDOM**, the indicator plot will be displayed in the **Price column** above the corresponding price row.



In the image above, you can see an orange highlighted price row at 1959.75, rounded to nearest price from the calculated EMA indicator (1959.63). Hovering your mouse cursor over the indicator plot will display a tool tip which will give you details pertaining to input settings of the indicator.

Note: It is possible for indicators to be calculated out of range of the current **Price Ladder Display**. You can right click on the **SuperDOM** and uncheck **Auto Center** which will allow you to scroll up or down on the **Price Ladder Display** to locate the indicator that has been added.

How to remove an indicator

Removing an Indicator

To remove an indicator from your NinjaTrader **SuperDOM**:

- Open the Indicators window *(see the "Understanding the Indicators window" section above)*, select an indicator from the **Configured** indicators list, press the Remove button, and then press the OK button to exit the Indicators window.

Custom indicator development

In addition to the over 30 price action indicators that come pre-built with the NinjaTrader application, you also have the ability to create custom indicators of your own. For example, you could create your own custom multi-series indicators to apply to your NinjaTrader **SuperDOMs**.

> **Note**: In order for a custom indicator to show up in the list of available **SuperDOM Indicators**, you must set the IsOverlay property to **true** in the indicator's **State.SetDefaults.**

For more information on using NinjaScript to build custom indicators please see the NinjaScript section of the user help guide. Click here to view NinjaScript tutorials.

The option to hire a NinjaScript Consultant to build your custom indicators is also available.

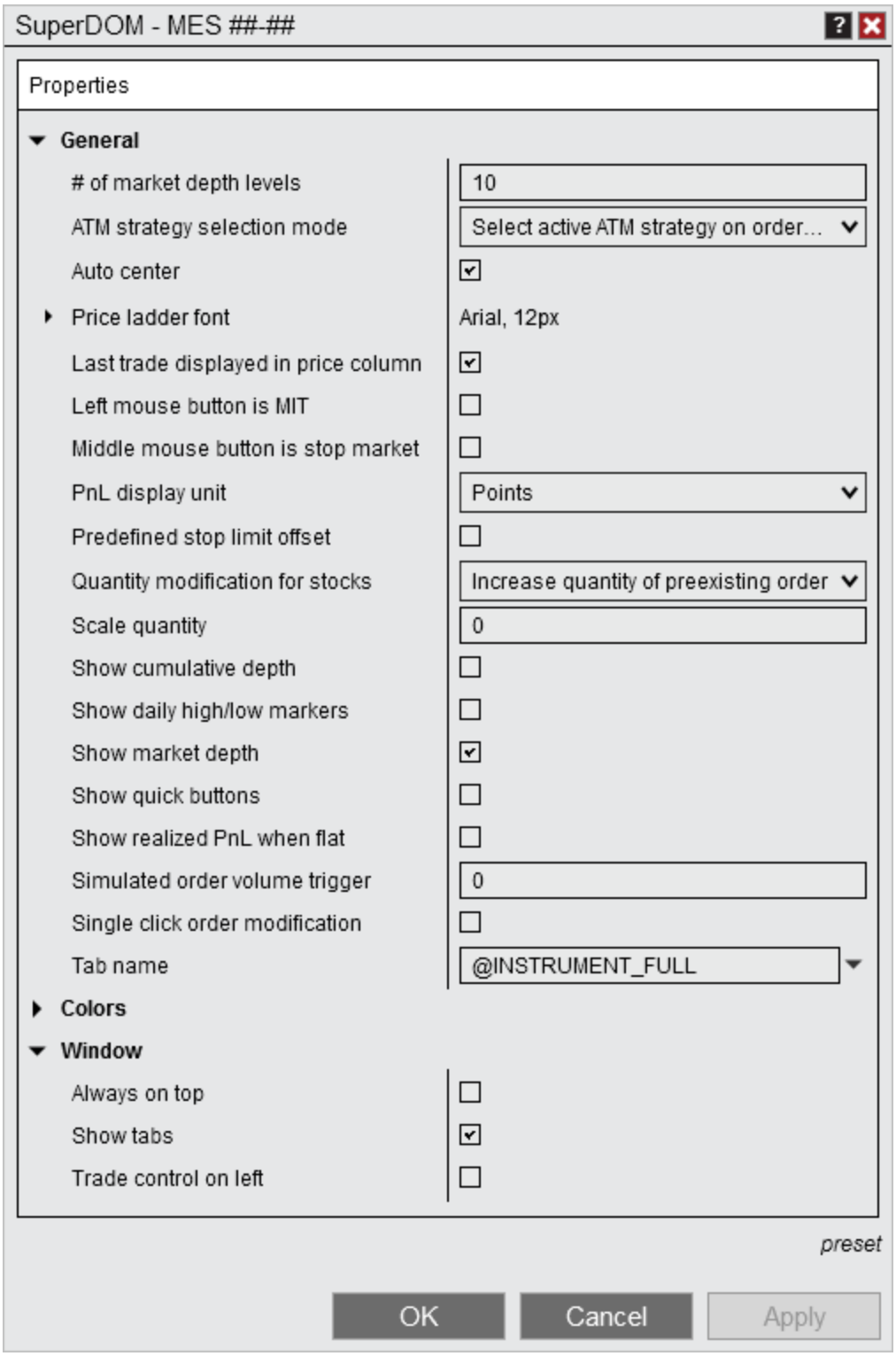
10.23.13. Properties

The SuperDOM is highly visual by design but can also be customized to each trader's preferences.

How to access the SuperDOM properties menu

You can access the SuperDOM properties dialog window by clicking on your right mouse button within the SuperDOM border and selecting the menu **Properties**.

Available properties and definitions



General	

# of market depth levels	Sets the number of market depth (Level 2) rows displayed
ATM strategy selection mode	Sets the behavior mode of the price ladder display and strategy selector (more information here)
Auto center	Enables or disables auto centering of the last traded price when it trades outside of range
Price ladder font	Sets the font options for the price ladder
Last trade displayed in price column	When true, the last trade volume is displayed in the center price column otherwise it is displayed in either the buy or sell column
Left mouse button is MIT	Sets if the left mouse uses a MIT order (Limit order by default)
Middle mouse button is stop market	Sets if the middle mouse (scroll wheel) button is stop-market (Stop-Limit by default)
PnL display unit	Sets the display unit for profit and loss
Quantity modification for stocks	Sets if new orders submitted at the same price will modify the quantity of exiting orders, or an entirely new order is submitted (stacked) at the same level
Scale quantity	Sets the scale order quantity amount

Show cumulative depth	Enables or disables cumulative market depth to be shown
Show daily high/low markers	Enables or disables the daily high and low markers to be shown
Show market depth	Enables or disables market depth
Show quick buttons	Enables or disables the quick buttons rapid order entry section
Show realized PnL when flat	Displays realized profit and loss for the selected account when flat
Simulated order volume trigger	Sets the value for a simulated order volume trigger (for entry and exit orders and NOT used for stop loss)
Single click order modification	Enables or disables single click stop loss and profit target order modification
Predefined stop limit offset	Sets the offset the limit price is away from the stop price for entry/exit stop-limit orders. Set to 'Off' to disable single click stop-limit order submission.
Tab name	Sets the tab name
Colors	

Action button	Sets the set the color for any action button's background
Ask price	Sets the color of the ask price font
Bid price	Sets the color of the bid price font
Buy button	Sets the color of the buy button
Buy column background	Sets the color of the buy column background
Buy column foreground	Sets the color of the buy column font
Daily high price	Sets the color of the daily high price marker
Daily low price	Sets the color of the daily low price market
Entry price	Sets the color of the average entry price marker
Highlight background	Sets the color for row and button highlighting
Last trade	Sets the color for the last trade market
Order - (...)	Sets the color for various orders displayed
Price column background	Sets the color for the price column background

Price column foreground	Sets the color for the price column font
Sell button	Sets the color for the sell button
Sell column background	Sets the color of the sell column background
Sell column foreground	Sets the color of the sell column font
Window	
Always on top	Sets if the window will be always on top of other windows
Show tabs	Sets if the window should allow for tabs
Trade control on left	Sets if the SuperDOM Trade Control is displayed on the left of the price column

How to set the default properties

Once you have your SuperDOM Properties set to your liking, you can left mouse click on the **preset** button, then click **save**. Presets will be applied to all windows of that type opened in the future.

If you change your settings and later wish to go back to the original factory settings, you can left mouse click on the **preset** button, then click **restore** to return to the factory settings.

Using Tab Name Variables

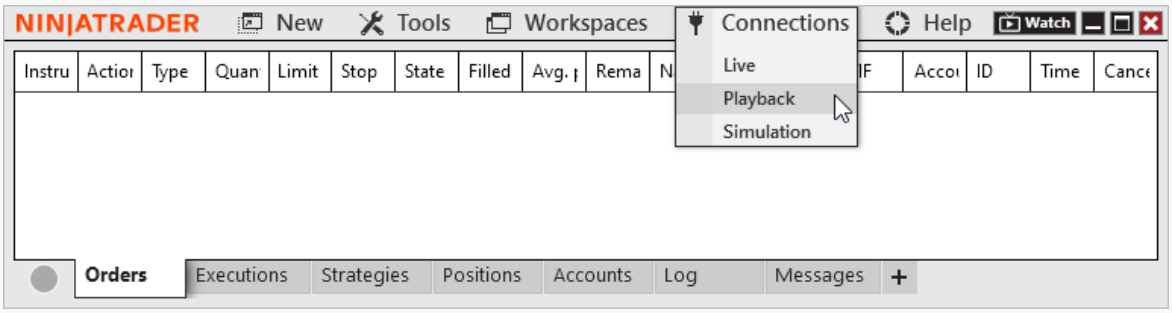
Tab Name Variables

A number of pre-defined variables can be used in the "Tab Name" field of the **SuperDOM Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.24 Playback Connection

Playback Connection Overview

The **Playback Connection** can be accessed by left mouse clicking on the **Connection** menu within the NinjaTrader Control Center and selecting the **Playback** menu item. You must close all other connections before connecting to **Playback**.



Playback utilizes the ability to download or record market data and replay it at another time. It is the same idea as recording your favorite TV show during the day and watching it at some other more convenient time. Unlike most products that only allow you to replay one market at a time, NinjaTrader provides synchronous replay of any and all recorded markets and delivers this market data to all NinjaTrader windows as if it was happening in real-time. Therefore, you can have multiple SuperDOMs and charts replaying different markets all at the same time. You can trade in simulation against this data at varying levels of replay speed.

- Set Up
- Playback
- Data Files

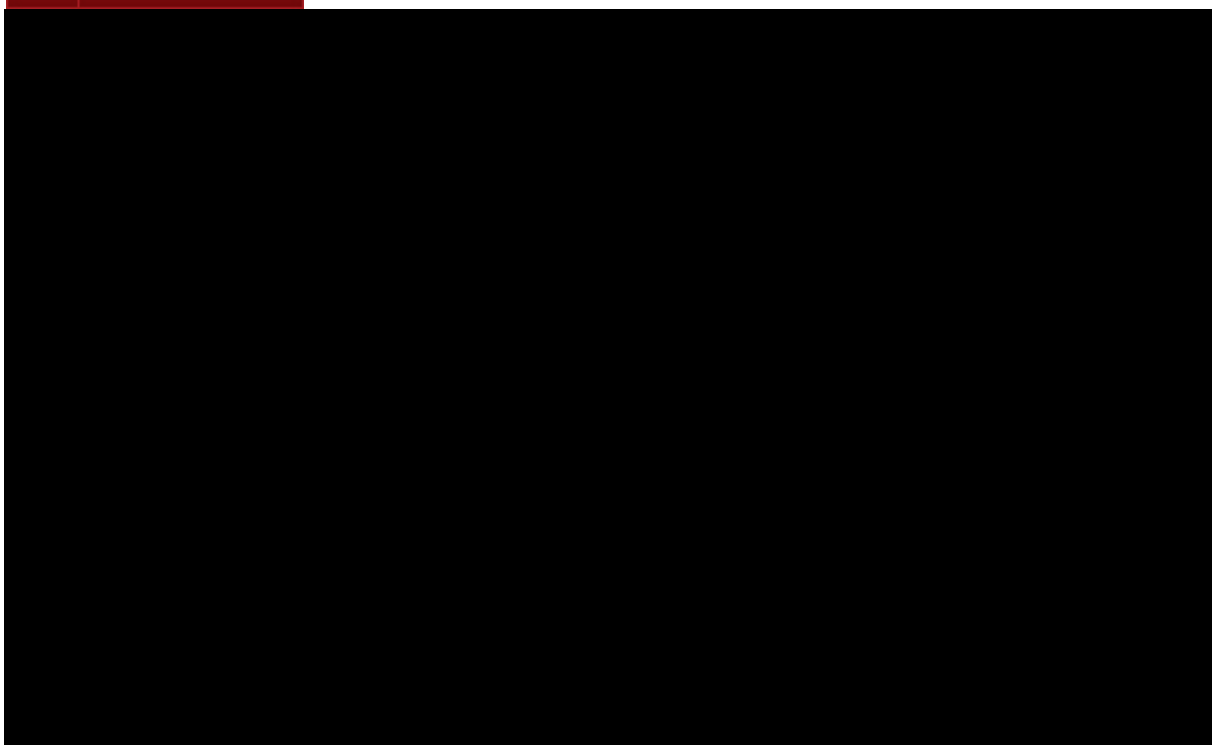
10.24.1 Set Up

You can **Playback** either **Market Replay data** or **historical tick data**. For the most accurate reflection of live market conditions you would want to use **Market Replay data**.

Notes:

1. **Market Replay data** holds the exact sequence level I and Level II (market depth) data and must be recorded or downloaded by NinjaTrader.
2. Alternatively, **historical tick data** can be used to playback chart data (without market depth). The granularity and accuracy of **historical** mode will be dependent on your data provider.
3. The Playback101 account properties (e.g., Commissions, Risk, etc.) are created from the Sim101 account when connecting to the Playback Connection and cannot be reset while connected. The Playback101 account will reset with current Sim101 account properties when reconnecting.
4. TimeInForce logic such as order expiration / cancellation at the end of a session, is not supported for the Sim101 or Playback101 accounts.
5. It is recommended to close and save your live workspaces then use dedicated workspaces for Playback before connecting to Playback to ensure items, such as drawing objects, are not affected in your live workspaces.

Play Video



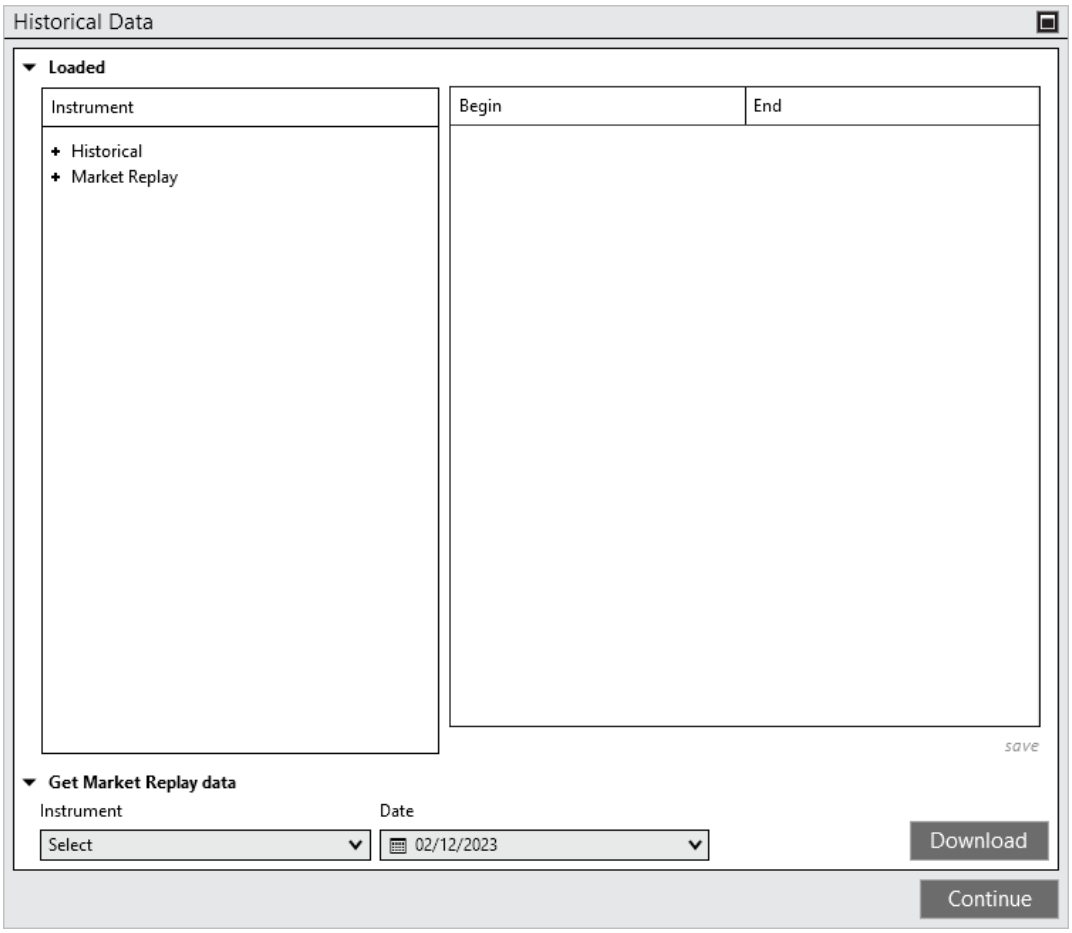
How to download playback data from the NinjaTrader server

Downloading Market Replay data for the Playback connection

Market Replay data holds the exact sequence level I and Level II (market depth) data. NinjaTrader offers a limited amount of **Market Replay data** free to download for playback purposes. Only the most common instruments are currently available.

Notes:
1. **Enable market recording for playback** must be disabled from the **Market Data** category of the **Options** menu before downloading replay data.
2. Downloading **Market Replay data** is **NOT** available when connected to the Playback connection. You must disconnect from **Playback connection** prior to downloading.

To download playback data:
1. Open the **Historical Data** window. The **Historical Data** window will open when initially connection to Playback or by going to the Control Center and selecting **Tools** then **Historical Data**. Here the section "**Get Market Replay data**" can be expanded.
2. Select the instrument and date of the desired replay data and press the OK button to begin the download.



The status of the download will appear in the lower right hand corner of the **Historical Data Window**.

> **Note:** Closing the Historical Data Window will cancel the download.

Downloading historical tick data for the Playback connection

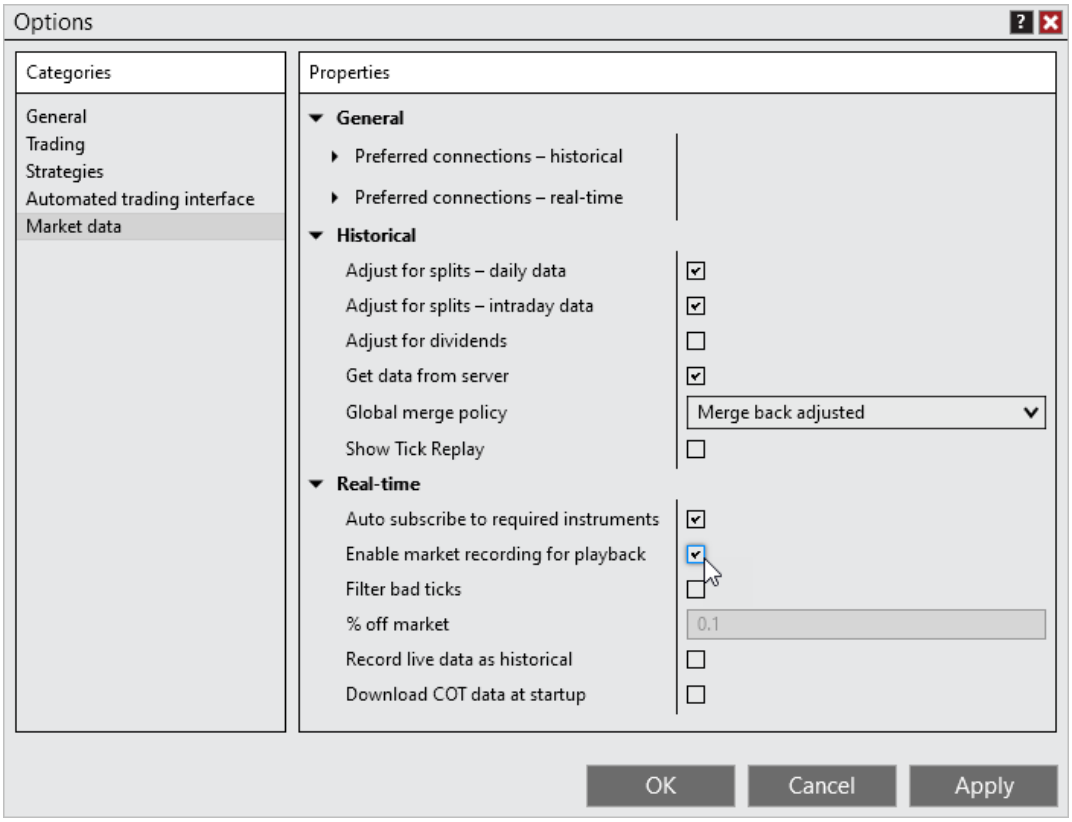
If **Market Replay data** is not available, or you do not need the accuracy that **Market Replay data** provides, you can optionally use playback using **historical tick data** offered from your data provider. You can download, export, import **historical tick data** via the Historical Data Window.

How to enable the market replay recorder

Enabling the Market Replay Recorder

To enable the replay recorder:

1. Left mouse click on the **Tools** menu and select the menu item **Options**.
2. In the **Market Data** category enable the option "**Enable market recording for playback**".



All live data from instruments that are active in any NinjaTrader window will now be recorded for playback. (see the "How to record live market data" section below)

How to record live market data

Recording Data

Once **Enable market recording for playback** is enabled (see the "*How to enable the market replay recorder*" section above), data is recorded for any instrument in any NinjaTrader window that is receiving live market data. Level II (market depth) data is only recorded if a Level II, SuperDOM, or FX Pro window is open and receiving data for the instrument. The Market Analyzer window is the recommended recording window as multiple instruments can be added to one Market Analyzer window and all recorded at the same time.

10.24.2 Playback

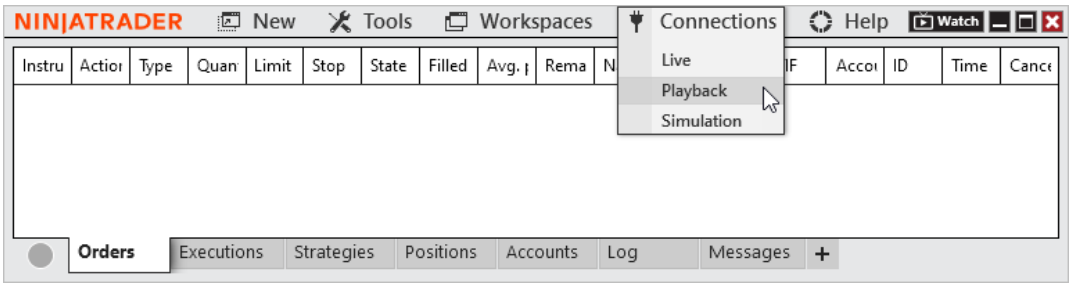
Once market replay data or historical tick data is available by either recording or downloading (See the "*Set Up*" page of the Help Guide), it can be replayed in all NinjaTrader windows.

How to connect to Market Replay data

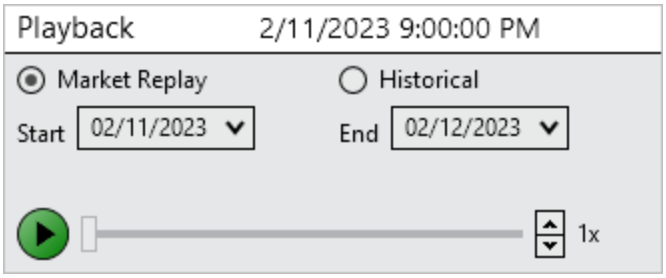
Connecting to Replay Data

To connect to Market Replay data:

1. Left mouse click on the **Connections** menu in the Control Center
2. Select the menu item Playback **Connection** menu item



The **Playback** connection should now be connected and the **Playback** Control should be visible.



Note: When disconnecting, the Playback account's trade history will be reset.

How to work with replay data

Playback Control

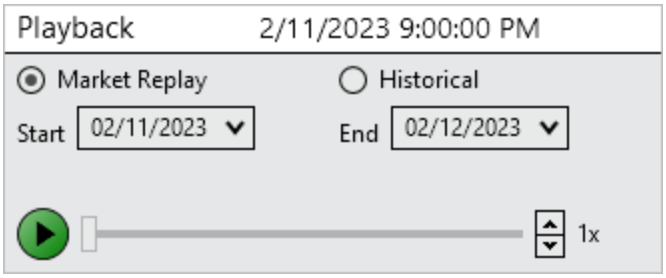
Once connected to the **Playback** connection (see the "*How to connect to Market Replay data*" section above for how to connect), the **Playback** control window will appear.

In the caption bar of the **Playback** control you will see the current date and time of where the play head is located.

Controls

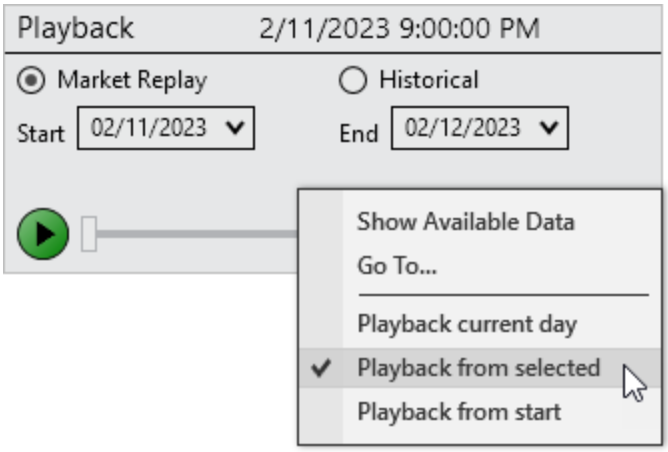
The **Playback** control is set up much like a DVD player. The following controls are available:

Playback Type	Select either "**Market Replay**" or "**Historical**"
Start	Sets the start date for the left side of the slider
End	Sets the end date for the right side of the slider
Play	Starts the market replay.
Slide control	Selects a point in time to start replay (sliding during playback will reset the Replay101 account trade history)
Speed control	Each successive click increases the speed of the playback. Playback of "**Max**" will process data at fastest possible speed.



Right Click Menu

Right mouse clicking in the Replay control window will bring up the right click menu with the two following menu items:



Show Available Data...	Brings up the Historical Data Window window. Instruments with replay data will be displayed with the level 1 (L1) and level 2 (L2) Begin and End dates and times.
Go To...	Brings up the Go To window where you can specify a date and time to jump the replay file to. There must be recorded data available for the selected time.
Playback Current Day	Only the current day will be played back from Market Replay when dragging the slider between multiple days, for past days historical data would be loaded.
Playback from selected	All historical data is loaded from historical data and Market Replay data is only used going forward. This is the fastest and default mode. This mode is sufficient when there's no NinjaScripts that need the sequence of events for each historical tick.
Playback from start	Market Replay data is played back for every day between the start point of the slider and the end point of the slider. This will be slower as NinjaTrader must process more data, but is useful when you are back testing a strategy in playback.

Tip: Should you be using the **Playback** for testing a NinjaScript strategy, please be sure the chart you apply the strategy onto has bars populating it prior to the start time of your replay.

Understanding how the Playback works

Playback supports running on Market Replay data or Historical data. Market Replay data is the most accurate and holds both level I and level II (market depth) data. If you do not have market replay data for a time frame, you can choose to playback historical tick data. However using historical tick is less accurate as there is no level II data.

Market Replay Data

NinjaTrader stores level I and level II together in a single market replay file to ensure that level I and level II events are perfectly in sync per instrument.

Market replay files have the ability to record time stamps down the 100 nanosecond level. However please note that we use the time stamp provided by the market data providers when storing data. This means that you are limited to the granularity of the provider if the time stamp is natively provided. Please see the Historical & Real-Time Data section of the help guide for more information.

Note: When using market replay, the NinjaTrader core market data updates occur at the granularity provided by the market data provider. However, the NinjaTrader user interface only visually updates in 1-second intervals for performance optimizations. Even though the NinjaTrader UI's are only visually updating at 1-second intervals, orders, indicators, and strategies will calculate just as they were running in real-time.

Historical Data

When using Historical data for playback NinjaTrader will use historical tick data for playback. If the tick data from your provider is stamped with ask and bid data then NinjaTrader will use that to simulate the ask and bid price during playback. If your historical data provider does not support ask/bid stamped tick data then NinjaTrader will simulate the ask and bid price by setting it either to last price or last price +/- 1 tick at random.

Note: Ask and Bid Volume during playback with Market Replay or historical data will be simulated and set to "1" except for Equities and Forex, where "100" is used for Equities and "100,000" for Forex.

Order Processing Differences in Playback

When submitting orders to the Playback101 account, these orders are processed immediately and synchronously. This enables reproducible results for strategy developers that run a strategy on the playback connection. Playback101 and Sim101 work differently when executing, since simulated internet latency delay simulation is not present in playback.

10.24.3 Data Files

Playback can use two types of data which is selected by the user via the Playback controller.

Market Replay

Market Replay data is recorded and stored in compressed files located in the Documents\NinjaTrader 8\db\replay directory. These files can be shared by copying the contents of this folder to another NinjaTrader installation, or by using the Backup & Restore utilities to create a backup file of the replay data and restoring this to another PC.

Historical

For **Historical** playback NinjaTrader uses historical **Tick** data. You can export and import Tick data via the Historical Data Window. Please see the exporting and importing market data sections of the help guide for more information.

10.25 Risk

Risk Overview

The **Risk** window allows you to define margin control and limits to be used on simulation accounts.

› Using the Risk window

10.25.1 Using the Risk window

Within the **Risk** window, **Risk** Templates hold the risk definitions for local simulation accounts. A risk definition holds the amount of margin needed per contract, it also limits

the amount of contracts allowed to trade. To define your Live or Simulation NinjaTrader risk, you can go to the **Client Dashboard** under **Tools**.

Understanding Risk Templates

Risk Templates

A **Risk** Template is a collection of risk definitions that can be used by Simulation accounts to track the amount of Margin being used.

Where Risk Templates can be Applied

Risk Templates can be applied via the **Control Center** or **Account Data** Accounts tab.

- Right click on a Simulation Account
- Select "Edit Account"
- Change the selected "Risk" template parameter

> **Note**: Risk definitions are activated as soon as the template is applied. However, there are some values (i.e., Intraday margin, Initial margin, Maintenance margin) which are calculated only as the position is updated. Should you be in a position when the risk template is applied, these values would **NOT** calculate until the position is updated. You can force this calculation by disconnecting and reconnecting to your data account connection.

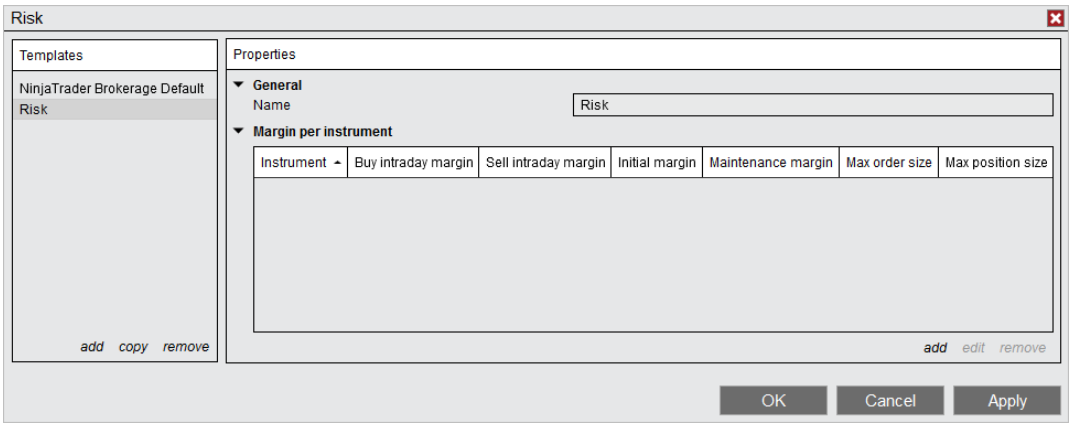
How to create and edit a Risk Template

Creating a Risk Template

If your desired session risk settings are not found within the pre-loaded **Risk Templates**, you can create a new template.

To create a **Risk Template**:

1. Left mouse click on "add"
2. Type in the name of the **Risk Template**
3. Select "add" to add a new risk definition, see "**Understanding risk definitions**" below for more information. Repeat for as many risk definitions as required.
4. Press the Apply button to save the configured session times in the **Risk Template**.



Working with Risk Templates

A saved **Risk Template** can be selected via the Template section to the left of the **Risk** window. Selecting the template will allow you to configure individual risk definitions for that template.

Editing Risk Templates

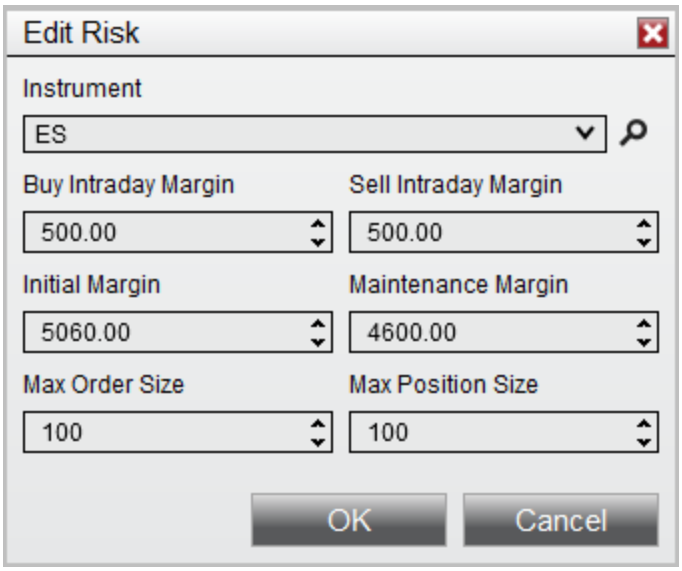
Risk Templates can be edited in the following ways:

- Left mouse click the "copy" button in the templates section and insert a new template name to copy the current **Risk Template.**
- Left mouse click the "remove" button in the templates section to delete the selected **Risk Template.**

Understanding risk definitions

Understanding Risk Definitions

Each risk definition applies to an individual instrument. You can only have one instrument definition per instrument.



Buy Intraday Margin	Sets the intraday margin required for buy orders.
Sell Intraday Margin	Sets the intraday margin required for sell orders.
Initial Margin	Sets the initial margin required.
Maintenance Margin	Sets the maintenance margin required.
Max Order Size	Sets the max allowable order size.
Max Position Size	Sets the max allowable position size.

10.26 Simulator

Simulator Overview

The **Simulator** can be accessed by selecting the Sim101 account in any of the NinjaTrader order entry features.
NinjaTrader provides a state of the art internal simulation engine that can be used to test trading ideas and hone your skills. The simulation engine is not a simple algorithm that fills your order once the market trades at your order price. The engine uses a scientific approach to determine fill probability by including a number of variables including: ask/bid volume, trade volume, time (to simulate order queue position), and random time delays for switching between order states.

Simulation Accounts

- The Sim101 Account
- Multiple Simulation Accounts

Paper Trading

- Live/Simulation Environment
- Global Simulation mode
- Trading in Simulation

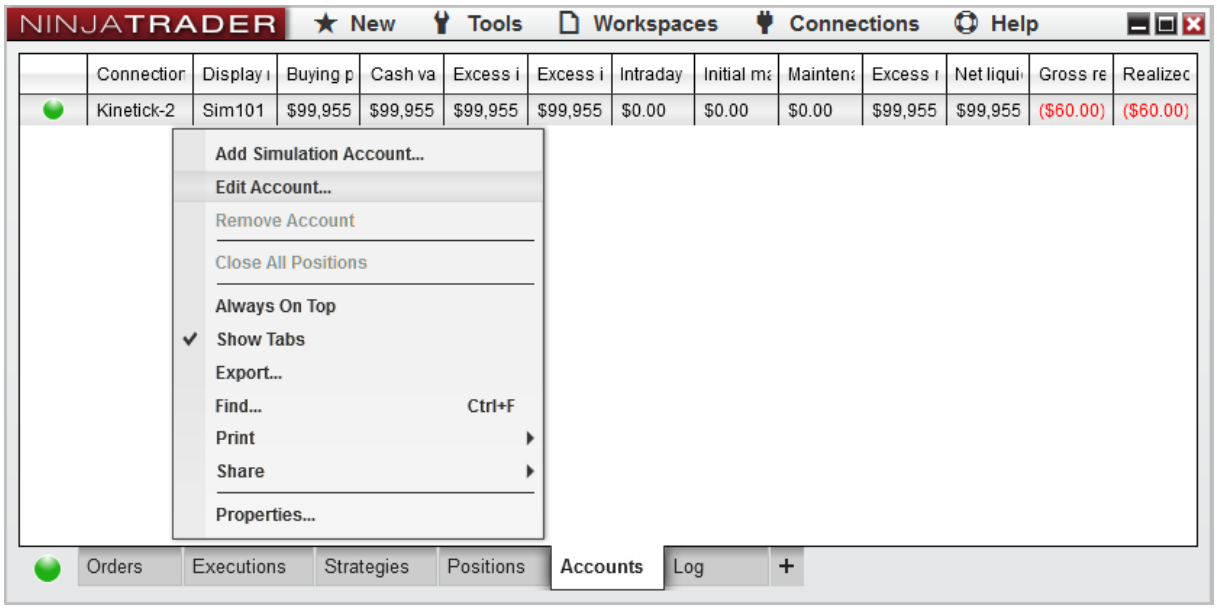
10.26.1 The Sim101 Account

What is the Sim101 account?

The Sim101 account is a default account that represents your own simulated account through which you place simulated trades. The Sim101 account behaves identical to a live account in that it has a cash balance, profit and loss and other financial parameters. For example, when placing orders to the simulator, the Sim101 account is checked to ensure that you are not exceeding your buying power.

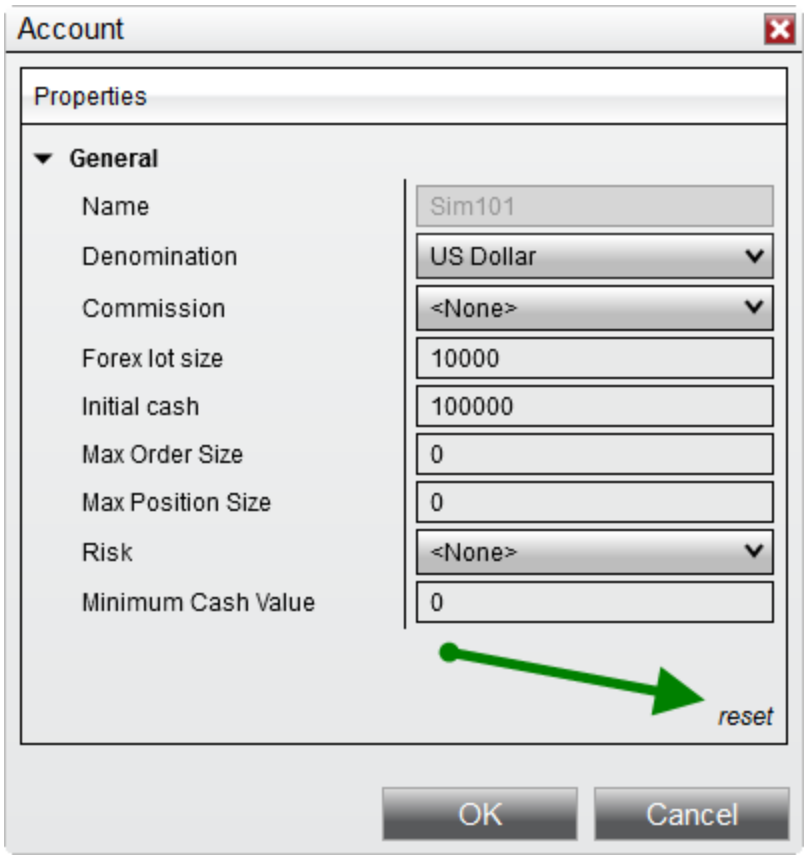
How to customize the Sim101 account

You can set initial Sim101 account values, reset simulator values, and clear order history. To access these settings open the Control Center window select the "**Accounts**" tab. If the account tab is not visible select the "+" tab button and select 'New accounts'



Resetting Initial Cash value on the Sim101 account

To reset the initial cash value on your account please edit the account as shown above.



In the Simulation Accounts window:

1. Set the "**Initial cash**" property to what you want to reset the account to.
2. Click "reset"

> **Note:** Simulation **Denomination** currently only supports US Dollar.

10.26.2 Multiple Simulation Accounts

You can create an unlimited number of simulation accounts in NinjaTrader.

Steps to Create Multiple Simulation Accounts

1. Open the NinjaTrader **Control Center**
2. Select the **Accounts** tab
3. Right click on the accounts tab and select **New simulation account...**
4. Configure your new account and click the **OK**

> **Note**: The account will be active the next time you connect to a data provider.

10.26.3 Live/Simulation Environment

NinjaTrader is a true mixed live/simulation platform. You can have multiple entry windows open and using the account drop down menu, simultaneously route orders to your live broker in one window while routing orders to the simulator in another. This provides you the flexibility to trade live while testing different methods or ideas in simulation.

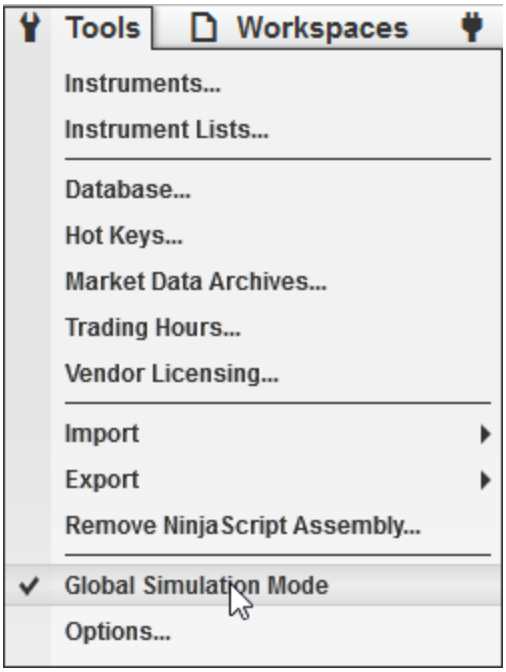
10.26.4 Global Simulation Mode

What is Global Simulation Mode?

When Global Simulation Mode is enabled, all order entry interfaces (SuperDOM, Chart Trader, etc.) will only allow selection of a **simulation** account . Enabling this is not necessary in order to route orders through simulation, because you can still set any order entry interface to the Sim101 account individually. However, Global Simulation Mode provides you a method to ensure that you do not accidentally place an order to your live trading account.

How to enable or disable Global Simulation mode

1. From the NinjaTrader Control Center select the **Tools** menu.
2. Then select the menu item **Global Simulation Mode**
3. When the check mark appears next to the menu item **Global Simulation Mode** it is active, and when the check mark is not showing Global Simulation Mode is disabled. Free license users are not able to disable Global Simulation Mode.
4. In addition, you can set NinjaTrader to always start in simulation mode via the Simulator Tab in the Options window.



10.26.5 Trading in Simulation

NinjaTrader routes orders based on the account that you select in any of the order entry interfaces (SuperDOM, Chart Trader, etc.). Simulation is no different. You can select the Sim101 account from any of the NinjaTrader order entry interfaces to submit your orders in simulation. Optionally you can set NinjaTrader to change the background color of the trading interface when a simulation account is selected, this is set via the "Simulation color" property in the NinjaTrader Trading Options window. Its default setting is "**Transparent**" which means it is disabled.

Notes:
1. Trading in simulation should be done only when you have data within market hours streaming. Simulation outside of market hours can result in fill prices which are seemingly far off the last traded price based on the erratic bid/ask prices commonly seen during these hours.
2. The simulator account(s) shift(s) to the next trading day at 4:15 EST every day, should NinjaTrader be running at this time it will occur on the next start-up.
3. When the simulator account is reset, the realized PnL is reset back to 0 and the CashValue has the commissions deducted from it and set back to 0.
4. Commissions for the simulation account(s) are continuously totaled and the *Total commissions* Account Statistic will reflect that throughout the session.
5. TimeInForce logic such as order expiration / cancellation at the end of a session, is not supported for the Sim101 account.

10.27 Strategy Analyzer

Strategy Analyzer Overview

The Strategy Analyzer can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center, and selecting the menu item **Strategy Analyzer**.

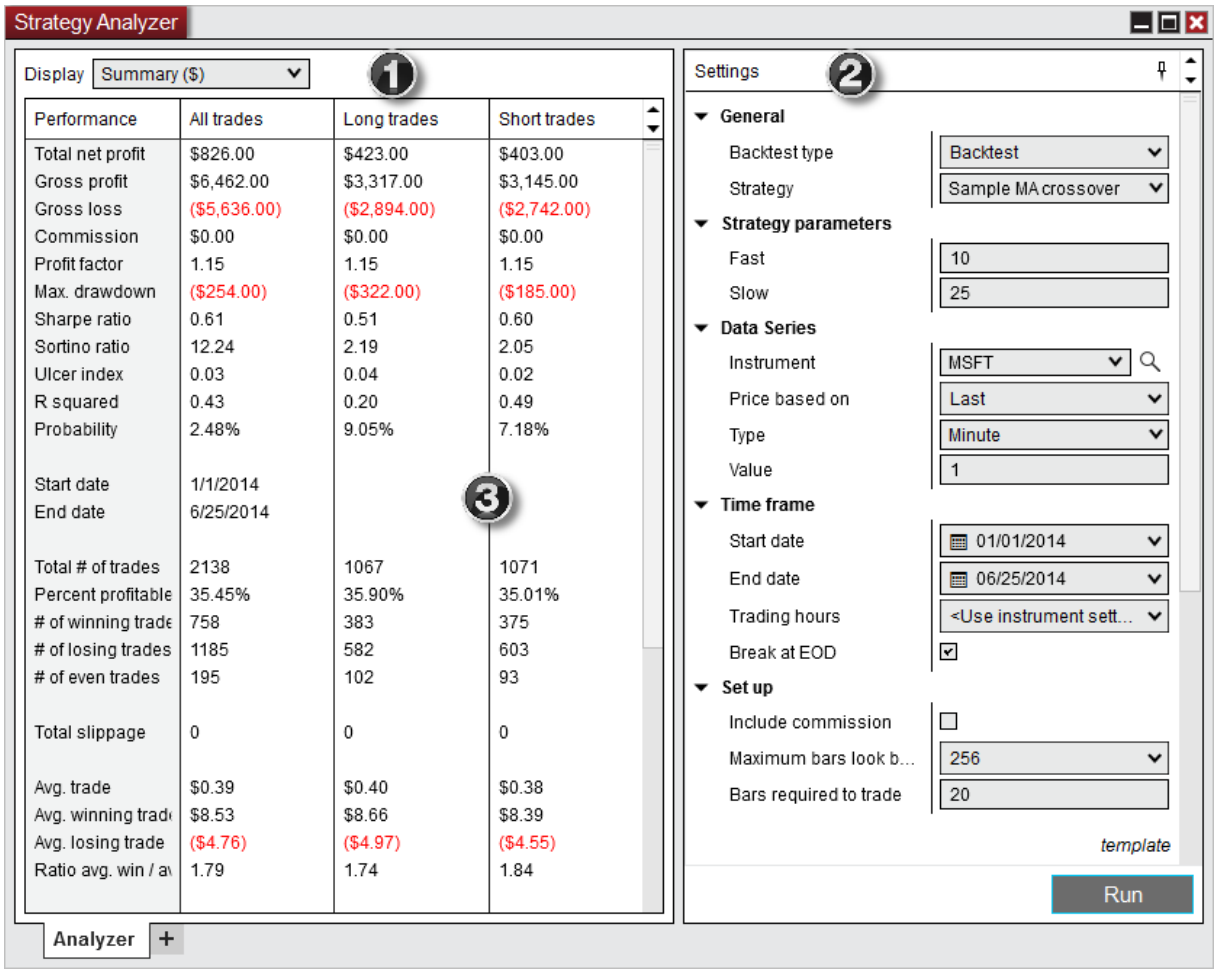
The Strategy Analyzer allows you to run historical analysis on your NinjaScript based automated trading strategies

10.27.1 Understanding the Layout

Layout

The **Strategy Analyzer** window contains the following items:

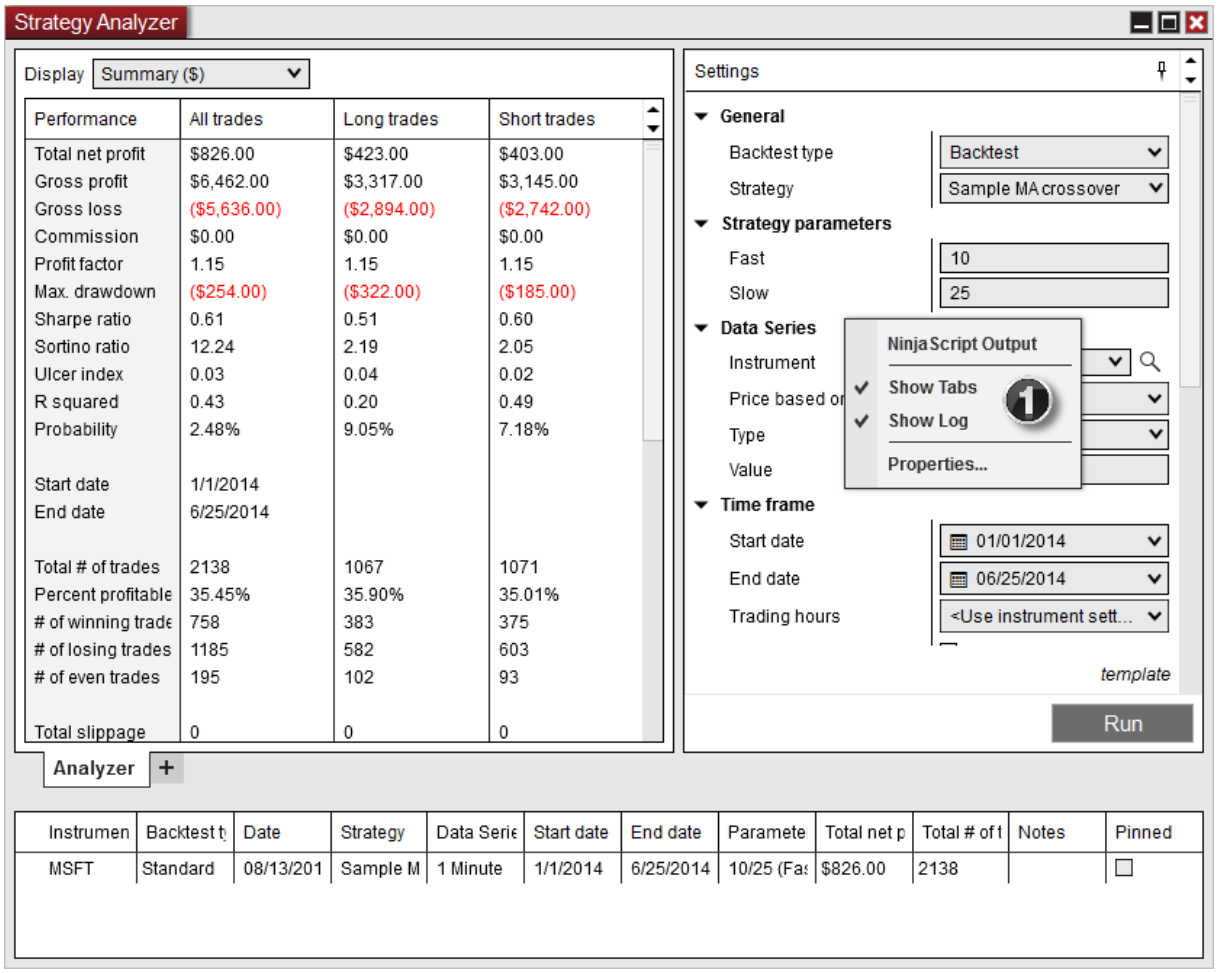
1. The **Display Selector** sets what performance results to view and the format to view the results in.
2. The **Settings** panel sets the parameters to be used for the strategy backtest.
3. Where Performance results are displayed based on the display selection.



Log Grid

You can toggle the log to be displayed, this shows summary details from all previous strategy backtests.

1. To show the log right click on the **Strategy Analyzer** and select "**Show Log**".
2. The Log will be made visible as shown below.



10.27.2 Backtest a Strategy

A backtest allows you to analyze the historical performance of a strategy. In order to run a backtest you will need:

- Access to historical data
- Custom NinjaScript *strategy

> **Tip**: There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore.

Play Video

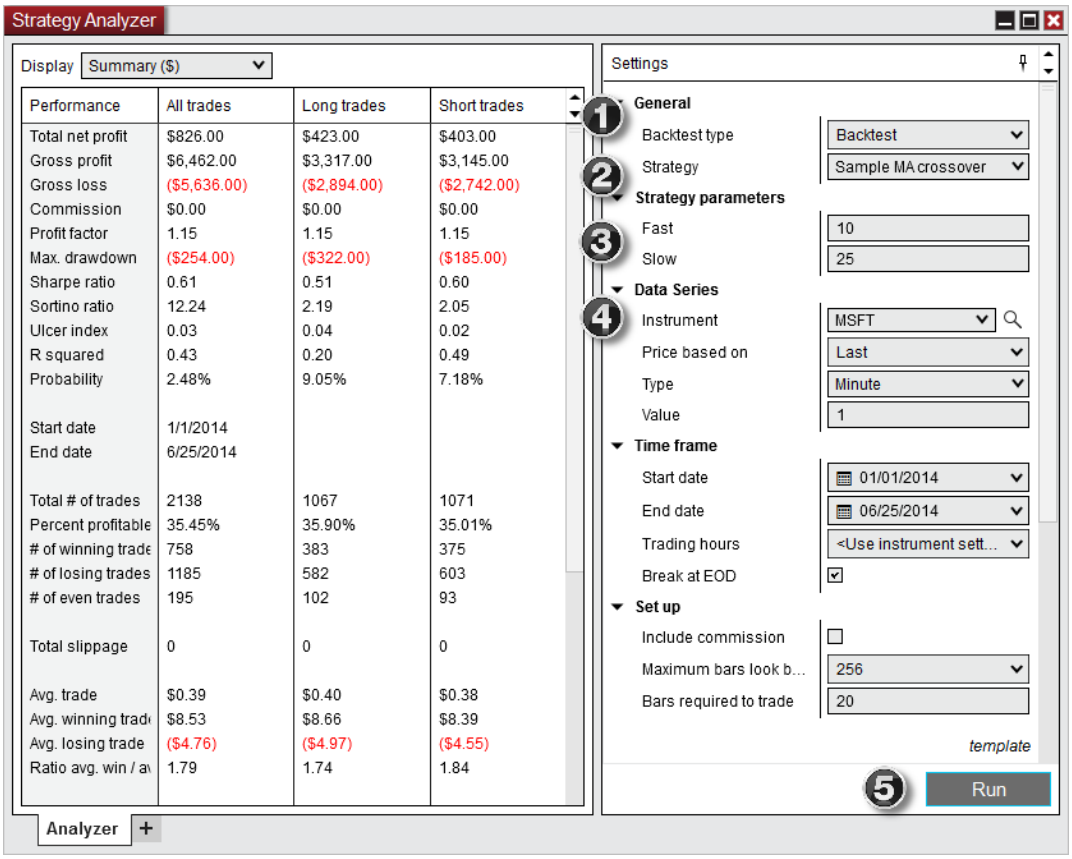
Notes:

1. By default, the **Strategy Analyzer** downloads data from your market data provider which can slow down backtest progress for larger tests. If you wish to disable this feature and operate using existing data in your database, right click on the **Strategy Analyzer** > select **Properties** > enable **Use Local Data Only**
2. The IncludeTradeHistoryInBacktest property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for backtesting. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as SystemPerformance.AllTrades.Count that rely on references to **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state. For more information, see the Working with Historical Trade Data page.
3. A certain level of discrepancy between realtime and backtest results would be expected, especially on more exotic barstypes like Point & Figure and Renko, please also review this page for more details.

How to run a backtest

Start a Backtest

To run a **Backtest** of a strategy:



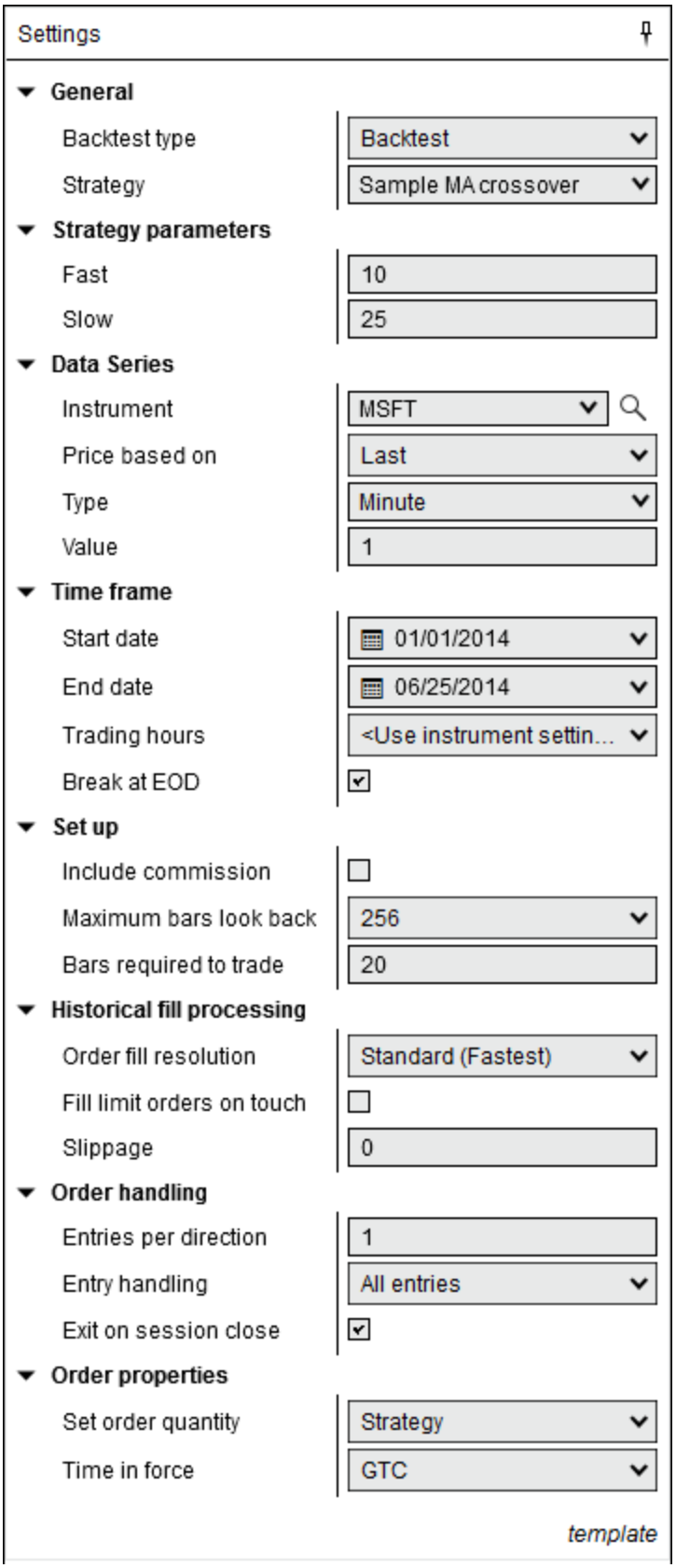
1. Select the **Backtest type** of "**Backtest**"
2. Select the strategy you would like to backtest
3. Set the strategy and backtest parameters (See the "*Understanding backtest properties*" section below for property definitions)
4. Select the instrument and Data Series you would like to backtest
5. Select the "**Run**" button to start the backtest

Tip: You can optionally configure a sound to be played when the Backtest completes. To enable this option, right click on the **Strategy Analyzer** > **Properties** > **Play sound on complete** > Choose the sound file you wish to play (must be a .WAV)

Understanding backtest properties

Backtest Properties

The following properties are available within the Backtest window:



General	
Backtest type	Sets the backtest type. 1. Backtest 2. Optimization 3. Walk Forward Optimization 4. Multi-Objective Optimization
Strategy	Sets the strategy you would like to test.
Strategy parameters	
Parameters (...)	Each strategy parameter is listed dynamically depending on the strategy selection
Data Series	
Instrument	Sets the instrument or list you wish to test on
Price based on	Sets the type of market data used to drive the Data Series
Type	Sets the bar type of the Data Series.
Value	Sets the Data Series value.
Time frame	
Start date	Sets the start date for the test period

End date	Sets the end date for the test period
Trading hours	Sets the trading hour template for the Data Series. (See the "Trading Hours" section of the Help Guide for more information)
Break at EOD	Enables or disables the bars being reset at EOD (End Of Day). (See the "Break at EOD" section of the Help Guide for more information)
Set up	
Include commission	Enables or disables commissions in the backtest performance results (See the "Commission Tab" section of the Help Guide for more information)
Maximum bars look back	Max number of bars used for calculating an indicator's value. The "TwoHundredFiftySix" setting is the most memory friendly.
Bars required to trade	Sets the minimum number of bars required before orders will be allowed to be submitted
Historical fill processing	
Order fill resolution	Sets the order fill resolution to be used for the backtest. (See the "Understanding Historical Fill Processing" section of the Help Guide for more information)
Fill limit orders on touch	Enables or disables the filling of limit orders on a single touch of price action.

Slippage	Set the amount of slippage in ticks to apply to market / stop market / Market-if-touched order executions (default is 0) Note: In the **Summary** the **Slippage** is displayed in points.
Order handling	
Entries per direction	Sets the maximum number of entries allowed per direction while a position is active based on the "Entry handling" property
Entry handling	Sets the manner in how entry orders are handled. If set to "AllEntries", the strategy will process all entry orders until the maximum allowable entries set by the "Entries per direction" property has been reached while in an open position. If set to "UniqueEntries", strategy will process entry orders until the maximum allowable entries set by the "Entries per direction" property per each uniquely named entry.
Exit on session close	When enabled, open positions are closed on the last bar of a session
Order properties	
Set order quantity	Sets how the order size is determined, options are: "by default quantity" - User defined order size "by strategy" - Takes the order size specified programmatically within the strategy
Time in force	Sets the order's time in force

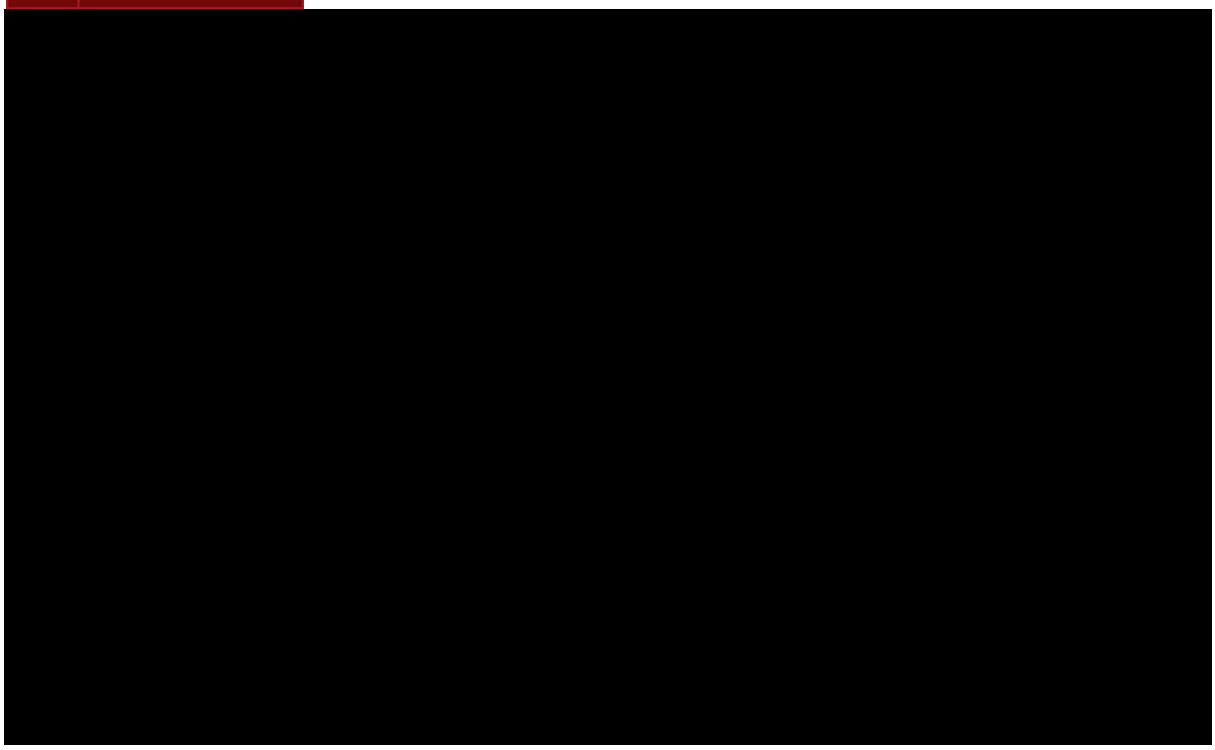
10.27.3 Optimization

You can fine tune the input parameters of a strategy through optimization. Optimization is the process of testing a range of values through iterative backtests to determine the optimal input values over the historical test period based on your optimization fitness. To run an optimization you will need:

- Access to historical data
- Custom NinjaScript *strategy
- A thorough understanding of the Strategy Analyzer's backtesting capabilities

Tip: There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore.

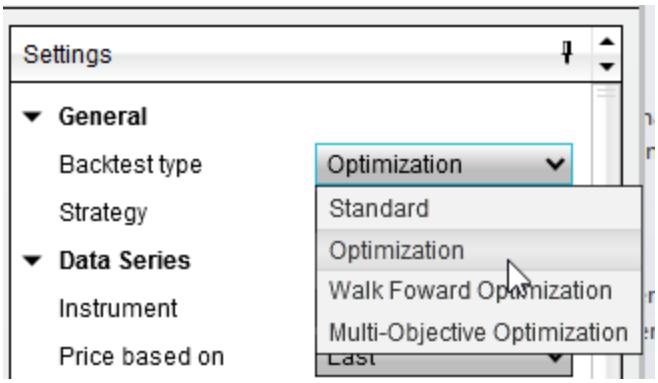
Play Video



How to run an Optimization

Running an Optimization

To run an **Optimization** select the **Backtest type** of "**Optimization"** in the settings panel of the **Strategy Analyzer**.

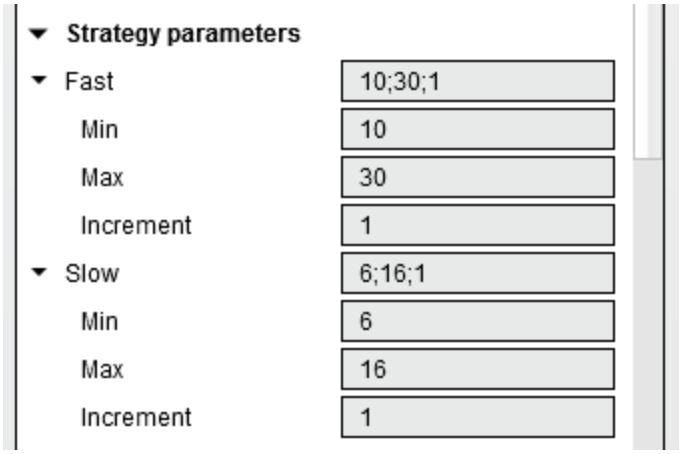


Note: When making the selection additional parameters to configure your optimization will be made visible.

Setting the Test Range

You can the test range of strategy parameters to be tested by left clicking on the triangle to expand the strategies sub parameters.

Note: If you don't see the triangle make sure that the **Backtest type** is set to "**Optimization**".



Min. - The starting value you want to test
Max. - The last value to test
Increment - The increment value (step value) used to increment the starting value by for each subsequent optimization pass

In the image above, the input "Fast" has a starting (initial) value of 10 and an ending value of 30 with an increment of 1. This means that the first value tested

will be 10, then 11, then 12 all the way through 30. The input "Slow" has a starting value of 6, ending value of 16 with an increment of 1. Based on these settings, a total of 200 (20 unique values for "Fast" multiplied by 10 unique values for "Slow") backtest iterations will be processed in order to find the optimal combination of input values based on the best optimization fitness.

Setting the Optimization Fitness

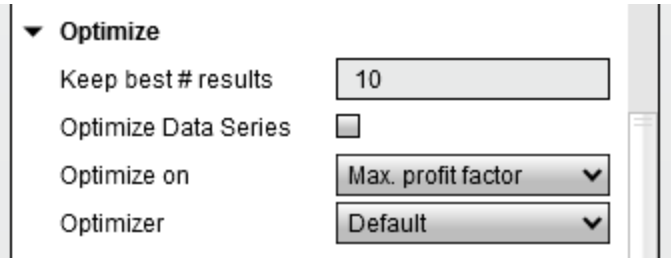
Optimization is based on the best optimization fitness you select. If you set the property "Optimize on..." to "Max. net profit", the optimizer will seek the optimal input values that return the maximum profit possible. There are over 10 different optimization criterion you can select and can be customized via NinjaScript. Please see the "*Understanding Optimization properties*" section below for more information.

Understanding optimization properties

Optimization Properties

Apart from the optimization specific properties described below, the properties are identical to the ones found in the backtest properties window. Please see the "*Understanding backtest properties*" section of the Backtest a Strategy page of the Help Guide for more information.

The following Optimization specific properties are available:



Tip: You can optionally "**Optimize on**" multiple objectives by using a Multi-Objective optimization

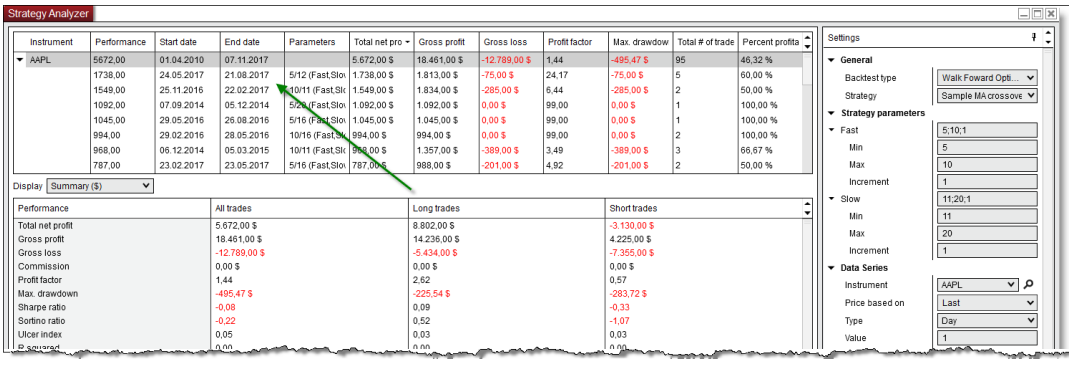
Keep best # results	Sets the number of best results to display

Optimize data series	If set to true, the Data Series Value property will be available for optimization (Not supported for Kagi, Point and Figure, Line Break and Heiken Ashi period Types)
Optimize on...	Sets the optimization fitness to base the optimization results on
Optimizer	Sets the optimization algorithm that is used. NinjaTrader comes with "Default" and "Genetic" optimizer algorithms. When the "Genetic" option is selected, the genetic algorithm's optimization properties fields will appear below the Optimizer selection You can program your own optimization algorithm using NinjaScript.

Understanding optimization results

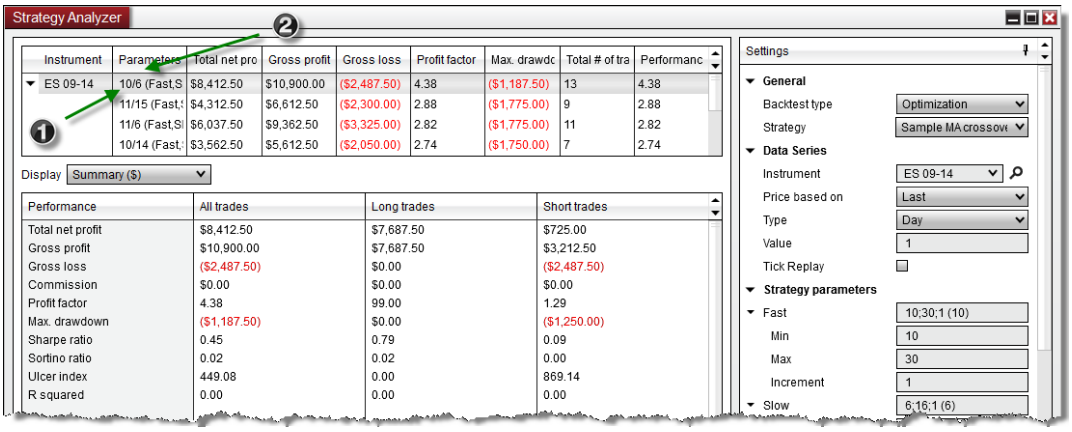
Understanding Optimization Results

Once the optimization process is complete, you will see a the Optimization Results Grid appear in the Analyzer tab. The results will be grouped per instrument and shows the parameter combination that achieved the highest performance. The "Performance" column is dynamic and will always be the Optimization Fitness that you selected for the "**Optimize**" parameter when you ran the optimization.



The Top Optimization Results

The Optimizer tab will display the top number of results based on the value you set for the "Keep best # results" property in the Optimizer dialog window. The column Parameters displays the optimized input values.

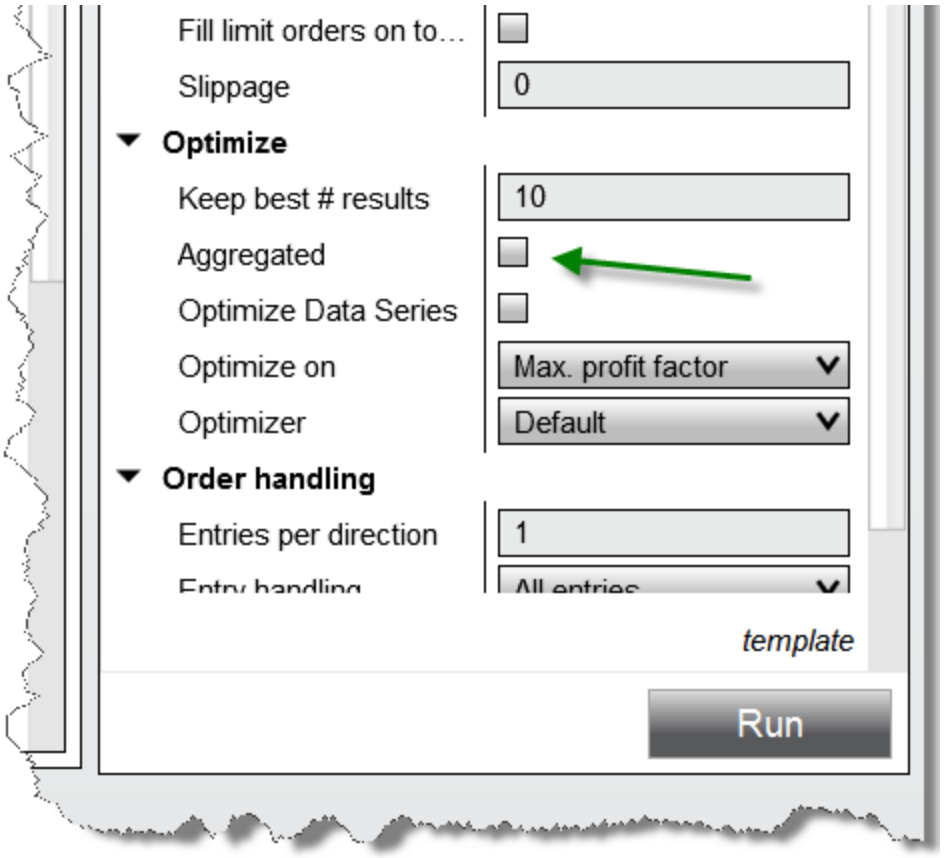


1. The optimal value for the "Fast" input for the demonstration strategy used for this optimization
2. The optimal value for the "Slow" input for the demonstration strategy used for this optimization

Running a basket test

Basket test

Running an optimization across a list of instruments works very much the same as running a regular basket backtest. For general information, please refer to the Basket testing multiple instruments page. However, when running an optimization across multiple instruments, an optional "Aggregated" option will be available.



Aggregated	If set to True, NinjaTrader attempts to find the optimal results for the whole basket of instruments. The COMBINED row in the results tab will show an aggregation of results across the basket of instruments. (This parameter is only available when an Instrument List is selected for optimization.)

Understanding factors that affect optimization performance

32 bit vs 64 bit

When you run an optimization in the **32 bit** version of NinjaTrader to consume less memory we do not store any trade data for each backtest that is run. Therefore if you want to do trade analysis on one of the backtest results returned from an optimization NinjaTrader must re-run the backtest to get the trade data, this adds a small delay when switching between tests. The **64 bit** version of NinjaTrader will

take advantage of the extra RAM available to NinjaTrader and will keep the trade results for each kept backtest, allowing you to quickly change between backtest result reports.

Keep best # results
If you are finding that you are running low on system memory during your backtests reduce this number of results to keep will make a significant improvement to the memory used by NinjaTrader.

Running multiple tests at a time

You will not get more done in a smaller time frame by separating multiple tests out manually and running them at the same time on the same PC. NinjaTrader will efficiently use all CPU cores for any optimization for fastest possible testing.

CPU Resources

Please insure that you have as much system resources available to the optimization as possible, this usually means making sure all other applications are closed. Furthermore as as the NinjaTrader optimization engine is optimized to take advantage of as much system resources as possible it is advisible not to trigger an optimization during a time where you would need to be using the PC. For example it is not advised to start an optimization while you are managing the exit of a trade.

Historical Trade Data

The IncludeTradeHistoryInBacktest property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for optimization. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as SystemPerformance.AllTrades.Count that rely on references to **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state, but this can result in greater memory usage. For more information, see the Working with Historical Trade Data page.

Running Efficient Optimizations

Strategy optimizations are expected to consume a good deal of CPU resources, simply due to the nature of the iterative data processing they perform. Strategies with a relatively large number of parameters for optimization can multiply this impact. When working with strategies with a large number of parameters, avoid using "1" as the increment value for the optimizer, to avoid forcing the optimizer algorithm to run the maximum number of permutations. Changing the increment

value to as little as "2" can cut the number of permutations in half, and increasing this value can have progressively less of an impact.

The Genetic Algorithm can offer an alternative solution to increasing parameter increment values. Rather than running brute-force tests by iterating over all permutations, the Genetic Algorithm intentionally ignores parameter combinations which are likely to produce sub-optimal results.

Especially on larger parameter sets with finer increment values, the upper limits of potential permutations / parameter combinations could be reached for both approaches to optimization - the error message *"The strategy needs at least one parameter to optimize"* would be then an indication to rework the # of parameters or increase the increment values to reach a more meaningful permutation count.

Using a virtual / cloud server

If you are using a virtual or cloud server as basis for your setup when running optimization testing in the Strategy Analyzer, please keep in mind that such environments can typically allocate available resources on demand. NinjaTrader will still take advantage of all available threads for it's processing, however those resources available would be determined at the start-up of the NinjaTrader platform. So if your virtual resources would have changed while you were in a working session, then please restart fresh to ensure performance will be optimal.

10.27.3.1 Genetic Algorithm

Very simply put the **Genetic Algorithm** attempts to find the most optimal set of parameters for a strategy. It does this not by brute force testing each individual combination as the default optimization method does, but instead using the concept of evolutionary theory borrowed from biology where only the fittest parents (combined with mutation and crossover) produce children for the next generation. Through testing of multiple generations you should have narrowed down on the most optimal parameters and therefore saving you time from having to test every single parameter combination.

Understanding the Genetic Algorithm

Overview

The general idea of how the GA solves an optimization problem is analogous to the concept of how evolution via natural selection adapts a species to the environment. In biology, only the strongest individuals will be able to reproduce and pass on their superior genes to the next generation. Assuming each generation can only pass on the strongest genes, after several iterations we would be left with the optimal attributes for the environment. Through this same mechanism, the GA will test a random preset of your parameters. Through multiple generations of testing, the parameters will zero in on an optimum solution.

> **Note**: It is important to understand that GA will find approximate optimum solutions. Since it does not test every combination possible there is no guarantee its solutions are absolute optimums.

How the GA calculates

The GA determines its solution through the following steps:

1. Begin with an initial population size consisting of randomly selected individuals (parameter setting combinations)

2. Compute the fitness (Optimize on...) for each individual in the population and assign probabilities to the population based on the fitness results. More fit results have more probability in being selected for breeding of the next generation.
3. Generate a new population for the next generation by selecting individuals from the prior generation to produce offspring via crossover and mutation (see below)
4. Repeat from step 2 till you reach the number of generations in your test

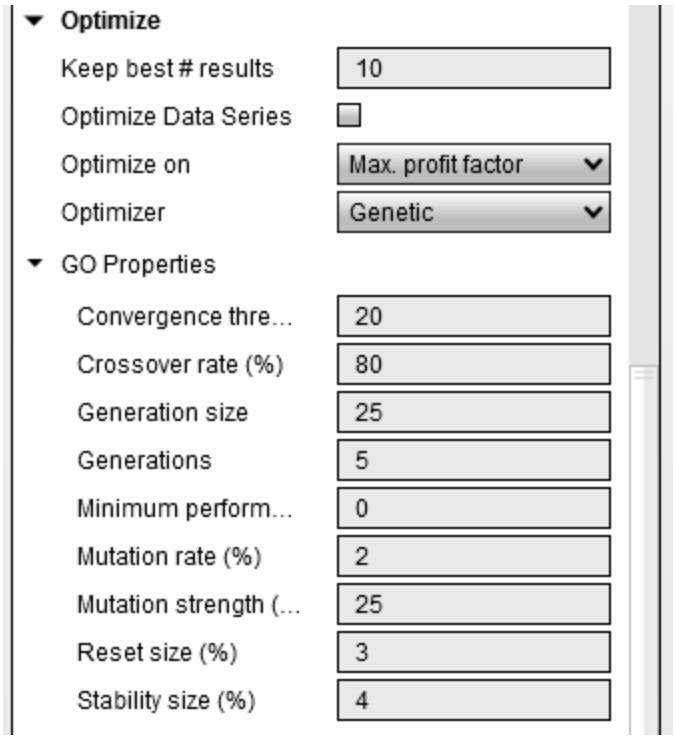
Crossover and Mutation

Crossover is the process in generating offspring that are not 100% identical to their parents. It is done by taking half of the parameter settings from parent A and mixing it with the other half from parent B. Crossover allows GA to test different combinations of parameters and hone in on the optimal solution. Crossover alone however will eventually yield identical offsprings in the population through several generations and so through mutation, some random parameter settings will be interjected in a few of the offsprings to allow for an adaptive quality to the algorithm.

Understanding Genetic Algorithm parameters

Please see the "Optimize a Strategy" article for how to run an optimization.

When you select the Genetic optimizer you will see the following optimization properties after you left click the triangle to the left of "GO Properties" to expand the properties.



Convergence threshold	Setting this will terminate the Genetic Optimization if there is more than a certain number of duplicate children in a single generation, defined by the Convergence Threshold value. This allows the optimization to terminate if no new work is getting done because it has already converged in on the most optimal solution. Example: In the screenshot above Generation size is set to 25, therefore each generation will contain 25 children, if 20 of these children are duplicates that have already been tested then the optimization will be terminated.
Crossover rate (%)	Each new generation is created from a combination of randomly generated offspring and offspring created from combining (crossing over) parent parameters. Crossover Rate determines the percentage of the new generation that is generated from the crossover process.

Generation size	Sets the number of combinations to test in each generation (children). The higher the size, the more variety of combinations that will be tested in each generation. You want to make sure to set this high enough to test enough parameter combinations to get good coverage of the problem domain but not so high that each possible parameter combination is being tested in a single generation.
Generations	Sets the number of generations to test. Each generation will hold the number of children set in "Generation Size". The number of total parameter combinations tested is equal to the Generation Size * Generations.
Minimum performance	If this performance value is reached before all generations are evaluated the optimizer will end and present results immediately, where the type of this value is directly tied to your used optimization fitness metric (i.e. Profit Factor). A Value of 0 means no minimum performance is in use.
Mutation rate (%)	Sets the probability that a crossover offspring will contain some mutated parameters (applies to all parameter types)
Mutation strength (%)	Sets the maximum offset from crossover values that an offspring marked for mutation can have its parameters changed (applies only to input parameters of type `double`)
Reset size (%)	When each new generation is created, all individuals from previous generations are possible parents for the new offsprings. If the top performing x% (stability size %) of children from the newly created generation is the same as the top performing x% of parents, reset all parents and repopulate a new generation randomly while leaving only the top performing y% of parents

	(reset size %) for future generations. Note: This occurs before convergence threashold is tested.
Stability size (%)	See "Reset Size %"

10.27.3.2 Optimization Fitness Metrics

Optimization fitness metrics are used as the targets of optimization tests to determine the optimal mix of strategy parameter values. Below is a list of all pre-loaded optimization fitness metrics and their definitions. Custom optimization fitness metrics can be developed via NinjaScript, as well.

▼ Understanding Max % Profitable

Max % Profitable
This metric represents the percentage of profitable trades compared to the total number of trades placed in an iteration.

```
Number of winning trades / Total number of trades
```

▼ Understanding Max Avg. Favorable Excursion

Max Average Favorable Excursion
This metric represents the average maximum run-up in profit during an iteration.

```
See the "Percent" formula for Average MFE on the Performance Statistics
page.
```

▼ Understanding Max Avg. Profit

Max Avg. Profit
This metric represents the average profit of all trades in an iteration.

```
See the "Percent" formula for Average Trade on the Performance Statistics
page.
```

Understanding Max Net Profit

Max Net Profit

This metric represents the net profit achieved for all trades of an iteration.

```
Total gross profit / Total gross loss
```

Understanding Max Profit Factor

Max Profit Factor

This metric provides a ratio of total earnings to total loss in an iteration.

```
See the Profit Factor formula on the Statistics Definitions page.
```

Understanding Max R Squared (R^2)

R Squared (R^2)

Sometimes called the Coefficient of Determination, this metric measures how closely an iteration's results come to a fitted regression line.

```
((Total Trades * (Summation of Trades * Summation of Profit)) - (Summation
of Trades * Summation of Profit) / SQRT((Total Trades * Total Summation of
Trades ^ 2 - Summation of Trades ^ 2) * (Total Trades * Total Summation of
Profit ^ 2 - Total Profit ^ 2))) ^ 2
```

Understanding Max Sharpe Ratio

Max Sharpe Ratio

This metric calculates risk-adjusted return.

```
(% Profit per month - risk free return) / monthly std. deviation

* if the monthly standard deviation is approximately 0, then set to 1
```

Understanding Max Sortino Ratio

Max Sortino Ratio

This metric modifies the Sharpe ratio by taking the standard deviation of negative returns into account to differentiate harmful volatility from general volatility.

```
(% Profit per month - risk free return) / monthly Ulcer Index

* if the monthly Ulcer index is approximately 0, then set to 1
```

Understanding Max Ulcer Ratio

Max Ulcer Ratio

This metric measures downside risk, with values increasing as the market price moves farther from a recent high.

See the Ulcer Index formula on the Statistics Definitions page.

Understanding Max Win/Loss Ratio

Max Win/Loss Ratio

This metric presents a ratio of the profit of winning trades to the loss of losing trades.

```
% average profit of winning trades / absolute value of % percentage average
loss
```

Understanding Minimum Avg. Adverse Excursion

Minimum Avg. Adverse Excursion

This metric represents the average run-down of trades in an iteration.

```
See the "Percent" formula for Maximum Adverse Excursion on the Statistics
Definitions page
Min Avg. Adverse Excursion finds the lowest value from the Maximum Adverse
Excursion statistic
```

Understanding Min Drawdown

Minimum Drawdown

This fitness metric represents the smallest decrease (draw-down) in account size experienced from the highest high seen in each trade, and is used to find the iteration with the lowest draw-down.

```
See the Maximum Drawdown formula on the Statistics Definitions page
Min Drawdown = the smallest single drawdown
```

Understanding Max Strength

Max Strength (Work in progress, implementation could possibly change in the future)
This fitness metric finds the 'steadiest' strategy represented by the highest linear regression slope of the equity curve. It favors strategies with as many profitable trades as possible while keeping draw-downs as small as possible.

10.27.4 Walk Forward Optimization

Walk Forward optimization is the process by which you optimize strategy input parameters on a historical segment of market data, then test the strategy forward in time on data following the optimization segment using the optimized input values. The central idea is that you evaluate strategy performance data on the test data, not the data used in the optimization. This process is then repeated by moving the optimization and test segments forward in time. To run a walk forward optimization you will need:

- Access to historical data
- Custom NinjaScript *strategy
- A thorough understanding of the Strategy Analyzer's backtesting and optimization capabilities

Tip: There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore.

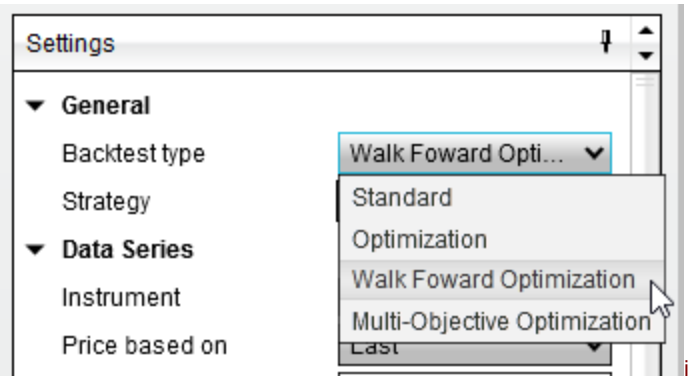
Play Video

Note: The IncludeTradeHistoryInBacktest property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for optimization. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as SystemPerformance.AllTrades.Count that rely on references to **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state. For more information, see the Working with Historical Trade Data page.

How to run a Walk Forward Optimization

Start a Walk Forward Optimization

To run a **Walk Forward Optimization** select the **Backtest type** of "**Walk Forward Optimization"** in the settings panel of the **Strategy Analyzer**.

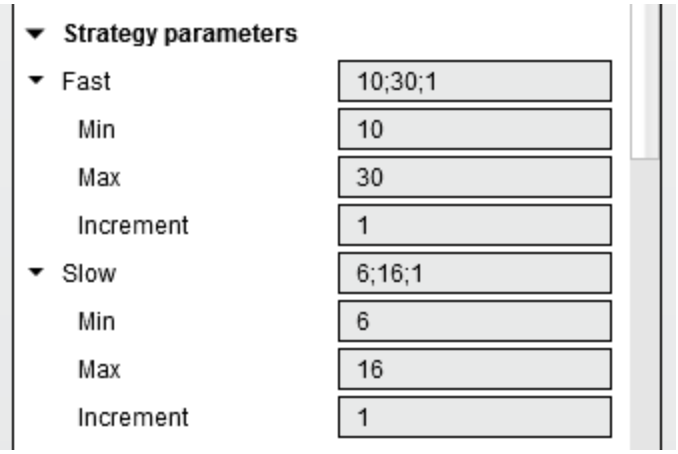


Note: When making the selection additional parameters to configure your optimization will be made visible.

Setting the Test Range

You can the test range of strategy parameters to be tested by left clicking on the triangle to expand the strategies sub parameters.

Note: If you don't see the triangle make sure that the **Backtest type** is set to "**Walk Forward Optimization**".



Min. - The starting value you want to test
Max. - The last value to test
Increment - The increment value (step value) used to increment the starting value by for each subsequent optimization pass

In the image above, the input "Fast" has a starting (initial) value of 10 and an ending value of 30 with an increment of 1. This means that the first value tested

will be 10, then 11, then 12 all the way through 30. The input "Slow" has a starting value of 6, ending value of 16 with an increment of 1. Based on these settings, a total of 200 (20 unique values for "Fast" multiplied by 10 unique values for "Slow") backtest iterations will be processed in order to find the optimal combination of input values based on the best optimization fitness.

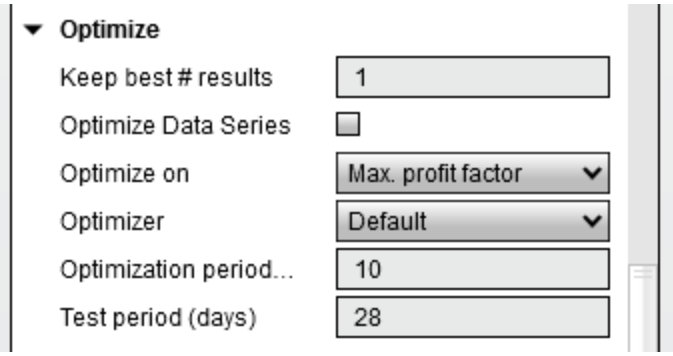
Setting the Optimization Fitness

Optimization is based on the best optimization fitness you select. If you set the property "Optimize on..." to "Max. net profit", the optimizer will seek the optimal input values that return the maximum profit possible. There are over 10 different optimization criterion you can select and can be customized via NinjaScript. Please see the "*Understanding Walk Forward properties*" section below for more information.

Understanding Walk Forward properties

Walk Forward Properties

Apart from the walk forward optimization specific properties described below, the properties are identical to the ones found in the Optimization properties window. Please see the "*Understanding optimization properties*" section of the Optimize a Strategy page of the Help Guide for more information.



Tip: You can optionally "**Optimize on**" multiple objectives by using a Multi-Objective optimization

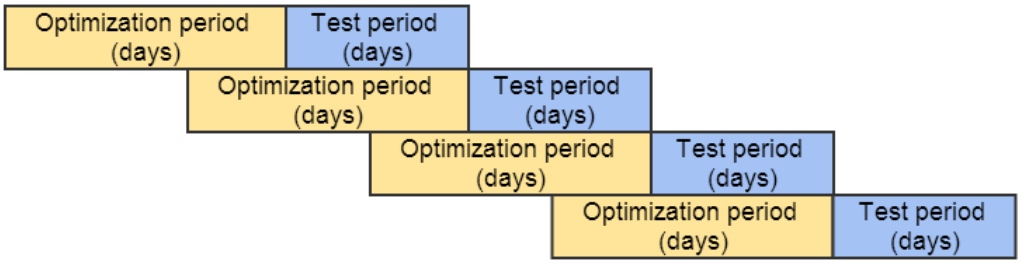
Keep best # results	Sets the number of best results to display

Optimize data series	If set to true, the Data Series Value property will be available for optimization (Not supported for Kagi, PointAndFigure, and Line Break period Types)
Optimize on...	Sets the optimization fitness to base the optimization results on
Optimizer	Sets the optimization algorithm that is used. NinjaTrader comes with "Default" and "Genetic" optimizer algorithms. When the "Genetic" option is selected, the genetic algorithm's optimization properties fields will appear below the Optimizer selection You can program your own optimization algorithm using NinjaScript.
Optimization period (days)	Sets the number of days used for the "in sample" optimization data set
Test period (days)	Sets the number of days used for the "out of sample" real backtest using the optimized input values generated from the "in sample" period

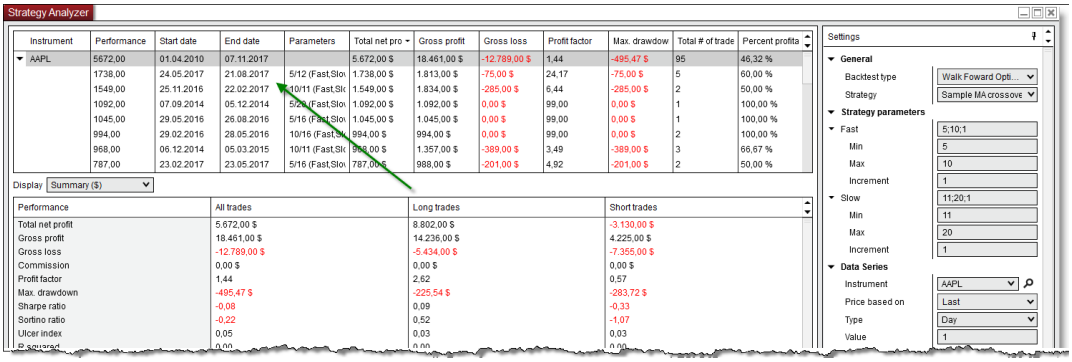
Understanding Walk Forward results

Understanding Walk Forward Test Results

From the Start date to the End date the walk forward optimization will do a standard optimization on the number of days set for parameter "Optimization period (days)". This is known as the "In Sample" test period. After the optimization period NinjaTrader will use the best parameter combination found and test that forward on non-optimized data that has not been seen yet for the number of days set for parameter "Test period (days)". This is known as the "Out of sample" test period. Please see the graph below for a better understanding of how the walk forward results are found.



The results for each "Test period" are returned and shown in the Optimization Results Grid along with the Start date, End date, and the best combination found by the optimization period.



Note: NinjaTrader does save the "Keep best # results" for each Optimization period, if you want to see each individual optimization results you can right click on the walk forward result and select "Open Optimization Results".

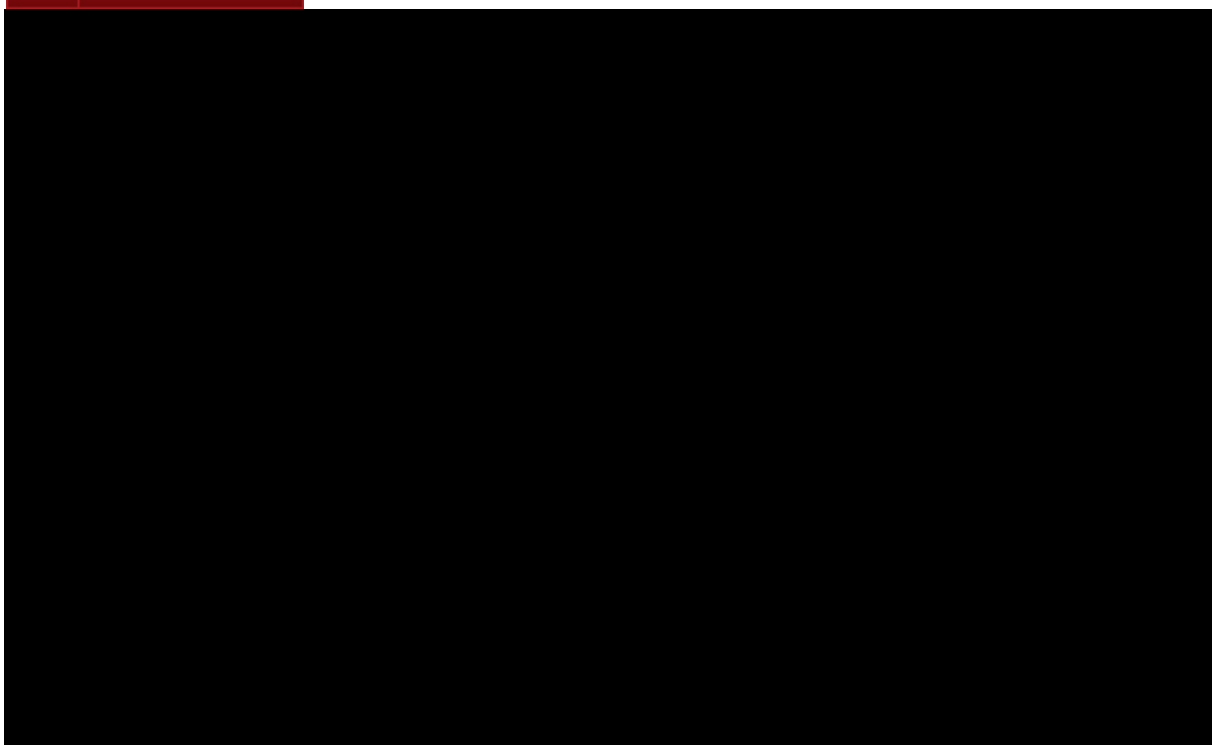
10.27.5 Multi-Objective Optimization

Multi-Objective optimization takes standard optimization a step further by allowing you to choose multiple objectives to test for. When results are returned instead of a singlular list of best results ranked from best to least best instead you will be presented a graph. With multiple objective there is no single best result, instead its up to the trader to choose what is the best tradeoff between two objectives. To run a Multi-objective optimization you will need:

- Access to historical data
- Custom NinjaScript *strategy
- A thorough understanding of the Strategy Analyzer's backtesting and optimization capabilities

Tip: There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore.

Play Video

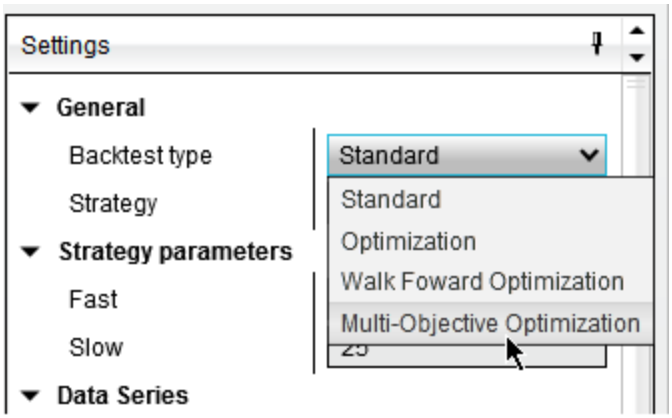


Note: The IncludeTradeHistoryInBacktest property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for optimization. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as SystemPerformance.AllTrades.Count that rely on references to **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state. For more information, see the Working with Historical Trade Data page.

How to run a Multi-Objective Optimization

Start a Multi-Objective Optimization

To run a **Multi-Objective Optimization** select the **Backtest type** of "**Multi-Objective Optimization"** in the settings panel of the **Strategy Analyzer**.

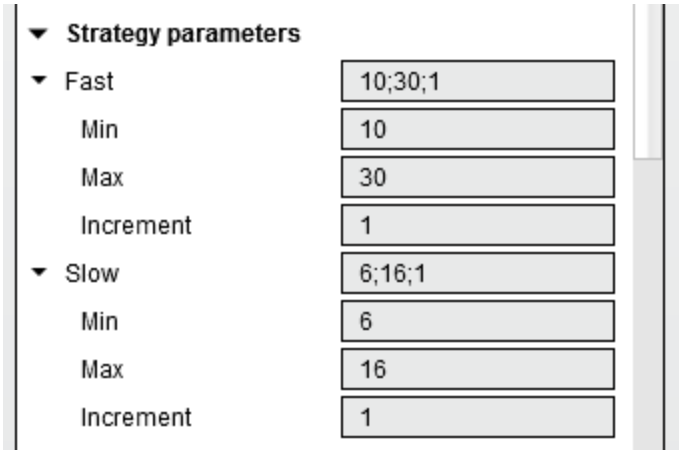


Note: When making the selection additional parameters to configure your optimization will be made visible.

Setting the Test Range

You can set the test range of strategy parameters to be tested by left clicking on the triangle to expand the strategies sub parameters.

Note: If you don't see the triangle make sure that the **Backtest type** is set to "**Multi-Objective Optimization**".



Min. - The starting value you want to test
Max. - The last value to test
Increment - The increment value (step value) used to increment the starting value by for each subsequent optimization pass

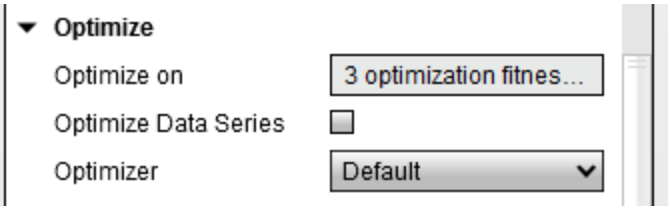
In the image above, the input "Fast" has a starting (initial) value of 10 and an ending value of 30 with an increment of 1. This means that the first value tested will be 10, then 11, then 12 all the way through 30. The input "Slow" has a starting value of 6, ending value of 16 with an increment of 1. Based on these settings, a total of 200 (20 unique values for "Fast" multiplied by 10 unique values for "Slow") backtest iterations will be processed in order to find the optimal combination of input values based on the best optimization fitness.

Understanding Multi-Objective properties

Setting Multiple Optimization Fitness

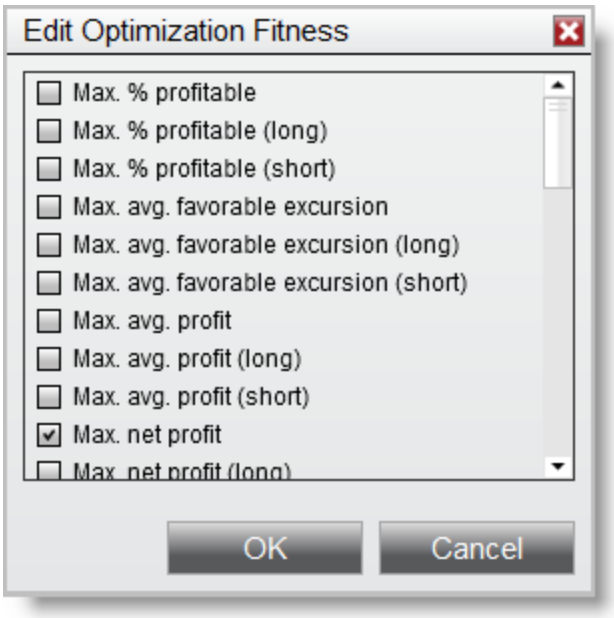
Apart from the "**Optimize on**" property described below, the properties are identical to the ones found in the Optimization properties window. Please see the "*Understanding optimization properties*" section of the Optimize a Strategy page of the Help Guide for more information.

Multi-Objective Optimization is based on the best optimization fitness you select. If you set the property "**Optimize on**" to "Max. net profit", "Max profit factor", and "Min. draw down" the optimizer will seek the optimal input values based on those three optimization fitness objectives. There are over 10 different optimization criterion you can select and can be customized via NinjaScript.



Optimize on...	Sets the optimization fitness to base the optimization results on, left clicking on the field will open the "Edit Optimization Fitness" window where you can enable what optimization fitnesses you want to be tested and to be available for multi-objective analysis.

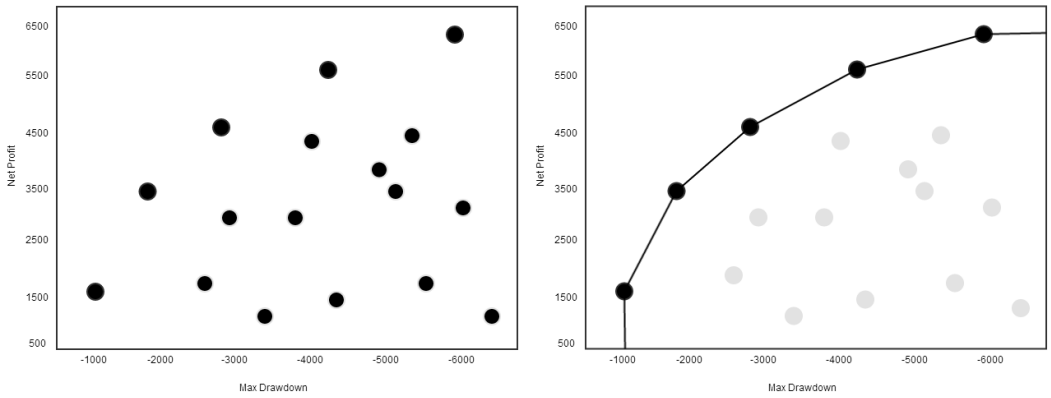
Note: For running Multi-Objective Optimizations the Default optimizer will be used.



Understanding Multi-Objective results

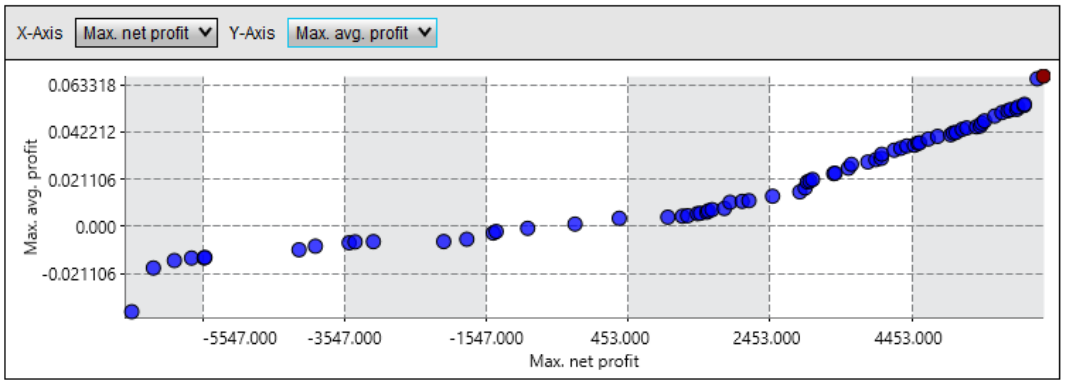
Understanding Multi-Objective Results

Multi-objective results are displayed on a graph instead of a grid. The reason we use a graph is with a **multi-objective** problem there is no one best solution and instead you must compare individual tradeoff between two often competing objectives. Please see the image below to the left with some sample data, each optimization has been performed and the results of each test plotted on the graph. We can narrow down our solution further by only showing results that have the best tradeoff between both objectives known as a Pareto optimal result. In the graph to the right the line drawn connects the 5 single results that are Pareto optimal forming the Paretor frontier. Any result that falls behind the Pareto frontier is discarded leaving us with 5 best tradeoff solutions between the two objectives.



Using the Multi-Objective Graph

There are two combo box selections to choose the optimization fitness will be graphed. You will be able to choose any optimization fitness that you have enabled in the optimize on field in the optimization strategies. See the multi-objective optimization properties section above for more information.



Left clicking on one of the dots will select that optimization run and NinjaTrader will run a backtest with these strategy parameters to retrieve the detailed trade data for further analysis.

10.27.6 AI Generate

The AI Generate optimizer is an experimental tool designed to help traders find new strategy approaches. It can combine up to 73 NinjaTrader default indicators, 25 Candlestick patterns, and single series custom indicators.
Internally a Genetic Algorithm is used to search through the potential entry and exit combinations possible to find the best performing ones according to the Max Strength optimization criterion.
To prevent against potentially over-fitting against historical data, the AI Generate will check its own results after each generation using a Monte Carlo Simulation, it finds the 95% confidence interval.

We are excited to bring you this new tool to enhance your NinjaTrader strategy trading and are looking for feedback to further enhance it.

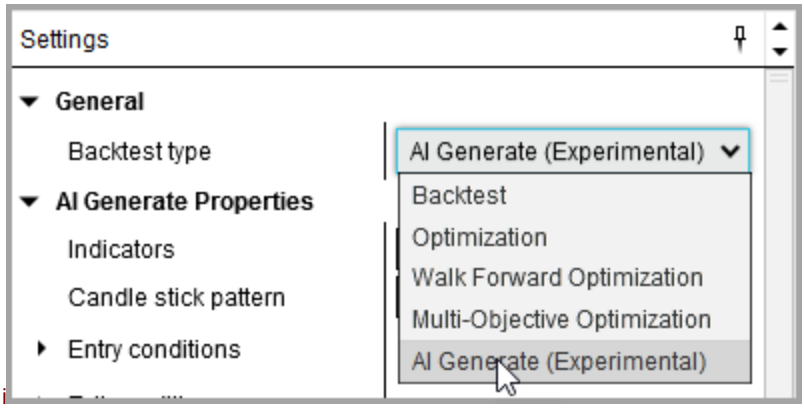
To run an AI Generate optimization you will need:

- Access to historical data
- A thorough understanding of the Strategy Analyzer's backtesting and optimization capabilities

How to run an AI Generate Optimization

Start a AI Generate Optimization

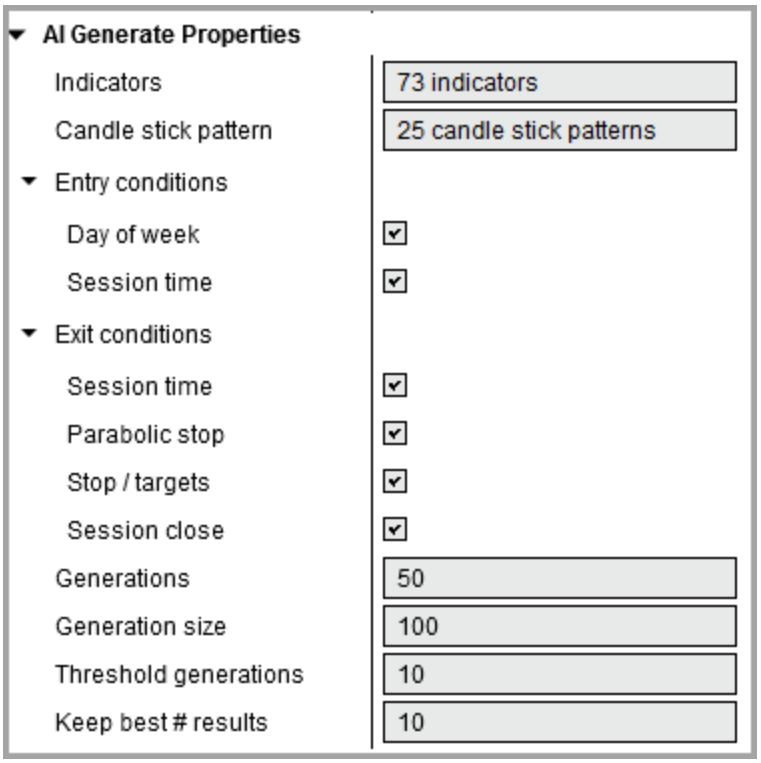
To run a **AI Generate Optimization** select the **Backtest type** of "**AI Generate (Experimental)"** in the settings panel of the **Strategy Analyzer**.



Note: When making the selection additional parameters to configure your AI Generate optimization will be made visible.

Setting the AI Generate Properties

You can set the various AI Generate strategy parameters by left clicking on the triangles to expand the sub parameters (Entry conditions and Exit conditions)



Indicators	Select up to 73 NinjaTrader default indicators which to include in your AI Generation optimization and/or custom indicator that are single series
Candle stick pattern	Select up to 25 NinjaTrader default Candle stick patterns which to include in your AI Generation optimization
Day of week	If checked, the AI Generate optimization will include or exclude certain days of the week as part of the generated entry conditions for the strategies
Session time	If checked, the AI Generate optimization will include or exclude certain parts of the trading session via time filters as part of the generated entry conditions for the strategies or would include them in its exit conditions to allow for time exits

	• for entries between 0 and 60 minutes after session opening, for a duration of max. 120 minutes in 15 minutes steps • for exits between 0 and 60 minutes before session close, going back max. 120 minutes in 15 minutes steps
Parabolic stop	If checked, the SetParablicStop from NinjaScript could be used as an exit for the strategies
Stops / Targets	If checked, would allow for SetStopLoss, SetTrailStop, SetProfitTarget from NinjaScript could be used as exit for the strategies
Session close	If checked, would allow the scripts to exit any open positions by the session end time
Generations	Sets the number of generations to test. Each generation will hold the number of children set in "Generation Size". The number of total parameter combinations tested is equal to the Generation Size * Generations.
Generation size	Sets the number of combinations to test in each generation (children). The higher the size, the more variety of combinations that will be tested in each generation. You want to make sure to set this high enough to test enough parameter combinations to get good coverage of the problem domain but not so high that each possible parameter combination is being tested in a single generation.
Threshold of generations	Determines if the optimization process can be aborted if for the property number of consecutive generations the average of the performance values of the 'stable individuals' (the best 1/5 of the population is not touched on next generation = 'stable individuals') results did not improve. This allows for 'infinite' runs which would be terminated if

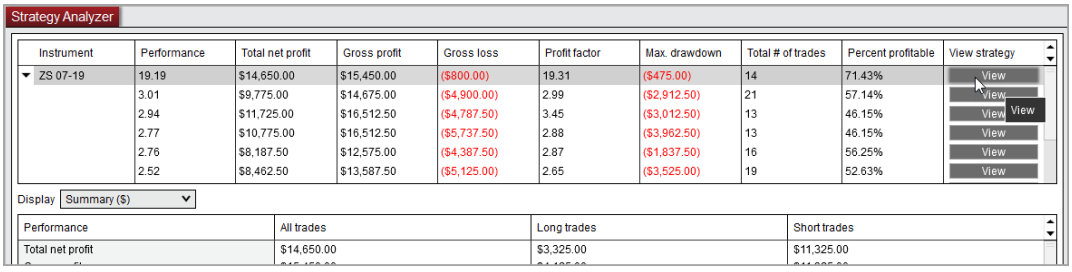
	no improvement is found. This logic is disabled if this property is set to 0.
Keep best # results	Sets the number of best results to display

Notes:

1. You can press 'Abort' to abort the AI Generate optimization, however you would have to wait until the 'generation size' iterations have passed to see the best found solutions so far.
2. In its current experimental state, the AI Generate sits on top the existing optimization framework inside NinjaTrader, as part of that you could see Strategy added indicators as well as the name from the last selected strategy (prior to switching over to the AI Generate optimization) still to appear on the Strategy Analyzer charts.

Viewing and saving results of the AI Generate optimization

Pressing the 'View' button in the optimization results section would let you open the individual generated strategy code in the NinjaScript editor. From there you could then review and also save and further customize.



10.27.7 Understanding Historical Fill Processing

NinjaTrader uses advanced historical fill processing methods and techniques to get the most realistic results possible on historical backtests.

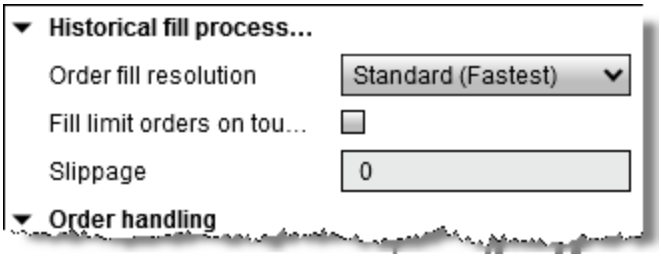
Our Historical Fill Algorithm will run on existing data that you are backtesting and simulate historical orders using the method descibed below in "Understanding the Historical Fill Algorithm". You can optionally choose to bring in a secondary data series to be used to get

more granular fill on orders and is explained in the section "Understanding Order Fill Resolution

Understanding the Historical Fill Algorithm

Historical Fill Algorithm

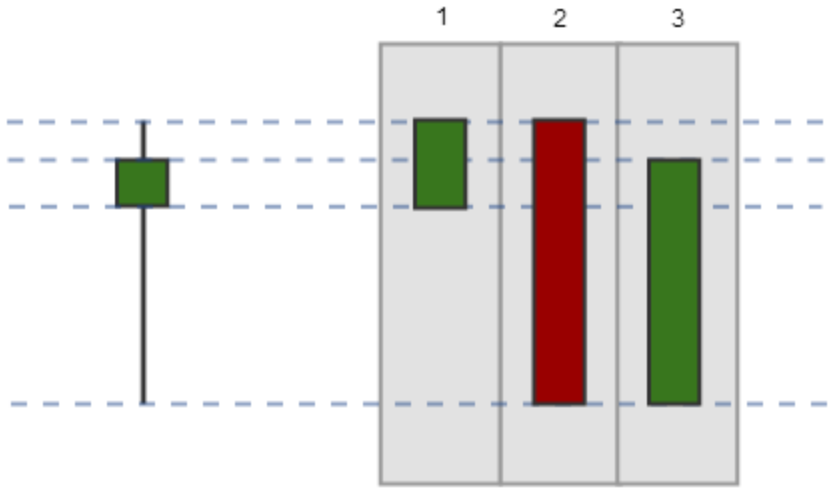
NinjaTrader provides two options to control the granularity of historical order fill processing: Standard and High. The Standard order fill resolution uses an algorithm to break each historical bar into three virtual bars to mimic the movement of price within each bar's timeframe. The virtual bars are created based on the proximity of the Open price to the High and Low prices. This provides more realistic intra-bar fills compared to traditional backtesting algorithms which only use static OHLC values.



The Standard setting creates virtual bars according to the following logic:

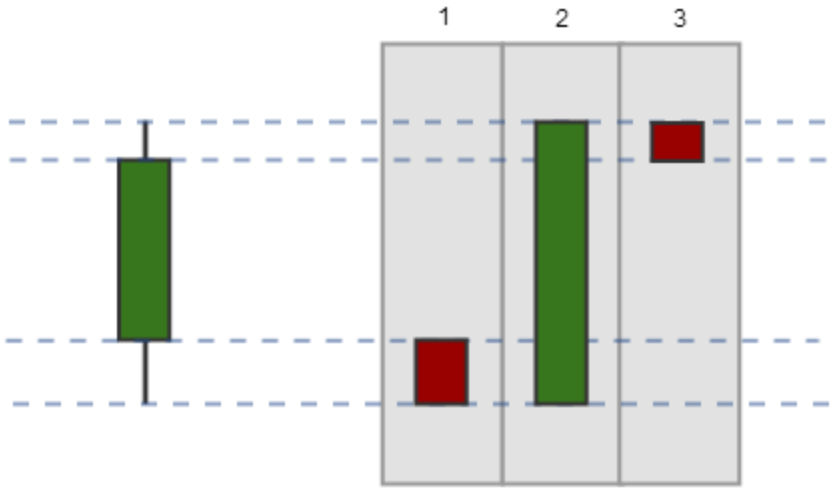
When the **Open** price of the bar is closer to the **High** price than the **Low** price:

1. **Open** price to the **High** price
2. **High** price to the **Low** price
3. **Low** price to the **Close** price



When the **Open** price of the bar is closer to the **Low** price than the **High** price:

1. **Open** price to the **Low** price
2. **Low** price to the **High** price
3. **High** price to the **Close** price



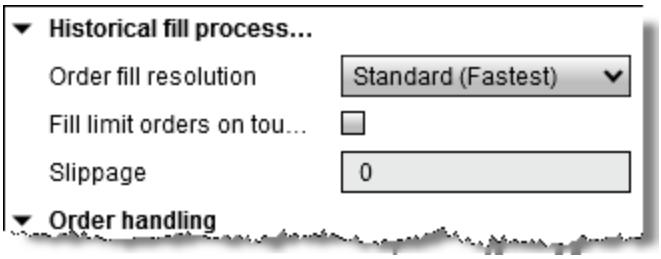
Slippage

Slippage can be added to your order fills to help mimic real market conditions. The value is expressed in "ticks", the minimum value of fluctuation for an instrument, and is only applied to market, stop-market and Market-if-touched orders. NinjaTrader will add the slippage to each order however you cannot have more slippage then the high/low price of the next bar.

Understanding order fill resolution

Order Fill Resolution

NinjaTrader allows you to pull in additional historical data that will be more granular than what you are using for the strategy backtest to be used to give you more data points of which to fill orders. Allowing for more accuracy in the order fill simulation.



Order fill resolution of "**Standard (Fastest)**" is the default setting and will use the existing bar type and interval that you are running the backtest on to fill your orders. This means that the historical fill algorithm will use the same Open, High, Low, Close, Time values that are available to the strategy for running the order fill simulation.

Selecting **order fill resolution** of "**High**" will allow you to set a secondary bar series to be used as the price data to fill your orders, this allows you to bring in more granular data then you are currently running the strategy on. For example you may have a strategy that you run on "**Daily**" bars but then want to bring in "**Minute**" bars for the **historical fill algorithm** to be based on.

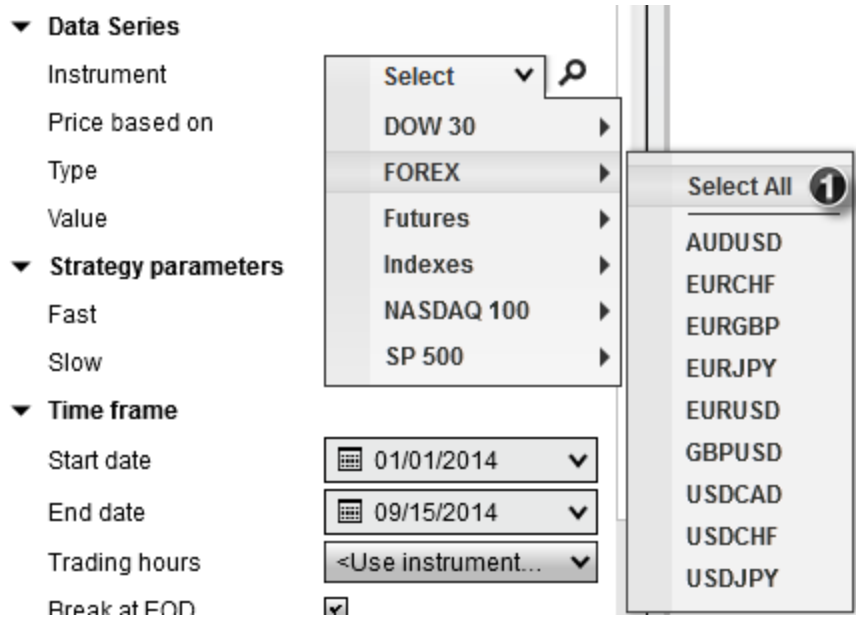
The secondary bar series will mimic the 'price based on' setting in your Strategy Analyzer settings, should you wish to mix different prices types, for example generate signals of last based data and execute those to a bid / ask series, this could be achieved with further custom programming.

Notes:

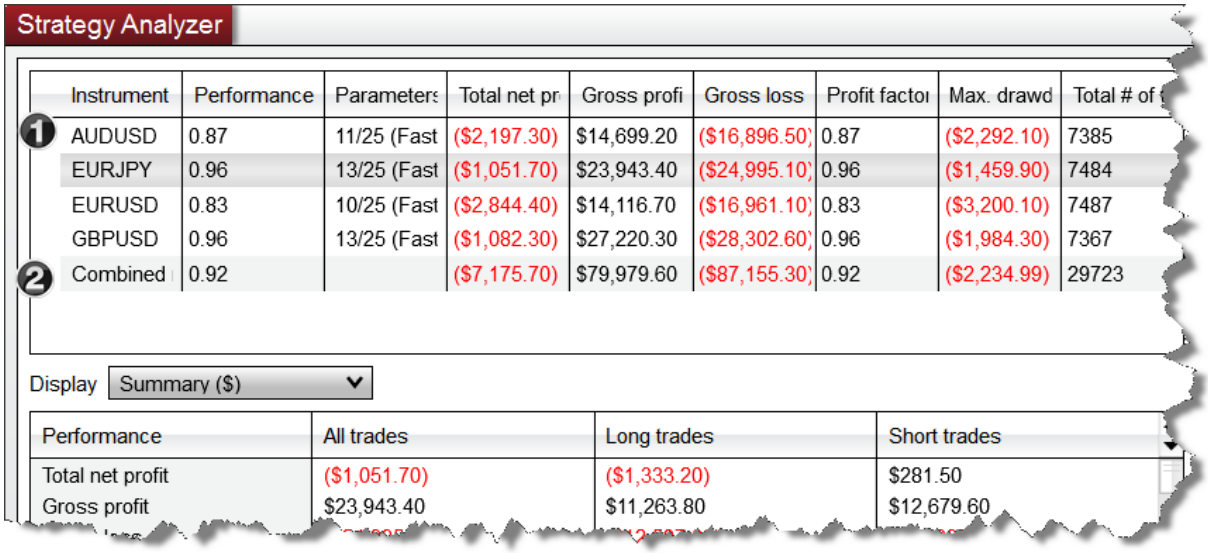
- You could choose to always use the most granular **order fill resolution** such as a 1 Tick Data Series**.** However this forces NinjaTrader to process this additional data for use in the **historical fill algorithm**. This results in longer backtest times due to the additional data that needs to be processed. NinjaTrader will only start the backtest after we have loaded historical data for both the strategy and the order fill resolution.
- **Order fill resolution** cannot be used with Multi Time Frame/Multi Instrument strategies, or when Tick Replay is used. For those cases, a strategy should be written to submit orders to a single tick data series.

10.27.8 Basket testing multiple instruments

You can Backtest, Optimize or Walk Forward optimize a basket of instruments by selecting an instrument list using the instrument selector in the settings panel.



Once the test is complete, a listing of all the results will be displayed.



1. Each instrument's backtest results are displayed individually

2. The combined backtest results of ALL instruments are shown at the bottom of the results

Selecting an individual row from the results grid will display the results in the Performance tabs individual performance results.

Reviewing Combined Results

When reviewing the following combined results, some values will be the total summation across all instruments, while others will be weighted to the total number number of trades.

The following results will be a summation across all instruments:

- Total net profit
- Gross profit
- Gross loss
- Commission
- Total # of trades
- # of winning trades
- # of losing trades
- # of even trades

For all other statistics, the combined results will be a weighted average (exception here is the RSquared statistic).

Calculating Weighted Combined Results

In order to understand how weighted combined results are calculated, lets use a simplified example which focuses on the Max. Drawdown across 4 different instruments:

Instrument	Max. Drawdown	Total # of trades
AUDUSD	($250.00)	200
EURJPY	($150.00)	105
EURUSD	($200.00)	20
GBPUSD	($50.00)	90
Combined Results	**(178.92)**	**415**

As you can see, the Max.Drawdown column is **NOT** equal to the sum of the individual Max.Drawdown values for the that column. This is because the total # of trades for the individual instrument and the total # of trades taken across all instruments is used to help provide more accurate statistics. Working from the table above, the formula used to calculate these weighted averages can be expressed as follows:

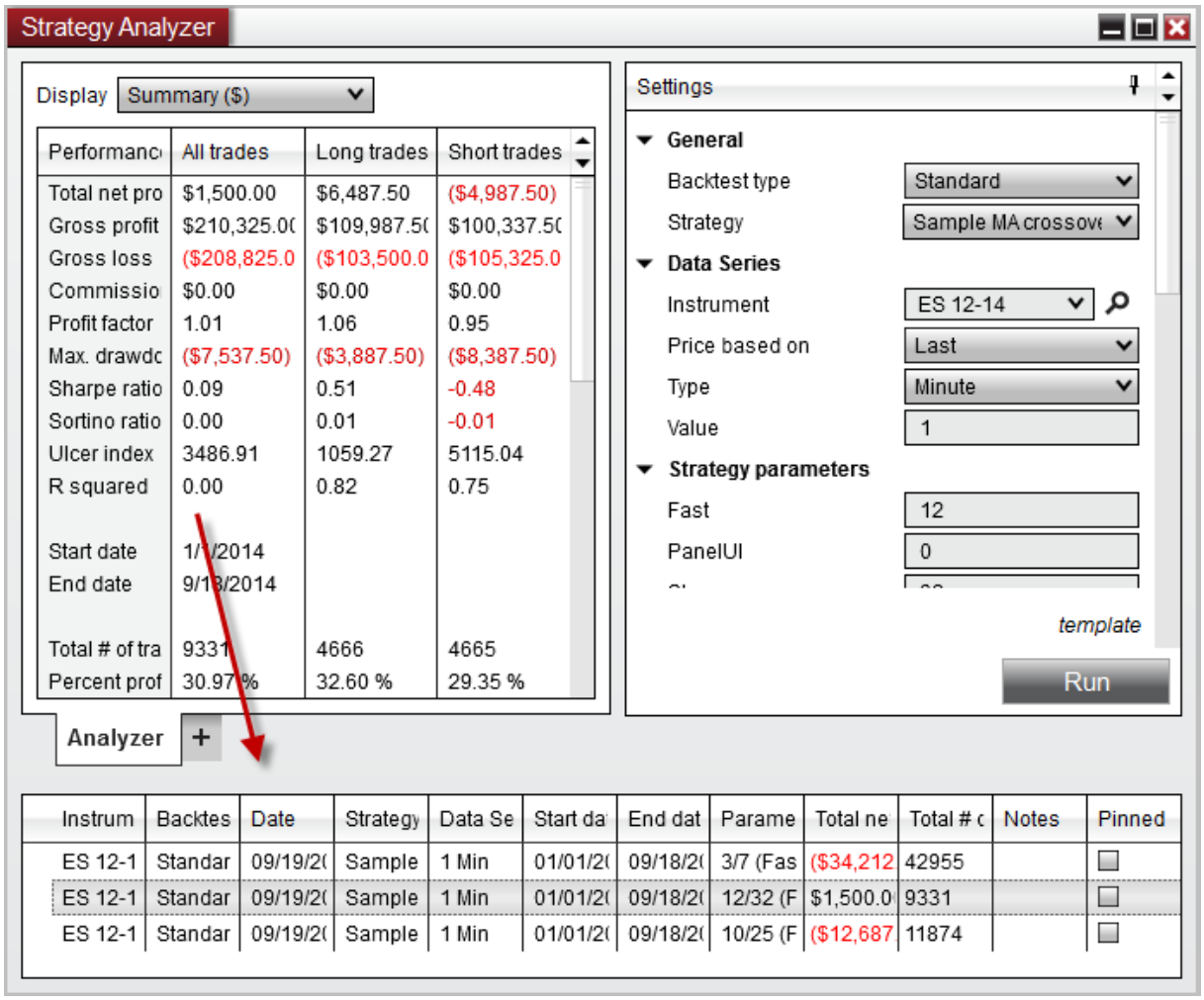
```
Combined Max.Drawdown = SUM((AUDUSD Drawdown * AUDUSD Trade Count) + (EURJPY Drawdown
* EURJPY Trade Count) + (EURUSD Drawdown * EURUSD Trade Count) + (GBPUSD Drawdown *
GBPUSD Trade Count)) / Total Trade Count of All Instruments
```

10.27.9 Understanding Backtest Logs

The Strategy analyzer saves a log on each backtest. The logs can be seen by right clicking on the Strategy Analyzer and selecting "**Show Logs**". Logs offer a convenient way to keep a history of backtest results. They can be used as you work to develop a strategy and fine tune parameters and code to compare previous backtests to current backtests easily.

The log also contains a saved snapshot version of the code used for the backtest, making it possible to look at or revert to previous code used.

> **Note**: Code save functionality only works on open and unlocked NinjaScript Strategies. Strategies which are protected by the vendor cannot be used to save code.



Understanding what is saved in the logs

Understanding Logs

NinjaTrader saves a log each time you perform a backtest in the strategyt analyzer. It saves several key information in the log which makes it easier to iterate on a strategy over time.

The log saves the following information per test:

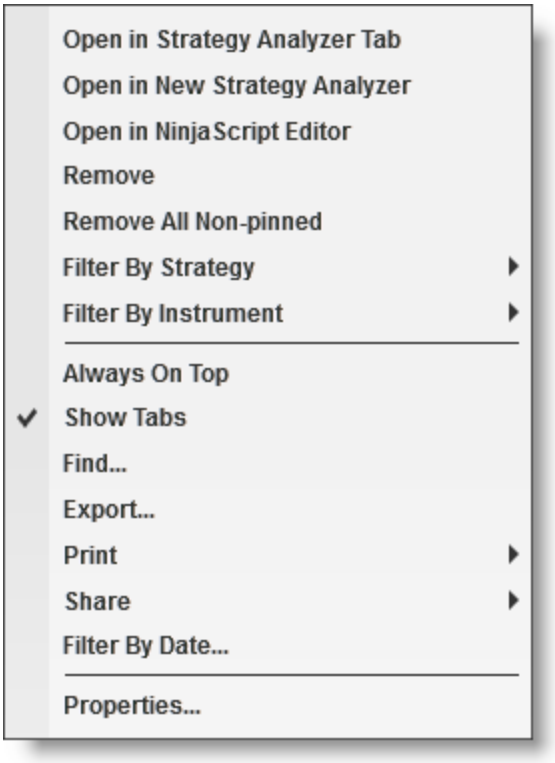
Instrument	The instrument the test was performed on.
Backtest	The type of backtest that was performed
Date	The date the backtest was performed
Strategy	The strategy used for the backtest
Data Series	The data series used for the backtest
Start date	The start date used for the backtest
End date	The end date used for the backtest
Parameters	The parameters used for the backtest
Total net profit	The total net profit for the backtest
Notes	An optional field to add user defined notes to more accurately recall the test. Double click the field to begin editing and when complete press enter on the keyboard to set the note.
Pinned	An optional field to Pin a result to the top. Pinned results are useful for saving a specific backtest of note for reference later.

▼ Using Logs

Using Logs

Logs are integrated with the Strategy Analyzer and can be double clicked to quickly restore the parameters and backtest information for that backtest. Giving you freedom to experiment with different configurations while maintaining the ability to compare previous backtests and restore a previous backtest at any time.

Right clicking on a backtest log yields the following context menu:



Open In Strategy Analyzer Tab	Opens the backtest in a new tab in the current Strategy Analyzer window
Open in New Strategy Analyzer	Opens the backtest in a new Strategy Analyzer
Open in NinjaScript Editor	Opens the saves revision of the code as used when the backtest was run. You can restore any set of backtest

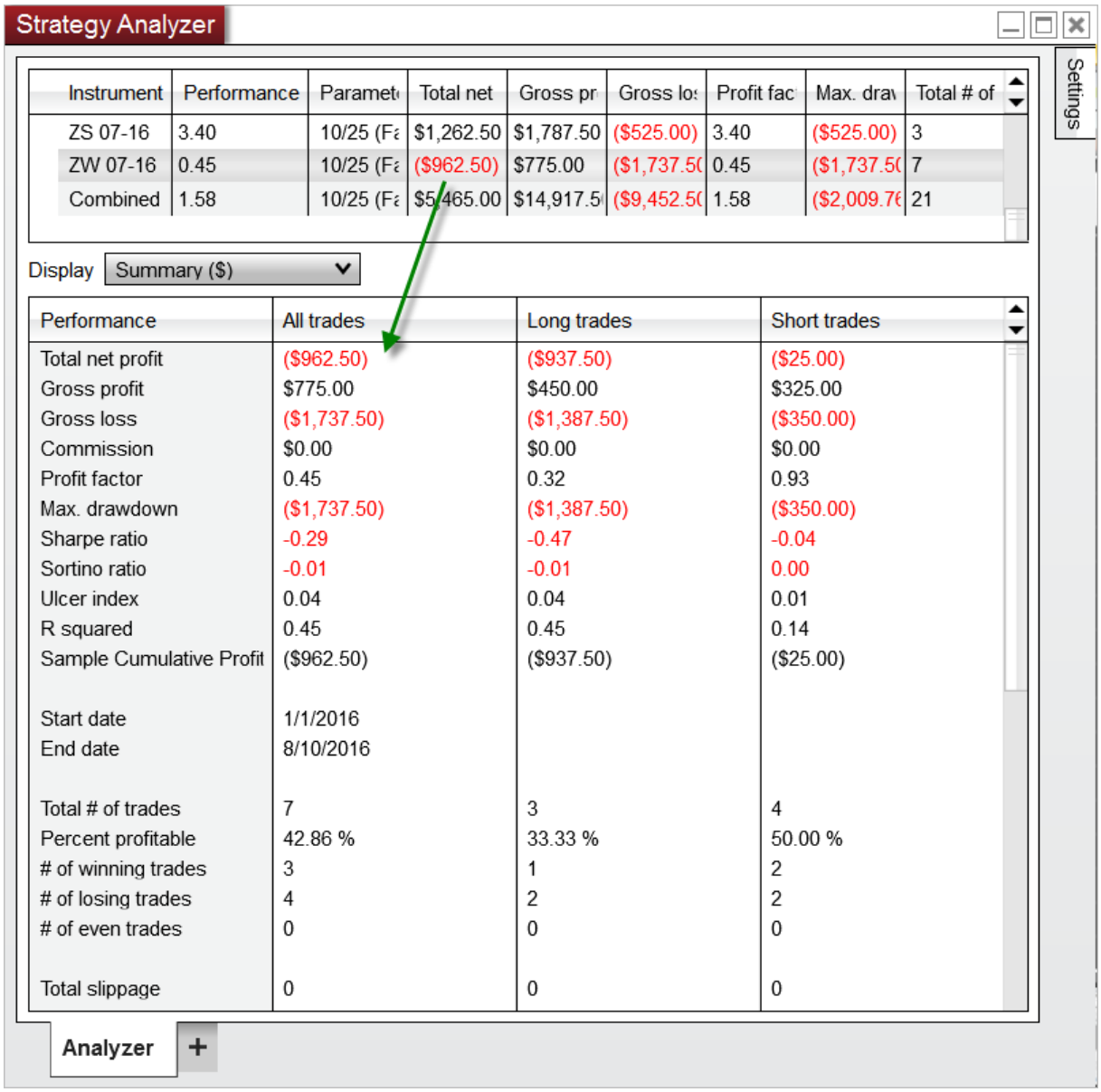
Remove	Removes the selected backtest log
Remove All Non-Pinned	Removes all backtest logs that are not pinned.
Filter By Strategy	Only view backtest logs for a specific strategy
Filter By Instrument	Only view backtest logs for a specific instrument
Filter By Date	Only view backtest logs for a specific date range

10.27.1 Reviewing Performance Results

Strategy Analyzer generates performance data that can be viewed in Performance Displays. When working with Optimizations or basket tests you can choose open an individual tab or new Strategy Analyzer window to analyze each individual backtest. Selecting an individual row from the results grid will display the results in the Performance tabs individual performance results.

Notes:

- When viewing combined backtest or optimization results, many of the values shown are a weighted average based on the total number of trades. This is used to provide a more accurate representation of combined trade performance. Please see the page on Basket testing multiple instruments for more information.
- Strategy performance statistics can be found under the Trade Performance Statistics Definition Page



10.27.1 Monte Carlo Simulation

Monte Carlo Simulation Overview

Monte Carlo Simulation is a mathematical technique used to study data that is highly random in nature. When used for trading, it is a method of randomizing trade results and running those results in a series of simulations to analyze the probability of multiple outcomes. This type of analysis will help you recognize if your strategy runs the risk of wiping out your account before it can turn a profit or not. Monte Carlo Simulation can be selected in the display drop down after a backtest has been run.

› Running a Monte Carlo Simulation

10.27.11. Running a Monte Carlo Simulation

The following page covers how to set up and run NinjaTrader's Monte Carlo Simulation

Understanding Monte Carlo simulation

What is Monte Carlo Simulation?

Monte Carlo Simulation is a mathematical technique that uses repeated random sampling ("sampling with replacement") to compute a range of possible results with their respective probability. NinjaTrader runs Monte Carlo Simulation by randomly combining the trade results in a defined series of simulations. A graph of the results are plotted with the statistic values or Profit/Loss on the Y - axis and the probability on the X - axis as a percentage.

Why use Monte Carlo Simulation?

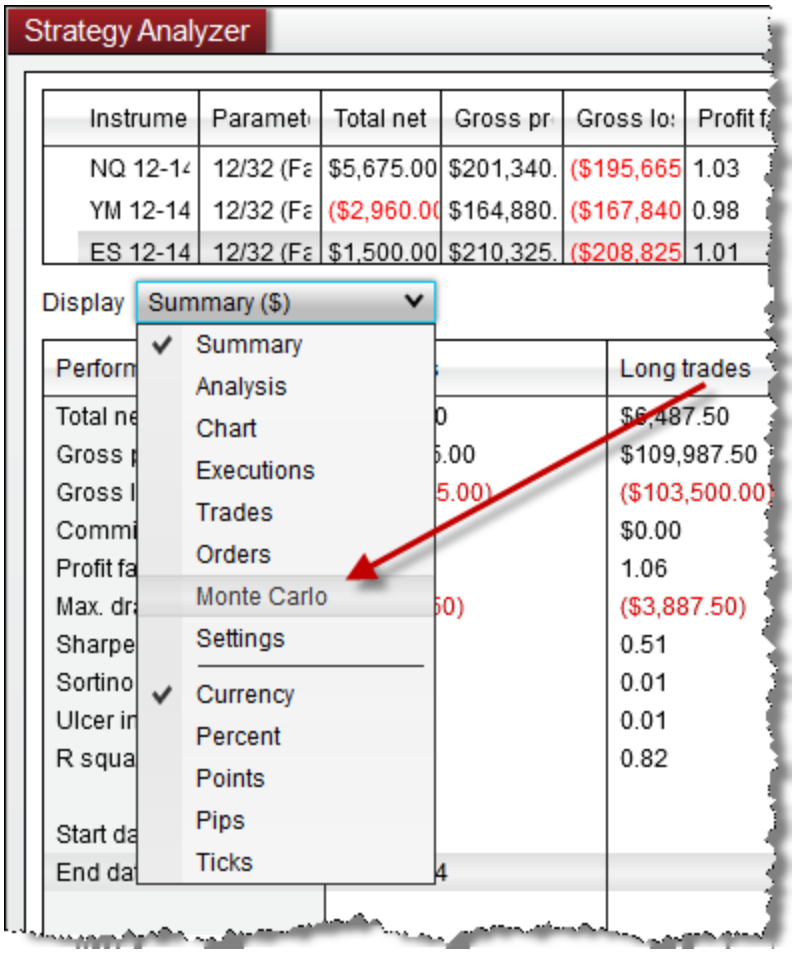
Although a backtest of a NinjaScript strategy may produce profitable results, those results may have just been due to good luck. In real life, you may have a string of bad trades that can wipe out the account before the good trades appear, therefore it would be helpful to understand the probability of such a string of bad trades. Monte Carlo Simulation will randomize your trade results over and over again in multiple simulations to provide you with a normal distribution of simulation performance. The trader can use this information to see the top or bottom percent of trades (outliers) that will cause the most variability in the strategy as well as the most statistically probable results.

How to run a Monte Carlo simulation

Monte Carlo Simulation window

To open the Monte Carlo Simulation window:

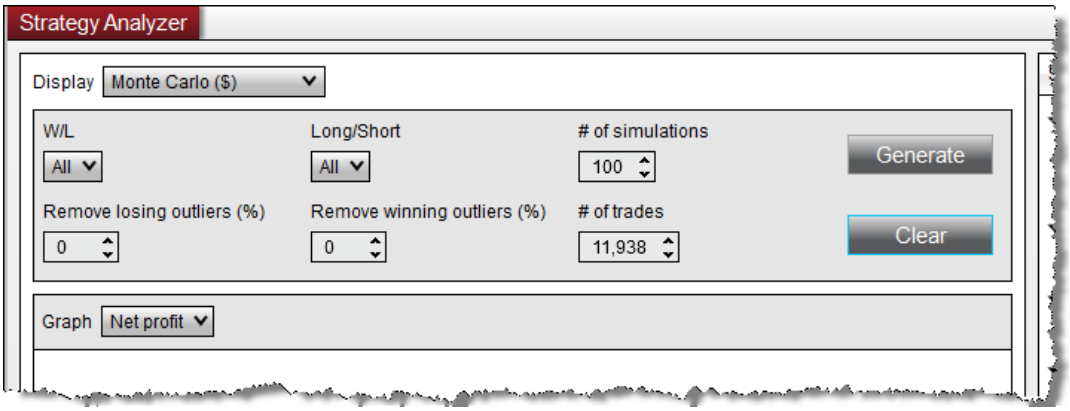
1. Run a Backtest, Optimization, or Walk-Forward Optimization.
2. Left mouse click on the Trades tab within any of the reports
3. Right mouse click in the data grid and select the item **Monte Carlo Simulation...**



Running a Monte Carlo Simulation

To run a Monte Carlo Simulation:

1. Open the Monte Carlo Simulation display (see sub-section above for how to open)
2. Set desired simulation parameters and press the Generate button.



Monte Carlo Simulation Parameters

The following parameters are adjustable when running a Monte Carlo Simulation:

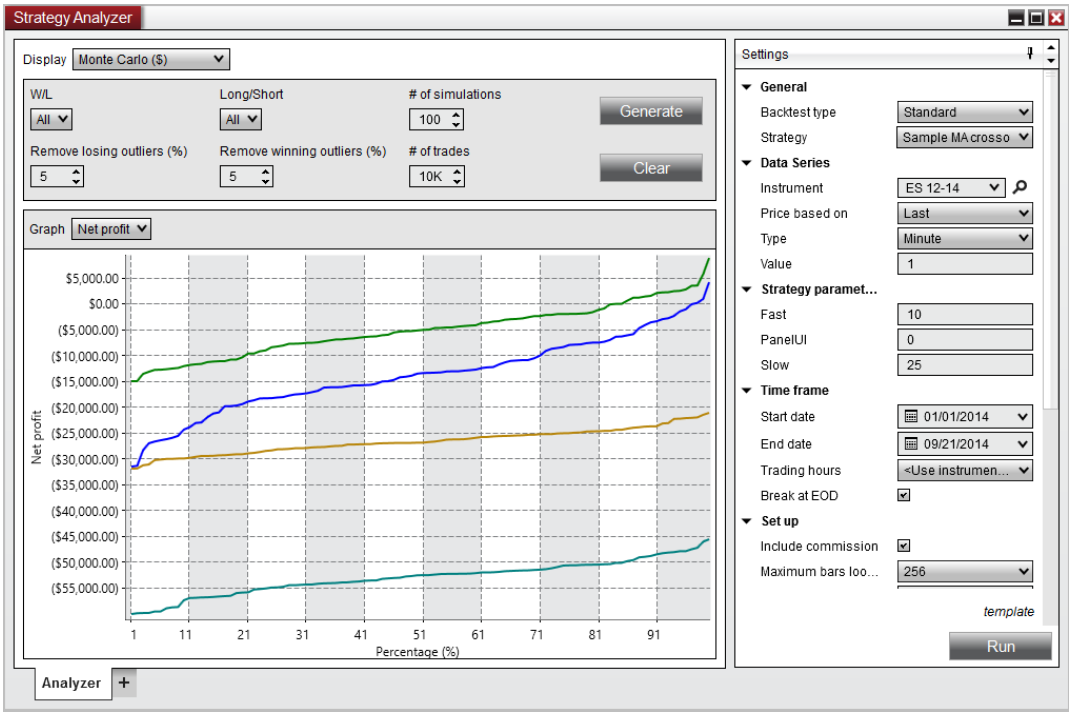
Graph	Sets the statistic to generate the report on
W/L	Sets the results to show only winners, only loser, or both
Long/Short	Sets the results to show only long trades, only short trades, or both
Remove winning outliers (%)	Removes the top % outliers from the results
Remove losing outliers (%)	Removes the bottom % outliers from the results
# of simulations	Sets the # of simulations to run
# of trades per	Sets the # of trades in each simulation (will default to the # of trades in the Trades tab)

simulation	

Understanding the Monte Carlo Simulation report

Monte Carlo Simulation Report

The results of the Monte Carlo Simulation are displayed in a graph below the parameters.



X-Axis

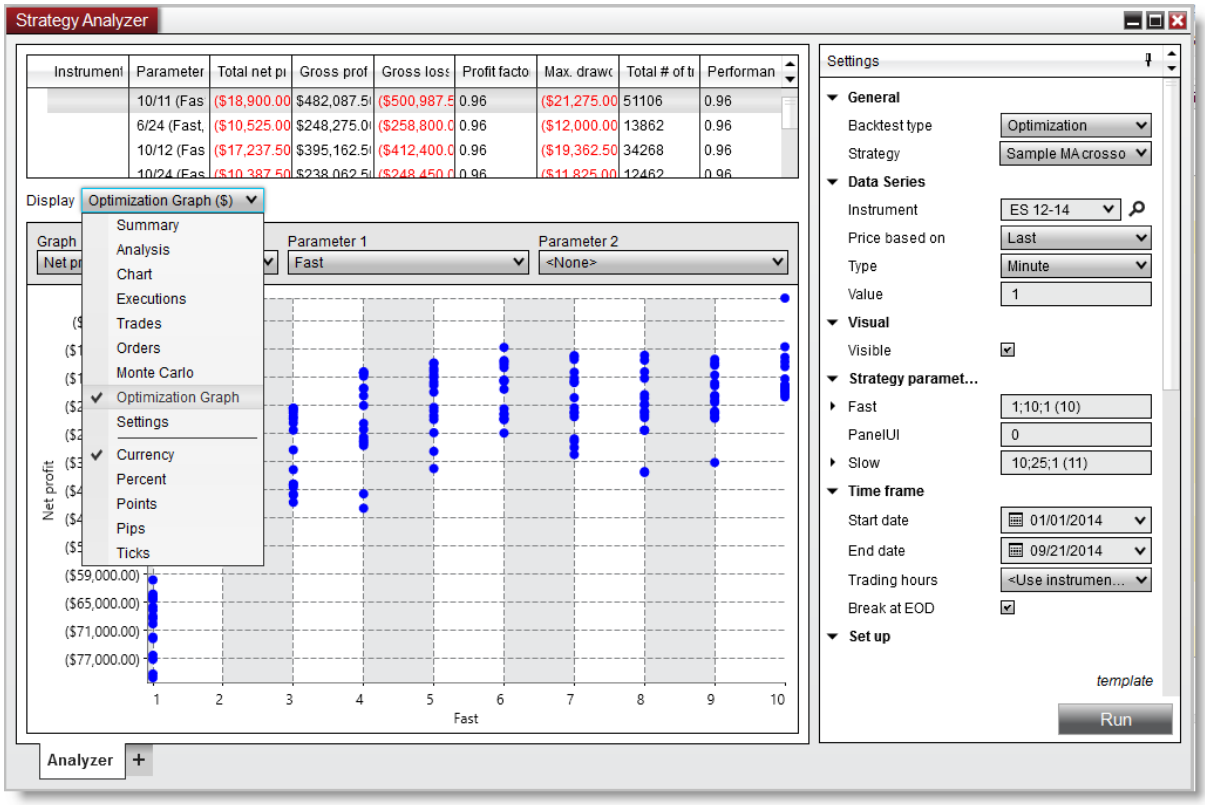
The horizontal axis of the Monte Carlo Simulation graph shows the percentage of simulations that have fallen below the Y - axis value. For example, if you run a Monte Carlo Simulation setting the # of Simulations to "100" and using the Cumulative Profit graph, the intersection of the 50% X - value and the associated Y value means that 50 of your simulations will be below that cumulative profit/loss value, and oppositely the remaining 50 simulations will have a greater cumulative profit/loss. This type of report allows you to analyze if the risk/reward ratio between worst and best case scenarios is acceptable or not.

Y-Axis

The vertical axis of the Monte Carlo Simulation graph displays the measured unit for the **Graph** item selected such as Profit/Loss, statistical information, or time and changes based on the Graph selection.

10.27.12 2D & 3D Optimization Graphs

The **Optimization Graph** can only be selected in the **Display** selector only after an optimization has been run. The optimization graph can be displayed in a 2D or 3D graph. A 2D graph is used when only graphing a single parameter. If you graph 2 parameters then a 3D graph is displayed.



Understanding the 2D optimization graph

Understanding the 2D Optimization Graph

The 2D optimization graph displays each and every test run for the optimization. This allows you to see the entire range of results produced from an optimization run. Allowing you to take a look over the entire solution domain to determine if your top results are stable. Instead of choosing the absolute best parameter set that might be an outlier you may instead desire to choose a parameter that has a gradual build up which may indicate stability in the result set.

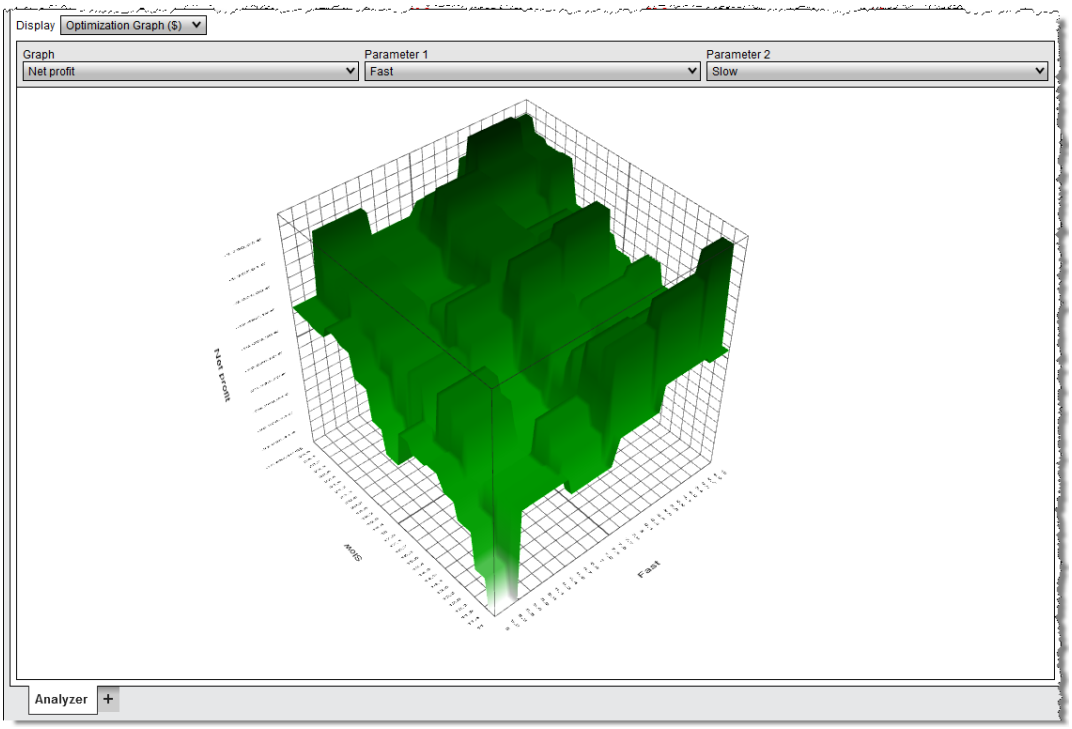
The 2D Optimization graph will be displayed when you have only selected a single parameter and is the default graph view.

> **Note**: Selecting a 2nd Parameter will switch to the 3d graph.

Using the 2D Optimization Graph

Each dot signifies a backtest result, graphed by the X-Axis and the Y-Axis. The X-Axis can be changed by selecting the **Graph** parameter.

Understanding the 3D optimization graph



Understanding the 3D Optimization Graph

The 3D optimization graph expands upon the 2D optimization graph by allowing an additional axis to place an additional parameter. You must have at least 2 parameters being optimized and with the 'Parameter 2' combo box select the secondary parameter. This will trigger the display of the 3D optimization graph. Select 'None' to return to the 2D optimization graph.

Using the 3D Optimization Graph

Using the following mouse controls you can interact with the 3D optimization graph.

Pan	Press the Middle Mouse Button to pan the graph
Orbit	Pres the Left Mouse Button to rotate / orbit the graph
Zoom	Use the Scroll Wheel to zoom in / out

10.27.13 Discrepancies: Real-Time vs Backtest

You should expect that a strategy running real-time (live brokerage account, live market simulation, Playback connection etc...) will produce different results than the performance results generated during a backtest. This difference may be more easily seen on certain Bars types (e.g. Point and Figure or Renko) than others due to their inherent nature in bar formation.

Getting Filled on an Order

- Fills are determined based on 4 data points, OHLC of a bar since that is the only information that is known during a backtest.
- During simulation using real-time live market data or **Playback**, the fill algorithm is dynamic in that it uses incoming market data (both price and volume) to determine if an order should be filled or not.
- During real-time live brokerage trading, orders are filled according to market dynamics.

As you can see, there are three distinctly different models for how and when an order may be filled. This is why you may see orders NOT fill in real-time that you may otherwise expect to see filled based on your backtesting results.

The Fill Price of Orders

- During a backtest assumptions are made on the fill price of an order is based on the OHLC of a bar and the price of the order itself. You can also have differences depending on which fill algorithm you choose.
- During simulation using real-time market data or **Playback**, the fill price is based on incoming market data and volume, you may receive better or worse fill prices dependant on where the bid or ask price is and what volume is available at this market prices.
- During real-time brokerage trading, orders are filled according to market dynamics.

As you can see, there are three different models on what price an order can be filled at.

Running a Strategy at the Close of a Bar or Tick by Tick

- During backtest, strategies can ONLY be processed at the close of each bar.
- During real-time operation, you have a choice to run a strategy tick by tick (Calculate set to 'On Each Tick') which can produce different results. This is because you can have a signal that executes an order at the close of a bar but when running tick by tick, while in a bar a signal condition can be true although its false at the close of the same bar.

Differences in chart data

- If you run a strategy in real-time on DAY1 and then DAY2, you are now backtesting your strategy on DAY1 data instead of processing like it did in real-time so there could be differences. You should understand how chart bars are built.
- If using tick based charts, all it takes is a single tick difference between real-time and historical data to generate completely different looking charts. This in turn would impact the calculations of your strategy should the data sets be different.

Backtesting Renko bars and HeikenAshi bars

- Standard Renko bars can be challenging for backtesting as the barstypes use functionality to remove the last bar in the data series and replace the bar with a new Open price.This is done with reversals and cannot be simulated historically, because the bars are already built. Testing may be done with the Playback Connection to fully simulate standard Renko bars with accurate reversals and order fills.
- Some custom Renko bars aim to work around this limitation and often substitute a "fake" open price in the bar. This will result in Standard Order Fill Resolution estimating fill prices using OHLC values that are not 100% reflective of actual price action which can create discrepancies with results. High Order Fill Resolution should be used in cases like this, or the strategy should be written to submit orders to a single tick data series.
- Similarly to custom Renko bars which use a "fake" open price, HeikenAshi bars alter the OHLC values of the data series to use the HeikenAshi moving average values. Since these are not real prices, it would be best to consider the same workarounds as noted above, or to use a HeikenAshi indicator. Please inquire with platformsupport@ninjatrader.com for more information on how to find a HeikenAshi indicator.

10.27.14 Strategy Parameter Templates

NinjaTrader allows a convenient way to save strategy parameters to easily transition to a live running strategy.

Saving a Template

Using the 'template' button on the bottom of the settings button shows 'Save' and 'Load'. Selecting **'Save'** allows you to save the selected settings for this strategy. If you have performed an optimization the selected optimization result set will be saved. This is signified by the "(" + ")" number directly to the right of the strategy parameter control.

If you save as 'Default' the template will be automatically loaded as you load the strategy.

Loading a Template

Using the 'template' button on the bottom of the settings button shows 'Save' and 'Load'. Selecting **'Load'** opens the loading dialog box where any templates specific to this strategy can be loaded. This allows you to have multiple configurations customized per instrument.

10.27.15 Strategy Analyzer Properties

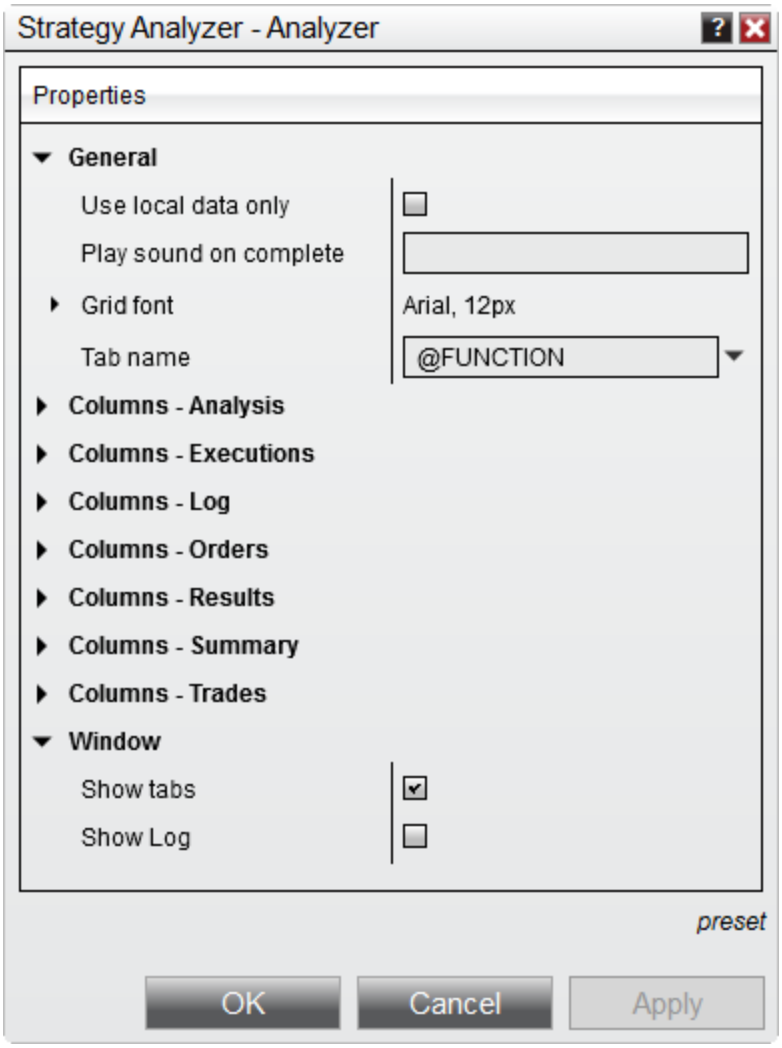
Many of the **Strategy Analyzer** visual display settings can be customized using the **Strategy Analyzer Properties** window.

How to access the Strategy Analyzer Properties

You can access the Strategy Analyzer Properties dialog window by clicking on your right mouse button and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the **Strategy Analyzer** Properties window:



Property Definitions

General	
Use local data only	When enabled the strategy analyzer will not make a request for historical data from the provider and used stored data in the repository only.
Play sound on complete	Once a back-test or optimization is complete, the

	chosen alert sound will be triggered.
Grid font	Sets the font options
Tab name	Sets the name of the tab, please see Using Tabs for more information.
Columns - Analysis	
Columns - Executions	
Columns - Log	
Columns - Orders	
Columns - Results	
Columns - Summary	
Columns - Trades	
Window	
Show Tabs	Sets if the tabs are visible or not.
Show Log	Set if the log feature is enabled or disabled.

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "template" text located in the bottom right of the properties dialog. Selecting the option "save" and naming it "Default" will save these settings as the default settings used every time you open a new window/tab. Saving the template with other names will allow you to save additional configuration that you could load.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "template" text and select the option to "reset" to return to the original settings.

10.27.16 Working with Historical Trade Data

The Trade class allows you to directly access information about historical trades. However, **Trade** objects are not always accessible from a NinjaScript strategy by default. The IncludeTradeHistoryInBacktest property determines whether references are made in memory to **Trade** objects, allowing you access them programmatically after a trade has completed, or whether no references are made, freeing up memory for other uses.

IncludeTradeHistoryInBacktest = True

When the **IncludeTradeHistoryInBacktest** property is set to **true**, **Trade** objects will be saved for later reference. This provides a reference to the object in memory, allowing you to access them in your script. For example, this setting would allow you to evaluate the Max Adverse Excursion statistic of an individual trade placed by the strategy in the past. While this can be convenient to address specific needs, it uses more memory than the alternative option. To maximize performance in cases in which you know you will not need to access historical **Trade** objects, it is recommended to set **IncludeTradeHistoryInBacktest** to **false** in the **Configure** state in your script (or in **SetDefaults** state when adding the strategy from the strategy tab).

IncludeTradeHistoryInBacktest = False

When the **IncludeTradeHistoryInBacktest** property is set to **false**, **Trade** objects will not include a reference in memory. Once a trade is completed, no **Trade** object will be accessible to the script. This setting allows for leaner memory management by avoiding the storage of a potentially large number of objects that may never be used. That being said, if you know that you will need to access these objects after trades have completed, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state in your script (or in **SetDefaults** state when adding the strategy from the strategy tab).

Notes:

- Since trade information is not stored you will only see entry/exit executions plotted on the chart with no connecting PnL trade lines.
- **IncludeTradeHistoryInBacktest** is always defaulted to **true**, except when the strategy is running on the strategy tab.

10.28 Strategy Builder

Strategy Builder Overview

The **Strategy Builder** is used to generate NinjaScript based strategies for automated systems trading. The **Strategy Builder** can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center, and selecting the menu item **Strategy Builder**.

In conjunction with understanding how to build strategies using the **Strategy Builder**, it is **imperative** that you:

- Understand the overall concepts of developing strategies and how they work
- Understand the backtesting options available in the Strategy Analyzer

Once you have developed a NinjaScript strategy you can run it live in full automation.

This **Strategy Builder** help guide section is divided into the following categories:

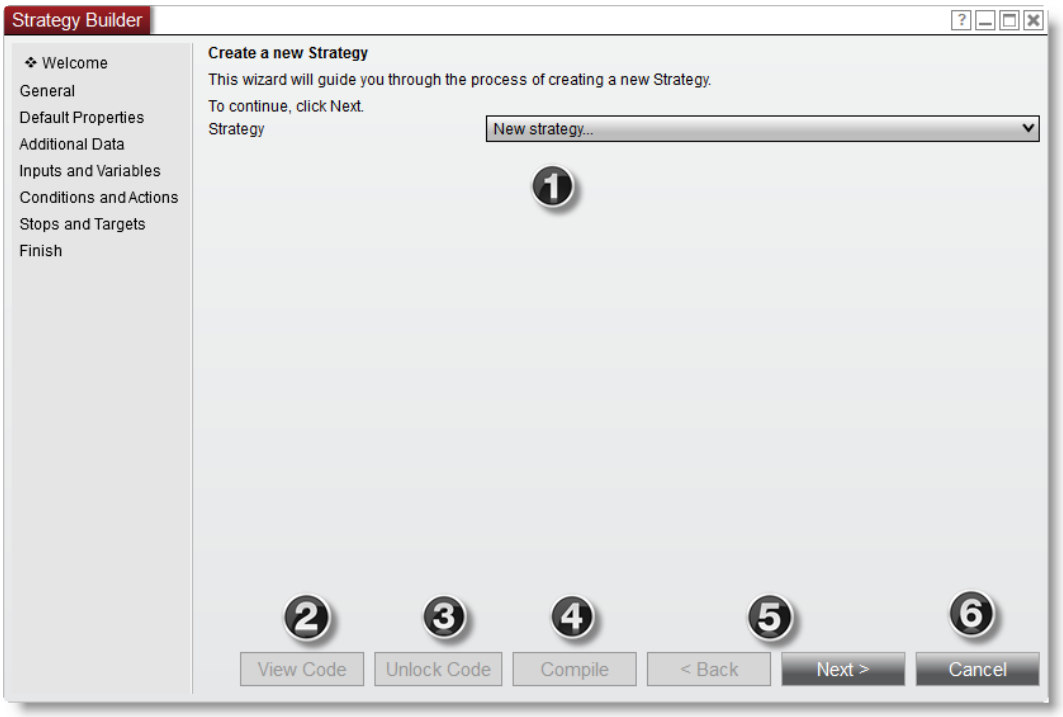
10.28.1 Builder Screens

The Builder point and click interface is a powerful entrypoint into NinjaScript strategy development for non programmers. Even if you target more deeper custom coding later on in the development cycle, the Builder can provide a great foundation to start with. To get started directly into full fledged programming a strategy object in the NinjaScript editor, please check into NinjaScript Wizard.

Understanding the Welcome screen

Welcome Screen Layout

This is the first screen and starting point in the **Strategy Builder**.

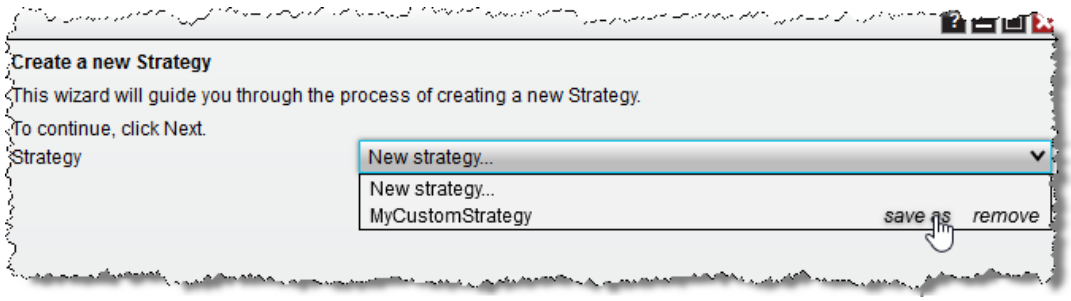


1. In the **Strategy** drop-down select **New Strategy** to create a new strategy script - all other Builder made scripts will be listed as well, so should you wish to modify a script - please select the desired one and proceed through the screens.
2. Press the **View Code** button at any time to view the Builder generated NinjaScript code.
3. Press the **Unlock Code** button at any time to open the NinjaScript editor and edit your strategy code.

Once the code is unlocked, you can no longer use the Builder for subsequent strategy editing

4. Press the **Compile** button at any time to compile your strategy code.
5. Press the **<Back** or **Next>** buttons to move back or forth between Builder screens - you can also directly jump to a specific screen by using the left side navigation menu.
6. Press the **Cancel** button to leave the **Strategy Builder**

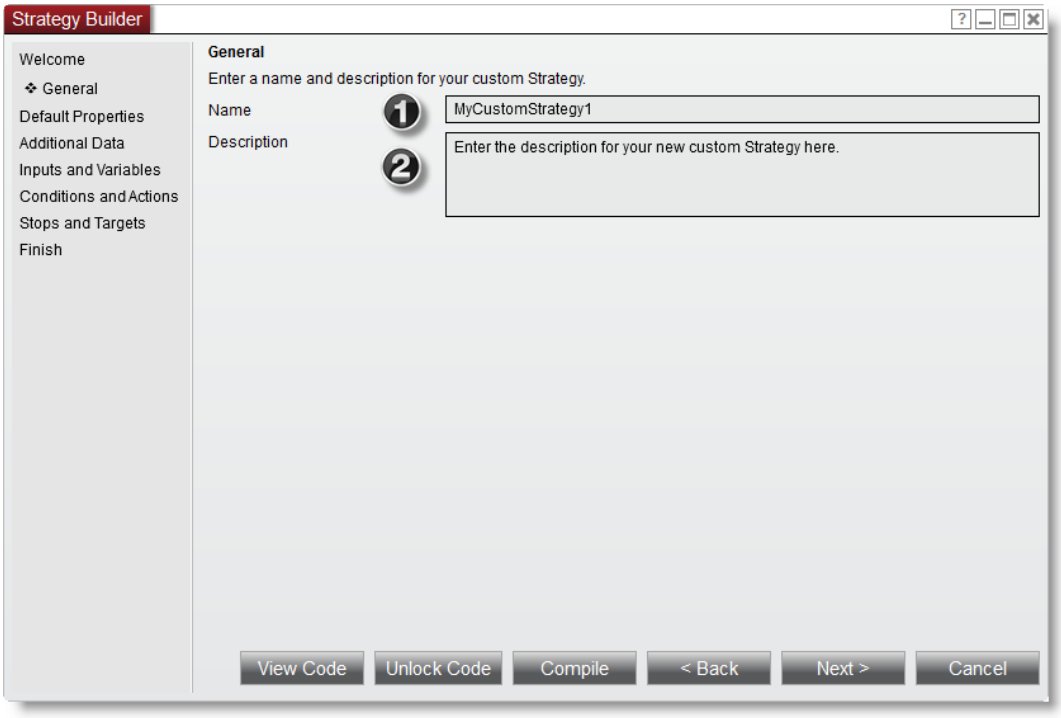
Note: Should you want to make a copy of your strategy, you can select your saved script in the **Strategy** drop-down and select '**save as**' - this opens a file dialog, where you can enter a new name to save the script copy under.



Understanding the General screen

General Screen Layout

The **General** screen is where you enter the name and description of your strategy.

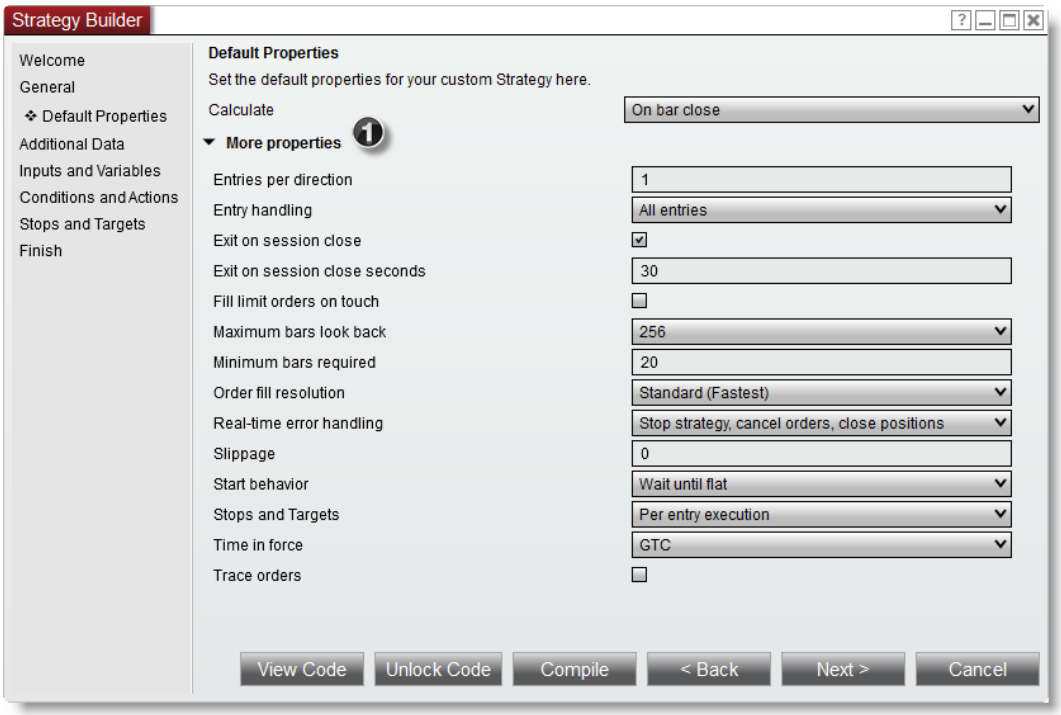


1. Sets the name of the strategy
2. Sets the description of the strategy

Understanding the Default properties screen

Default properties screen Layout

The Default properties screen is where you can set the default values for your custom strategy properties.



1. Per default only the **Calculate** section is visible, click the **More properties** to expand the selection to include all strategy default properties as well to set for your Builder script.

Calculate	Sets the **Calculation Mode** for the strategy. Possible values are "On Each Tick," "On Price Change," or "On Bar Close"
Entries per direction	Sets the maximum number of entries allowed per direction while a position is active based on the "Entry handling" property
Entry handling	Sets the manner in which entry orders are handled. If set to "AllEntries", the strategy will process all entry orders until the maximum allowable entries set by the "Entries per direction" property have been reached while in an open position. If set to "UniqueEntries", the

	strategy will process entry orders until the maximum allowable entries set by the "Entries per direction" property per each uniquely named entry have been reached.
Exit on close	When enabled, open positions will be closed on the last bar of a session
Exit on session close seconds	Sets the number of seconds prior to the end of a session at to close any open positions held by the strategy
Fill Limit Orders on Touch	Enables the filling of limit orders when touched for the historical portion of the chart
Maximum Bars Look Back	Sets the maximum number of historical bars to use for strategy calculations. The TwoHundredFiftySix setting is the most memory friendly
Minimum Bars Required	Sets the minimum number of historical bars required to start taking trades
Order Fill Resolution	Sets the way that simulated historical orders will be processed by the strategy. See the Understanding Historical Fill Processing page for more information.
Real-time error handling	Defines the behavior of a strategy when a strategy generated order is returning in a "Rejected" state. See the Real-time Error Handling page for more information.
Slippage	Sets the slippage amount in ticks for the historical portion of the chart

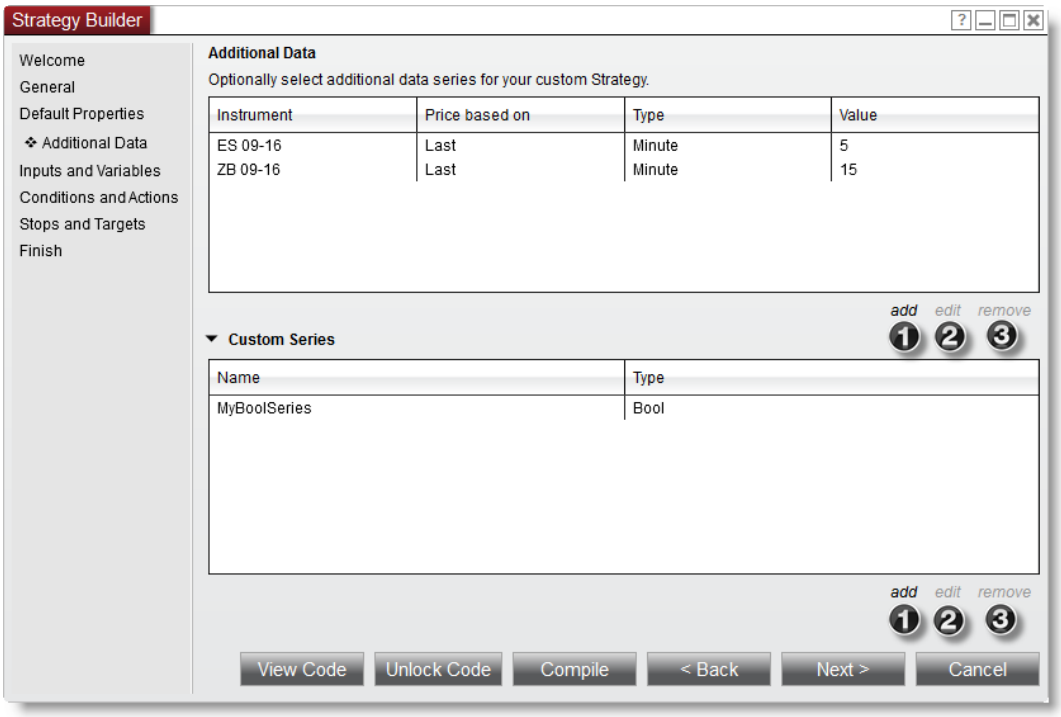
Start Behavior	Sets the starting behavior of the strategy, based upon the account position. See the Syncing Account Positions page for more information.
Stops and Targets	Sets how stop and target orders are submitted
Time in force	Sets the order's time in force. Possible values are DAY and GTC
Trace orders	Enables sending more detailed order debug info to the NinjaScript output window

Understanding the Additional data screen

Additional data screen Layout

The Additional data screen is where you can optionally select additional instrument data or custom series for your strategy.

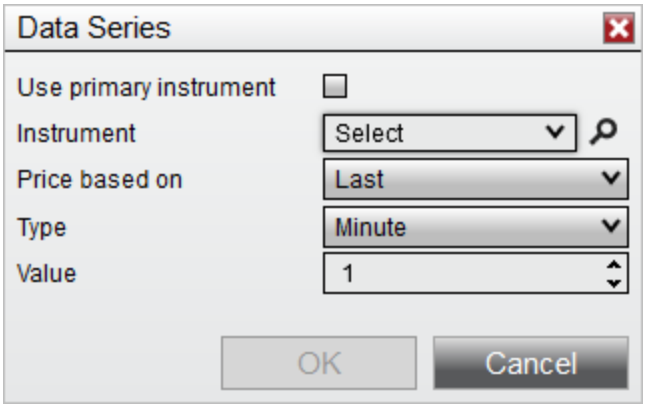
Critical: You will want to make sure to add any additional series in this section that a hosted / called MultiSeries indicator in your Builder script would use, such as for example the Pivots, Camarilla Pivots or Fibonacci Pivots indicators.



1. Press the **add** button to be able to configure a new series to add
2. Press the **edit** button to be able to configure an existing series
3. Press the **remove** button to be able to remove an existing series

Data Series Selector Layout

Select your instrument data series to add here

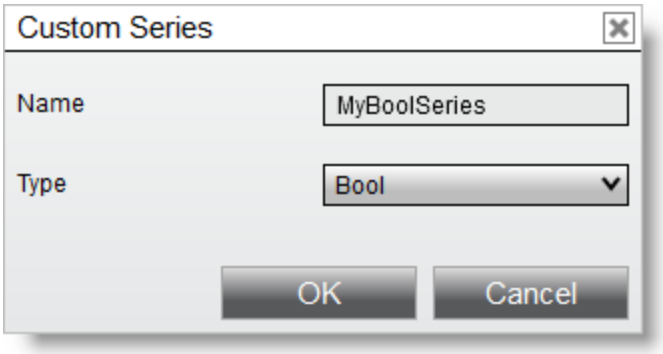


Use primary instrument	Checking this will use the primary instrument the strategy is applied to

Instrument	Select your instrument from the favorite or list selector or by using the search feature (press the magnifying glass)
Price based on	Selects the price type the data series is based on, possible values are Last, Bid, Ask
Type	Selects the bars type your series will use, possible values for the Builder interface are - • Tick • Minute • Day • Week • Month • Year • Volume • Range • Second
Value	Sets the bars period type value for your series

Custom Series Selector Layout

Select your custom series to add here



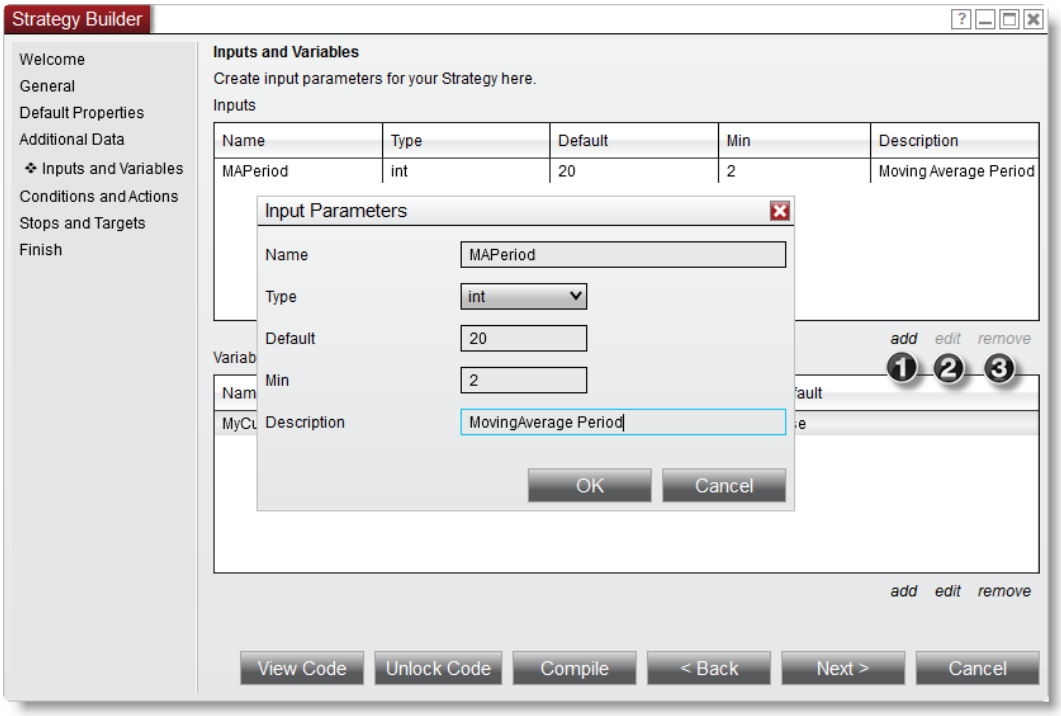
Name	Set the name for your custom series
Type	Selects the data type of the custom series, possible values for the Builder interface are -

	• Bool • Double • DateTime • Int • String

Understanding the Inputs and Variables screen

Inputs and Variables screen Layout

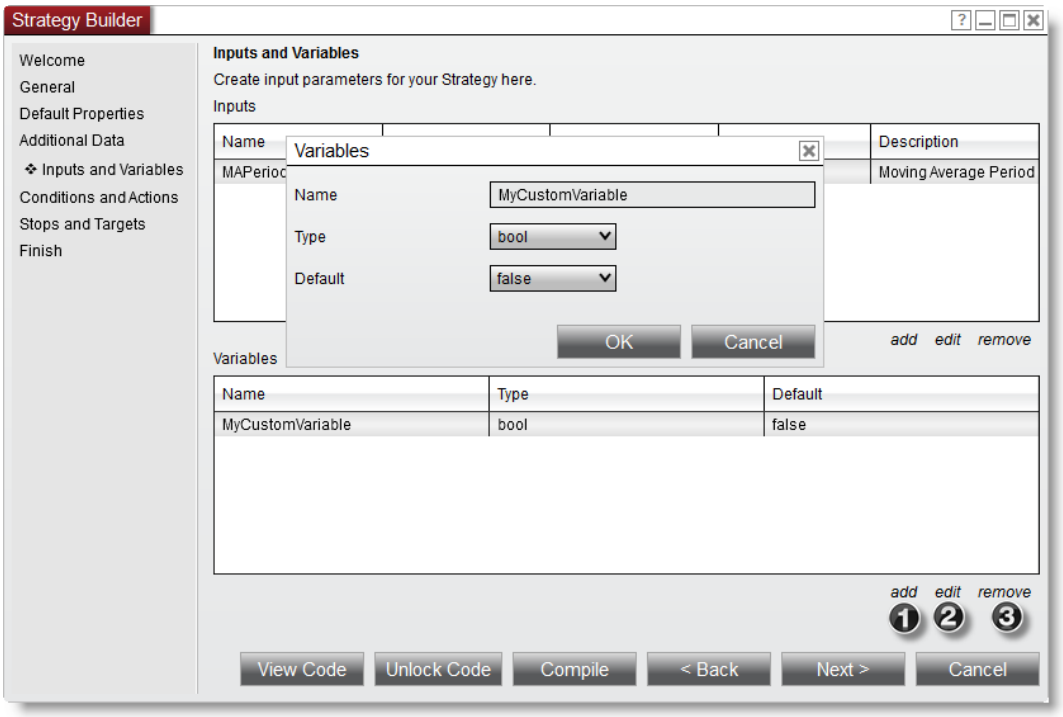
The Inputs and Variables screen allows you to define the user inputs of your strategy. User inputs are important if you require input values that may vary the performance of your strategy. If for example you have a simple moving average cross over system, you may want to create an input for the fast moving average and another for the slow moving average. This then allows you to change the values of the moving averages at run time from the UI. Inputs are also required if you plan to use the NinjaTrader Strategy Analyzer's optimization capabilities.



1. Press the **add** button to add a new user input.

2. Press the **edit** button to edit an existing, selected user input.
3. Press the **remove** button to remove the selected user input.

Name	Set the name for your user input
Type	Selects the data type of the user input, possible values for the Builder interface are - • Bool • Double • String • Int • Time
Default	Set the default value your user input will have
Min	Set the minimum value your user input will have
Description	Enter an optional description for your user input here



1. Press the **add** button to add a new user variable.
2. Press the **edit** button to edit an existing, selected user variable.
3. Press the **remove** button to remove the selected user variable.

Name	Set the name for your user variable
Type	Selects the data type of the user variable, possible values for the Builder interface are - • Bool • Double • String • Int • Time
Default	Set the default value your user variable will have

Note: If an input is named the same as the generated code for an indicator, the

strategy will not be able to successfully compile.

Understanding the Conditions and Actions screen

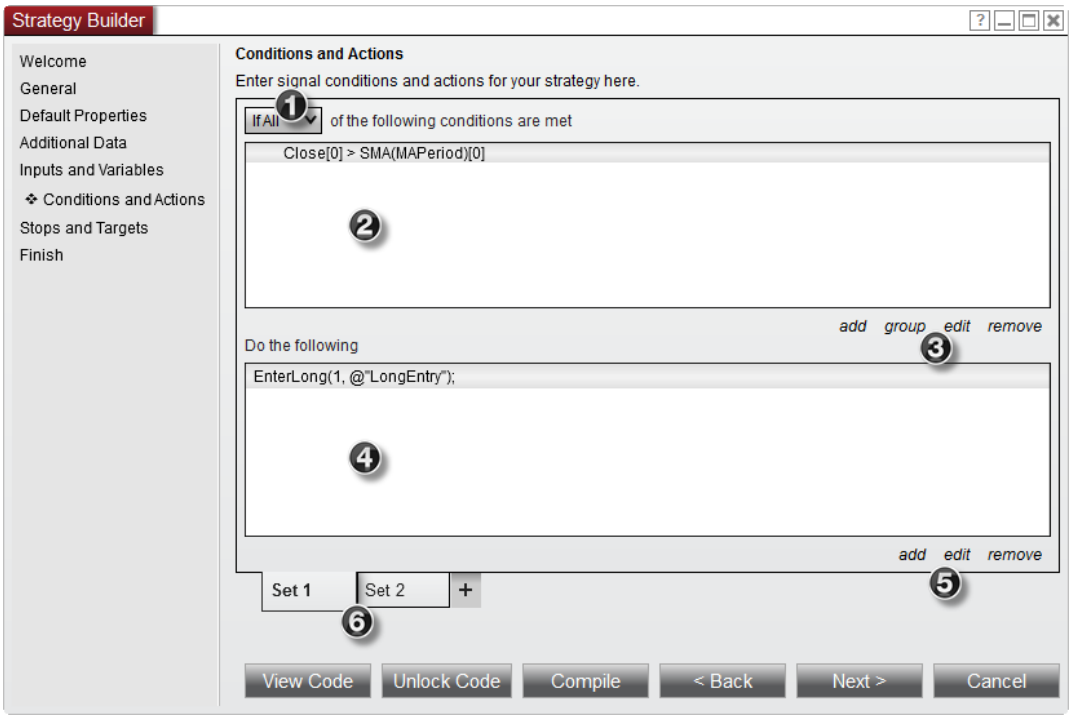
Conditions and Actions screen Layout

The Conditions and Actions screen allows you to set conditions and subsequent actions that control the flow of your strategy.

Conditions - Take the specified action when true
Actions - Execute an action (submit orders, draw objects on the chart etc ...) based on its parent condition evaluating to true

Via the Builder, you can have an unlimited set of conditions with related actions and you also group conditions into a condition group (for example for a certain set of filter rules like time)

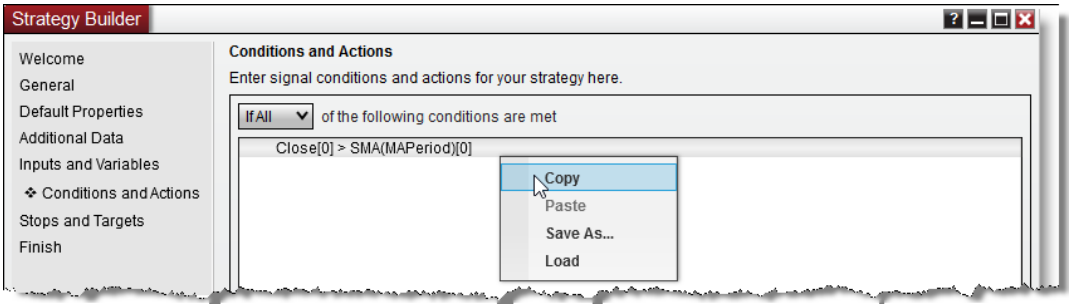
Conditions and condition groups are created using the Condition Builder. Actions are specified by the Strategy Actions window.



1. Selects **if all** of the individual conditions have to be met in order to trigger an action, or **if any** will be sufficient.
2. Displays the conditions associated with the currently selected condition set
3. Adds, opens condition grouping(*), edits or removes a condition (a double click on selected item will also allow editing)
4. Displays the actions associated with the currently selected condition set
5. Adds, edits or removes an action (a double click on selected item will also allow editing)
6. Selects the condition set you wish to edit

* For an example on working condition groups, please see "How to create a Time Filter' in the Condition Builder section

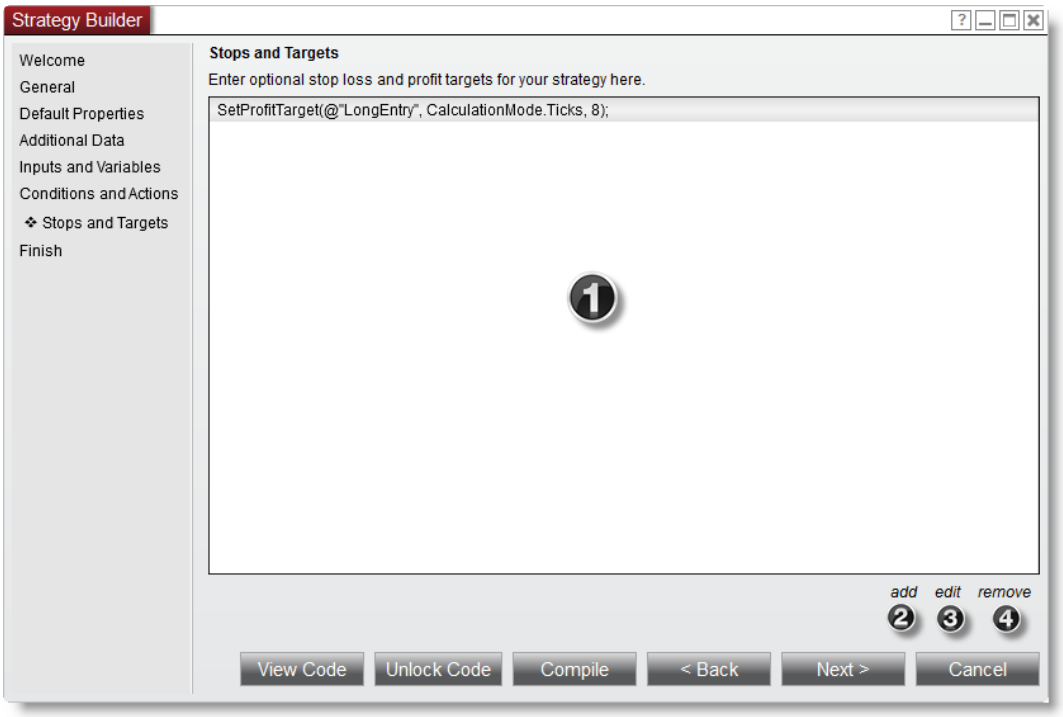
You can copy and paste conditions from one set to another and you can even save a condition set as a template and load for future use via the right mouse button click context menu as show in the image below. To save a condition set as a template, select the **Save As...** menu item and then to re-use it in another strategy or condition set at a later time, select the **Load...** menu item.



Understanding the Stops and Targets screen

Stops and Targets screen Layout

The Stops and Targets allows you to set stop loss, trail stop, parabolic stop (R15 and higher) and profit target orders that are automatically submitted and managed once your strategy opens a position.

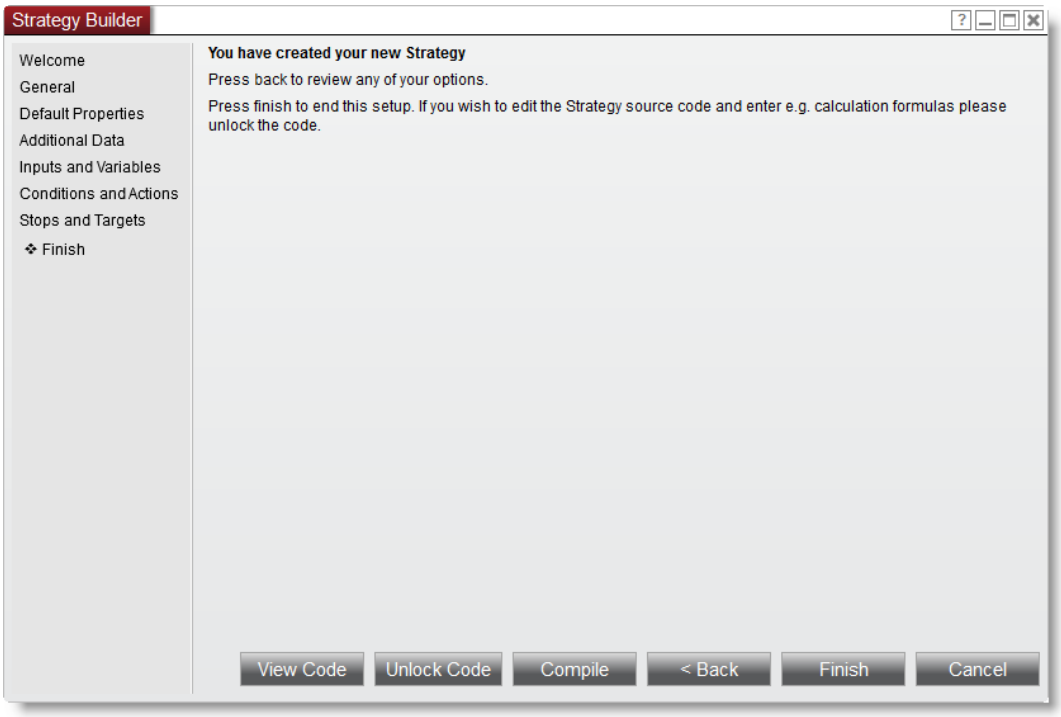


1. Displays stops and targets associated with your strategy
2. Adds a stop or target to your strategy
3. Edits the selected stop or target in your strategy (a double click on the selected item will also allow editing)
4. Removes the selected stop or target from your strategy

Understanding the Finish screen

Finish screen Layout

Once you reach this screen you are finished with developing your strategy. Press the **Finish** button to compile your strategy which will then be ready for backtesting or live execution.



10.28.2 Condition Builder

The **Condition Builder** is a very powerful feature that allows you to define complex conditions for your automated trading systems without having to know how to program.

Understanding the Condition Builder

Condition Builder

Most if not all automated trading system code wizards are limited in scope in that they provide canned predefined expressions and only allow you to change a few parameters on those expressions. The NinjaTrader **Condition Builder** is advanced in that you can develop powerful expressions without limitations. Due to its power and flexibility, it is extremely important that you read through and understand its capabilities.

The **Condition Builder** is also a very powerful aid for those of you learning NinjaScript or learning how to program. You can build your conditions within the **Condition Builder** and instantly see NinjaScript code generated by having the NinjaScript Editor open (by pressing the **View Code...** button in the Builder screen).

The **Condition Builder** can be accessed via the Conditions and Actions screen in the NinjaTrader Strategy Builder.

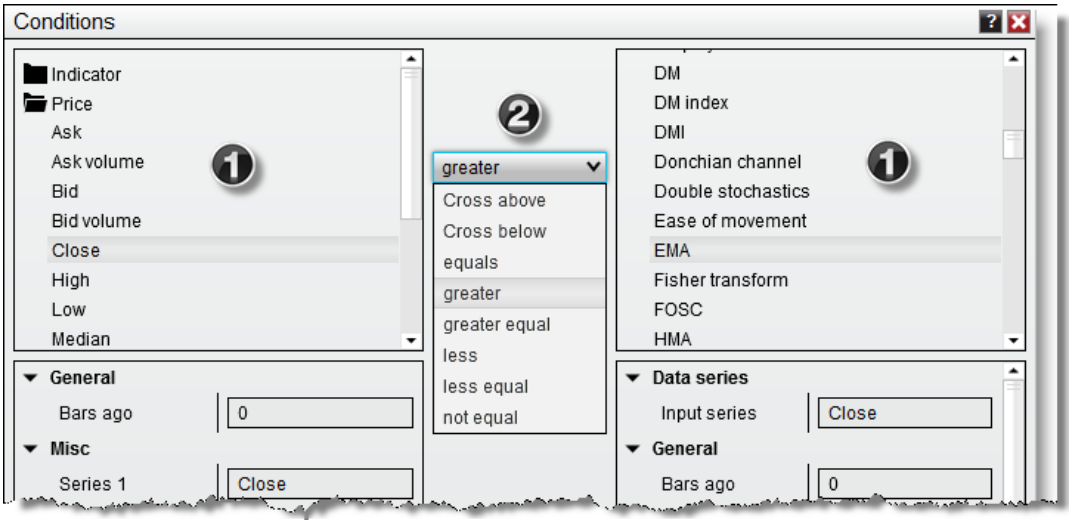
Basic Operation

The general concept of the **Condition Builder** is to generate a Boolean expression also known as comparison expressions or conditional expressions. What does that mean? It is simply an expression that results in a value of either TRUE or FALSE. For example, the expression

2 < 7 (2 is less than 7)

is a Boolean expression because the result is TRUE. All expressions that contain relational operators are Boolean. Boolean expressions or "Conditions" as they are known in NinjaTrader is used to determine when to take a specified action such as submitting an order or drawing on the chart.

Looking at the image below, you can instantly see that the **Condition Builder** is set up like a Boolean expression. Select an item from the left window (1), compare it to a selected item in the right window (1) and then select the relational operator (2).



1. Available items such as indicators, price data, etc. to use for the comparison
2. List of relational operators

Relational operator invalid comparisons

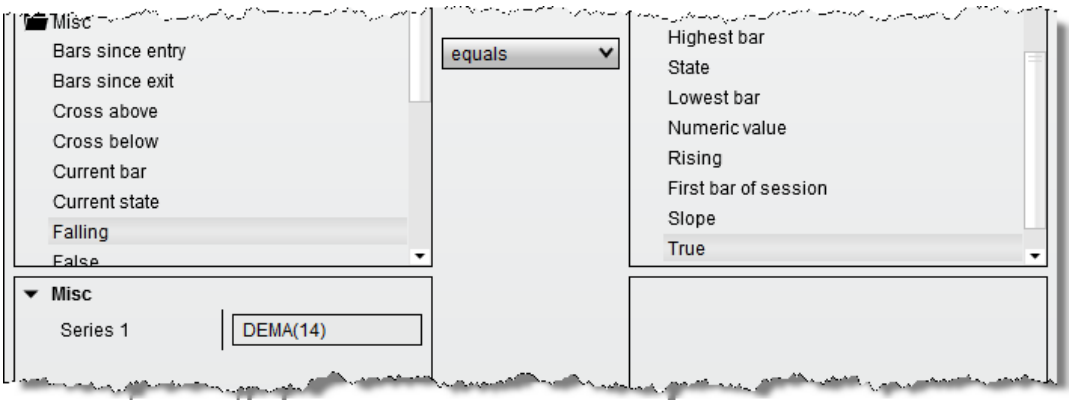
Since the relational operator will let you select any items from the left to compare to the right in the Condition Builder, you need to be mindful what you attempt comparing. For example comparing a price based value like the **DEMA** indicator

value to the Misc category **Falling** would not be possible, and prompt the Condition Builder to issue an error like shown below -

"Type of left expression and right expression do not match, please select similar expressions"

To work around, you would need to select expressions with a similar return value that would allow for a programmatic comparison. In the example used above, the **DEMA** indicator provides a double value in return that is attempted to be compared to a boolean (true / false) value, which **Falling** would return.

The correct approach is shown below, the **DEMA** indicator would be passed into **Falling** as input series and then the return value could be compared to **True** from the Misc category to create a successful condition.

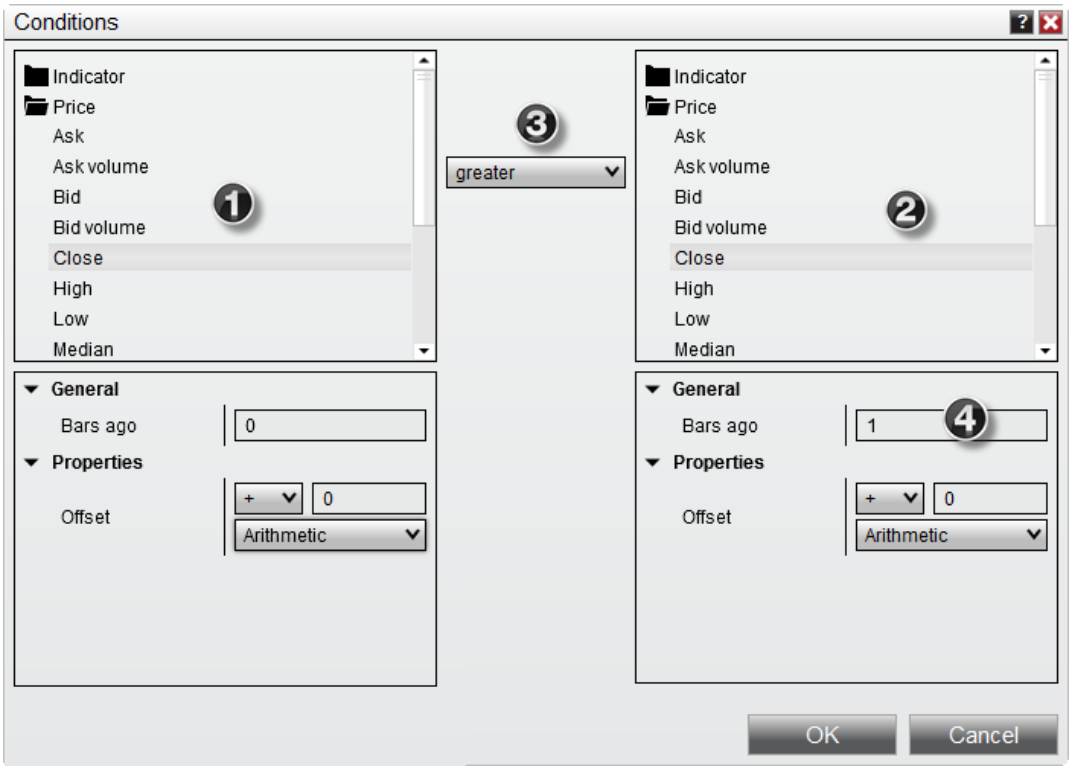


How to make price data comparisons

Price Data Comparisons

You can compare a bar's price data such as checking for a higher close. The following is an example and represents one of many possible combinations.

1. Expand the **Price** category on the left side and select the **Close**.
2. Expand the **Price** category on the right side and select the **Close**.
3. Select the **greater** relational operator
4. Set the **Bars ago** parameter to a value of "1"



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current closing price is greater than the closing price of 1 bar ago"

How to offset an item value

Offsetting an Item Value

You can offset the value of most items available in the **Condition Builder**. An offset is a value that is added, subtracted, multiplied or divided from / into the actual item's value. When an item is selected such as an indicator or price data,

the **Offset** and **Offset type** parameters become visible in the window directly below the item selected. This is shown as numbers 5 and 6 in the image below.

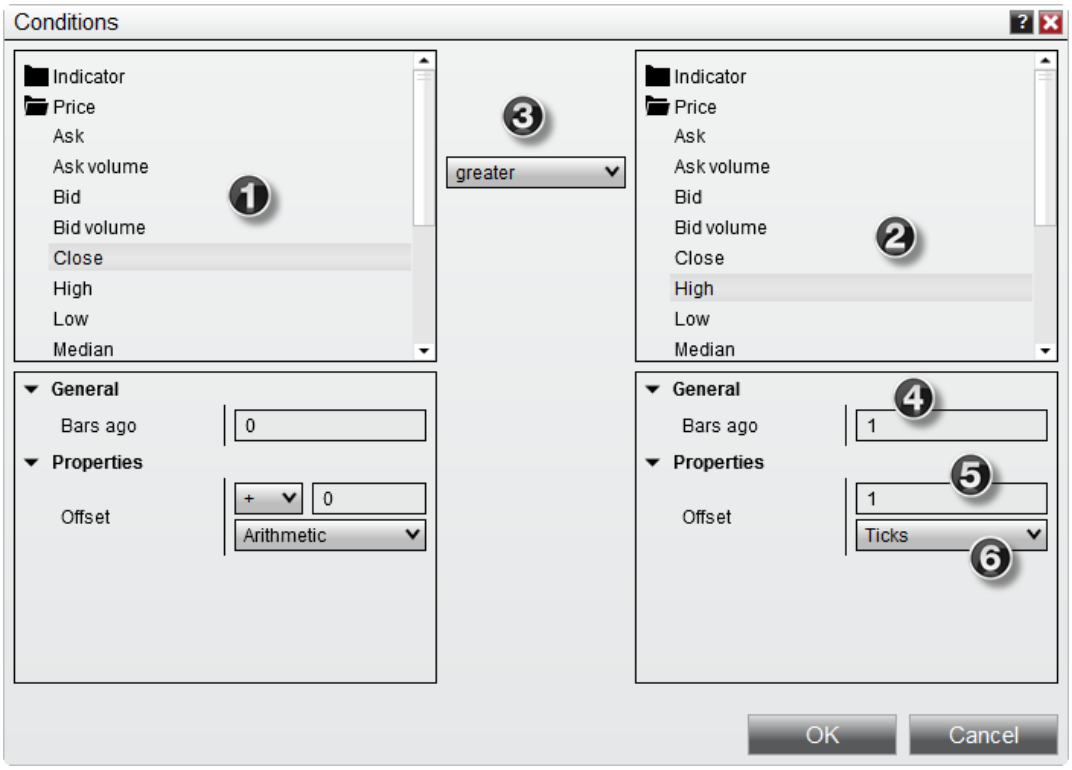
Offset type can be set to:

Arithmetic	Offsets by an arithmetic equation you can setup by the absolute value and the arithmetic offset operator to the left (+ - * /)
Pips	Offsets by the specified amount of pips
Percent	Offsets a percentage value of the item's value. A value of 1 is equal to 100% where a value of 0.1 is equal to 10%.
Ticks	Offsets by the specified amount of ticks

Once the **Offset type** is selected, you must set the value **Offset**. In addition to the example below, you can see the "*Checking for Volume Expansion*" section below for another example that uses the **Percent Offset type**.

The following is an example and represents one of many possible combinations:

1. Expand the **Price** category and select the **Close**
2. Expand the **Price** category and select the **High**
3. Select the **greater** relational operator
4. Set the **Bars ago** parameter to a value of "1"
5. Set the **Offset type** parameter to **Ticks**
6. Set the **Offset** parameter to a value of "1"



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current closing price is greater than the high price of 1 bar ago + 1 tick"

How to make indicator to value comparisons

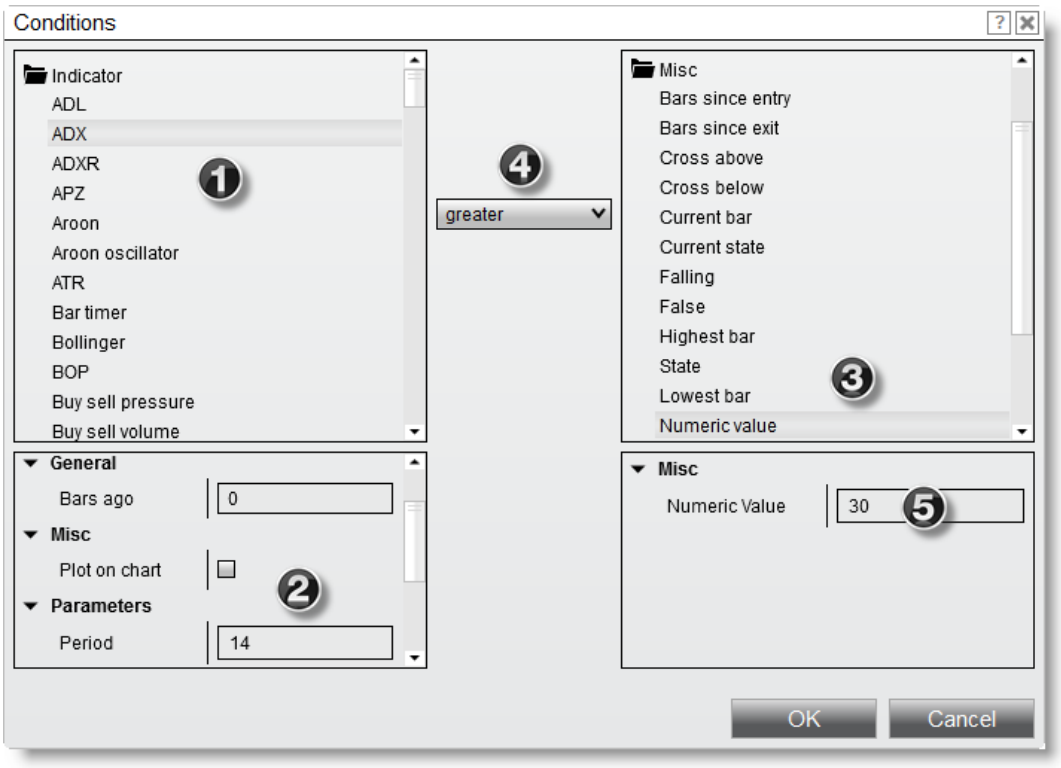
Indicator to Value comparisons

You can compare an indicator's value to a numeric value. This can come in handy if you wanted to check if ADX is over a value of 30 (trending) or if Stochastics is under a value of 20 (oversold) or any other conditions you can think of.

The following is an example and represents one of many possible combinations:

1. Expand the **Indicator** category and select the **ADX** indicator
2. Set the parameters of the indicator, for our example with the default values no changes are needed
3. Expand the **Misc** category and select **Numeric value**
4. Select the **greater** relational operator

5. Enter the numeric value you want to compare the indicator to (30 in our example)



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current value of a 14 period ADX is greater than 30"

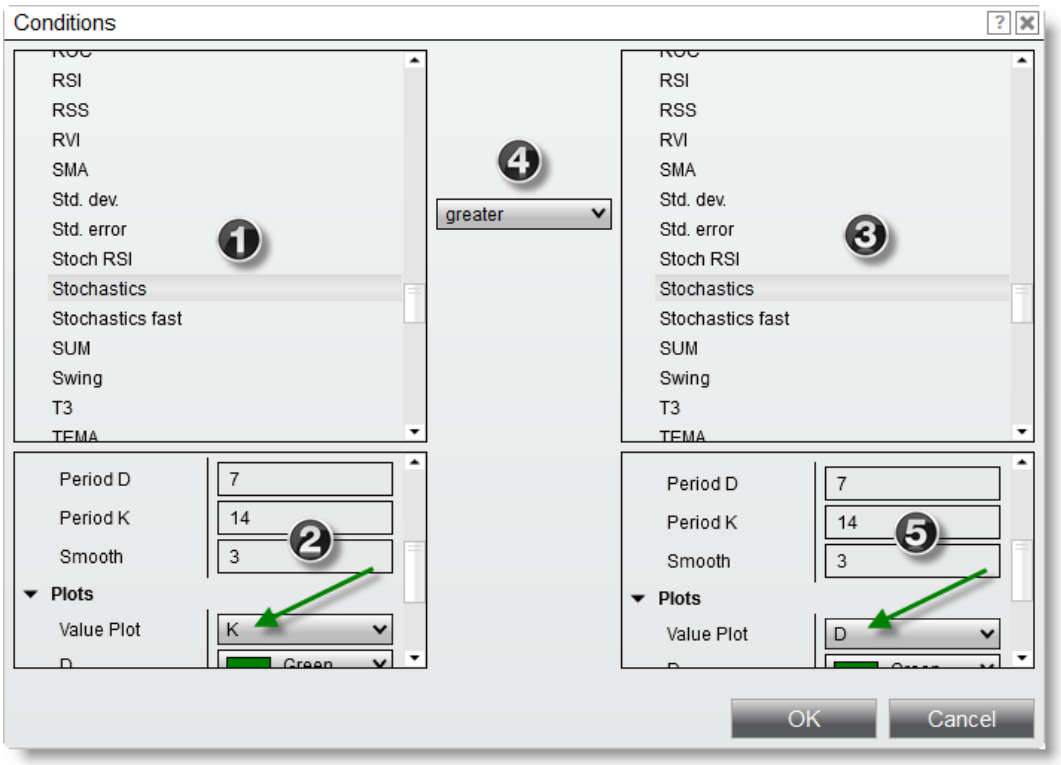
How to compare plot values of multi-plot indicators

Comparing Plot Values of Multi-Plot indicators

You can compare plots in the same indicator or select any individual plot within an indicator to create a condition.

The following is an example and represents one of many possible combinations:

1. Expand the **Indicator** category and select the **Stochastics** indicator
2. Set the indicator input parameters and select the **K** plot (green arrow)
3. Expand the **Indicator** category and select the **Stochastics** indicator
4. Select the **greater** relational operator
5. Set the indicator input parameters and select the **D** plot (green arrow)



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current K plot value of a Stochastics indicator is greater than the current D plot value of the same Stochastics indicator"

How to use user inputs & variables

User Inputs & Variables

User inputs are simply variables that can be used in place of absolute values. They increase the flexibility of your strategy since you can substitute a variable for the period parameter of a simple moving average instead of provide an absolute value.

SMA(9) is how you express a 9 period simple moving average in NinjaScript. If you run a strategy, you would always be using a 9 period simple moving average. At run time, you might want to change this value to 10. User defined inputs accomplish this. If you created an input named "MyInput", you could express the simple moving average as SMA(MyInput). At run time, you can then configure your

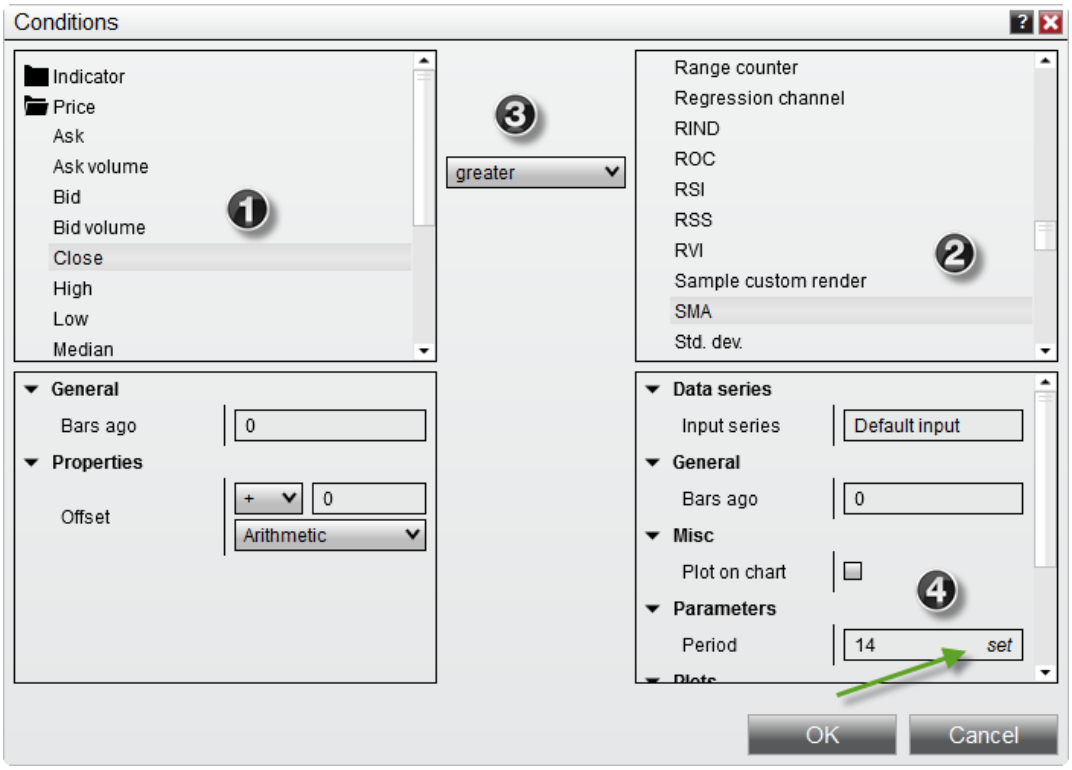
strategy by setting the value of "MyInput" to whatever value you like. In addition, user inputs are required when optimizing a strategy.

User variables (not to be confused with inputs) behave in the same manner with the exception that they can not be configured when starting a strategy but can only be set programmatically during run time.

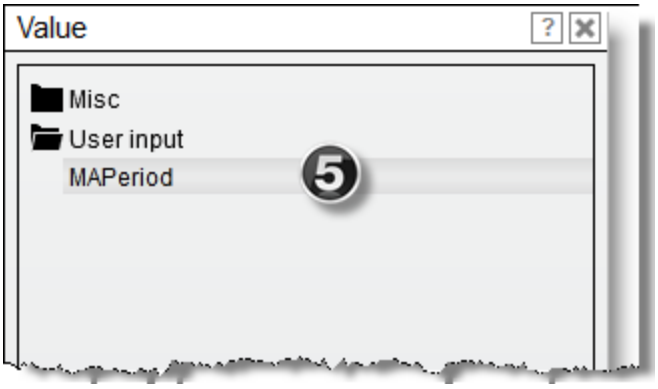
- User inputs are created from the Builder screen
- User variables can be set in the strategy logic through the **Condition Builder** (see the sections above)

The following is an example and represents one of many possible combinations, the example demonstrates the use of a user input however the sample approach applies to user variables.

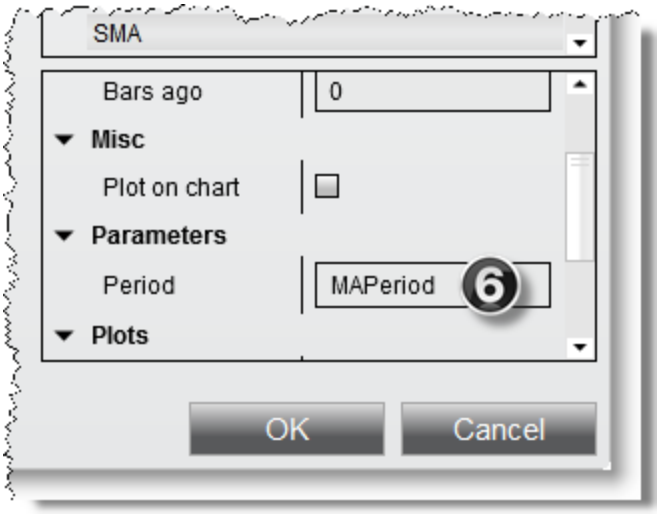
1. Expand the **Price** category and select the **Close**.
2. Expand the **Indicator** category and select the **SMA** indicator
3. Select the **greater** relational operator
4. Set the **Period** parameter to a user defined input by pressing the "**Set**" button (green arrow) to open the **Value** window



5. Expand the **User input** category and select the value **MAPeriod** and press the **OK** button



6. The Condition Builder will now look as per the image below with the user input "MAPeriod" assigned to the parameter Period. When you apply this strategy to a chart, you will be able to set the value for the user input directly from the UI which will then be used to drive the SMA indicator.



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current closing price is greater than the user defined Period simple moving average"

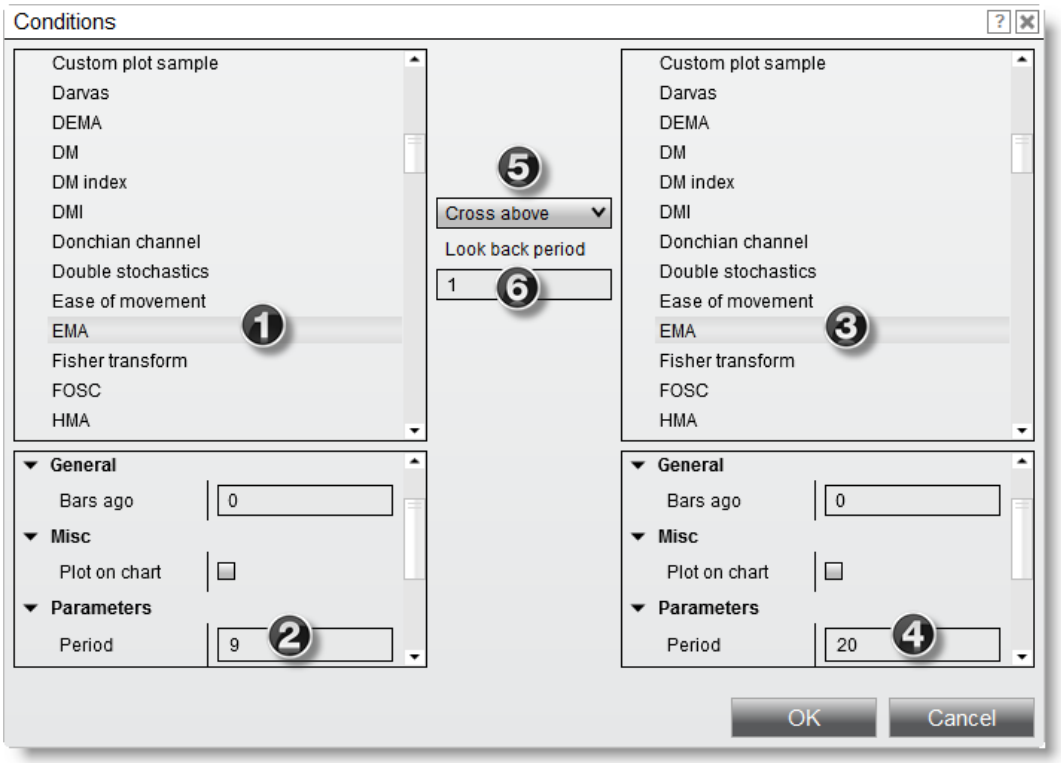
How to create a cross over condition

Cross Over Conditions

You can check for either a **CrossAbove** or **CrossBelow** condition with a user defined look back period. The look back period sets the number of bars to look back to check for the cross over condition.

The following is an example and represents one of many possible combinations.

1. Expand the **Indicator** category and select the **EMA** indicator
2. Set the **Period** parameter to the desired value ("9" is used in this example)
3. Expand the **Indicator** category and select the **EMA** indicator
4. Set the **Period** parameter to the desired value ("20" is used in this example)
5. Select **CrossAbove** relational operator
6. Set the **Look back period**



Once the **OK** button is pressed, a condition is created that would translate to the following:

"9 period exponential moving average crosses above the 20 period exponential moving average in the last bar"

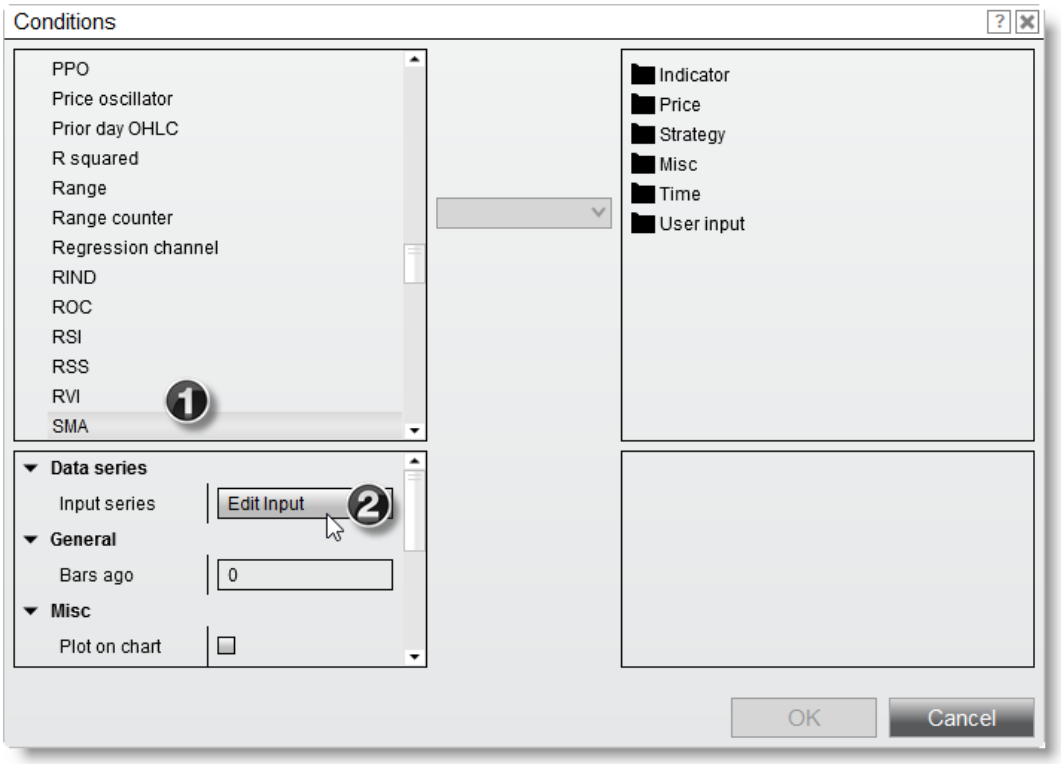
How to use indicator inputs in other indicators

Indicator on Indicator

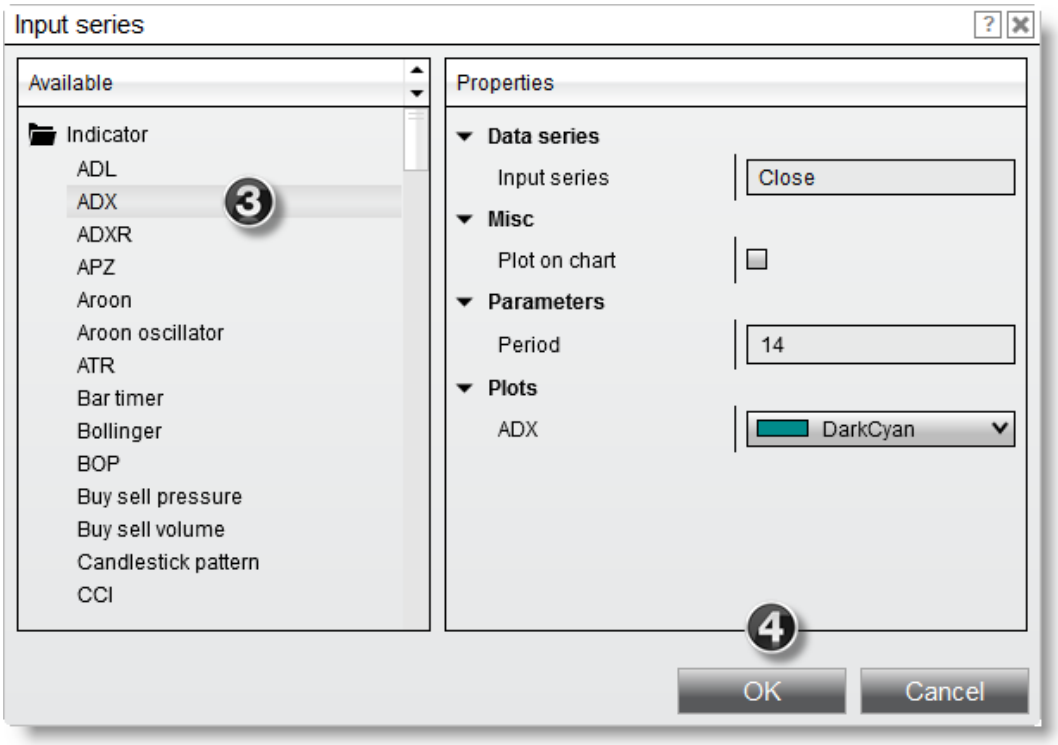
You can use indicators as input for other indicators ... actually, you can nest indicators within indicators infinitely if you really wanted to!

The following example is an example of applying a simple moving average (**SMA**) to a 14 period **ADX** indicator and is one of many possible combinations.

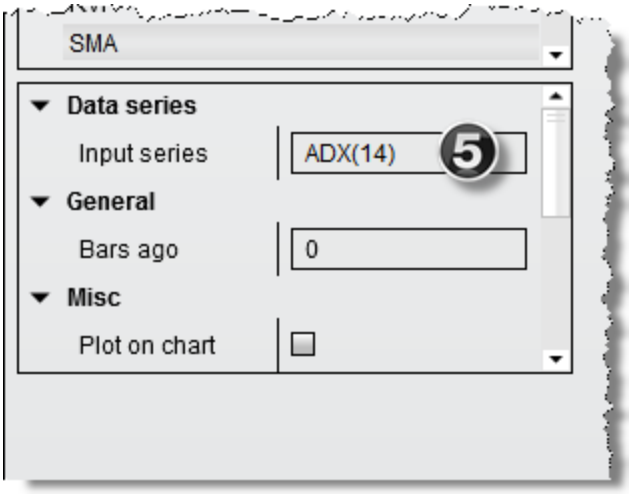
1. Expand the **Indicator** category and select the **SMA** indicator
2. Set Input series to the **ADX** indicator by pressing the "**Edit Input**" button to open the **Value** window
3. Select the **ADX** indicator and set any properties in the **Parameters** window



3. Select the **ADX** indicator and set any properties in the **Properties** window
4. Press the **OK** button



5. Once you have pressed the **OK** button, you will notice on the left lower window, the "Input series" parameters has now been set to the **ADX**(14) which is the 14 period **ADX** indicator.



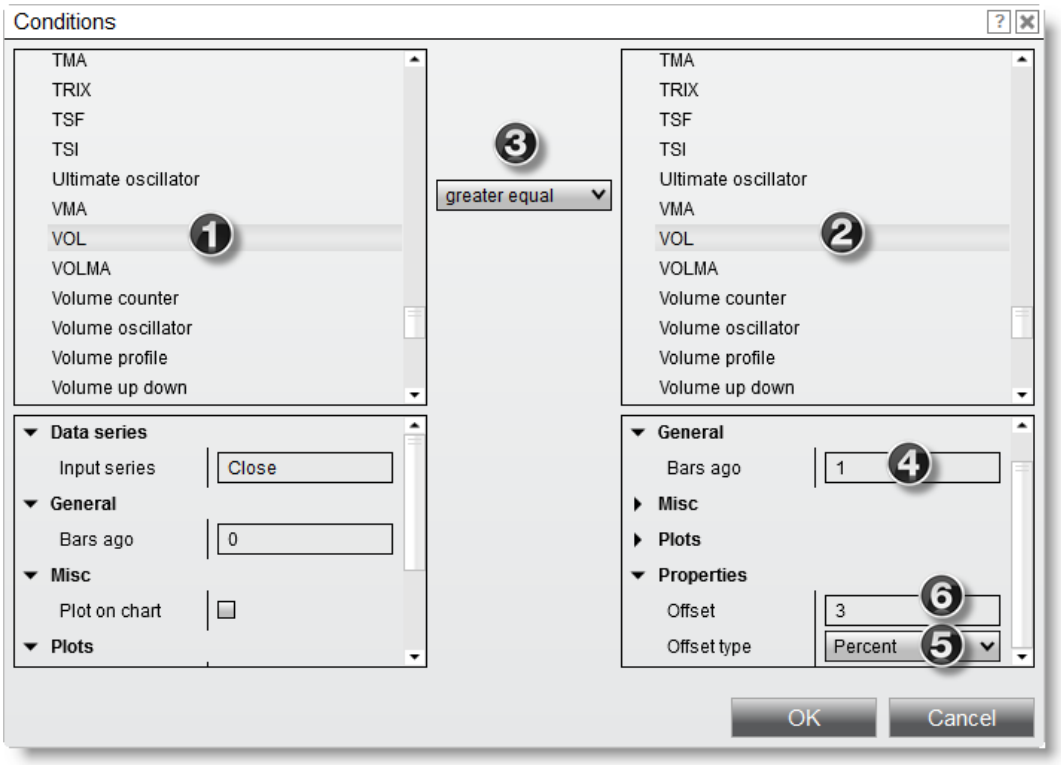
How to check for volume expansion

Checking for Volume Expansion

You can compare if the current bar's volume is greater than the prior bar's volume plus an offset amount.

The following is an example and represents one of many possible combinations.

1. Expand the **Indicator** category and select the **VOL** indicator
2. Expand the **Indicator** category and select the **VOL** indicator
3. Select the **greater than or equal** relational operator
4. Set the **Bars ago** parameter to a value of "1"
5. Set **Offset type** parameter to **Percent**
6. Set the **Offset** parameter to a value of "3" - *3 equals 300% percent here, i.e. 10% would be 0.1*



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current value of Volume is greater than or equal to the value of Volume of 1 bar ago + 300%"

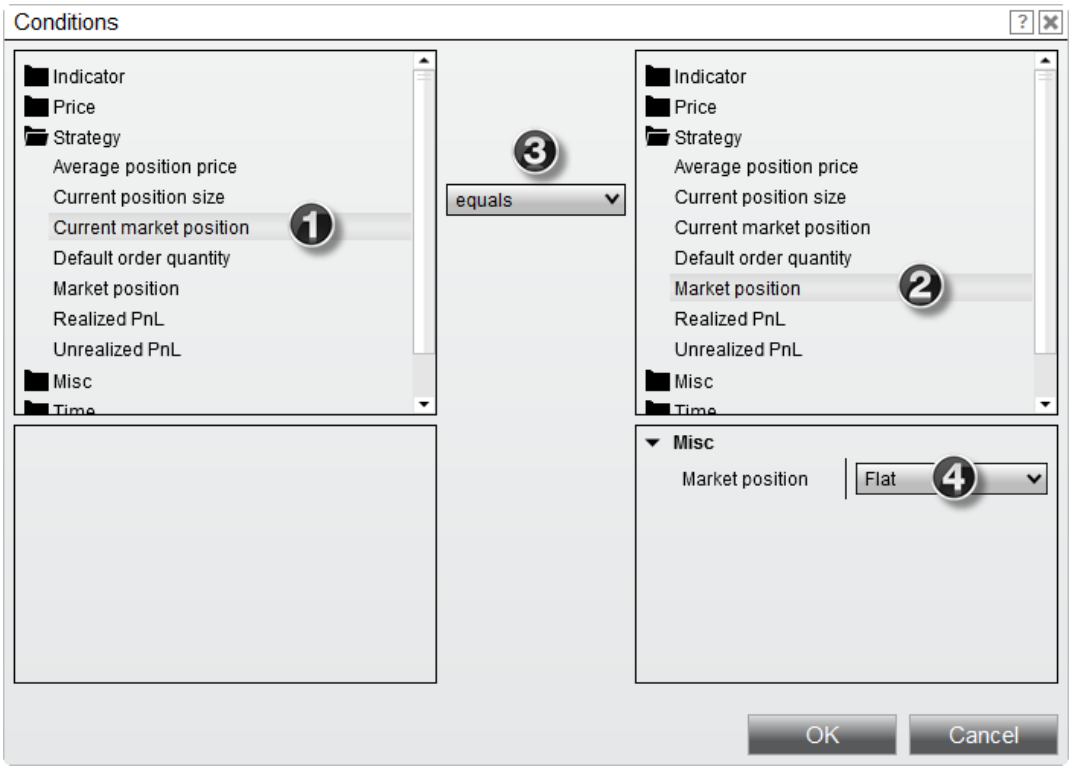
How to create market position comparisons

Creating Market Position Comparisons

You can compare strategy state information such as but not limited to current market position or current position size.

The following is an example and represents one of many possible combinations.

1. Expand the **Strategy** category and select **Current market position**.
2. Expand the **Strategy** category and select **Market position**
3. Select the **equals to** relational operator
4. Select **Flat** from the Market position drop-down under Misc



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current market position equals flat"

How to create time comparisons

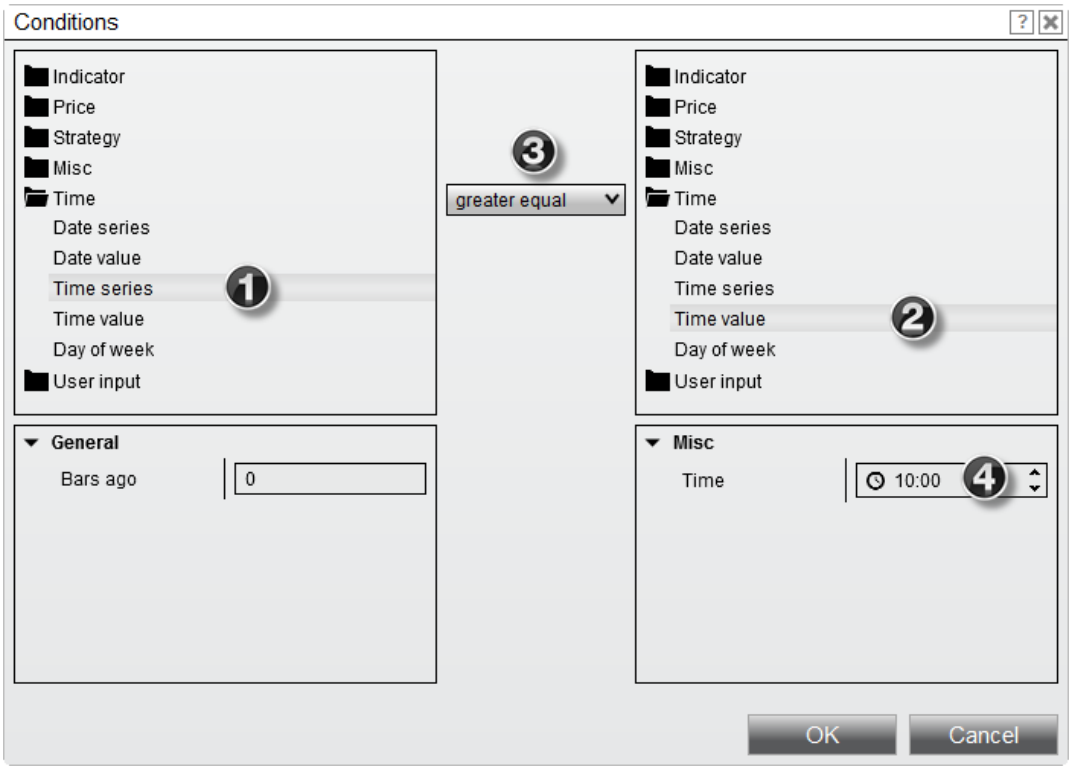
Creating Time Comparisons

You can compare a bar's time data to a user defined time or date value.

The following is an example and represents one of many possible combinations.

Note: Time series represents a collection of bar Date / Time values of a bar series

1. Expand the **Time** category and select **Time series**
2. Expand the **Time** category and select **Time series**
3. Select the **greater than or equal** relational operator
4. Set the **Time** parameter to a user defined value of "10:00"



Once the **OK** button is pressed, a condition is created that would translate to the following:

"Current bar's time is greater or equal to 10:00 AM"

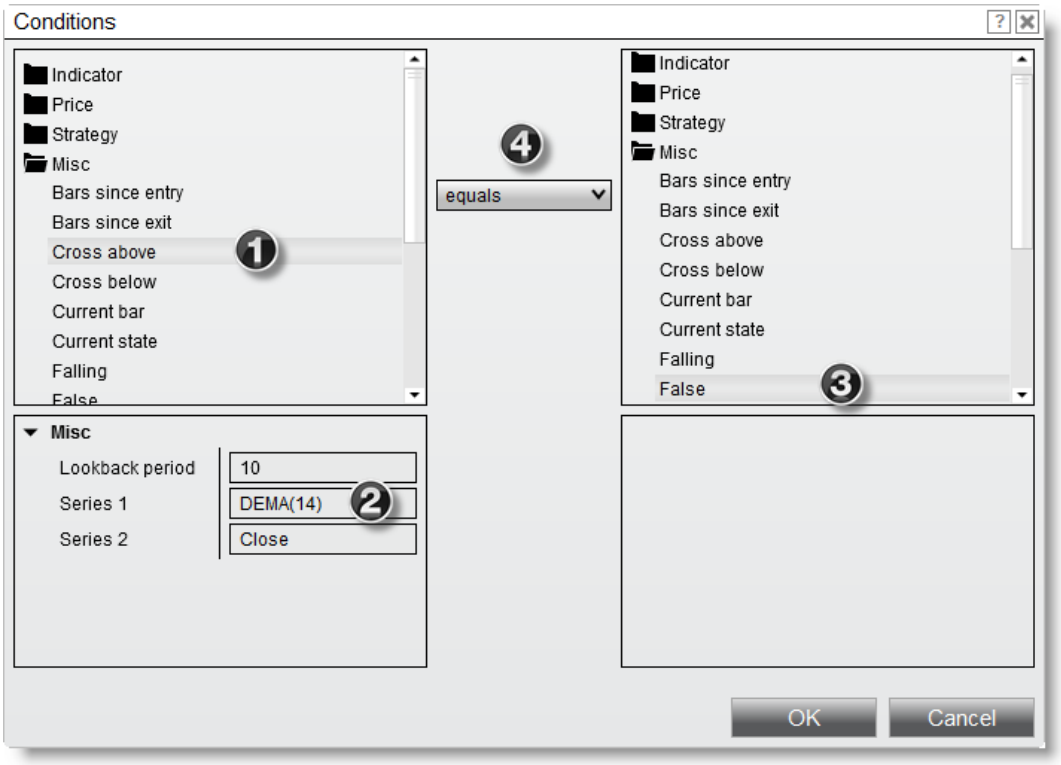
How to negate a condition

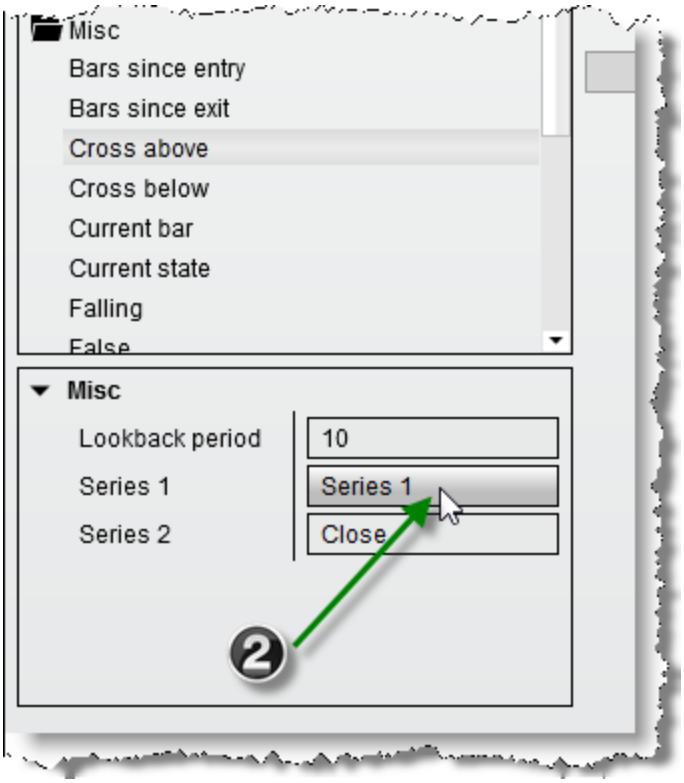
Negating a Condition

You can also negate a condition, so allowing for example to have a certain filter or technical indicator setup being the opposite and evaluate to false.

The following is an example and represents one of many possible combinations.

1. Expand the **Misc** category and select the **Cross above**
2. Click the **Series 1** input field and select the **DEMA** indicator as series for the cross comparison to use
3. Expand the **Misc** category and select the **False**
4. Select the equals relational operator





Once the **OK** button is pressed, a condition is created that would translate to the following:

"The DEMA(14) indicator has not been crossed by the Close price within the last 10 bars"

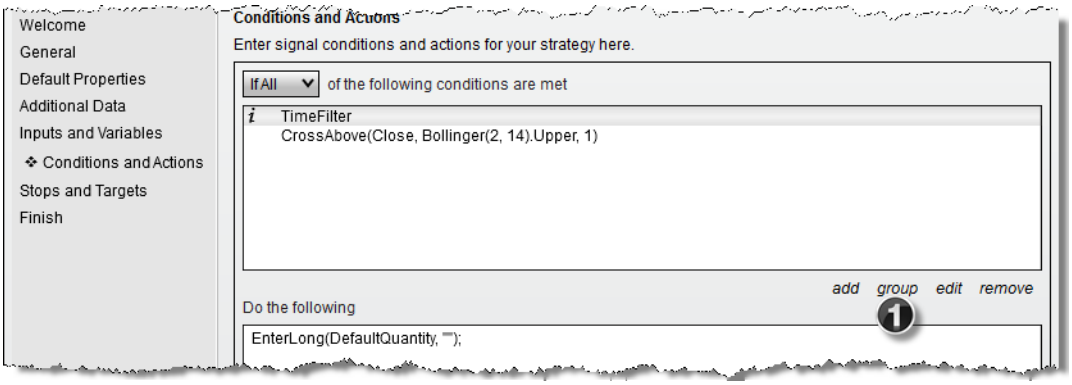
How to create time filters

Creating a Time Filter

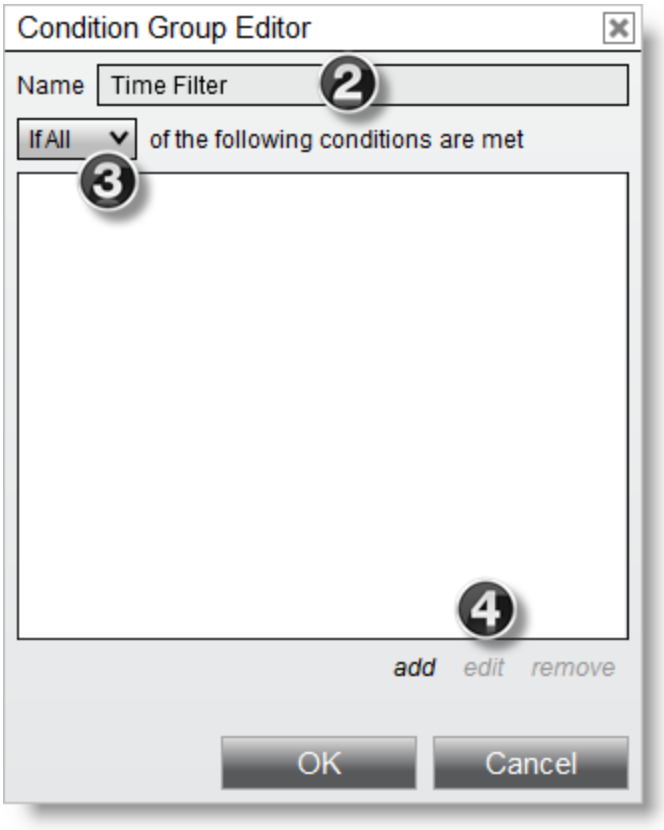
Time filters can be a useful tool of your custom strategy to help make its trades more efficient and devise a way to test for various parts of the trading session. The Condition Group Editor is ideally suited to set those up for your Strategy Builder scripts.

The following is an example and represents one of many possible combinations (as well as the actual time filter times below) :

1. Press the **group** icon on the Conditions and Actions screen to open the Condition Group Editor

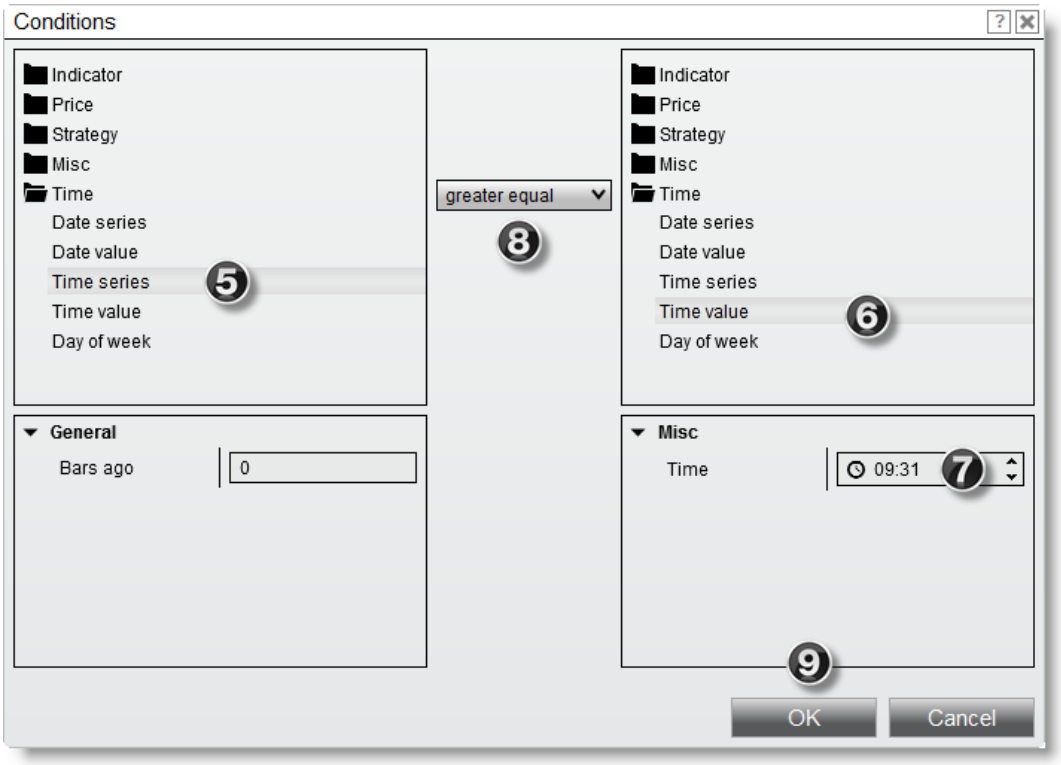


2. Optionally set a custom name for your Condition Group, i.e. Time Filter.
3. Selects **if all** of the individual conditions in the group have to be met in order to allow for a 'true' result evaluation, or **if any** will be sufficient.
4. Press **add**, **edit** or **remove** to add new condition into the group or manage existing ones.

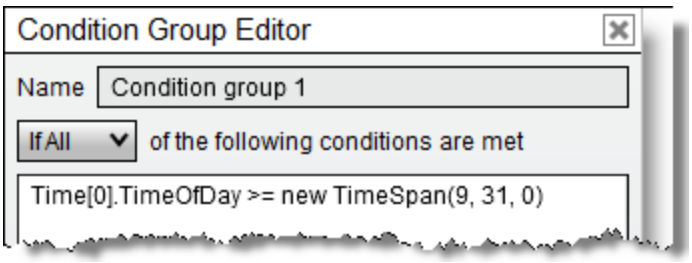


5. **Add** a new condition in and expand the **Time** category and select **Time series**
6. Expand the **Time** category and select **Time value**

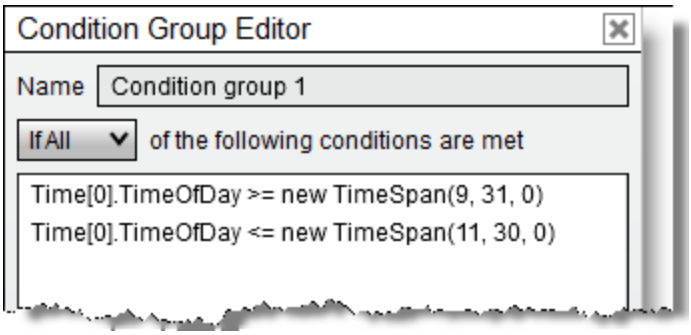
7. Enter your desired **Time** under Misc for the start of the time filter, i.e. 9:31
8. Select the **greater equal** relational operator



9. Press the **OK** button then to return to the Condition Group Editor with your first filter condition created.



Having setup the second, opposing condition as well the Condition Group for the time filter could look like :



Press **OK** now in the Condition Group Editor to exit out of it and return to the Conditions and Actions screen to setup other criteria, such as your trade entry as well as the resulting actions to take.

The time filter created would translate to :

"Allow this condition group to be true only if the Time of day is greater or equal to 9:31 AM and less or equal to 11:30 AM"

10.28.3 Actions

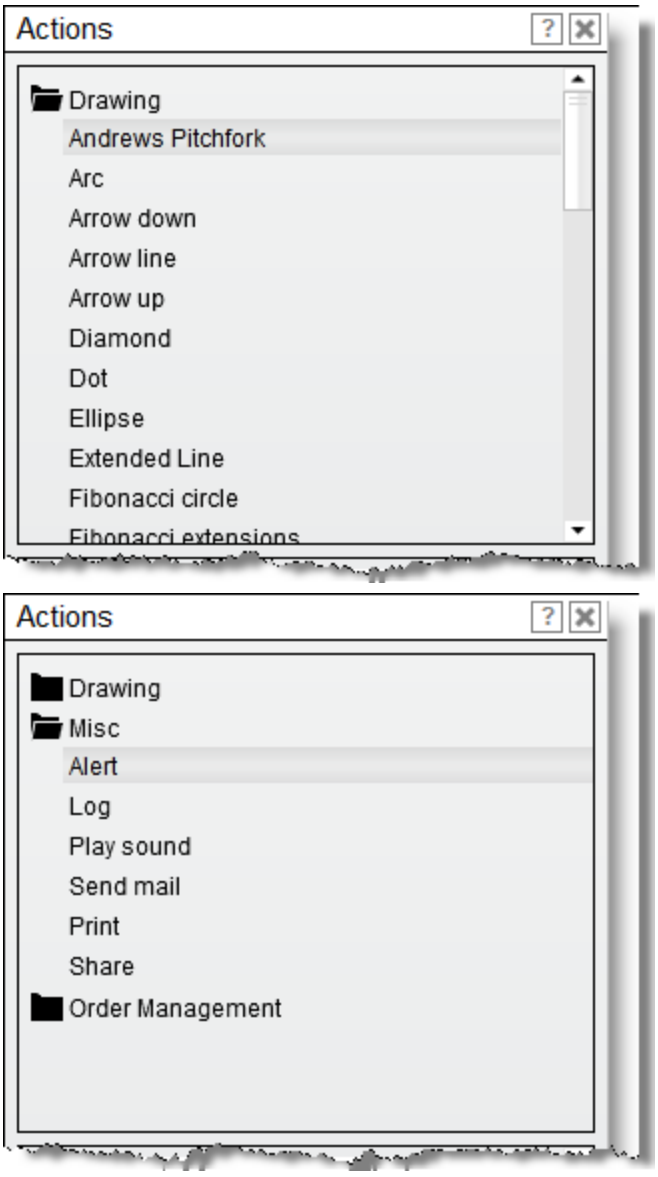
The **Actions** window allows you to select actions to execute for your script's conditions, for example executing an order or visualizing outcomes via draw objects.

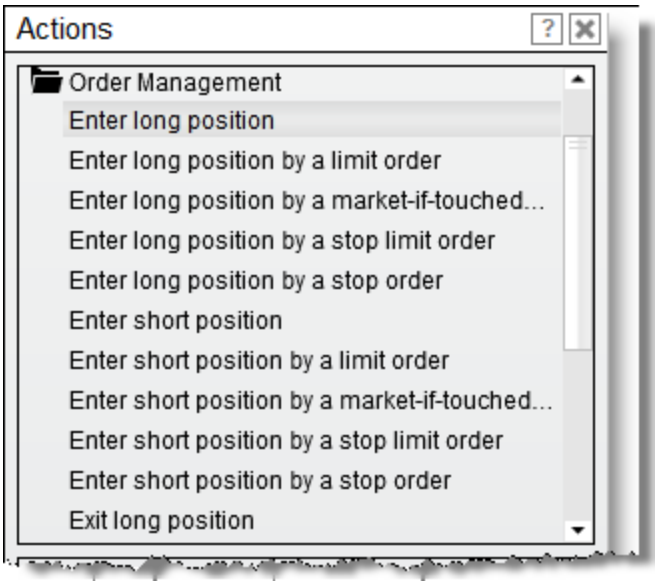
Understanding the Actions window

Strategy Action Window

The **Actions** window allows you to select actions to execute. Actions are executed when a strategy condition is true. The **Actions** window can be accessed via the **Conditions and Actions** Builder screen.

Within a NinjaScript strategy you can invoke miscellaneous actions, submit various order types for entering and exiting market positions, and have access to various drawing methods as shown in the images below.





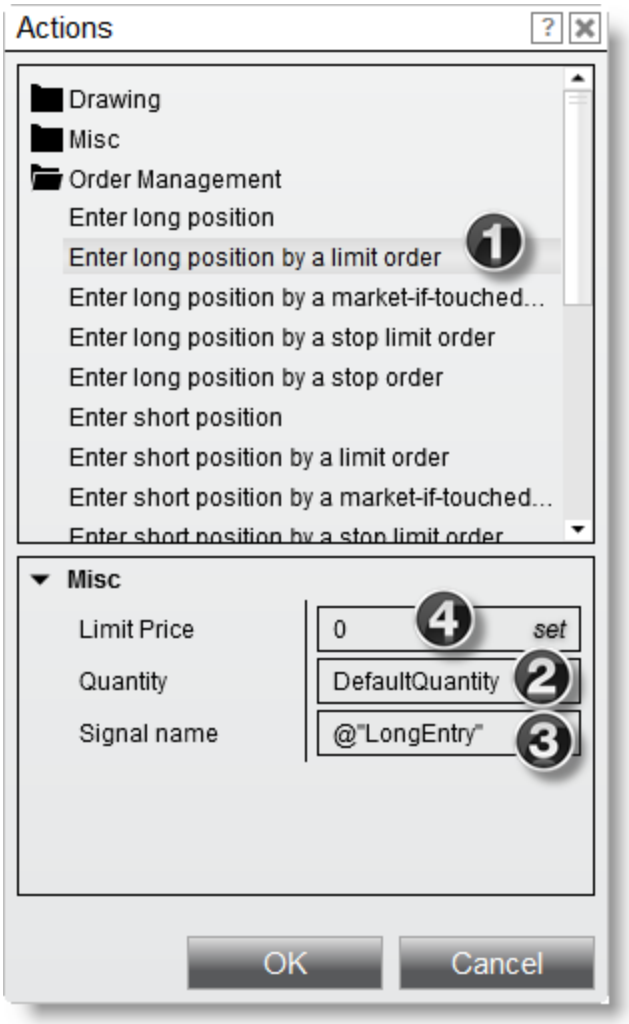
How to enter a market position

Entering a Market Position

Using the various **Order management** actions, you can enter a position using market, limit, market-if-touched, stop limit and stop market orders.

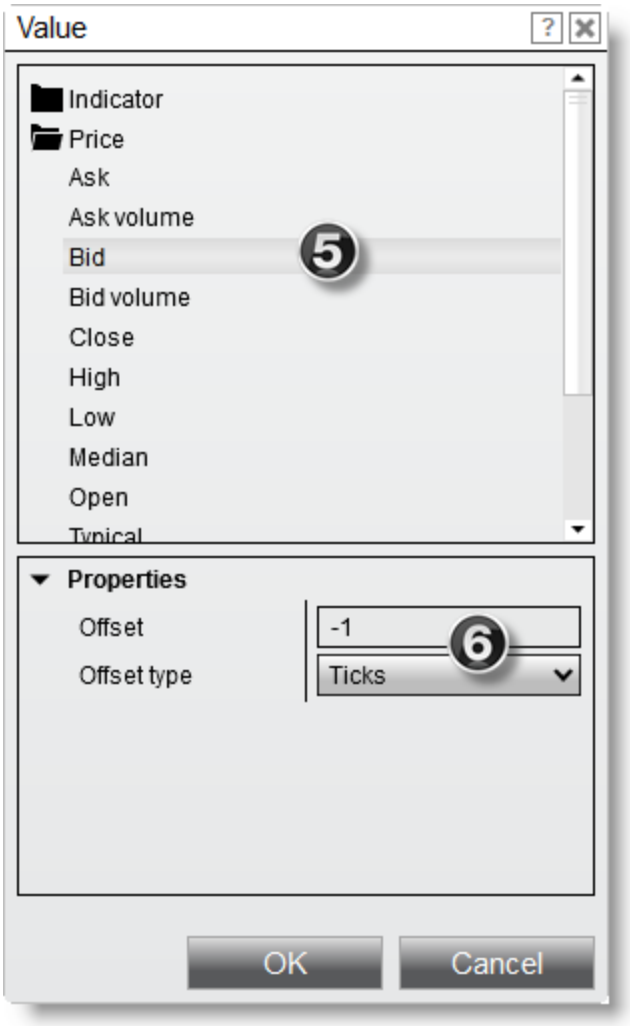
Following is an example and represents one of many possible combinations.

1. Expand the **Order management** category and select **Enter a long position by a limit order**
2. You can optionally set the number of contracts / shares for the order or leave the **DefaultQuantity** value which allows you set the quantity when starting a strategy
3. Set the ***Signal name** property to any user defined value to identify the entry (you can also leave this name blank) - here we used LongEntry
4. We can set the limit price dynamically by setting it to another item's value, press the "**Set**" button to open the **Value** window



*__Signal names__ are important in that they are used as unique identifiers if you have more than one unique entry in a strategy. By providing unique entry signal names for each entry on a strategy, you can then identify which position you want closed via the exit position methods. Signals names are also used to identify executions on a chart visually.

5. Expand the **Price** category and select **Bid**
6. Set the **Offset type** to **Ticks** and enter a value of "-1" for **Offset** (see "*How to offset an item value*" section of the Condition Builder page of the Help Guide for more information)



Once the **OK** button is pressed, an action is created that would translate to the following:

"Enter a buy limit order at a price 1 tick below the current bid price to enter a long position"

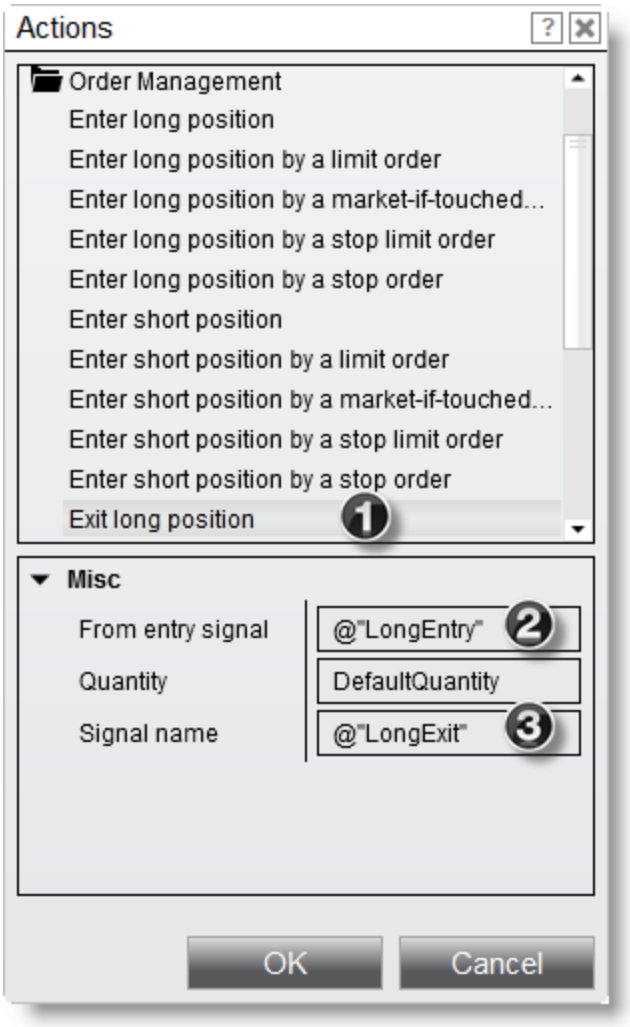
How to exit a market position

Exiting a Market Position

Using the various **Order management** actions, you can exit a position using market, limit, stop market and stop limit orders.

Following is an example and represents one of many possible combinations.

1. Expand the **Order management** category and select **Exit long position** (exits via market order)
2. Set the **From entry signal** property to a named entry signal within the strategy (tied to our prior example, LongEntry is used). Providing a value will exit only the quantity associated to the position created by the named signal. Leaving it blank will exit the total net position.
3. Set the **Signal name** property to any user defined value to identify the entry (we use LongExit here, but you can also leave this name blank)



Once the **OK** button is pressed, an action is created that would translate to the following:

"Enter a sell market order to exit from entry signal 'Long Entry'."

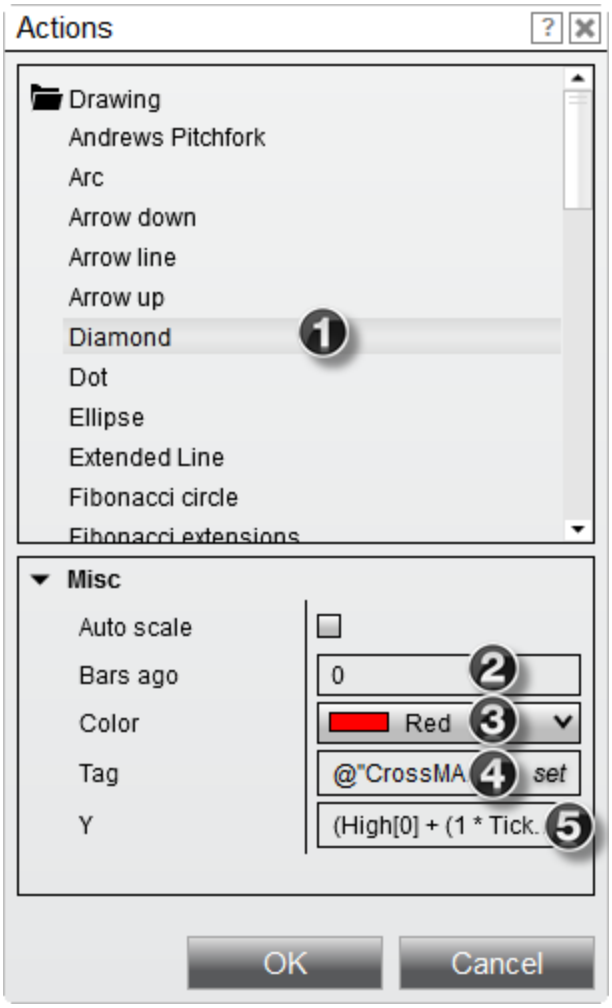
How to draw on a chart

Drawing on a Chart

Using the various **Drawing** methods, you can draw lines, text, squares and more on a chart. You can review detailed information on supported drawing methods in the NinjaScript Language Reference section of this Help Guide.

Following is an example and represents one of many possible combinations.

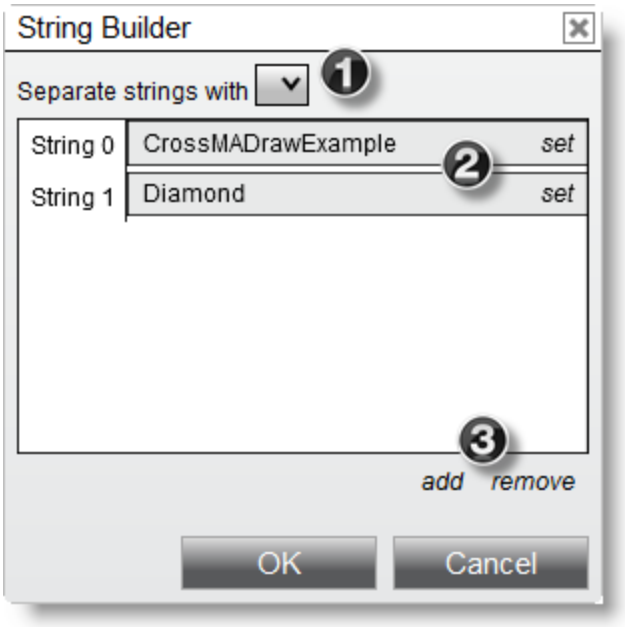
1. Expand the **Drawing** category and select **Diamond**
2. Set the **Bars ago** parameter to "0" which will draw the diamond on the current bar x location
3. Set the **Color** parameter to any desired color
4. Set the **Tag** parameter with a user defined name that identifies this drawing object. Providing a tag is of value if you are going to draw more than one of the same draw type object (Diamond in this case) on the same bar. Per default the builder will set this to the script name plus the draw object type, pressing the "set" button will display the String Builder window that would let you customize this further.
5. Set the **Y** parameter to the "High" of the current bar plus one tick by pressing the "set" button (not seen below, but same concept as in step 4) to display the **Value** window



Once the **OK** button is pressed, an action is created that would translate to the following:

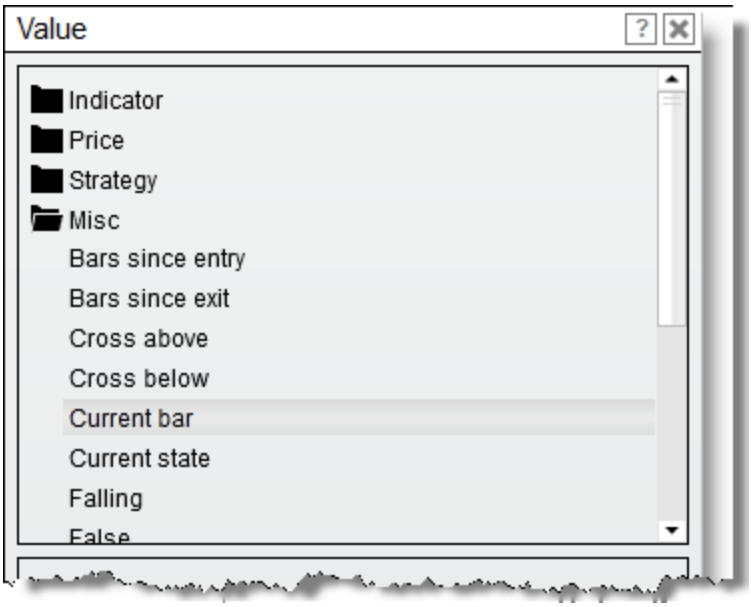
"Draw a red diamond above the high of the current bar plus one tick"

If you want to further customize the drawing object tag's used, then the String Builder will offer the following :



1. Select your string separator here, possible values are - ; : or blank (which is the default)
2. Enter custom text, or items from the **Value** window in the String fields
3. Press the "**add**" or "**remove**" buttons to add new string fields in or remove any of the currently added ones, the last filed will stay in any case, as a tag is needed for the object created.

For example if we added a 3rd string field in and added the Current bar from the Value window misc category, our drawing object would plot on each occurrence of the condition, so also for any historical triggers.



10.29 Time & Sales

Time & Sales Overview

You can access the **Time & Sales** window from within the NinjaTrader **Control Center** window by left mouse clicking on the menu **New**, and then selecting the menu item **T & S**

The **Time & Sales** window displays the current Bid/Ask price and volume as well as color coded last traded time, price and size. You can optionally filter for large trades blocks (B) by setting the block size in the **Time & Sales** Properties dialog window.

- Using the Time & Sales Window
- Time and Sales Properties
- Window Linking

10.29.1 Using the Time & Sales Window

Play Video

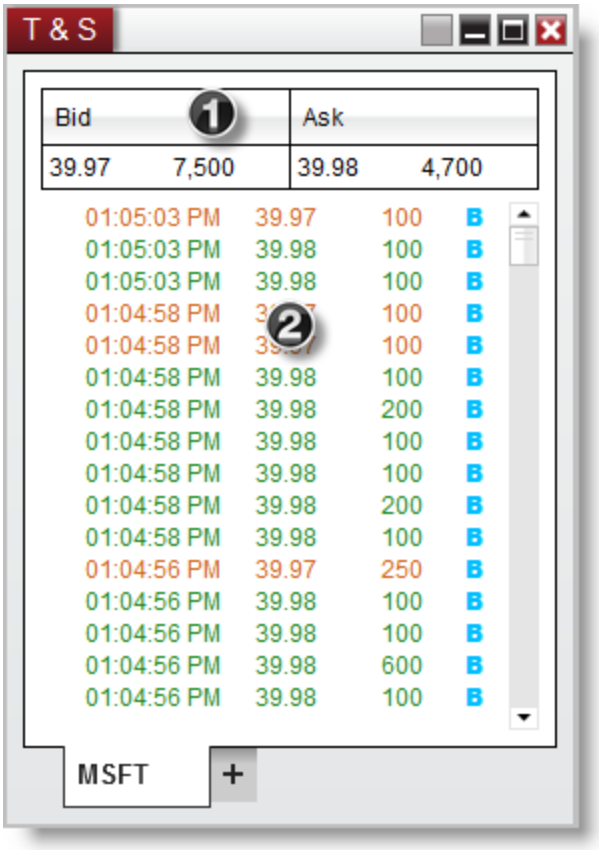
Selecting an Instrument

There are multiple ways to select an Instrument in the **Time & Sales** window.

- Right clicking on the **Time & Sales** window and selecting the menu **Instruments**.
- With the **Time & Sales** window selected begin typing the instrument symbol directly on the keyboard. Typing will trigger the **Overlay Instrument Selector.**

For more Information on instrument selection and management please see Instruments section of the Help Guide.

Understanding the layout of the Time & Sales window



Quotes

The Quotes section displays the current bid and ask price.

Bid	The current bid price followed by the number of contracts at the current bid.
Ask	The current ask price followed by the number of contracts at the current ask

You can disable the Quotes section by clicking on your right mouse button and deselecting the menu item **Show Quotes**.

Time & Sales Grid

The **Time & Sales Grid** displays Bid, Ask, and Last data.

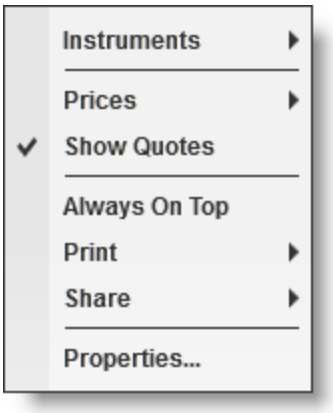
Daily High/Low	Displays an 'H' if the trade occurred at or above the current session high price Displays an 'L' if the trade occurred at or below the current session low price
Time	Displays the time stamp of the trade, The time stamp format can be configured via the Time & Sales properties dialog window
Price	Displays the price of the trade
Size	Displays the volume of the trade
Block	Displays a 'B' if the last trade size was greater then the **"Block alert trade size"** configured via the Time & Sales properties dialog window

You can enable or disable the columns by right mouse clicking within the **Time & Sales** window and then selecting the menu item **Properties...**

> **Note**: Selecting an instrument outside of market hours will display snapshot data from the data provider. This can result in the first price information being from the previous close.

Right Click Menu

Right mouse click on the **Time & Sales** window to access the right click menu.



Instruments	Selects the instrument
Prices	Selects what level 1 data to display, you can choose to display bid, ask, and last data in the **Time & Sales** window.
Show Quotes	Sets if the quotes section is displayed
Always On Top	Sets if the window should be always on top of other windows
Print	Displays Print options
Share	Displays Share options
Properties...	Sets the Time & Sales properties

Using tabs

Using Tabs
Please see the "Using Tabs" section of the help guide for more information.

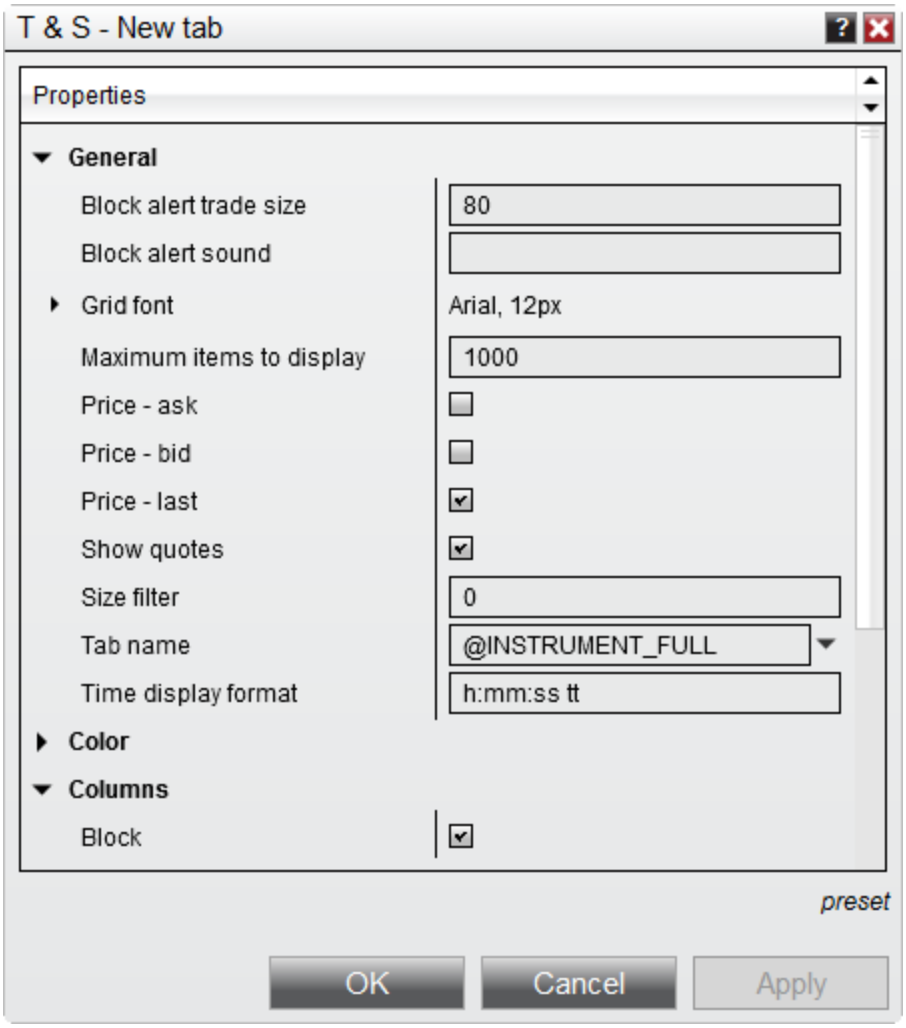
10.29.2 Time & Sales Properties

Many of the **Time & Sales** visual display settings can be customized using the **Time & Sales Properties** window.

How to access the Time & Sales Properties window

You can access the Time & Sales Properties menu by right clicking in the Time & Sales window and selecting the menu name **Properties**.

Available properties and definitions



Property Definitions

General	

Block alert trade size	Sets a value indicating the minimum last trade size required to register a block trade alert
Block alert sound	Select a sound file to play when the block alert is triggered
Grid font	Sets the font options
Maximum items to display	Sets the number of display rows
Price - ask	Sets if the level 1 ask data will be displayed
Price - bid	Sets if the level 1 bid data will be displayed
Price - last	Sets if the level 1 last data will be displayed
Show quotes	Sets if the quotes section is displayed
Size filter	Sets a value indicating the minimum trade size (trades less than this size are filtered out)
Tab name	Sets the tab name
Time display format	Sets the display format for the time column, format can be customized using the symbols below: d — The day of the month, from 1

	through 31.
dd	The day of the month, from 01 through 31.
f	The tenths of a second in a date and time value.
ff	The hundredths of a second in a date and time value.
fff	The milliseconds in a date and time value.
hh	The hour, using a 12-hour clock from 01 to 12.
HH	The hour, using a 24-hour clock from 00 to 23.
mm	The minute, from 00 through 59.
s	The second, from 0 through 59.
ss	The second, from 00 through 59.
tt	The AM/PM designator.

Color	
Above ask background	Sets the back color for trades above the ask price
Above ask foreground	Sets the text color for trades above the ask price

Ask background	Sets the back color for ask data
Ask foreground	Sets the text color for ask data
At ask background	Sets the back color for trades at the ask price
At ask foreground	Sets the text color for trades at the ask price
At bid background	Sets the back color for trades at the bid price
At bid foreground	Sets the text color for trades at the bid price
Below bid background	Sets the back color for trades below the bid price
Below bid foreground	Sets the text color for trades below the bid price
Between background	Sets the back color for trades between the bid and ask price
Between foregrou	Sets the text color for trades between the bid and ask price

nd	
Bid background	Sets the back color for bid data
Bid foreground	Sets the text color for bid data
Block alert	Sets the block icon color.
Daily high	Sets the high icon color
Daily low	Sets the back color for trades at the daily low
Display Background	Sets the back color of the display rows
Columns	
Block	Sets if the block column is displayed
Daily High/Low	Sets if the daily high/how column is displayed
Price	Sets if the price column is displayed
Time	Sets if the time column is displayed
Volume	Sets if the volume column is displayed
Window	
Always on top	Sets if the window will be always on top of other windows.

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

Using Tab Name Variables

Tab Name Variables

A number of pre-defined variables can be used in the "Tab Name" field of the **TIme & Sales Properties** window. For more information, see the "*Tab Name Variables*" section of the Using Tabs page.

10.29.3 Window Linking

Please see the Window Linking section of the Help Guide for more information on linking the **Time & Sales** window.

10.30 Trade Performance

Trade Performance Overview

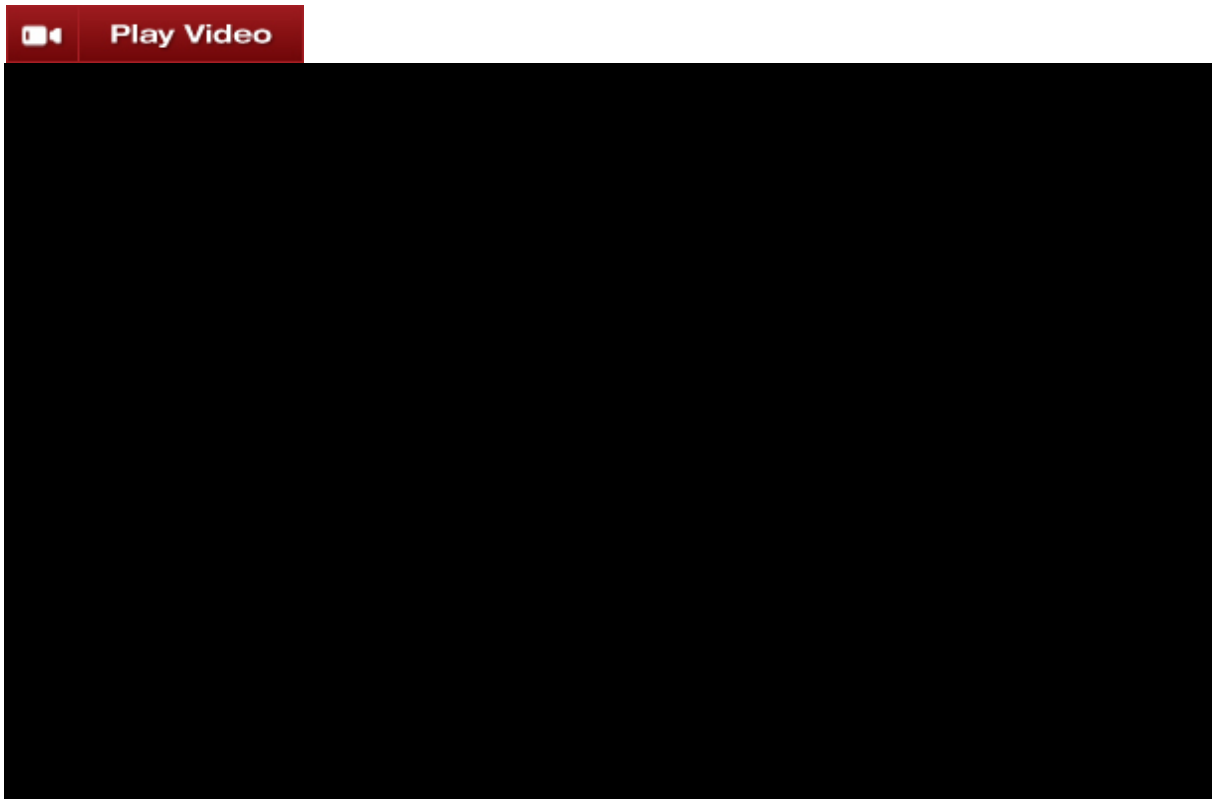
The **Trade Performance** window can be opened by left mouse clicking on the **New** menu within the NinjaTrader Control Center and selecting the menu item **Trade Performance**.

The **Trade Performance** window allows you to generate customized performance data against your trade history for your accounts.

- Using Trade Performance
- Performance Displays
- Statistics Definitions
- Trade Performance Properties

10.30.1 Using Trade Performance

You can access the **Trade Performance** window from within the NinjaTrader Control Center window by left mouse clicking on the menu **New**, and then selecting the menu item **Trade Performance**.

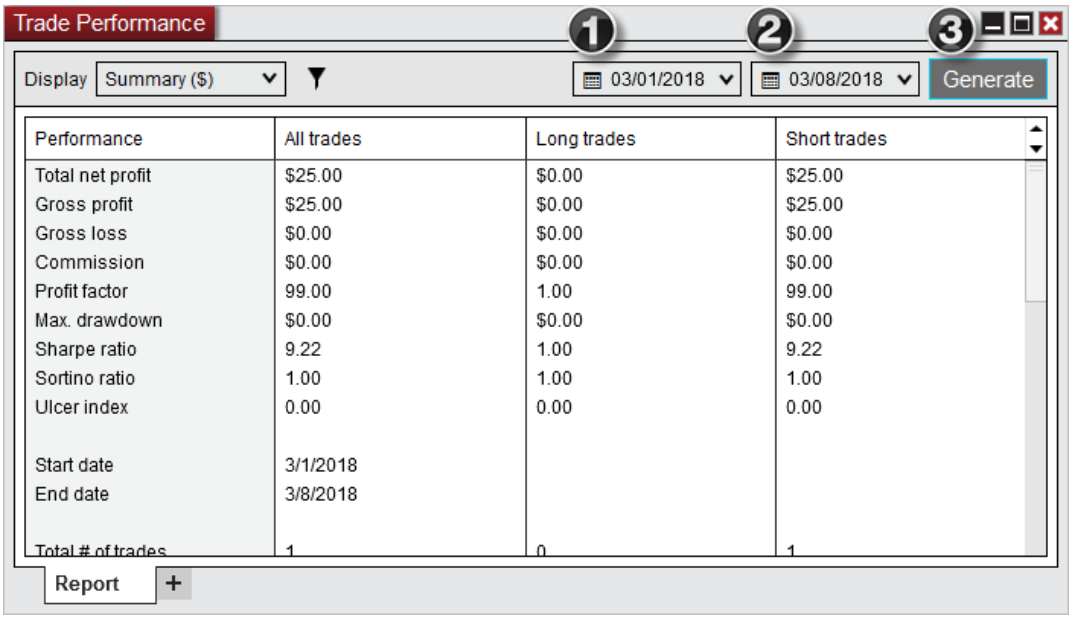


Understanding the performance report

Performance Report
To generate a performance report:

1. Select the From date
2. Select the To date
3. Press the **Generate** button

Performance data is generated and displayed in the various Performance Displays.



The **Generate** button only needs to be pressed when adjusting the From and To dates or if since generating the report new trades have been placed within the From and To dates.

> **Note**:
> It is possible to have before the **From** date being listed in your report. NinjaTrader generates reports from the last time you were flat. If a particular instrument was already in the middle of a position at the beginning of the **From** date, NinjaTrader will report all trades prior to this date up to the point where the position was flat. This will ensure you have a complete picture in terms of your performance on any specific date instead of "jumping" into the middle of a position which may cause inaccurate overall performance.

Display Options

Use the **Display** selector to select both what to display and how to display it.

Available Display Views

- Summary
- Analysis
- Executions
- Trades
- Orders
- Journal

Available Display Units

- Currency
- Percent
- Points
- Pips
- Ticks

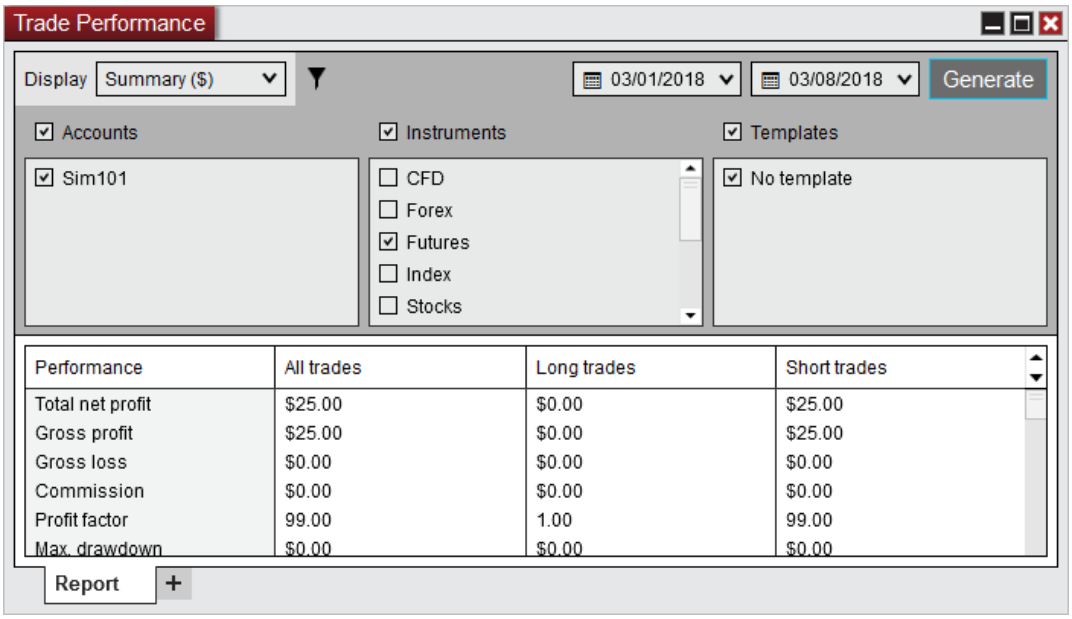
> **Note**: As **Forex** trade quantities are typically in multiples of 1,000, 10K, and 100K a NinjaTrader feature is to divide (normalize) your **Forex** trades by your account lot size. Account lot size is recorded per execution and is set by the connection. A micro **Forex** account is 1,000, a mini **Forex** account is 10K, a standard **Forex** account is 100K (FX lot size is set automatically if your broker supports it and if not set manually via the property "FX Lot Size" in the connection setup)
>
> Example: Account is a mini **Forex** account (10K). A trade is made with quantify of 10k and gain a 1 pip. Instead of recording that profit as 10,000 pips it is recorded as 1 pip of profit.

Understanding Filter Options

Filter Options

Pressing the **Filter** icon will expand the Performance tab to include parameters that you can use to filter your performance reports. This filtering is done on an executions basis and not a trades basis.

> **Notes**:
> Adjustments to the filters will automatically update within the Trade Performance window. The **Generate** button does not need to be selected.
> If the From and To dates are adjusted and then **Generate** is pressed unchecked filter items will become rechecked since the **Generate** button created a new scan to see what item are applicable for the time window.



1. Accounts	Sets the account(s) to be included in the performance report
2. Instruments	Sets the instruments name or type to be included in the performance report
3. Templates	Sets the ATM strategies to be included in the performance report

Tip: The checkbox inline with the filter label will toggle the check mark for all items in that list. Allowing you to quickly select or deselect the entire list.

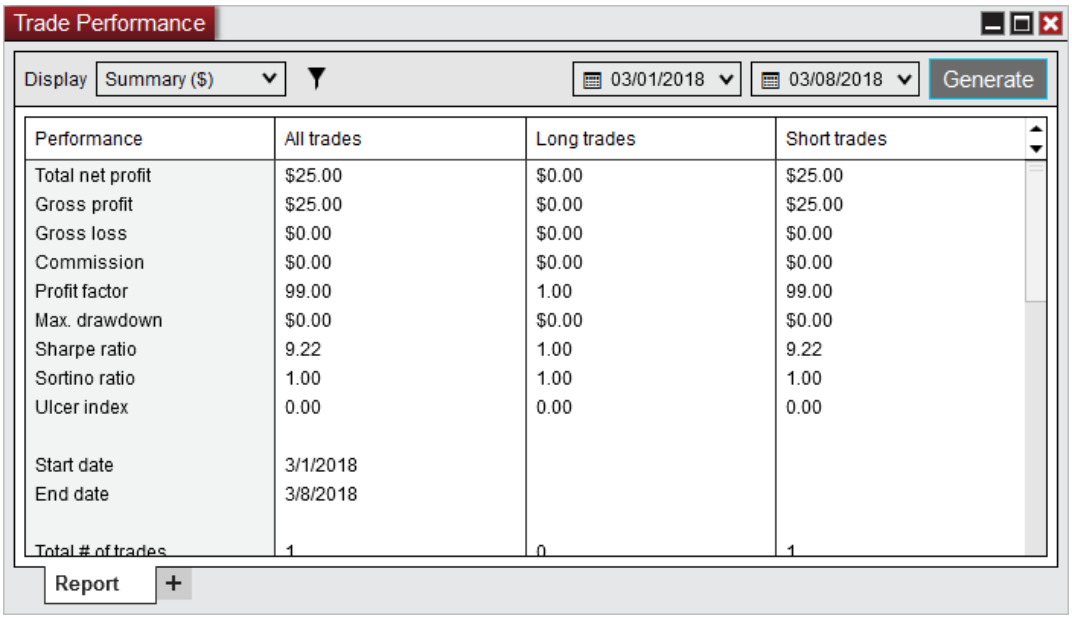
10.30.2 Performance Displays

The **Account Performance** window displays performance data in a variety of ways.

▼ Understanding the Summary display

Summary Display

Displays all performance statistics and metrics.



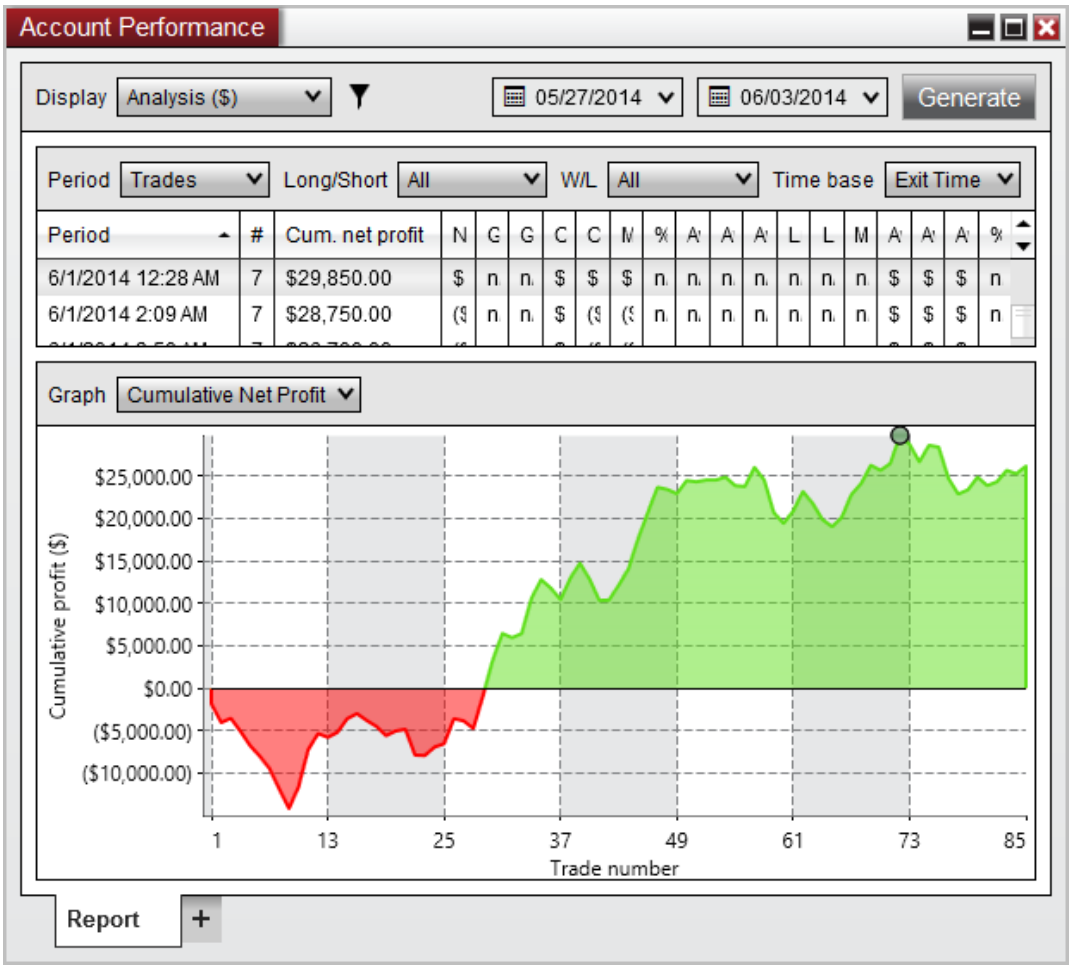
Please see the Statistic Definition section of the help guide for details on how each statistic is calculated.

Tip: You can add your own **Custom Performance Metrics through** NinjaScript programming, or install other performance metrics developed by 3rd parties. Please see this section of the help guide for more information on how to develop and add a custom performance metric

Understanding the Analysis display

Analysis Display

Displays data based on various time periods for analysis.



Analysis view displays both a grid of data in the selected period format and a graph that you choose to display based on the period data. It allows an easy way to see trends in the data set and make correlations.

Period Grid

The period grid has options that let you select what data to display, note that the data in the grid drives the data shown in the selected graph type below.

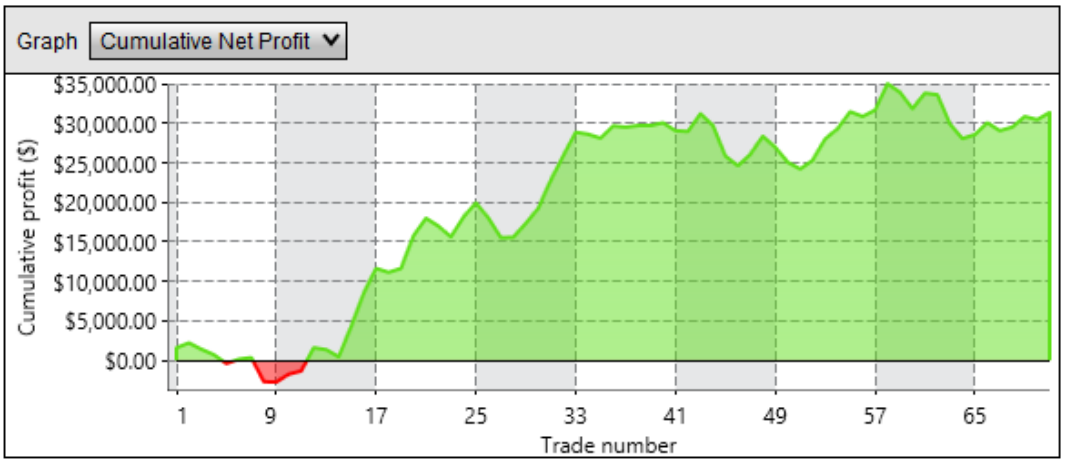
Period	Sets the periodicity you want the trade data displayed in, this also drives the graph below. **Available Period Selections** • Daily • Weekly • Monthly

	• Yearly • Trades • Half-hour of day • Hour of day • Day of week Note: Half-hour of day, Hour of day, and Day of week are distribution period selections, meaning that the trades that make up the collection will not be in synchronous order, therefore cumulative statistics such as **Cumulative Net Profit** are disabled.
Long /Short	Sets if you just want long trades displayed or short trades displayed.
W/L	Sets if you want only trades that have a Net Profit greater than 0 displayed or less than or equal to 0.
Time base	Sets if you want to include the the trade in the period based on the entry time or the exit time of the trade.

Tip: Selecting a graph row will highlight the row and also highlight the data point that is associated to this data on the graph below.

Graph

The graph displays data from the period grid above.



You can select what data you would like to view from the **Graph** selection combo box. As you move your mouse over the **Graph** a dot will be displayed indicating that it is a data point. Left clicking on the data point will select it and also select the same data in the **Period Grid** above the **Graph**.

Available Graph Types

- Cumulative Net Profit
- Net Profit
- Cumulative Max. Drawdown
- Avg. MAE
- Avg. MFE
- Avg. Entry Efficiency
- Avg. Exit Efficiency
- Avg. Total Efficiency

Please see the Statistic Definition section of the help guide for details on how each graph type is calculated.

> **Note**: Some graph types are not available for some period types, in this case the graph type will be disabled.

Understanding the Executions display

Executions Display

The Executions display shows all historical executions in a data grid. The columns listed in the data grid use the same layout you would see from the **Executions Tab** of the Control Center. For definitions of each column, please see the Understanding the executions tab section.

Charting Executions

You can go to the exact chart location of an execution by doing the following:

1. Select the execution
2. Right mouse click and select the menu item **Chart**.

NinjaTrader will open a temporary chart to the location of the execution

Notes:

- The trade performance Chart is a non-configurable 1-minute interval and does not have all the standard features of a regular chart
- Charting Executions only work if you have access to historical data for that date range via a connection or in your local database

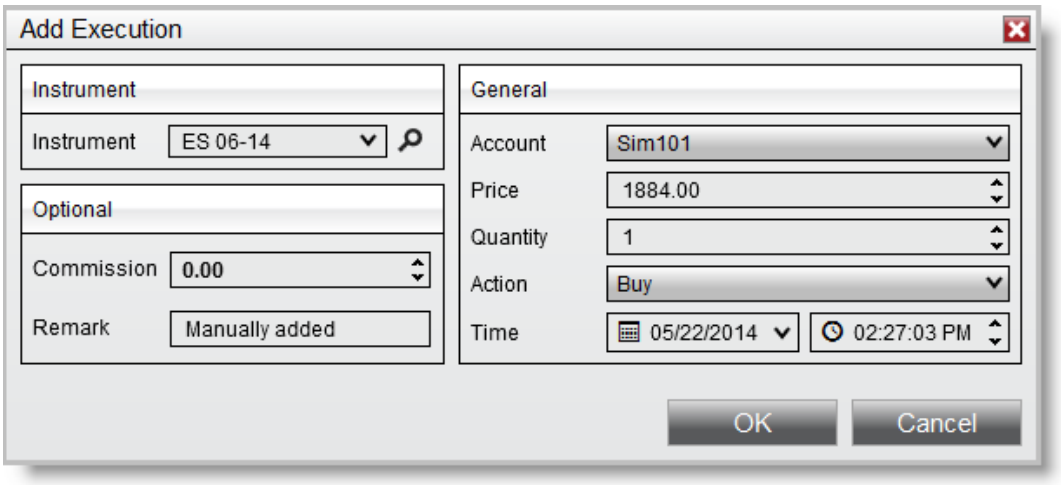
Adding Executions

There may be situations where you will want to manually add or remove an historical execution. Historical executions are used to generate performance data in the **Account Performance** window. If an execution is missing, the performance data will be incorrect. This could happen since not all brokers provide historical execution. Let's say you placed a good till cancelled (GTC) order on Monday, did not connect on Tuesday at which time your order filled, then connected NinjaTrader on Wednesday, NinjaTrader would never receive the execution report for Tuesday's order fill. You would then have to add this historical execution to the database if you want your performance reporting to be accurate.

To add an execution to the database:

1. Right mouse click in the **Executions** display and select the **Add...** menu item. The Add Execution window appears.
2. Input your desired execution parameter values in the image below
3. Press the OK button

The execution is now added to the database and will be used in performance reporting.



Removing Executions

You can also remove an execution by right mouse clicking the execution you wish to delete and selecting "**Remove**".

Understanding the Trades display

Trades Display

The Trades display shows all historical executions in a data grid. A Trade defined is a completed buy/sell or sell/buy transaction sorted by time and matched by the market position and quantity of the execution. Positions which have been scaled in or scaled out will be considered as separate trades.

> **Note**: If you are trading multiple NinjaScript strategies simultaneously on the same account, the logic used to pair trades is agnostic of executions which belong to a particular strategy, and will match trades based on the overall account position. This may cause the trade performance to calculate some statistics (such as Entry/Exit Pairing, Bars) differently than you are expecting. You can view a strategies individual performance from the Strategies tab

Trade number	The trade numbered by the sequence it occurred in the trade collection
Instrument	The Instrument on which the trade took place

Account	The Account the trade took place
Strategy	The NinjaScript or ATM strategy which generated the trade **Notes**: 1. ATM Strategies which have not been saved as a template will not be reflected (i.e., "Custom") 2. The strategy needs to be active for its name to be shown
Market position	Indicates the position of the trade (long or short)
Quantity	The quantity of the execution
Entry price	The entry execution price of the trade
Exit price	The exit execution price of the trade
Entry time	The execution time of the entry of the trade
Exit time	The execution time of the exit of the trade
Entry name	A name for the entry execution of the trade (if specified by the strategy or action)
Exit name	A name for the exit execution of the trade (if specified by the strategy or action)
Profit	The profit of the individual trade
Cumulative net profit	Summation of all the profit earned by all your trades

Commission	Summation of commission applied to the entry and exit executions
MAE	Maximum adverse excursion (i.e., worst price trade reached – entry price)
MFE	Maximum favorable excursion (i.e., best price trade reached – entry price)
ETD	End Trade Drawdown (i.e., MFE - profit)
Bars	The number of bars between the entry and exit executions **Note**: Only applicable to NinjaScript strategy executions

Tip: Please see the Statistic Definition section of the help guide for additional details regrading trade value calculations

Charting Trades

You can go to the exact chart location of an trade by doing the following:

1. Select the trade
2. Right mouse click and select the menu item **Chart**.

NinjaTrader will open a temporary chart to the location of the trade.

Notes:
- The temporary Chart is a non-configurable 1-minute interval and does not have all the standard features of a regular chart
- Charting Executions only work if you have access to historical data for that date range via a connection or in your local database

Understanding the Orders display

Orders Display

The **Orders** display shows all historical orders in a data grid.

Understanding the Journal display

Journal Display

The Journal tab is only visible in the **Account Performance** window. The Journal tab allows you to keep journal entries on your trading activities. Enter your comments in the text area and press "add". The data grid will display your journal entries by date.

> **Tip**: You can also add Journal entries based on a Execution or Trade via the Executions Display and Trades Display. Right click on an execution and select "**Add Journal Entry**".

10.30.3 Statistics Definitions

The following are definitions and formulas used for Trade Performance statistics.

> **Notes**:
> - Quantity is defined as the number of contracts traded
> - Point value is defined as the monetary conversion of each point (e.g. 100 for currency pairs)
> - FX lot size is the default Forex Lot Size for the account
> - Please also review the information on Profit and Loss Calculation Modes where noted as applicable

Understanding Profit

Profit

The difference in price between the entry and exit execution. This value may be positive or negative and is used to determine winning vs losing trades

- `(exit price – entry price) for long trades`
- `(entry price – exit price) for short trades`

Note: This statistic may also display in selected **Display Units** (percent, points, pips or ticks). To see the base calculation behind each execution, view the Profit and Loss Calculation Modes page.

Understanding Total Net Profit

Total net profit

This statistic returns a monetary value representing a final cumulative profit of the all profitable trades and all unprofitable trades.

Currency, Pips, Points, Ticks	`SUM(gross profit and gross loss) of all trades`
Percentage	`SUM((1 + gross profit in percent) * ( 1 + gross loss in percent) - 1) of all trades`

Notes:

- *See also Understanding Gross Profit* and *Understanding Gross Loss* on this page
- This statistic may also display in selected **Display Units** (percent, points, pips or ticks). To see the base calculation behind each execution, view the Profit and Loss Calculation Modes page.

Understanding Gross Profit

Gross Profit

This statistic returns a monetary value representing a summation of all the money earned across all your trades.

Currency, Pips, Points, Ticks	`SUM(profit * quantity) of all winning trades`
Percentage	`SUM((1 + Current gross profit in percent) * ( 1 +`

	gross profit in percent) - 1) of all winning trades

Note: This statistic may also display in selected **Display Units** (percent, points, pips or ticks). To see the base calculation behind each execution, view the Profit and Loss Calculation Modes page.

Understanding Gross Loss

Gross Loss

This statistic returns a monetary value representing a summation of all the money lost across all your trades.

Currency, Pips, Points, Ticks	SUM(profit * quantity) of all losing trades
Percentage	SUM((1 + Current gross loss in percent) * (1 + gross loss in percent) - 1) of all losing trades

Note: This statistic may also display in selected **Display Units** (percent, points, pips or ticks). To see the base calculation behind each execution, view the Profit and Loss Calculation Modes page.

Understanding Commission

Commission

This statistic returns a monetary value that is the summation of all the commission fees associated with the trades executed.

```
SUM(commission of all traded executions)
```

Note: Commissions must be setup on the account using a Commission template

Understanding Profit Factor

Profit Factor

This statistic returns a ratio that can be used as a performance measure for your strategy. It gives you an idea of how much more money your strategy earns then it loses. A higher ratio can be considered characteristic of a high performing strategy. A ratio less than one indicates your strategy loses more money than it earns.

```
Gross Profit / Gross Loss
```

Understanding Max. Drawdown

Max. Drawdown

The maximum drawdown statistic provides you with information regarding the biggest decrease (drawdown) in account size experienced from the highest high seen. Drawdown is often used as an indicator of risk.

```
Drawdown = local maximum realized profit – local minimum realized loss
Max Drawdown = single largest Drawdown
```

As an example, your account rises from $25,000 to $50,000. It then subsequently drops to $40,000 but rises again to $60,000. The drawdown in this case would be $10,000 or -20%. Take note that drawdown does not necessarily have to correspond with a loss in your original account principal.

> **Note**: This statistic may also display in selected **Display Units** (percent, points, pips or ticks). To see the base calculation behind each execution, view the Profit and Loss Calculation Modes page.

Understanding Sharpe Ratio

Sharpe Ratio

This statistic returns a ratio that measures the risk premium per unit of risk of your strategy. It can help you make decisions based on the excess risk of your strategies. You may have a high-return strategy, but the high returns may come at a cost of excess risk. The Sharpe ratio will help you determine if it is an appropriate increase in risk for the higher return or not. Generally, a ratio of 1 or greater is good, 2 or greater is very good, and 3 and up is great.

```
(Profit per Month – risk free Rate of Return) / standard deviation of
monthly profits
```

Notes:
- See also *Understanding Profit Per Month* on this page
- NinjaTrader hard sets "risk-free Rate of Return" to a value of zero
- The Sharpe Ratio is set to a value of "1" if there is insufficient data to calculate the monthly standard deviation of profits (i.e., there is only 1 month of trade history or less)
- A month is defined as 30.5 days which is the (number of days) / (number of months in a year considering leap year)

Understanding Sortino Ratio

Sortino Ratio

This statistic is used the same as Sharpe Ratio, the only difference being that Sortino only takes into account the downside deviation. You would want to use this statistic if you wanted to differentiate between harmful volatility from volatility in general (Sharpe Ratio).

```
(Profit per Month – risk free Rate of Return) / standard deviation of
monthly drawdown
```

See also *Understanding Profit Per Month* on this page.

Notes:
- NinjaTrader hard sets "risk-free Rate of Return" to a value of zero
- The Sortino Ratio is set to a value of "1" if there is insufficient data to calculate the monthly standard deviation of profits (i.e., there is only 1 month of trade history or less)
- A month is defined as 30.5 days which is the (number of days) / (number of months in a year considering leap year)

Understanding Ulcer Index

Ulcer Index

This statistic measures downside risk, the Ulcer Index becomes higher as profit declines from the max realized profit achieved and lower as profit rises. The lower the value the better as this indicates there is overall less downside risk.

Currency	`SQRT(Summation((cumulative currency profit - maximum realized currency profit) ^2 ) / Total # of trades)`
Percent	`SQRT(Summation((1 + cumulative percent profit / (1 + maximum realized percent profit) - 1) ^2 ) / Total # of trades)`
Points	`SQRT(Summation((cumulative point profit - maximum realized point profit) ^2 ) / Total # of trades)`
Pips	`SQRT((Summation((cumulative point profit - maximum realized point profit) ^2) / PipSize ) / Total # of trades)`
Ticks	`SQRT((Summation((cumulative point profit - maximum realized point profit) ^2) / TickSize ) / Total # of trades)`

Understanding Probability

Probability

This statistic determines how likely a trade is to occur that would return the same PnL as your **Avg. trade**. This is based on how many trade's PnL fall within a standard deviation of the **Avg. trade**. Student's t-distribution is used to find probability.

Understanding Winning, Losing, Even, and Total Number of Trades

Trade totals

These are a simple statistics used to gauge the overall performance of the performance report.

Total # of trades	The total number of trades taken between the start and end date in the collection
# of winning trades	The total number of trades which profit in point is greater than 0
# of losing trades	The total number of trades which is less than 0
# of even trades	The total number of trades which profit in point is equal to 0
Percent profitable	The total number of profitable trades divided by the total number of trades

Note: The winning and losing trades are factored by their calculated profits solely in points. It is possible to have trades which are technically profitable in percent, but are not profitable based on their point value (or vice versa)

Understanding Average Trade

Average Trade

This statistic returns a value representing the average profit you experience from all of your trades. It is useful for getting an idea of how much you could expect to earn on future trades.

Currency	`SUM(profit * quantity * point value) of all trades / # of trades`
Percen	`SUM(profit * quantity / entry price) of all trades / # of traded lots`

t	
Points	SUM(profit * quantity) of all trades / # of trades
Pips	SUM(profit * quantity / FX lot size) of all trades / # of trades
Ticks	SUM(profit * quantity / tick size) of all trades / # of trades

Understanding Average Winning Trade

Average Winning Trade

This statistic returns a value representing the average profit you experience from all of your winning trades. It is useful for getting an idea of how much you could expect to earn on winning trades.

Currency	SUM(profit * quantity * point value) of all winning trades / # of winning trades
Percent	SUM(profit * quantity / entry price) of all winning trades / # of winning traded lots
Points	SUM(profit * quantity) of all winning trades / # of winning trades
Pips	SUM(profit * quantity / FX lot size) of all winning trades / # of winning trades
Ticks	SUM(profit * quantity / tick size) of all winning trades / # of winning trades

Understanding Average Losing Trade

Average Losing Trade

This statistic returns a value representing the average loss you experience from all of your losing trades. It is useful for getting an idea of how much you could expect to lose on losing trades.

Currency	`SUM(loss * quantity * point value) of all losing trades / # of losing trades`
Percent	`SUM(profit * quantity / entry price) of all losing trades / # of losing traded lots`
Points	`SUM(profit * quantity) of all losing trades / # of losing trades`
Pips	`SUM(profit * quantity / FX lot size) of all losing trades / # of losing trades`
Ticks	`SUM(profit * quantity / tick size) of all losing trades / # of losing trades`

Understanding Ratio Avg Win / Avg Loss

Ratio Avg Win / Avg Loss

This statistic returns a ratio that can be used as a performance measure for your strategy. A value greater than 1 signifies you win more than you lose. A value less than 1 signifies you lose more than you win.

```
Average Winning Trade / Average Losing Trade
```

Understanding Maximum Consecutive Winners

Max. consec. winners

This statistic returns the largest number of winning trades which followed a previous winning trade. Once a losing trade is detected, the consecutive winner count is reset until another winning trade is found

Understanding Maximum Consecutive Losers

Max. consec. losers

This statistic returns the largest number of losing trades which followed a previous losing trade. Once a winning trade is detected, the consecutive loser count is reset until another losing trade is found

Understanding Largest Winning Trade

Largest winning trade

This statistic returns the the most profitable trade value from the collection of trades

> **Note**: This statistic may also display in selected **Display Units** (percent, points, pips or ticks). To see the base calculation behind each execution, view the Profit and Loss Calculation Modes page.

Understanding Largest Losing Trade

Largest losing trade

This statistic returns the the least profitable trade value from the collection of trades

> **Note**: This statistic may also display in selected **Display Units** (percent, points, pips or ticks). To see the base calculation behind each execution, view the Profit and Loss Calculation Modes page.

Understanding Average # of trades per day

Average # of Trades per Day

This statistic returns a value that represents the average # of trades you take per day. This is useful so you can determine if you are overtrading. This statistic excludes weekends and holidays by using a 252 trading days in a year constant.

```
SUM(of all trades) / (# of days between the date of the first trade and the date of the last trade) * 252 / 365
```

Understanding Average Time in Market

Average Time in Market

This statistic returns a value that gives you an idea of how long you can expect your positions to be open. You can use this by manually closing out a position if you feel it has been in the market for too long.

```
SUM(exit date/time – entry date/time) of all trades / # of trades
```

Understanding Profit Per Month

Profit Per Month

This statistic returns a value that can be used as a performance measure for your strategy. It gives you an idea of how much profit you can expect to make per month. A month is defined as 30.5 days which found by the following: (number of days) / (number of months in a year considering leap year)

Currency	`cumulative profit * (30.5 / # days)`
Percent	$(1 + \text{cumulative profit})^{(1 * (30.5 / \# \text{ days}))} - 1$
Points	`cumulative profit * (30.5 / # days)`
Pips	`cumulative profit * (30.5 / # days)`
Ticks	`cumulative profit * (30.5 / # days)`

Note: See the cumulative profit statistic below for its definition

Understanding Max. Time to Recover

Max. Time to Recover

The maximum time to recover statistic returns the largest time it took to recover back to the highest profit experienced. This indicates how long you waited before becoming profitable again.

Understanding Longest Flat Period

Longest Flat Period

This statistic returns the longest time duration that occurred between trades. This may be reflected in total minutes, or total days.

```
current trade entry time - last trade exit time
```

Understanding Average MAE

Average MAE (Maximum Adverse Excursion)

This statistic returns a value representing the average maximum run-down your trades experience. This information helps you gauge how poorly your entry conditions predict upcoming price movement directions. A low percentage here is desirable since it would imply that the price movement after you enter a position follows the direction of your intended trade.

Currency	`SUM(MAE * quantity * point value) of all trades / # of trades`
Percent	`SUM(MAE * quantity / entry price) of all trades / # of traded lots`
Points	`SUM(MAE * quantity) of all trades / # of trades`
Pips	`SUM(MAE * quantity / FX lot size) of all trades / # of trades`
Ticks	`SUM(MAE * quantity / tick size) of all trades / # of trades`

Note:
- MAE (max. adverse excursion) is defined as *worst price trade reached – entry price*
- For real-time trades, the maximum and minimum price seen is recorded live during the duration of the trade and stored per entry/exit execution. This value includes bid/ask prices which may not be reflected on the chart. If there are time periods where you are not receiving real-time price updates, those prices within that period cannot be used.
- When backtesting, the high and low of the bar series is used

Understanding Average MFE

Average MFE (Maximum Favorable Excursion)

This statistic returns a value representing the average maximum run-up your strategy experiences. This information helps you gauge how well your strategy's entry conditions predict upcoming price movements. A high percentage here is desirable since it would imply high profitability opportunities.

Currency	`SUM(MFE * quantity * point value) of all trades / # of trades`
Percent	`SUM(MFE * quantity / entry price) of all trades / # of traded lots`
Points	`SUM(MFE * quantity) of all trades / # of trades`
Pips	`SUM(MFE * quantity / FX lot size) of all trades / # of trades`
Ticks	`SUM(MFE * quantity / tick size) of all torades / # of trades`

Note:

- MFE (max. favorable excursion) is defined as (best price trade reached – entry price)
- For real-time trades, the maximum and minimum price seen is recorded live during the duration of the trade and stored per entry/exit execution. This value includes bid/ask prices which may not be reflected on the chart. If there are time periods where you are not receiving real-time price updates, those prices within that period cannot be used.
- When backtesting, the high and low of the bar series is used

Understanding Average ETD

Average ETD (End Trade Drawdown)

This statistic returns a value that is useful in giving you a measure of how effective your exit conditions capture the price movements after your strategy enters a position. It shows you how much you give back from the best price reached before you exit the trade. A small number here is generally desirable since it would imply highly optimized exit conditions that capture most of the price movement you were after.

```
Average MFE – Average Trade
```

Understanding Cumulative Profit

Cumulative profit

This statistic returns a value representing a summation of all the profit earned by all your trades. It can be interpreted as a performance measure.

Currency	`SUM(Profit in Currency) of all trades`
Percent	`SUM((1 + Current Profit in Percent) * (1 + Profit in Percent) - 1) for all trades`
Points	`SUM(Profit in Points) of all trades`

Pips	SUM(Profit in Points / Pip Size) for all trades
Ticks	SUM(Profit in Points / Pip Size) for all trades

Tip: Cumulative profit in % mode will reinvest your profits - as an example let's say you take 3 trades on a starting value of 10K - on these trades you made $500, $1000, and $750. The starting percentages for these trades are 5%, 10%, and 7.5%, respectively. On Trade 1, we made 5% of $10k, which is $500. Rolling that into Trade 2, we made 7.5 % of ($10k + $500), or $1050. Rolling that into trade 3, we made 7.5% of ($10k + $500 + $1050) or $866.25 . This means that after the 3rd trade, we made an extra $866.25 by cumulating profits. This means our cumulative net profit will show ($500 + $1000 + $750 + $866.25) / $10k or 31% of our investment, and not ($500 + $1000 + $750) / $10k = 22.5% of our investment.

Note: To see the base calculation behind each calculation mode, view the Profit and Loss Calculation Modes page.

Understanding Entry, Exit, and Total Efficiency

Following are the formulas for the calculation of the entry, exit, and total efficiency.

Assume the following:
- Enter long at price of 100
- Market moves down to a price of 90
- Market moves up to a price of 130
- Exit at a price of 110

Entry Efficiency is Calculated as:

```
(maximum price seen - entry price) / (maximum price seen - minimum price seen)
= (130 - 100) / (130 - 90)
= 75%
= The entry took 75% of the trade range
```

Exit Efficiency is Calculated as:

```
(exit price - minimum price seen) / (maximum price seen - minimum price
seen)
= (110 - 90) / (130 - 90)
= 50%
= The exit took 50% of the available trade range
```

Total Efficiency is Calculated as:

```
(exit price - entry price) / (maximum price seen - minimum price seen)
= (110 - 100) / (130 - 90)
= 25%
= The trade represented only 25% of the trade range
```

Note:
- The formulas are reversed for short
- The blue line on any efficiency graph represents the average
- For real-time trades, the **maximum** and **minimum** price seen is recorded live during the duration of the trade and stored per entry/exit execution. This value includes **bid/ask prices** which may **NOT** be reflected on the chart. If there are time periods where you are receiving real-time price updates, those prices within that period are not used.
- When backtesting, the high and low of the bar series is used
- These statistics may also display in selected **Display Units** (percent, points, pips or ticks). To see the base calculation behind each execution, view the Profit and Loss Calculation Modes page

10.30.4 Profit and Loss Calculation Modes

Trade Performance statistics are based on core PnL calculations, which differ for each selected **Display Units** (currency, percent, points, pips or ticks) calculation mode. Below is a list of the formulas used for each calculation mode.

Calculation Modes

Currency	Rate of Exit * Profit in Points * Lot Size of Exit * Point Value of Exit
Percent	(Profit in Points * Lot Size of Entry) / (Quantity * Higher of .01 or Absolute Value of Entry Price)

Points	(1 for Long position, or -1 for short position) * Quantity * (Exit Price - Entry Price - Entry Commission - Exit Commission) / (Exit Rate * Point Value) / Lot Size
Pips	Forex Instruments: Profit in Ticks * Tick Size Other Instruments: Profit in Ticks
Ticks	Profit in Points / Tick Size

Note:
- Since execution quantity is factored into the PnL calculation in Points, and since other calculations rely on Profit in Points, each calculation mode takes execution quantity into account by extension.
- It is possible to have trades which are technically profitable in percent, but are not profitable based on their point value (or vice versa)

Terms used

Entry	The last Entry execution
Exit	The last Exit execution
Rate	The currency conversation rate used back to the account demonstration (E.g., A rate of 1 means no conversion took place)
Lot Size	Default Forex Lot Size for the account. 1 for non-forex accounts.
Point Value	Instrument value per point define in the Instruments window

10.30.5 Trade Performance Properties

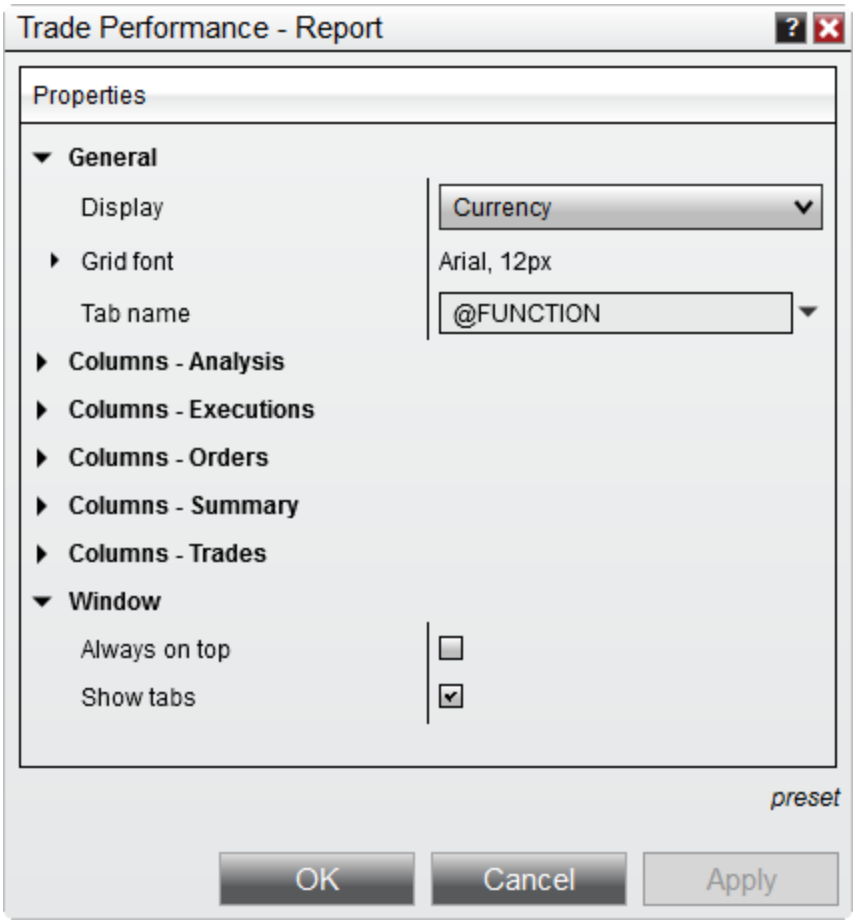
Many of the **Trade Performance** visual display settings can be customized using the **Trade Performance Properties** window.

How to access the Trade Performance Properties window

You can access the Trade Performance properties dialog window by clicking on your right mouse button and selecting the menu **Properties**.

Available properties and definitions

The following properties are available for configuration within the **Trade Performance** Properties window:



Property Definitions

General	
Display	Sets the currency display
Grid font	Sets the font options
Tab name	Sets the name of the tab, please see Managing Tabs for more information.

Columns - Analysis	
Columns - Executions	
Columns - Journal	
Columns - Orders	
Columns - Summary	
Columns - Trades	
Window	
Always on top	Sets if the window will be always on top of other windows.

How to preset property defaults

Once you have your properties set to your preference, you can left mouse click on the "preset" text located in the bottom right of the properties dialog. Selecting the option "save" will save these settings as the default settings used every time you open a new window/tab.

If you change your settings and later wish to go back to the original settings, you can left mouse click on the "preset" text and select the option to "restore" to return to the original settings.

10.31 Trading Hours

Trading Hours Overview

You can access the **Trading Hours** window from within the NinjaTrader **Control Center** window by left mouse clicking on the menu **Tools**, and then selecting the menu item **Trading Hours**

The **Trading Hours** window is used to create and configure **Trading Hour Templates**. **Trading Hour Templates** are set up to contain the session start and end times of a specific market or instrument. NinjaTrader maintains trading hour definitions on the data server and comes predefined with common **Trading Hour Templates** for the most common instruments.

› Using the Trading Hours window

10.31.1 Using the Trading Hours window

Within the **Trading Hours** window, **Trading Hour** Templates hold the session definitions for each day of the week can be created and edited based on any time zone.

▼ Understanding Trading Hour Templates

Trading Hour Templates

A **Trading Hour** Template is a collection of session definitions that can be used anywhere NinjaTrader utilizes data. When a template is applied, any data outside of the times in the session definitions will be ignored. NinjaTrader comes pre-loaded with the most common **Trading Hour** Templates and will also update these automatically from the NinjaTrader data server. You may also create your own custom **Trading Hour** Templates can also be created to suit your needs.

Where Trading Hour Templates can be Applied

Trading Hour Templates can be applied in the following NinjaTrader dialogue windows under the property "**Trading Hours**":

- Chart panel via the Data Series window
- Market Analyzer via customizing columns
- Strategy Analyzer window when configuring backtesting
- Strategies tab of the Control Center when starting a strategy

How to create and edit a Trading Hour Template

Creating a Trading Hour Template

If your desired session settings are not found within the pre-loaded **Trading Hour Templates**, you can create a new template.

To create a **Trading Hour Template**:

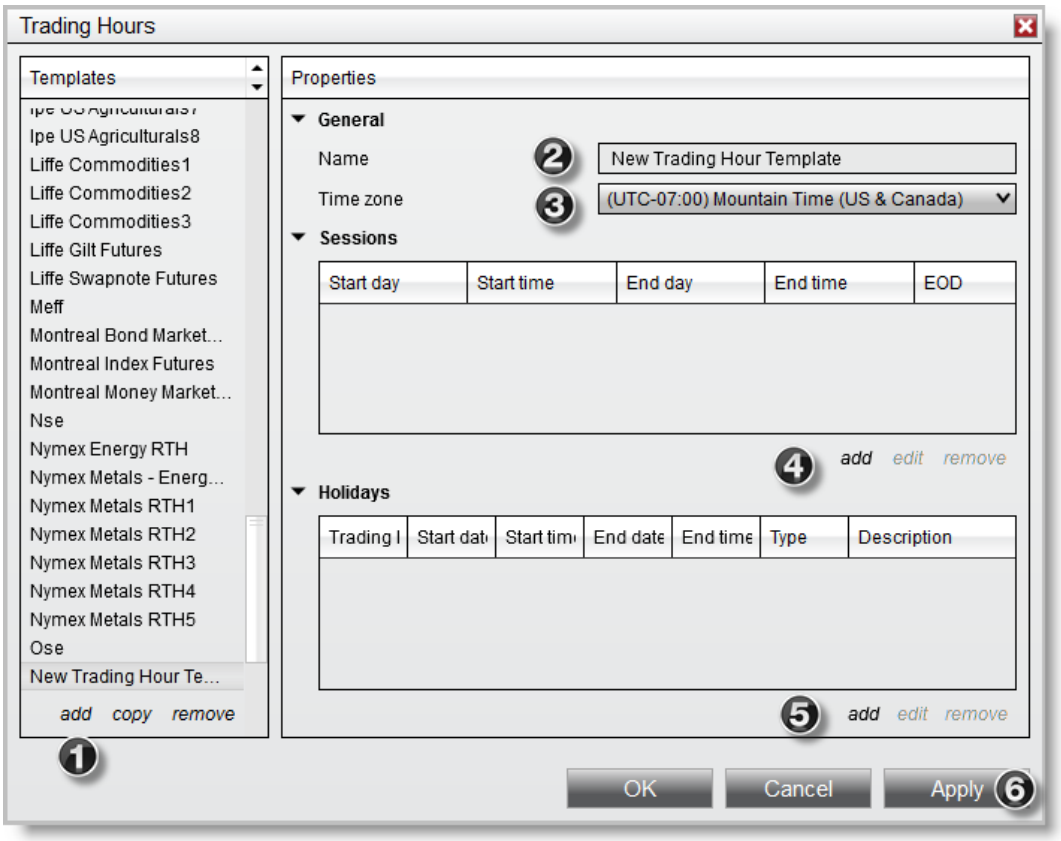
1. Left mouse click on "add"
2. Type in the name of the **Trading Hours Template**
3. Left mouse click on the time zone drop down menu and select the time zone that represents the time inputted in the session definitions
4. Select "add" to add a new session definition, see "**Understanding session definitions**" below for more information. Repeat for as many sessions as required

> **Note**: You can right mouse click on the first session added and select the menu item **Add Monday through Friday** to have NinjaTrader automatically add sessions for Monday through Friday with session definitions based on the selected row.

5. Optionally add any Trading Holidays, see "**Understanding trading holidays**" below for more information. Repeat for as many Trading Holidays as required.

> **Note**: You can right mouse click on the Holidays grid and select Load Holidays from Trading Hours. You will be prompted to select another **Trading Hours Template,** once selected NinjaTrader will import the Holiday session definitions from the selected template.

6. Press the Apply button to save the configured session times in the **Trading Hour Template**.



Working with Trading Hour Templates

A saved **Trading Hour Template** can be selected via the Template section to the left of the **Trading Hours** window. Selecting the template will allow you to configure individual session definitions and trading holiday definitions for that template.

Editing Trading Hour Templates

Trading Hour Templates can be edited in the following ways:

- Left mouse click the "copy" button in the templates section and insert a new template name to copy the current **Trading Hours Template.**
- Left mouse click the "remove" button in the templates section to delete the selected **Trading Hours Template.**

Editing Session Definitions

Individual session definitions can be edited in the following ways:

- Left mouse click on a session definition and press the "edit" button in the sessions section to edit the session.
- Left mouse click on the "remove" button in the sessions section to delete the selected session definition.

Editing Holiday Definitions

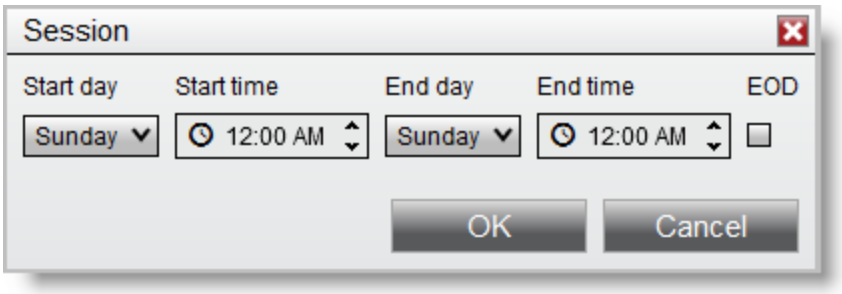
Individual holiday definitions can be edited in the following ways:

- Left mouse click on a trading holiday and press the "edit" button in the holidays section to edit the holiday.
- Left mouse click on the "remove" button in the holidays section to delete the selected session definition.

Understanding session definitions

Understanding Session Definitions

Each session is defined with a start day and time and end day and time. You can have multiple sessions per day, however on the last session of the day you would check mark "EOD(End of Day)". This tells NinjaTrader that this session signifies the ending session for the current trading day and the next session will be counted as the next trading dates session.



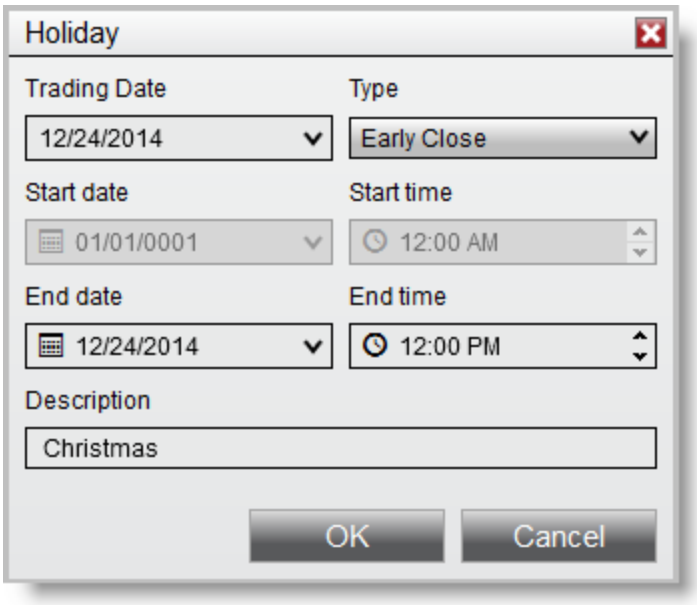
Start day	Sets the Start day of the session definition.
Start time	Sets the start time of the session definition.
End day	Sets the end day of the session definition.

End time	Sets the end time of the session definition.
EOD	Sets if the session is the last session for the trading day.

Understanding trading holidays

Understanding Trading Holidays

NinjaTrader will exclude trading holidays that are defined in the **Trading Hour Template**.



Each Holiday has a **Trading Date**, **Type**, **Start date**, **Start Time**, **End date**, **End time**, and **Description**. The Holidays type will determine what fields are available.

Holiday Types

Full Day	Any sessions between the EOD session of the **Holidays Trading Date** and the prior EOD marker are excluded.

Replace	Replace all session definitions for the **Holidays Trading Date** with the start and end time/date specified.
Early Close	Replace the end time/date for the **Holidays Trading Date** EOD session. (Note: If the end time is before the start time of the EOD session then the EOD session is no longer used and the previous sessions end time/date will be used as EOD and its end time adjusted accordingly.
Late Open	Replace the start time/date for the first session after the **Holidays Trading Date** prior days EOD session.
Modify	Modifies the starting sessions start time and ending sessions end time for the **Holidays Trading Date**. Note: In contracts to "replace" this keeps all existing sessions defined in between the start and end session.

Note: Trade Holidays are automatically updated from the NinjaTrader data server, to report an issue with a trade holiday or a missing holiday please contact platformsupport@ninjatrader.com

10.32 Windows

Windows Overview

NinjaTrader uses shared window controls and interfaces for a wide array of functionality. Details on this shared controls can be found in this section.

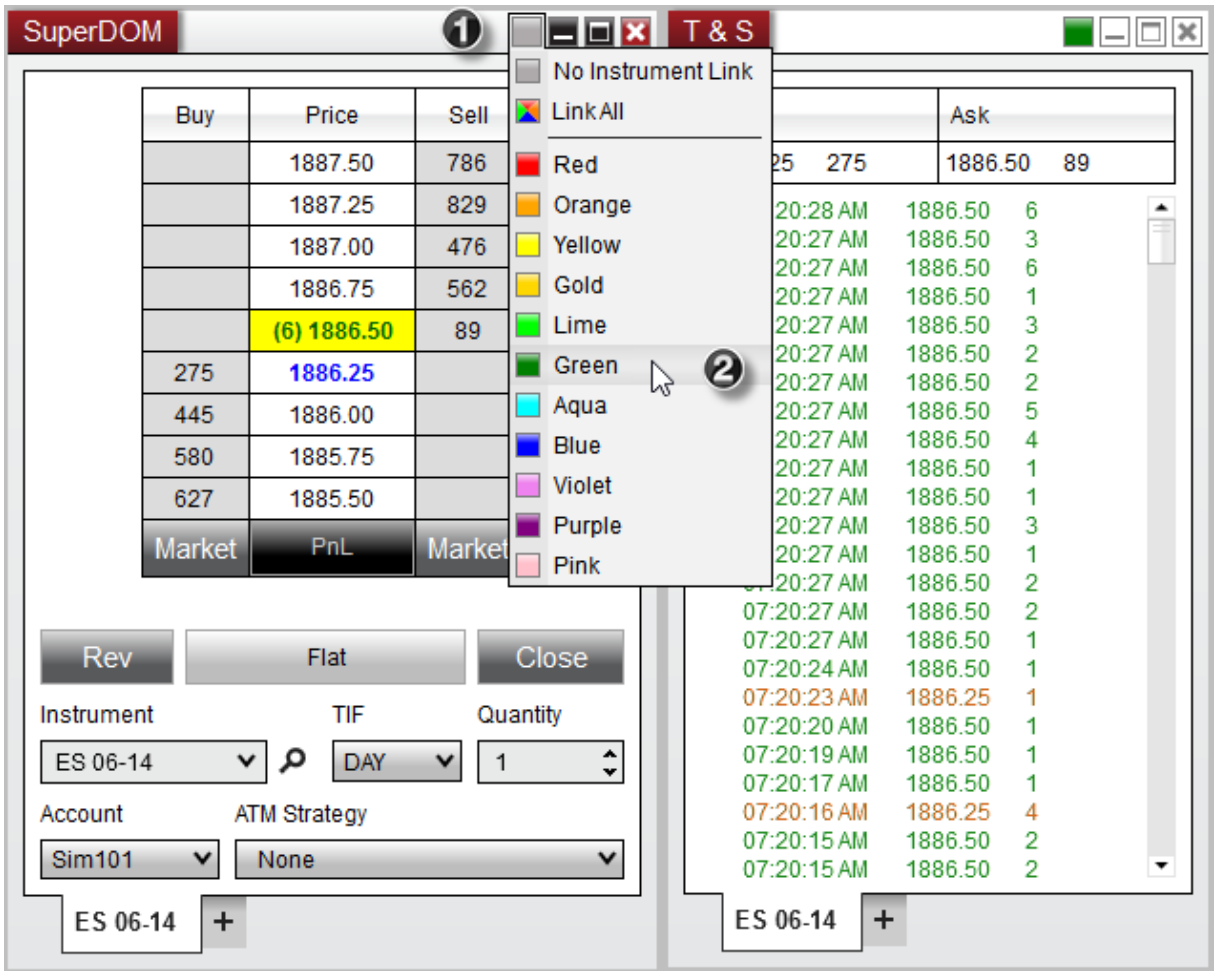
- Linking Windows
- Using the overlay instrument selector
- Using Tabs

Play Video

10.32.1 Using Window Linking

Instrument Link

Charts, Order entry, Alerts, Time and Sales, News, FX Board and Market Analyzer windows all have link buttons in the top right hand corner. Any window that is linked by color (each link button is set to the same color) will receive the same change of instrument request. That means that if you change, or select, an instrument in one window, all other linked windows will also change to that instrument.



Interval Link

Charts has additional link functionality where you can link chart intervals. Meaning that any time you change an interval selection on a chart all linked windows would also change their interval.

Link All

Selecting **Link All** will result in the specified window receiving transmission from windows with any link color selected.

10.32.2 Using the Instrument Selector

Instrument Selector

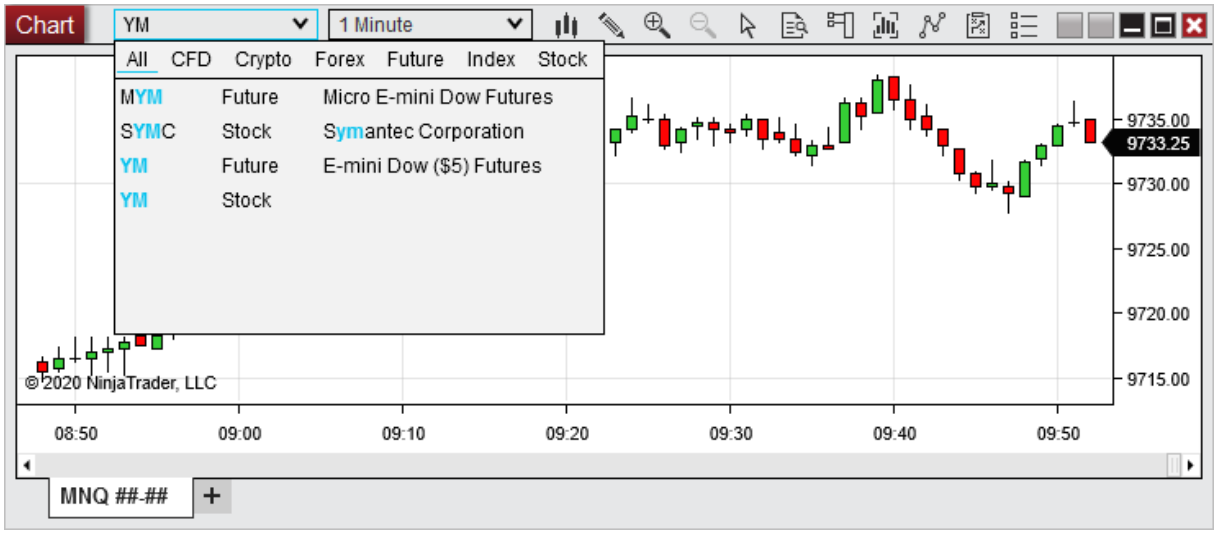
All order-entry and market-data windows feature an **Instrument Selector**, which can be used to quickly select the desired instrument.
This can be accessed either from the **Instrument Selector** directly on the window or in the right-click menu.

Using the Quick Search

To access the **Quick Search**,start typing directly into the **Instrument Selector**.

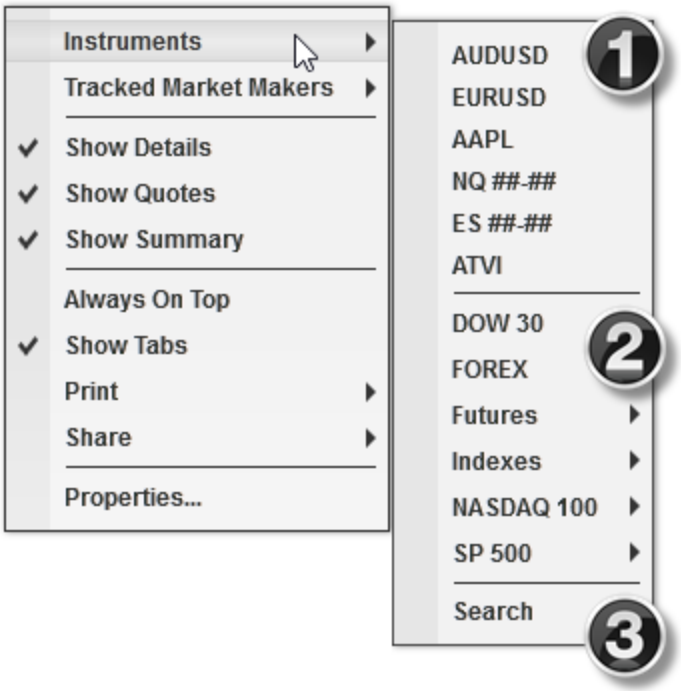
The **Quick Search** will appear and can be used to filter results by instrument type if desired. Last used filter settings are retained. Typing @, ^, or $ will automatically set the filter to the associated instrument type.
Double clicking the instrument from the results will load that instrument. When selecting a futures instrument, it will load the current expiry. Typing in the desired instrument and pressing **Enter** will load the entered instrument (including the desired expiry for futures).

Note: When entering a desired expiry or exchange, no quick search results will appear since it no longer matches the search items. Pressing **Enter** will still load the desired instrument.



Accessing the Instruments Menu

To access the **Instruments** menu, click the down arrow on the **Instrument Selector** or right mouse click in the window in which you wish to apply a new instrument then hover your mouse cursor over the **Instruments** menu item.



The **Instruments** menu is separated into three sections:

1) Pinned or recently viewed instruments

2) Instrument Lists

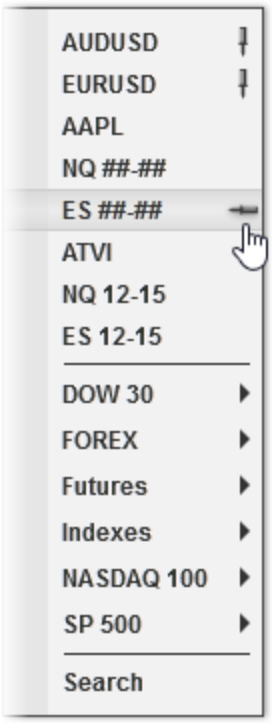
3) Search feature

To access any instrument in the top section, simply left mouse click the instrument name, and it will be applied in the specific window in which you are working. To access an instrument in an instrument list, first hover your mouse cursor over one of the lists displayed (all instrument lists will be displayed), and a list of instruments contained within will appear. You can then left mouse click on any instrument in the list to apply it in the window.

If you do not see your desired instrument listed, click the **Search** menu item to access the Instruments window, in which you can search your entire database of instruments. Use the "Search" dropdown menu near the top of the window to filter results by instrument type, such as stocks or futures, then enter search terms in the text field directly beside it, and the search results will appear as you type. Once you have located your desired instrument, select it in the list of search results, then click **OK** to close the window and apply the instrument. Once you have applied an instrument this way, it will then be saved in the list of recently viewed instruments, and can be pinned from there.

Pinning and Removing Instruments to the Instruments Menu

To pin an instrument in the **Instruments** menu, first view that instrument in any window, in order to add it to the list of recently viewed instruments. Once it is in the list, open the **Instruments** menu in any window, hover your mouse cursor over the instrument you wish to pin, then left mouse click the small icon resembling a push-pin laying horizontally that appears next the instrument name. Pinned instruments will display a vertically standing push-pin icon next to their names.



To remove any item from the list of pinned and recently viewed instruments, first hover your mouse cursor over the instrument you wish to remove, then click the Delete key on your keyboard.

10.32.3 Using the Overlay Instrument Selector

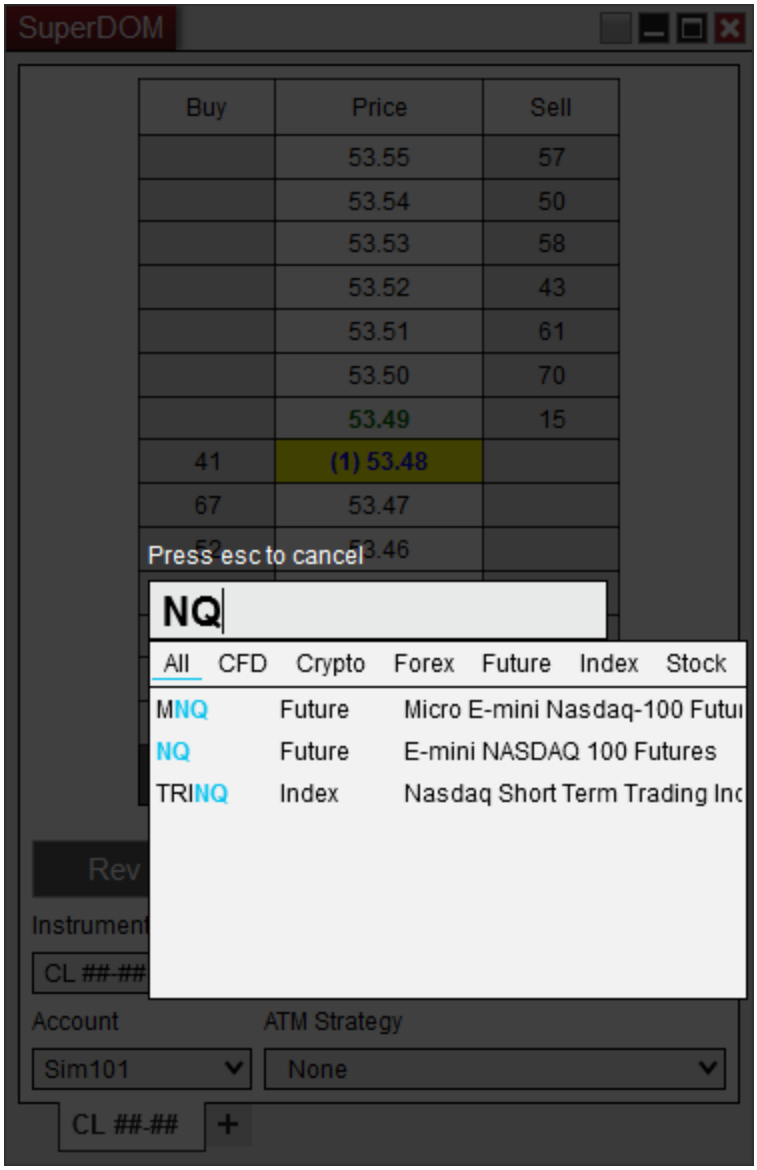
Charts, Level II, Order entry windows, Time and Sales, Instrument Lists, and Market Analyzer windows all have the ability to begin typing in the window to display the **Instrument Overlay Selector**.

Using the overlay instrument selector

The overlay instrument selector is a quick way to change or select an instrument.

- To access the **Overlay Instrument Selector** with the window selected and in focus begin typing on the keyboard the symbol of the instrument you wish to select. In the below image you can see the **Overlay Instrument Selector** over the top of the superDOM window.

- Once the symbol is typed in press "Enter" to complete the instrument selection or you can select the instrument from the **Quick Search** results. See the Using the Instrument Selector section for more information on the **Quick Search**.
- Press the "**Esc**" key on your keyboard to cancel.



Shortcuts available in the Overlay Instrument Selector

Using these shortcuts you can quickly add an additional instrument or switch instruments.

- Typing in a "**+**" at the start of the Instrument Symbol or Time Frame will tell NinjaTrader to add an additional instrument to the current tab if the window supports that.
- Typing in a "**++**" on any tab will open a new tab with that instrument selected.

10.32.4 Using Tabs

Various windows in NinjaTrader are now a tabbed interface, this gives you the ability to have multiple tabs in the same window.

Adding tabs

Adding Tabs

Pressing the **+** tab will create a new **Time & Sales** tab in this window.

Note: With the **Time & Sales** window selected you may also start typing "++" followed by the instrument symbol into the **Overlay Instrument Selector** to quickly create a new tab with that instrument preselected. Example: "++MSFT"

Removing tabs

Removing Tabs

Moving your mouse over the tab handle and selecting the **x** icon to remove that specific tab.

Note: You cannot remove the last remaining tab as you must have at least one tab per window.

Reordering tabs

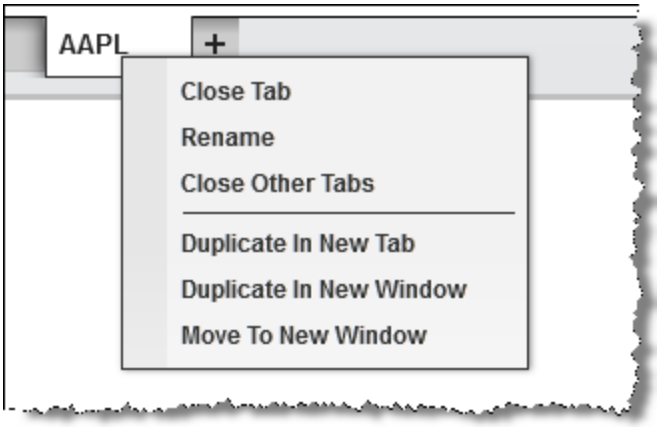
Reordering Tabs

Click and hold to drag the tab into the desired position in the tab area.

Tabs actions

Right Click Menu

Right mouse click on the tab to access the right click menu to perform a tab action.



Close Tab	Removed the selected tab
Rename	Opens the properties window with the tab name property selected for direct editing
Close Other Tabs	Removed all tabs except for the selected tab
Duplicate In New Tab	Duplicates the selected tab into a new tab in the same window
Duplicate In New Window	Duplicates the selected tab into a new tab in a new window
Move To New Window	Moves the selected tab into a new window

Duplicating a window and it's tabs

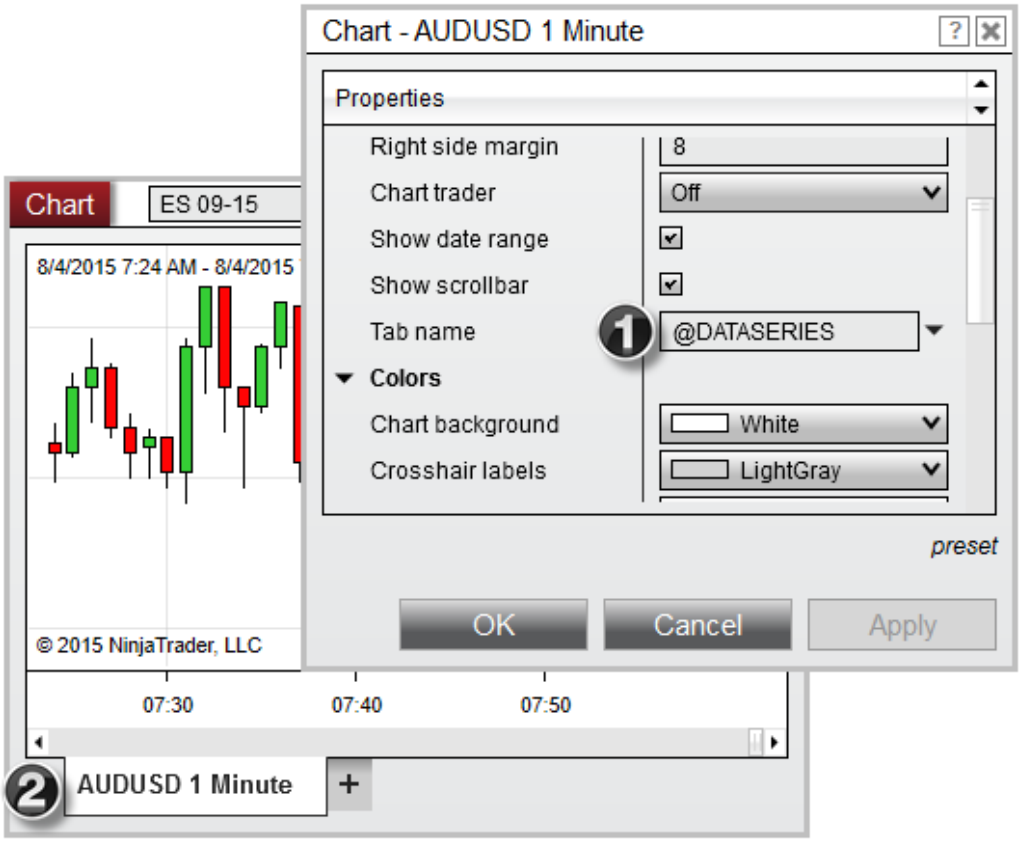
Duplicate Window

Right mouse click on the title bar of a window and selecting Duplicate Window will duplicate the window, including it's tabs.

Tab Name Variables

Using Tab Name Variables

Tabs throughout NinjaTrader allow for the use of pre-defined variables which will dynamically populate tab names with relevant labels, such as the instrument name, period, or account selected in the tab. To use one of the variables listed in the table below, first open the window's **Properties** dialogue, then enter your chosen variable in the "Tab Name" field.



1) The variable "@DATASERIES" has been entered in the "Tab Name" field of the **Chart Properties** window.
2) The "@DATASERIES" variable populates the instrument full name and period in the selected tab.

Notes:

- These variables are case sensitive, meaning that "@instrument" is not the same as "@INSTRUMENT."
- More than one variable can be used together in a single tab name. For example, using "@FUNCTION @ACCOUNT" would list the tab's function and selected account together.

Variable	Value	Applicable To

@INSTRUMENT	Displays the name of the primary instrument displayed in the tab	Level II, Time & Sales, Basic Entry, FX Board, FX Pro, SuperDOM, Order Ticket, Charts
@INSTRUMENT_FULL	Displays the full name of the primary instrument displayed in the tab (adds the expiry for futures contracts)	Level II, Time & Sales, Basic Entry, FX Board, FX Pro, SuperDOM, Order Ticket, Charts
@INSTRUMENT_ALL	Displays the names of all instruments displayed in the tab	FX Board, Charts
@INSTRUMENT_FULL_ALL	Displays the full name of all the instruments displayed in the tab (adds the expiry for futures contracts)	FX Board, Charts
@PERIOD	Displays the period configured on the primary instrument in the tab	Charts
@PERIOD_ALL	Displays the periods configured for all instruments in the tab	Charts
@ACCOUNT	Displays the account selected in the tab	Control Center (Account, Executions, Orders, Positions, Strategies Grids), Basic Entry, FX Board, FX Pro, SuperDOM, Order Ticket, Charts
@FUNCTI	Displays the function of the tab (examples:	All Tabs

ON	"Chart" or "Log")	
@ATM	Displays the selected ATM Strategy in the tab	Charts
@DATASERIES	Equivalent to "@INSTRUMENT_FULL @PERIOD" for the primary instrument in the tab	Charts
@DATASERIES_ALL	Equivalent to "@INSTRUMENT_FULL @PERIOD" for all instruments in the tab	Charts

▼ Switching tabs

Switching Tabs
You can switch between tabs by clicking on the desired tab or by pressing CTRL+Tab on your keyboard.

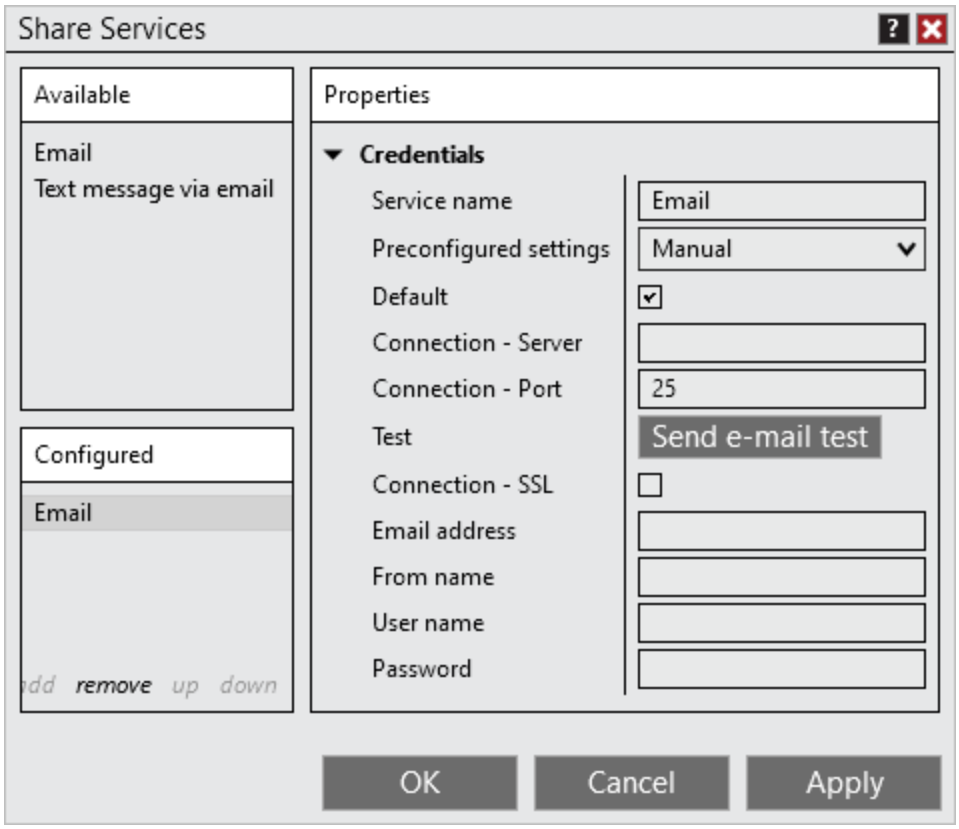
10.32.5 Sharing Content

NinjaTrader support sharing messages and images via **Email** or **Text message via email**.

Setting up Sharing Services

You must first setup your sharing services before you are able to share content.

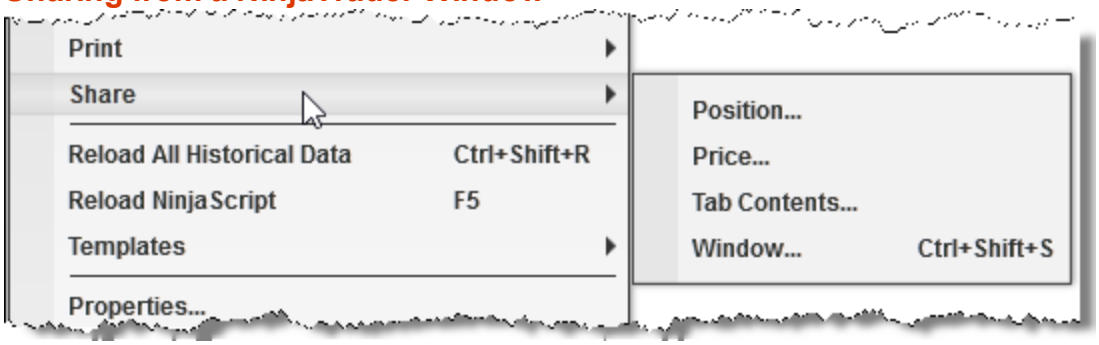
In the NinjaTrader Control Center under the "**Tools**" menu select "**Options**", here in the General category you can click to configure "Sharing Connections".



Select an available sharing service and double click or select "add" to configure that connection.
Once you have completed setup of the sharing service. You can now use this sharing service in NinjaTrader.

Note: The default check box can only be checked for a single account for each Sharing Services. This is the account that is used when any automated process attempts to share something, such as an strategy tweeting a new position is just got into. For more information on the NinjaScript method to share please see the following section of the help guide.

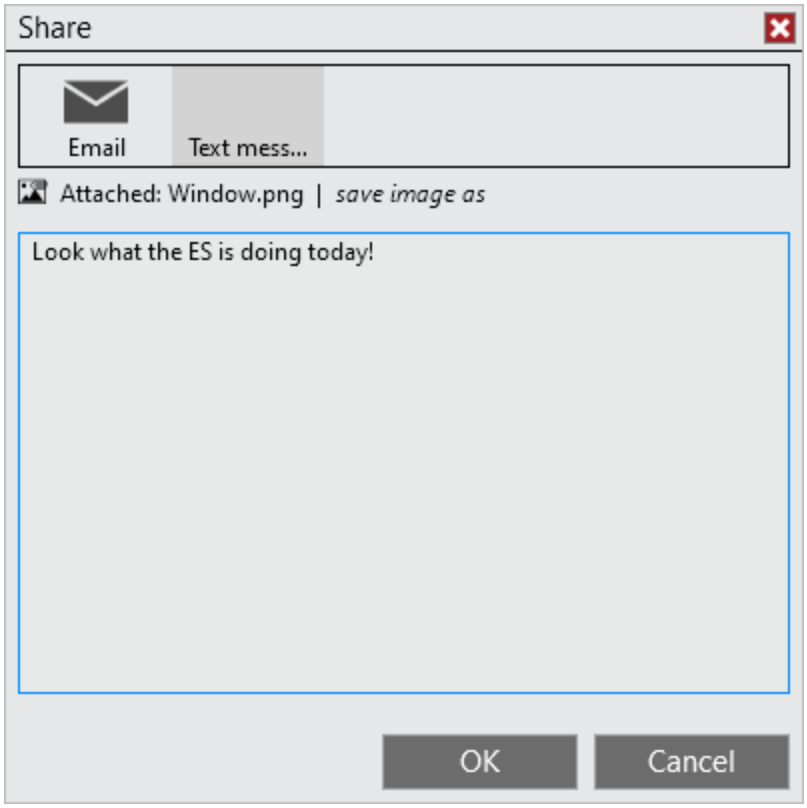
Sharing from a NinjaTrader Window



Right clicking on a NinjaTrader window that has sharing enabled you will see the the "**Share**" menu. On mouse over the "**Share**" all items that you can share will be available for selection. In the screenshot you have the following actions available, however please note that these will change depending on the window context you have right clicked in.

Position...	Opens the share window with the current instrument position pre populated for message.
Price...	Opens the share window with the current instrument price pre populated for message.
Tab Contents...	Opens the share window with a screenshot of the tab attached to the message.
Window...	Opens the share window with a screenshot of the window attached to the message.

Once you make a selection the Share dialog will be launched where you can customize the message and select what service you would like to share too.

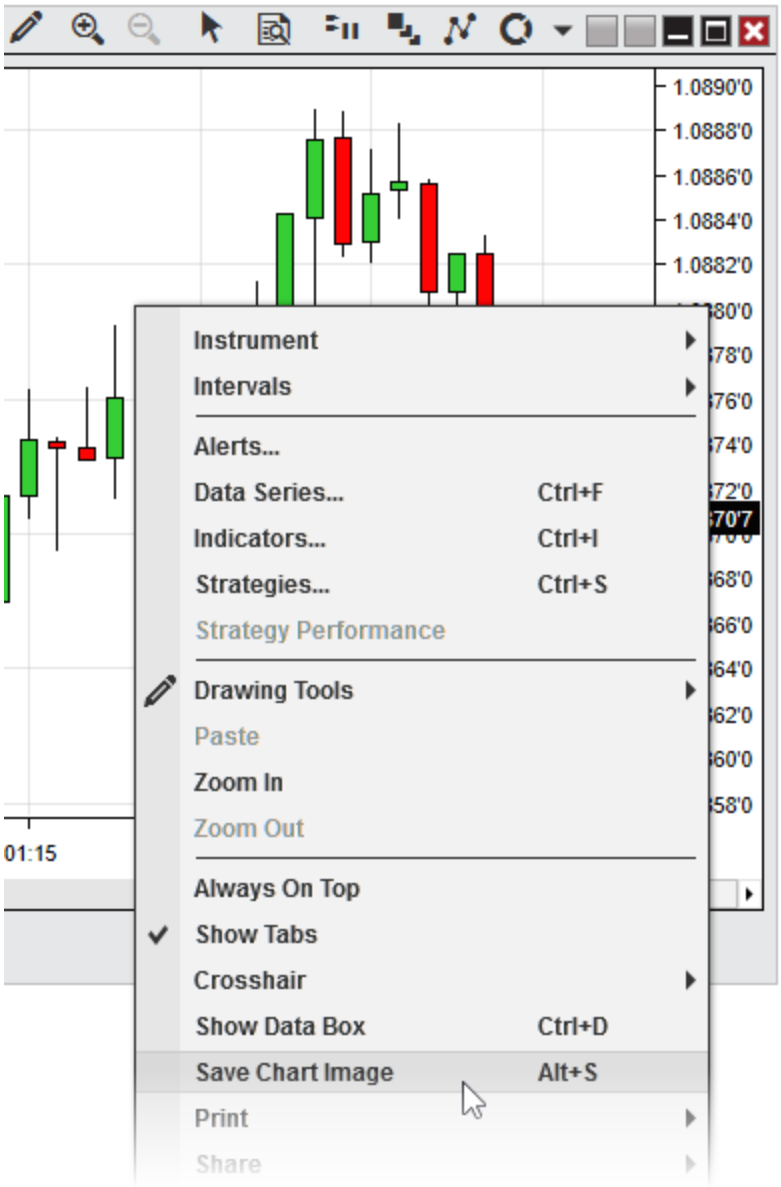


Note: Depending on what the window supports for sharing will change depending on what options you have for sharing for that window.

Saving Chart Images

In addition to sharing directly through NinjaTrader, you can also save images of chart windows to your PC locally, which you can store or share in other ways.

To save a chart image, first right click within the chart canvas area, then click "Save As Image," as seen in the screenshot below:



10.32.6 Printing Content

How to print content in NinjaTrader

NinjaTrader has a generic approach to printing which can be accessed via the right mouse click menu on any print enabled window.

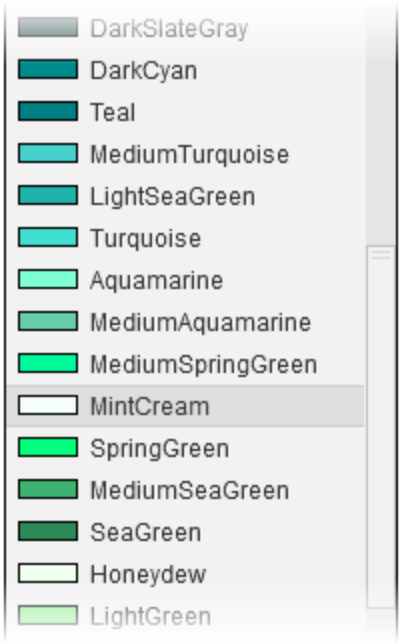
The print options available to you will vary depending on the window you choose to print from, in the screenshot above there are two options:

Tab Contents...	Opens the print dialog to configure print options for printing a screenshot of the Tab
Window...	Opens the print dialog to configure print options for printing a screenshot of the window.

10.32.7 Using Color Pickers

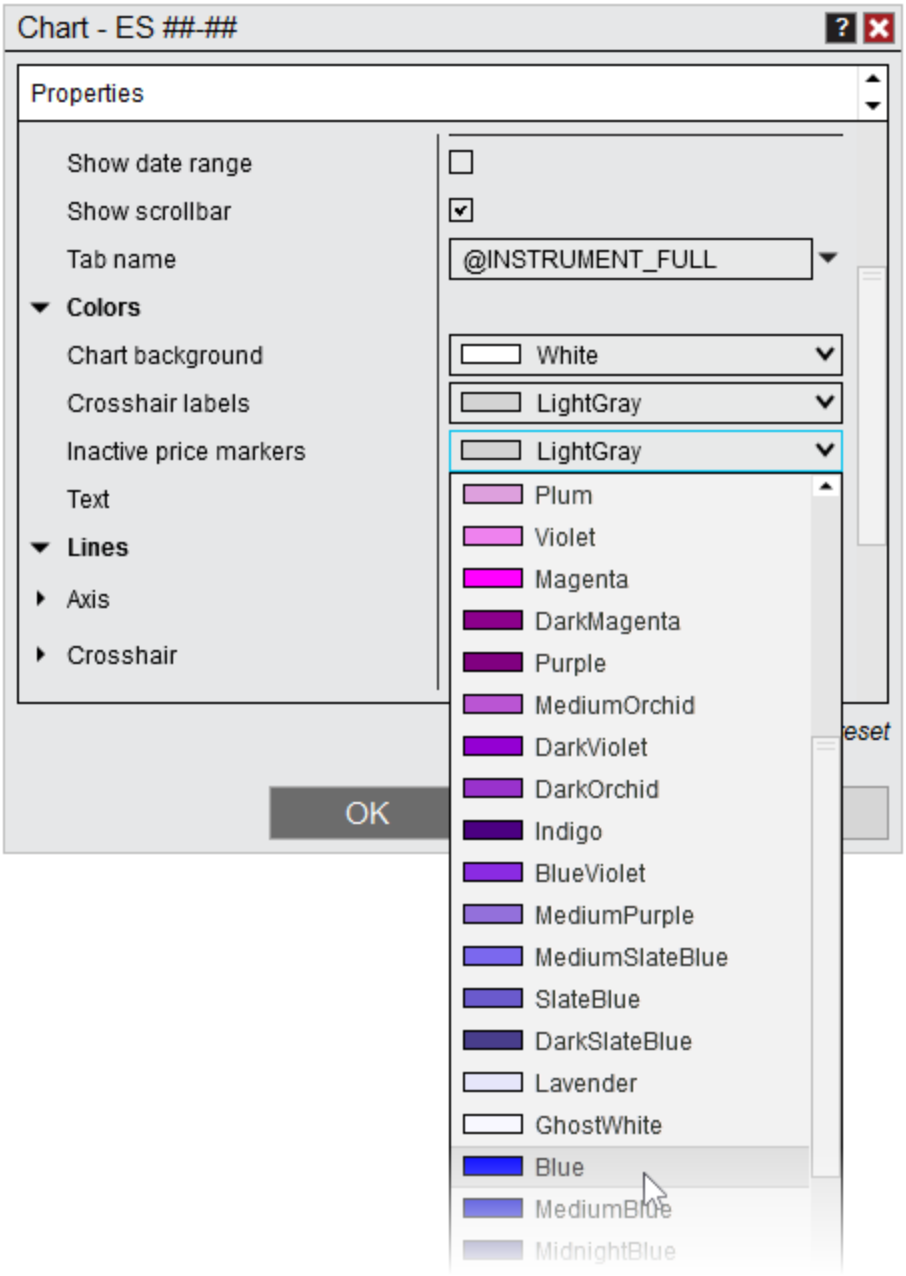
Color Picker

NinjaTrader features a standard **Color Picker** which can be used to quickly apply a set of predefined colors to any feature that allows for configuration of personalized colors (chart bars, indicator plots, Time & Sales rows -- anything which allows a color to be set). You can also use this tool to add custom color by defining either the underlying **Hexadecimal Value**, or by using a comma separated **Red, Green, Blue (RGB)** values.



Accessing a Color Picker

You will find the **Color Picker** in various areas of the product, such as a Properties window, or when setting up a new chart. **Color pickers** function identically to standard drop-down menus, showing a collection of .NET Brushes, or colors, which can be applied to the feature to which a particular **Color Picker** relates. **Brushes** are organized by Hue, and display both the name and a small sample of the Brush color.



Using Custom Colors

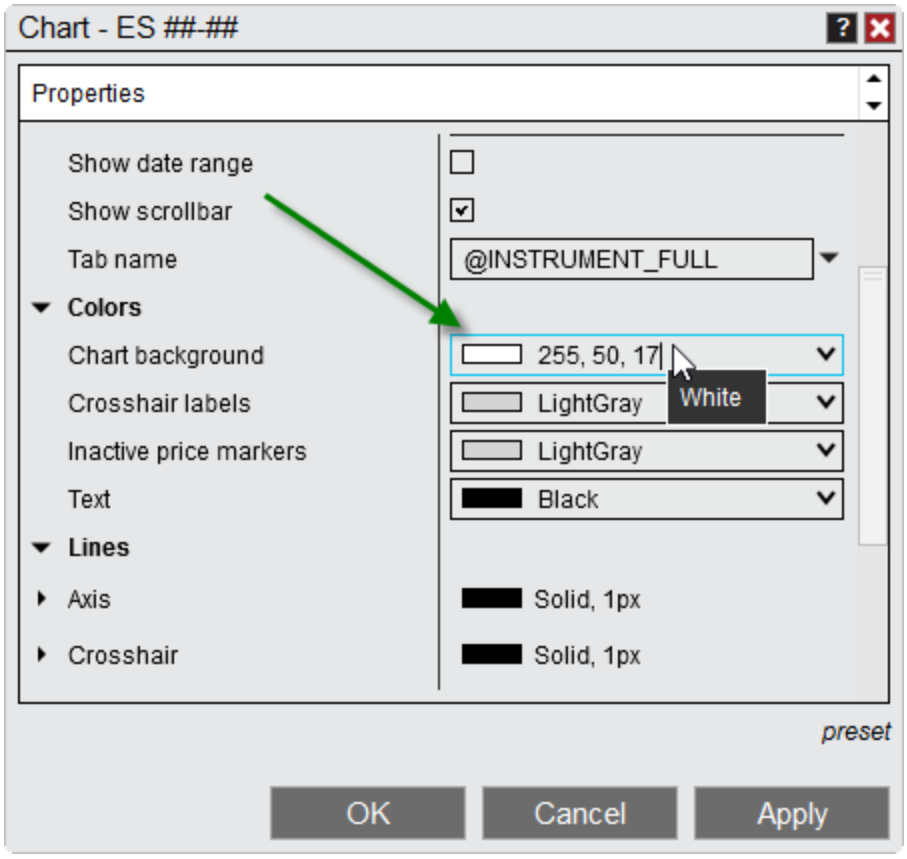
NinjaTrader's **Color Pickers** allow you to enter custom color values not defined in by default by clicking and typing directly into the field, rather than pulling down the menu. Values can be entered in one of 5 formats:

Method	Brush Type	Description

Name	Solid Brush	The name of your desired color (e.g., *Red*, *LimeGreen*, *Khaki*) if it already exists in the Color Picker
RGB (Red, Green, Blue) Value	Solid Brush	A comma-separated RGB value (eg. "*255, 192, 203*" for Pink)
ARGB (Alpha, Red, Green, Blue) Value	Transparent Brush	A comma-separated ARGB value (eg. "*50, 255, 192, 203*" for Pink with 50% Opacity)
Hexadecimal Value (#RRGGBB)	Solid Brush	A hexadecimal value representing a color (eg. *#6A5ACD* for Slate Blue)
Hexadecimal Value (#AARRGGBB)	Transparent Brush	Hexadecimal values can optionally include an Alpha component where 00 is fully transparent and FF is fully opaque (eg., *#806A5ACD*)

Alpha component	Opacity
FF	100%
E6	90%
CC	80%
B3	70%
99	60%
80	50%

66	40%
4D	30%
33	20%
1A	10%
00	0%



The image above shows an RGB value typed in to produce a White color.

Note: Custom colors typed in manually will only apply to the **specific Color Picker** in which they are typed, and will not be available after the next startup. However, colors can be added to all Color Pickers permanently by creating your own skin.

11 NinjaScript

NinjaScript Overview

NinjaScript is a C# based language that allows unlimited extensibility to NinjaTrader.

- Distribution
- Editor
- Educational Resources
- Language Reference

11.1 Code Breaking Changes

The following document is intended as a high level overview of the NinjaScript changes you can expect between NinjaTrader 7 and NinjaTrader 8. For specific information on a particular method or property, you can refer to the dynamically formatted **Code Breaking table** at the bottom of this page. We recommend using the **Filter** and **Sorting** features built into the table, as well checking the **Summary** column and expanding the **Details** section of each entry for general information. Referring to the conveniently linked NinjaTrader 8 and NinjaTrader 7 documentation will provide specific information on syntax, usage, and examples of any new implementation or element names.

Note: Information on this page focuses on **supported (documented)** NinjaTrader methods and properties shared between versions. NinjaTrader 8 has seen a significant increase in supported NinjaTrader code, however if you were using previously **undocumented** NinjaTrader 7 methods or properties, they will **NOT** be covered in this topic. You may be able to find more information on previously **undocumented** methods and properties in the NinjaTrader 8 Help Guide, or our support staff will also be happy to personally point you in the right direction.

Critical: If your product uses **unsupported (undocumented)** elements we strongly urge you to put your scripts through thorough testing to ensure they still behave as expected. There is **NO** guarantee that previously **undocumented** method or property behavior has not changed in the new version of NinjaTrader 8.

For questions or comments, please contact us at platformsupport@ninjatrader.com

▼ Implementation Changes Overview

Initialize(), OnStartUp(), OnTermination()

NinjaTrader 8 has simplified the methods used to set or release various resources during the lifetime of a NinjaTrader object to a single **OnStateChange()** method. This single method is guaranteed to be called for every change in **State** of the object. It is from this method you can monitor the progression of the object throughout its lifetime in order to setup various resources, set properties, or take action the moment **State** has changed. This method also exposes a **State** variable which can be used in various other methods, such as **OnBarUpdate(),** in order to tell your indicator or strategy to process data depending on the actual **State** of the object.

For example, pushing settings to the UI, or setting initial values for public properties can now be done use **OnStateChange()** when the state has reached **State.SetDefaults**:

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      // set the default properties
      Name = "My Indicator";
      Fast = 10;
      Slow = 25;
      IsOverlay = true;
      IsAutoScale = true;
   }
}
```

If you have custom resources that need to be setup before the NinjaTrader object is active and processing data, instead of using the **Initialize()** method, you can now set this up once the **OnStateChange()** method has reached **State.Configure** state:

```
protected override void OnStateChange()
{
   if (State == State.Configure)
   {
      // Add a 5 minute Bars object to the strategy
      AddDataSeries(Data.BarsPeriodType.Minute, 5);
      // setup a custom data series
      spread = new Series<double>(this);
      // setup a 20-period EMA indicator
      ema = EMA(20);
      // add indicator to strategy for visual purposes
      AddChartIndicator(ema);

   }
}
```

NinjaTrader 7 had no concept to detect when your NinjaTrader object was transitioning from processing Historical data to processing Real-time data. Now with NinjaTrader 8, the **OnStateChange()** method provides a **State.Transition** state which will notify you when this change is about to occur. If your NinjaTrader 7 indicators or strategies were using custom methods to try to detect this transition, your custom methods may be refactored under this new state:

```
protected override void OnStateChange()
{
   if (State == State.Transition)
   {
      Print("We're going to real-time data...");
      // setup your real-time data resources here
   }
}
```

When your NinjaTrader object is shutting down, and you need clean up any custom device resources, instead of using **OnTermination()**, you should now clean up these resources once the **OnStateChange()** method has reached the **State.Terminated** state:

```
protected override void OnStateChange()
{
   if (State == State.Terminated)
   {
      // release any device resources
      if(myTimer != null)
         myTimer = null;
   }
}
```

NinjaTrader previously used a **Historical** bool property to notify when an indicator or strategy bar was being processed historically or real-time. The NinjaTrader 8 **OnStateChange()** approach has now introduced a class level variable **State** where you can check for **State.Historical** or **State.Realtime** in any of the other event methods which will allow you to take action depending on the desired state:

```
protected override void OnBarUpdate()
{
   // only process on real-time data
   if (State == State.Historical)
      return;

   else if (State >= State.Realtime)
       // rest of logic
}
```

Strategies, Orders, and Accounts

Low level access has been provided to allow more flexibility with the information pertaining to trade data.

- IOrders, IExecution, and IPosition interfaces have all been replaced directly with the corresponding object
- The signatures of the related NinjaScript events have changed to match the NinjaTrader internal Update events
- Methods now return and update with the object instance generated, instead of the previously used interface

Tip: Since NinjaTrader 8 now exposes the direct **Order** object, rather than an

IOrder interface, it is possible to receive **null object reference errors** if you attempt to access an order object before the entry or exit order method has returned. To prevent these situations, it is recommended to assign your strategies **Order** variables in the **OnOrderUpdate()** method and match them by their **signal name** (`order.Name`). Please see the example beginning on line #22 below for demonstration of assigning order objects to private variables.

```
Order myOrder = null;

protected override void OnBarUpdate()
{
   if (Position.MarketPosition == MarketPosition.Flat &&
myOrder == null)
      EnterLongLimit(Low[0], "Entry");

   if (myOrder != null)
   {
      Print(myOrder.OrderState);

      if (myOrder.OrderState == OrderState.Cancelled ||
myOrder.OrderState == OrderState.Filled)
         myOrder = null;
   }
}

protected override void OnOrderUpdate(Cbi.Order order,
double limitPrice, double stopPrice,
   int quantity, int filled, double averageFillPrice,
   Cbi.OrderState orderState, DateTime time, Cbi.ErrorCode
error, string comment)
{
   // compare the order object created via EnterLongLimit
by the signal name
   if (myOrder == null && order.Name == "Entry")
   {
      // assign myOrder to matching order update
      myOrder = order;
   }
}
```

Data Series

Previously there had been type specific Data Series implementations (e.g., IntSeries, TimeSeries, BoolSeries, etc). Now there just is a template Series<T> class which could be used generically and even allows support for additional types:

```
Series<double> mySeries = new Series<double>(this);
Series<DateTime> myTimeSeries = new Series<DateTime>(this);
```

The **DataSeries.Set()** method used to assign Data Series or Plot values has been removed and values can now be stored using a single assignment operator:

```
protected override void OnBarUpdate()
{
   // set public plotting data series close value of
current bar
   MyPlot[0] = Close[0];
   // set custom Series<DateTime> to time value of current
bar
   myTimeSeries[0] = Time[0];
}
```

Drawing

The DrawObjects used in NinjaTrader have received a number of changes:

- All DrawObjects have been moved to a separate **NinjaScript.DrawingTools** namespace and are properly known as **DrawingTools**
- Drawing Methods called from indicators or strategies have been moved to a new static partial **Draw** class
- Drawing Methods have all received a signature change which requires you specify the owner (object) which drew the **DrawingTool** object
- Drawing Methods no longer returns an interface but rather an instance of the **DrawingTool** object itself
- Drawing Methods now use the System.Windows.Media.Brushes class instead of the System.Drawing.Color structure

Tip: DrawingTools are now completely unprotected and you can review their source code from the DrawingTools folder of the NinjaScript Editor's explorer

menu

```
// example syntax
Draw.Line(NinjaScriptBase owner, string tag, int
startBarsAgo, double startY, int endBarsAgo, double endY,
Brush brush)

// example usage
Draw.Line(this, "tag1", true, 10, Low[0], 0, Brushes.Red);
```

Casting a member of the **DrawObjects[]** collection must be done safely using the "as" keyword, otherwise you may receive exceptions at run time should another instance of the object (e.g., matching tag) exist from another owner:

```
NinjaScript.DrawingTools.Line myLine = DrawObjects["tag1"]
as DrawingTools.Line;
```

DrawingTools anchor fields such as "Time" or "Price", etc have been moved to a **ChartAnchor** object owned by the drawing tool, rather than a direct field on the drawing object interface. Please refer to the NinjaTrader 8 documentation for specific changes for each drawing tool:

```
double linePrice = myLine.StartAnchor.Price;
```

Objects which previously used **System.Drawing.Font** now uses new **NinjaTrader.Gui.Tools.SimpleFont** class:

```
Gui.Tools.SimpleFont myFont = new
Gui.Tools.SimpleFont("Arial", 12);
```

Properties and other methods/objects which previously System.Drawing.Color structure now use the System.Windows.Media.Brushes class:

```
BackBrush = Brushes.Blue;
```

Note: For custom **Brush** objects, it is important to .**Freeze()** the **Brush** due to the multi-threaded architecture of NinjaTrader 8. Please be sure to review the new information on using Brushes

Namespaces

The NinjaTrader 7 namespaces **NinjaTrader.Indicator** and **NinjaTrader.Strategy** have been renamed and moved to single **NinjaTrader.NinjaScript** namespace

```
//This namespace holds indicators in this folder and is
required. Do not change it.
namespace NinjaTrader.NinjaScript.Indicators
{
   public class MyCustomIndicator : Indicator
   {
   }
}

//This namespace holds Strategies in this folder and is
required. Do not change it.
namespace NinjaTrader.NinjaScript.Strategies
{
   public class MyCustomStrategy : Strategy
   {
   }
}
```

Partial Classes (Porting methods and properties from UserDefinedMethods.cs)

NinjaTrader 7 used a "UserDefinedMethods" class to define methods to be used across multiple NinjaScript indicators or strategies. In NinjaTrader 8, these pre-built partial classes have been removed to reduce a number of issues which could result from users sharing their UserDefinedMethods.cs files, or overwriting their existing files with copies from a new vendor. Partial classes are now best built manually and saved in the C:\Users\<user>\Documents\NinjaTrader 8\bin\Custom\AddOns folder.

Warning: If a partial class is saved in one of the folders used for specific NinjaScript objects other than AddOns (e.g., Indicators folder), auto-generated NinjaScript code may be appended to the end of the class by the NinjaScript Editor when compiled, which will cause a compilation error. Saving these files in the AddOns folder will ensure they are still accessible and will not generate code which may be cause conflicts.

You can use the template below as a starting point to create your partial class. If your partial class needs to inherit from a parent class, you can append the name of your desired parent class after the " : " to change the inheritance.

Note: Methods within your partial classes should be using the "public" modifier.

Partial Class Example Template

```
namespace NinjaTrader.NinjaScript.Indicators
{
    public partial class MyMethods // : parent class to
inherit from
    {
        //Sample method which calculates the delta of two
prices
        public double calculateDelta(double firstPrice,
double secondPrice)
        {
            return Math.Abs(firstPrice - secondPrice);
        }

        //Sample method which prints Position information
        public void printPositionInfo(Position position)
        {
            Print(String.Format("{0}: {1} {2} at {3}",
position.Instrument, position.Quantity,
position.MarketPosition, position.AveragePrice));
        }

    }
}
```

Below is an example of using one of the methods in this partial class from within an Indicator:

Partial Class Usage

```
protected override void OnBarUpdate()
{
    if (CurrentBar < 1) return;

    // Use the static calculateDelta method to calculate
the difference between the close of each bar
    double delta = MyMethods.calculateDelta(Close[0],
Close[1]);

    Print(delta);
}
```

Tip: At the time of the Beta implementation, the NinjaScript Editor does **NOT** include a partial class generator wizard, as it does for core NinjaScript Types such as Drawing Tools, Market Analyzer Columns, or Strategies. However, we are currently tracking a suggestion to implement a wizard for partial classes, under ID # **SFT-341**. Please feel free to contact platformsupport@ninjatrader.com if you would like to add your vote for this enhancement.

Prevention of Redundant Data Loading

In NinjaTrader 7, multiple Data Series could be added within a script, such as an indicator, and that script could then be hosted by another script, such as a strategy. While this is still possible in NinjaTrader 8, there is a new safeguard in place to prevent redundant data loading in both the hosting script and the hosted indicator.

When hosting an indicator which adds Data Series programmatically, the hosting script must include the same calls to the AddDataSeries() method as the hosted script. Without this, an error will result, which reads *"A hosted indicator tried to load additional data. All data must first be loaded by the hosting NinjaScript in its Configure state."* Without this safegaurd in place, it would be possible for unnecessarily large amounts of data to be loaded concurrently, as would be the case in a direct call to an indicator method on each OnBarUpdate(). By adding the calls to AddDataSeries() to the hosting script, you can ensure that the data is loaded when needed. Also, when this is done in the hosting script, all identical

calls to AddDataSeries() in the hosted script will be ignored, as the data is already available.

The examples below show this in action:

Hosted Indicator Loads Additional Data

```
public class MyCustomIndicator : Indicator
{
   protected override void OnStateChange()
   {
      if (State == State.Configure)
      {
            AddDataSeries("AAPL", BarsPeriodType.Day, 1);
            AddDataSeries("EURUSD", BarsPeriodType.Minute,
15);
        }
   }
}
```

Hosting Strategy Mirrors AddDataSeries() calls

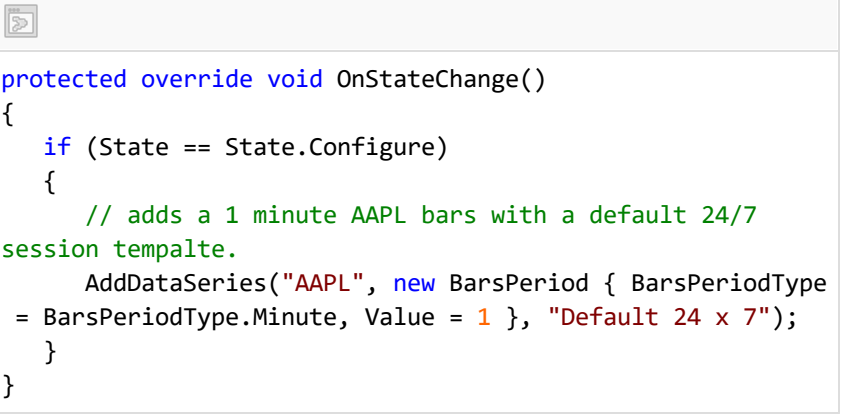
```
public class MyCustomStrategy : Strategy
{
   // Define a MyCustomIndicator
    MyCustomIndicator myIndicator;

   protected override void OnStateChange()
   {
      if (State == State.Configure)
      {
         // Instantiate the MyCustomIndicator and add it to
the chart
         myIndicator = MyCustomIndicator();
         AddChartIndicator(myIndicator);

         // These calls to AddDataSeries() mirror the calls
in the hosted indicator
         AddDataSeries("AAPL", BarsPeriodType.Day, 1);
         AddDataSeries("EURUSD", BarsPeriodType.Minute,
15);
      }
   }
}
```

Bars with 0 Volume

In previous versions, the NinjaTrader core was designed to replace a tick with a volume of 0 with a volume of 1. This resulted in all ticks having a volume value of at least 1. NinjaTrader 8 has removed that design policy and will now allow ticks with a volume of 0 to be processed. This policy change may require logic changes to any custom bar types, indicators, or strategies which may have previously assumed volume would always be greater than 0.

Multi-Series default "Trading Hours" templates

The default behavior in NinjaTrader 8 will ensure that a bars series added to a script using AddDataSeries() will use the same "TradingHours" template as the primary series configured by the user. In contrast, the NinjaTrader 7 behavior was highly dependent on a number of variables. We have updated this behavior to help with consistences and synchronization issues between multiple series; however if you your script relies on two times frames using different trading hours templates, you may consider using one of the new **tradingHours** string overloaded used in AddDataSeries():



```
protected override void OnStateChange()
{
   if (State == State.Configure)
   {
      // adds a 1 minute AAPL bars with a default 24/7
session tempalte.
      AddDataSeries("AAPL", new BarsPeriod { BarsPeriodType
 = BarsPeriodType.Minute, Value = 1 }, "Default 24 x 7");
   }
}
```

Miscellaneous

All of the NinjaTrader 7 reference samples posted in our support forum have been updated to demonstrate NinjaTrader 8 functionality. Please be sure to check the reference sample section to see other undocumented features and concepts which may not have been covered in the help guide:

Official NinjaScript reference code samples

There are several other changes to implementation which are not covered in detail on this overview, please see the code breaking changes table at the bottom of this page which will compare the implementation changes between both versions.

Signature Changes Overview

Signature

A large number of the NinjaTrader methods which were available in NinjaTrader 7 have remained largely the same and should not generate any errors on compilation. However there are a handful of existing methods signatures which have been updated in NinjaTrader 8 in order to fit within new framework which you would need to be aware of in order to transfer these functions from NinjaTrader 7 to NinjaTrader 8. In most cases, the fundamental argument type has been restructured, which may result in compile errors depending on the type of object that is being used within the methods signature.

Tip: Methods may now have additional signatures which add functionality which was not previously available. Be sure to check the NinjaTrader 8 documentation which will cover all the available signatures available.

Name Changes Overview

Renamed

During the NinjaTrader 8 development process, one of our goals to make sure that our core framework matched various coding standards which have been set out in the industry. As a result of meeting these coding standards, many NinjaTrader methods and properties needed to been renamed. While the functionality of these methods and properties remains the same, we chose to rename these variables to follow a semantically context specific naming convention which is generally agreed upon to favor readability. We feel that the renaming of these properties and methods more explicitly describes the intended function to the developer who may be reviewing code. The largest number of changes is in response to the name convention of bools, where they now follow a more strict verb-adjective or verb-noun structure.

For an example:

- The property **FirstTickOfBar** may have been hard to distinguish precisely what it represented without having to look up documentation. In NinjaTrader 8, this property has been renamed to **IsFirstTickOfBar**, which now gives this property a more readable identifier name when you read this line of code as "*is the first tick of bar true?*"

- Another example is the case of **BarsSinceEntry()** which was renamed to **BarsSinceEntryExecution()**, which now specifies that this method is looking for an entry *execution*.
- NinjaTrader 7 sometimes had methods or properties which shared names, but references different data or actions. For example **Add()** could have been used in reference to adding **DataSeries** to a script, adding a **Plot**, or adding a **Line**. To be more specific, NinjaTrader 8 has renamed these to **AddDataSeries()**, **AddPlot()**, and **AddLine()** respectively.
- There may be cases where the property or method name has changed simply because the type of data it interacted with has changed. (e.g., **BarColor** vs. **BarBrush**)
- There are other cases where properties may have used unnecessary brevity and was renamed to favor readability (e.g., **AvgPrice** vs **AveragePrice**)

These are just a few examples of the many name changes found in NinjaTrader 8 and some of the rational behind the number of these changes. For simplicity, you will find a list of all the renamed properties in the table at the bottom of this document by filtering by the "Renamed" keyword.

Code Breaking Table

Below you will find a reference table which lists all of the supported NinjaScript changes between NinjaTrader 7 and NinjaTrader 8.

11.2 NinjaScript Best Practices

There are some best practices to be aware of when developing NinjaScript classes. The following tables present a non-exhaustive list of considerations to keep in mind when designing and implementing your code.

> **Note**: NinjaTrader is multi-threaded and event driven. Always assume that any of the methods you implement in NinjaScript could be called from another thread.

▼ State management practices

Managing Resources

The OnStateChange() method is called anytime there has been a change of State and can be used to help you setup, manage, and destroy several types of resources. Where these values are setup is highly dependent on the kind of resource you are using. The section below will cover how to manage various resources throughout different states.

Setting Default UI Property Grid values

Reserve **State.SetDefaults** for defaulting any public properties you wish to have exposed on the UI property grid. You should also use this State for setting default desired NinjaScript property behavior which can be overridden from the property grid (e.g. Calculate, IsOverlay, etc.). For Plots and Lines you wish to configure, AddPlot(), AddLine() should also have their default values set during this State

> **Why:** Public values of the NinjaScript object in **SetDefaults** are pushed to the UI property grid for an opportunity to change settings of your object.

Best practice

```
protected override void OnStateChange()
{
   // these are the values that show up as default on the
UI
   if (State == State.SetDefaults)
   {
      Calculate = Calculate.OnPriceChange;
      IsOverlay = false;

      Period = 50;

      AddPlot(Brushes.Blue, "Plot Value");
      AddLine(Brushes.Gray, 100, "Threshold");
   }
}
```

For public properties you do **NOT** wish exposed to the UI property grid, set the Browsable attribute to false:

Best practice

```
[Browsable(false)] // prevents from showing up on the UI
property grid
public int Communicator { get; set; }
```

On indicators, properties you wish to set from other objects, set the NinjaScriptPropertyAttribute:

Best practice

```
[NinjaScriptProperty] // can now call MyIndicator(20) from
another object
public int Period { get; set; }
```

The default behavior is to serialize any public properties and fields to a Workspace or Template file when saving. However, not all objects can be serialized - or you may wish to exclude a property from being saved and restored. For these scenarios, set the XMLIgnore attribute to the property:

Best practice

```
[XmlIgnore]  // removes from serialization
public Brush DownBrush
{ get; set; }
```

As a best practice as well, your NinjaScript should not have any public fields, since those would get serialized as well - which means their state would be persisted, which in turn could lead to unexpected outcomes.

Tip: See the Working with Brushes section of the Help Guide for information on properly serializing brushes

Calculating run-time object values

Do not attempt to do advanced calculations or try to access object references in **State.SetDefaults**. This State should be kept as lean as possible, and any calculation logic should be delayed until at least **State.Configure**

Why: Your object will be called in situations you may not be expecting. You can read more about this subject on Understanding the life cycle of your NinjaScript objects

Practice to avoid

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
       // logic could take longer than desired as the list
of indicator names is populated
      for (int i = 0; i <= array.length; i ++)
         DoWork(i);

      // possible null reference exception since TickSize
is not set yet
      Period = 5 * TickSize;
   }
}
```

Best practice

```
protected override void OnStateChange()
{
   // Complex operations should be delayed to >=
State.Configure
   if (State == State.Configure)
   {
      for (int i = 0; i < = array.length; i ++)
         DoWork(i);
   }

   // information related to market data is not available
until at least State.DataLoaded
   else if (State == State.DataLoaded)
   {
      Period = 5 * TickSize;
   }
}
```

Setting class level variables

Do not set variables at the class level unless they are constant. You should delay setting or resetting variables until the **State** has reached **State.Configure**. You can use `const` keyword to differentiate values which do not change from variables which do change.

Why: Waiting to set up and define resources until the object has been configured ensures that values not set up and declared prematurely.

Best practice

```
// value is always 5, it can be made constant and declared
at the class level
private const int multiplier = 5;

// these values can change, may be better to delay setting
until State.Configure
private int counter;
private List<int> myList;

protected override void OnStateChange()
{
   if (State == State.Configure)
   {
      counter = 0;
      myList = new List<int>();
   }
}
```

Resetting class level variables for Strategy Analyzer Optimization

To take advantage of performance optimizations, developers may need to reset class level variables in the strategy otherwise unexpected results can occur.

Why: When optimizing a strategy, instances may or may not be recycled depending on the strategy IsInstantiatedOnEachOptimizationIteration setting.

Best practice

```
// examples of fields which need to be reset
private double myDouble;
private bool myBool;
private DateTime myDateTime;
private Order myOrderObject;
private Brush myBrushObject;
private Array myIntArray;
private List<object> myList;
private SMA mySMAIndicator;
private Series<double> mySeries;

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      // disabled to take advantage of performance gains
      // However any strategy state that would be mutable
after State.SetDefaults needed to be reset for the next
run.
      IsInstantiatedOnEachOptimizationIteration = false;
   }
   else if (State == State.Configure)
   {
      // Since these values are not dependent on bars, they
can be reset as early as State.Configure
      myDouble = double.MinValue;
      myBool = false;
      myDateTime = DateTime.MinValue;
      myOrderObject = null;
      myBrushObject = null;

      if (myIntArray != null)
         Array.Clear(myIntArray, 0, myIntArray.Length);
      else
         myIntArray = new int[20];

      if (myList != null)
         myList.Clear();
      else
         myList = new List<object>();
   }

   else if (State == State.DataLoaded)
   {
      // Since these values do are dependent on bars, they
should only reset during State.DataLoaded
      mySMAIndicator = SMA(14);
```

Accessing properties related to market data

Do not attempt to access objects related to instrument market data until the **State** has reached **State.DataLoaded**

> **Why**: Waiting to access objects that depend on market data until **DataLoaded** prevents access errors in all scenarios

Best practice

```
protected override void OnStateChange()
{
   if (State == State.DataLoaded)
   {
      // these objects and their related members are not
available until State.DataLoaded
      Print(Bars.Count);
      Print(Instrument.FullName);
      Print(BarsPeriod.BarsPeriodType);
      Print(TradingHours.TimeZone);
      Print(Input);
   }
}
```

> **Note**: All additional data series must be added in **State.Configure** (this includes series that any hosted script potentially needs as well - more info). Since objects such as Instrument, BarsPeriod, TradingHours, etc. are **NOT** guaranteed to be available until **State.DataLoaded**, you cannot reliably use the primary instrument properties as arguments in AddDataSeries(). Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided. In some cases, you may be able to use a BarsRequest() to obtain market data for other instruments and intervals.

Setting up resources that rely on market data

For objects which depend on market data, delay their construction until the **State** has reached **State.DataLoaded**

> **Why**: Waiting to construct objects that depend on market data until **DataLoaded** ensures that their underlying input contains significant values in all scenarios.

Best practice

```
// these resources depend on bars, wait until
State.DataLoaded to instantiated
private EMA myEMA;
private Series<double> mySeries;
private SessionIterator mySessionIterator;

protected override void OnStateChange()
{
   if (State == State.DataLoaded)
   {
      myEMA = EMA(20);
      mySeries = new Series<double>(this);
      mySessionIterator = new SessionIterator(Bars);
   }
}
```

Accessing element on the UI

For objects which exist on the UI (e.g., ChartControl, ChartPanel, ChartBars, NTWindow, etc.) wait until the State has reached State.Historical. This practice is correct for both reading properties or should you wish to add custom elements to the existing UI.

Why: NinjaTrader UI related objects are not guaranteed to be available until historical data processing has started.

Best practice

```
protected override void OnStateChange()
{
   // wait until at least State.Historical
   if (State == State.Historical)
   {
      // and double check UI object is not null before
accessing
      if (ChartControl != null)
      {
         Print(ChartControl.Properties.ChartBackground);
      }
   }
}
```

Transitioning order references from historical to real-time

When dealing with strategy based orders which have transitioned from historical to real-time, you will need to ensure that locally stored order references are also updated.

Why: As the core order object updates, NinjaTrader has no specific way to update your locally stored order references. You can read more about this subject on the Advanced Order Handling topic: Transitioning order references from historical to live

Best practice

```
private Order entryOrder = null;

protected override void OnBarUpdate()
{
   if (entryOrder == null && Close[0] > Open[0])
      entryOrder = EnterLongLimit("myEntryOrder", Low[0]);
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled,
double averageFillPrice, OrderState orderState, DateTime
time, ErrorCode error, string nativeError)
{
    // One time only, as we transition from historical
    // Convert any old historical order object references
to the live order submitted to the real-time account
    if (entryOrder != null && entryOrder.IsBacktestOrder &&
State == State.Realtime)
        entryOrder = GetRealtimeOrder(entryOrder);

    // Null entryOrder if filled or cancelled. We do not
use the Order objects after the order is filled, so we can
null it here
    if (entryOrder != null && entryOrder == order)
    {
        if (order.OrderState == OrderState.Cancelled &&
order.Filled == 0)
            entryOrder = null;
        if (order.OrderState == OrderState.Filled)
            entryOrder = null;
    }
}
```

Terminating custom resources

Use a flag to track when resources have been set up properly before attempting to destroy them.

Why: Checking that an object has been configured ensures that values not destroyed prematurely. You can read more about this subject on Understanding the life cycle of your NinjaScript objects

Best practice

```
protected override void OnStateChange()
{
   if (State == State.Configure)
   {
      myObject = new object();
      // set a flag to indicator object has been configured
      configured = true;
   }

   else if (State == State.Terminated)
   {
      // only dispose of object if it has been configured
      if (configured)
      {
         myObject.Dispose();
      }
   }
}
```

Error handling practices

Safely accessing reference objects

Although there are documented **States** where objects are available, the implementation could change. If you are accessing a reference object, please do so by first checking that the object is not null.

Best practice

```
// checking to ensure chart control is available in all
situations
// will help to ensure this logic below does not generate
errors at a later time
if(ChartControl != null)
{
   myBackgroundBrush =
ChartControl.Properties.ChartBackground;
}
```

Accessing objects which terminate

To protect against race conditions and access errors, you should temporarily check for reference errors any time you attempt to do something with an object.

Why: **OnStateChange()** runs asynchronous to other NinjaScript events. You can run into scenarios where you **State.Terminated** logic is called in the middle of OnBarUpdate(), OnRender() etc.

Best practice

```
protected override void OnStateChange()
{
   // this logic runs asynchronously to other events
   if (State == State.Terminated)
   {
      myObject = null;
   }
}
protected override void OnRender(ChartControl chartControl,
 ChartScale chartScale)
{
   if (myObject == null)
      return;

   // for safety, always check for null references before
attempting to access an object
   // even if you have once checked for null references
earlier run-time
   if (myObject != null)
      myObject.DoSomething();
}
```

Proving instructions for non-ninjascript properties

Do not attempt to modify existing UI "Properties" to meet your specific needs. These features are exposed to allow you to read the environment state and make decisions to alter how your code executes, but should not be relied on to modify settings on behalf of the user. While these objects from these classes have setters for technical reasons, you should not attempt to amend the values through code. Instead, you should issue warnings or log errors instructing users to modify settings when required:

Why: NinjaTrader makes no guarantee that the requested changes will take effect, and user settings always take precedences. This includes the user defined ChartControl.Properties, ChartBars.Properties, and ChartPanel.Properties. Furthermore, two different user scripts could be installed which also attempt to modify properties you are relying which could introduce conflicts.

Best practice

```
if (State == State.Historical)
{
   if (ChartControl.Properties.EquidistantBarSpacing ==
true)
   {
      Draw.TextFixed(this, "error", "This indicator works
best with Equidistant BarSpacing set to false.",
TextPosition.BottomRight);
   }
}
```

Modifying UI elements and multi-threading

When interacting with UI objects, such as obtaining UI information, or modifying the existing layout, always use the NinjaScript's Dispatcher asynchronously

Critical: Improper thread handling from a NinjaScript object is a common cause of application deadlocks. Please be sure to read more information on Multi-Threading Consideration for NinjaScript

Best practice

```
// using a Dispatcher will ensure that the corresponding
action executes on the associated thread
this.Dispatcher.InvokeAsync(() =>
{
   UserControlCollection.Add(new
System.Windows.Controls.TextBlock
   {
      Text = "\nAdded by the ChartControl Dispatcher."
   });
});
```

Properly implementing try/catch blocks

Unless you are specifically debugging a method, the use of a try-catch block should be scoped to a particular area of logic. Do **NOT** try to handle all of your execution logic under one giant try-catch block.

Why: Larger try-catch blocks can not only be harder to debug, but can introduce performance issues at run-time

Practice to avoid

```
protected override void OnBarUpdate()
{
   try
   {
      // encapsulates entire OnBarUpdate logic
   }
   catch (Exception ex)
   {
      // attempt to handle all errors in one catch
   }
}
```

Using WPF brushes

Try to use a static predefined Brush if possible. If you need to customize a new brush object, make sure to .Freeze() the brush before using it.

Why: The pre-defined brushes are thread safe and do not require any special

handling. Custom defined brushes, on the other hand, are **NOT** thread-safe and must be frozen otherwise cross-thread exceptions can occur.

Best practice

```
// predefined brush
BackBrush = Brushes.Blue;

// if you are using a custom brush to e.g., modify the
opacity
SolidColorBrush opaqueBlue = new
SolidColorBrush(Colors.Blue) {Opacity = .25f};

// or just using at custom color not available in pre-
defined brushes class
SolidColorBrush coolGreen = new
SolidColorBrush(Color.FromRgb(30, 255, 128));

// you must freeze these brushes after they are
constructed!
opaqueBlue.Freeze();
coolGreen.Freeze();
```

barsAgo indexer vs. absolute bar Index

As you probably know, you can quickly look up the bar value on the chart by calling a PriceSeries<T> barsAgo indexer, e.g., Close[0].
However, the internal indexer and pointers about the barsAgo value are only guaranteed to be correctly synced and updated during a market data event. As a result, you should favor using the absolute GetValueAt() methods during events which are not driven by price

Why: Attempting to call the barsAgo indexer in an event method that is not driven by market data can yield unexpected results.

Best practice

```
// OnRender is not a market data event; barsAgo pointers
are not guaranteed to be in sync
protected override void OnRender(ChartControl chartControl,
 ChartScale chartScale)
{
   Print(mySMA.GetValueAt(CurrentBar));
}

// same is true for you custom events
private void myCustomClickHandler(object sender,
MouseButtonEventArgs e)
{
   Print(Close.GetValueAt(CurrentBar));
}
```

Tip: If you have programming requirements which rely on a PriceSeries indexer, you can use the TriggerCustomEvent() delegate which will update the internal pointers and indexes before executing the logic you specify.

Casting safely

Avoid type casting and type conversion as much as possible. Casting from a mixed collection of types is also prone to exceptions especially in situations that may not occur when you originally test your code.

Why: The practice to avoid code below could work in some scenarios but would generate errors if other types were added to that collection that you were not anticipating.

Practice to avoid

```
// This would run without errors if there were _ONLY_ type
HoriztonalLine on the chart
// But you risk a likely 'System.InvalidCastException' when
other draw types are in that collection
foreach (HorizontalLine hLine in DrawObjects)
{

}
```

If you must cast, do so safely and avoid implicit casts to types which may not be guaranteed to succeeded

Best practice

```
// Use the base IDrawingTool type and then cast to the
desired type within the for loop
foreach (IDrawingTool hLine in DrawObjects)
{
   // Note:  to prevent further errors, your type casting
should be done using the "as" keyword
   // Opposed to a direct cast:
   // HorizontalLine myLine = (HorizontalLine) hLine;

   HorizontalLine myLine = hLine as HorizontalLine;

   // This will allow you to ensure the cast actually
occurred
   if (myLine != null)
   {
      Print(myLine.StartAnchor.Price);
   }
}
```

Performance practices

Referencing indicator methods

In general, when calling an Indicator return method, there is some internal caching which occurs by design to help reduce memory consumption.

Why: While the designed indicator caching improves general memory performance, there is an implied cost of actually looking up the cached indicator

Practice to avoid

```
// each time you call the SMA() return method there is a
small performance cost
// implied from the time it takes to look up the cached
instance
if (Close[0] > SMA(20)[0])
{
   Print(SMA(20)[0]);
   EnterLongLimit(SMA(20)[0]);
   Draw.Dot(this, Time[0].ToString(), false, 0, SMA(20)[0],
 Brushes.DarkGreen);
}
```

Note: Indicator caching **ONLY** occurs when an indicator is recalled with the same **EXACT** parameters and input from the **SAME** calling script. (i.e. when a previously called indicator is called a second time with new parameters in the same script, a second instance will be created / cached)

If you are reusing an indicator several times through your code (especially indicators with many parameters), you can take further steps to refine performance by storing a reference to the indicator instance yourself (although it is by no means a requirement, and this suggestion does not need to be followed strictly)

Best practice

```
private SMA mySma;

protected override void OnStateChange()
{
   // when the indicator begins processing
   // save an instance of the SMA indicator with the
desired input
   if (State == State.Historical)
   {
      mySma = SMA(20);
   }
}

protected override void OnBarUpdate()
{
   // use the referenced mySMA throughout the lifetime of
the script
   if (Close[0] > mySma[0])
   {
      Print(mySma[0]);
      EnterLongLimit(mySma[0]);
      Draw.Dot(this, Time[0].ToString(), false, 0,
mySma[0], Brushes.DarkGreen);
   }
}
```

Marking object references for garbage collection

While it is not always necessary to set objects to null, doing so will mark them for garbage collection sooner and help prevent unnecessary memory resources from being utilized.

Why: In general you should be diligent to set stored memory objects to null when you are done using them, especially in situations where a NinjaScript object may be running for an extended period.

Best practice

```
protected override void OnBarUpdate()
{
   // saving "myDot" creates an additional reference in
memory
   Dot myDot = Draw.Dot(this, "myDot" + CurrentBar, false,
Time[0], Close[0], Brushes.Blue);

   if (conditionToRemove)
   {
      // remove draw object will remove the object from the
chart
      RemoveDrawObject("myDot");

      // but your local object "myDot" is still stored in
memory.
      // Explicitly setting to null will ensure object is
marked for garbage collection
      myDot = null;
   }
}
```

Note: The example above demonstrates using a draw object, but the practice can be extended to any object you store in memory (e.g., orders, brushes, custom objects, etc)

Disposing of custom resources

Dispose of objects that inherit from IDisposable or put into a Using statement.

Why: NinjaTrader is not guaranteed to dispose of objects for you. To avoid unnecessary memory consumption, always manage your resources by creating a variable and dispose of the object.

Best practice

```
// example of object instantiated which need to be disposed
StreamWriter writer = new StreamWriter("some_file.txt");

// use the object
writer.WriteLine("Some text");

// implements IDisposbile, make sure to call .Dispose()
when finished
writer.Dispose();

// or put in "using" statement which implicitly calls
.Dispose() when finished
using (StreamWriter writer2 = new
StreamWriter("some_file.txt"))
{
   writer2.WriteLine("Some text");
}
```

Tip: This is most commonly applicable when using SharpDX resources for custom rendering. Please be sure to review the information on Best Practices for SharpDX Resources

Avoiding duplicate calculations

Be mindful where and when your potentially complex calculations would be recalculated and thus run the risk of being calculated redundantly. For example, you may have logic which only needs to calculate, e.g., once per instance, once per session, once per bar, etc.

Best practice

```
// get GetPreviousTradingDayEnd() is expensive to look up
// but value only needs to be looked up once a day -> only
calcualte on first bar of session
if (Bars.IsFirstBarOfSession)
{
   TradingHours.GetPreviousTradingDayEnd(Time[0]);
}
```

The same considerations would apply to variables or function calls that would not change their output value for the currently processed bar on

Calculate.OnEachTick or .OnPriceChange, thus there would be no need handling them outside of IsFirstTickOfBar

Best practice

```
// dedicated logic to cache the prior sum on each tick of
bar
// While it is a good practice, this can cause problems for
bar types which may remove last bar (see below)
if (IsFirstTickOfBar)
   priorSum = sum;

sum = priorSum + Input[0] - (CurrentBar >= Period ?
Input[Period] : 0);
Value[0] = sum / (CurrentBar < Period ? CurrentBar + 1 :
Period);
```

Caching values on bars which remove last bar

Building on the previous example, be careful when caching values on the first tick of bar if using bars types which are IsRemoveLastBarSupported. To see how to handle these situations best, take a look at the default SMA indicator which has an additional logic branch which disables caching on those bar types:

Best practice

```
// logic below disables first tick of bar caching only on
bar types which remove last bar
if (BarsArray[0].BarsType.IsRemoveLastBarSupported)
{
   if (CurrentBar == 0)
      Value[0] = Input[0];
   else
   {
      double last = Value[1] * Math.Min(CurrentBar,
Period);

      if (CurrentBar >= Period)
         Value[0] = (last + Input[0] - Input[Period]) /
Math.Min(CurrentBar, Period);
      else
         Value[0] = ((last + Input[0]) /
(Math.Min(CurrentBar, Period) + 1));
   }
}
```

Precomputing values instead of calculating in OnRender()

To preserve good performance, always err on the side of caution if you are using OnRender for any calculation logic.

Why: OnRender() is called frequently as you interact with the Chart, which can cause calculations to occur much more often than the related market data events and can cause unnecessary spikes in CPU consumption.

Practice to avoid

```
protected override void OnRender(ChartControl chartControl,
 ChartScale chartScale)
{
   // continually recalling the same value methods is
unnecessary in this situation
   double myValue = Bars.GetClose(CurrentBar) +
Bars.GetOpen(CurrentBar);

   // render myValue
}
```

Best practice

```
private double myValue;

protected override void OnBarUpdate()
{
   // myValue only needs to update when OnBarUpdate() is
called
   // and then can be passed to OnRender() for chart
rendering purposes
   myValue = Close[0] + Open[0];
}

protected override void OnRender(ChartControl chartControl,
 ChartScale chartScale)
{
   // if needed, you can always check that myValue has
actually been set
   if (myValue > double.MinValue)
   {
      // render myValue
   }
}
```

Restricting OnRender() calculations to visible ChartBars

Use the ChartBars.FromIndex and ChartBars.ToIndex to limit calculations to only what is visible on the chart

Why: Rendering should be reserved for rendering on what is visible on the Chart. Performing calculations on bar index which are not visible can cause random spikes in CPU consumption.

Best practice

```
protected override void OnRender(ChartControl chartControl,
 ChartScale chartScale)
{
   // restricting this loop to only the
ChartBars.From/ToIndex limits the loop to only what is
visible on the chart
   for (int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
   {
      Print(ChartControl.GetSlotIndexByX(barIndex));
   }
}
```

Using DrawObjects vs custom graphics in OnRender()

When using Draw methods, a new instance of the Draw object is created including its custom rendering and calculation logic. These methods are convenient in many situations, but can quickly introduce performance issues if used too liberally. In some situations, you may see better performance for rendering via SharpDX in OnRender().

Why: Each draw object instance will see its own OnRender() called to render values. If you instead implement custom rendering in the your object, you would only see a single OnRender() call for your custom created graphics.

Practice to avoid

```
protected override void OnBarUpdate()
{
   // this would draw a dot on every bar on the chart
   // each instance would need to call its own OnRender()
method
   // not a very efficient use a draw method
   Draw.Dot(this, "everyDot" + CurrentBar, false, 0,
Close[0], Brushes.Blue);
}
```

With just a little extra code (much less than what is in the Draw methods) custom SharpDX rendering greatly reduces CPU and Memory consumption

Please ensure a Direct2D1 factory would only be instantiated from OnRender() or OnRenderTargetChanged() (which run in the UI thread), as access from other threads outside those methods could cause a degradation in performance.

Best practice

```
protected override void OnRender(ChartControl chartControl,
 ChartScale chartScale)
{
   // achieves the same effect of drawing a dot on every
bar
   // but only needs to call your object's OnRender()
   for (int index = ChartBars.FromIndex; index <=
ChartBars.ToIndex; index++)
   {
      float price =
chartScale.GetYByValue(Close.GetValueAt(index));
      float bar = chartControl.GetXByBarIndex(ChartBars,
index);
      float radius = (float) chartControl.BarWidth;

      SharpDX.Direct2D1.Ellipse dot = new
SharpDX.Direct2D1.Ellipse(new SharpDX.Vector2(bar, price),
radius, radius);

      using (SharpDX.Direct2D1.SolidColorBrush brush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.Blue))
      {
         RenderTarget.FillEllipse(dot, brush);
      }
   }
}
```

Tip: One of the advantages of using a Draw.Method is the returned Draw Objects contains metadata which could be used later (such as for obtain the bar index or price value of the dot later on). If you would use this metadata later on, using a Draw method would be in your best interests. However, if you are solely looking to render figures on a chart, favoring your custom SharpDX methods can drastically improve performance.

Responding to user events

Do **NOT** use OnRender() for purposes other than rendering. If you need events to hook into user interactions, consider adding your own event handler. The example

below shows registering the ChartPanel MouseDown event and registering a custom WPF control

> **Why:** OnRender() may call more or less frequently than you anticipated. Using your own custom event handlers allows you control and isolate user event logic you are looking to capture

Best practice

```
protected override void OnStateChange()
{
   if (State == State.Historical)
   {
      // subscribe to chart panel mouse down event
      if (ChartPanel != null) ChartPanel.MouseDown +=
DoUserClickedChartPanelEvent;

      // subscribe to a custom UI element mouse down event
      if (myWPFControl != null) myWPFControl.MouseDown +=
DoCustomWPFControlClickEvent;
   }

   else if (State == State.Terminated)
   {
      // remember to unsubscribe when finished
      if (ChartPanel != null) ChartPanel.MouseDown -=
DoUserClickedChartPanelEvent;
      if (myWPFControl != null) myWPFControl.MouseDown -=
DoCustomWPFControlClickEvent;
   }
}

private void DoUserClickedChartPanelEvent(object sender,
MouseButtonEventArgs e)
{
   Print("User clicked on the ChartPanel, executing custom
mouse down logic...");
}

private void DoCustomWPFControlClickEvent(object sender,
MouseButtonEventArgs e)
{
   Print("User clicked on my button, executing button
logic...");
}
```

Delaying logic for a particular time interval

Do **NOT** call Thread.Sleep() as it will lock the Instrument thread executing your NinjaScript object.

> **Why:** Market data events exposed to NinjaScript run on the underlying Instrument thread pool shared by all Instruments. Sleeping the underlying thread of your object will cause the entire Instrument thread to sleep, adversely affecting other features using that same Instrument.

Practice to avoid

```
protected override void OnBarUpdate()
{
   if (IsFirstTickOfBar && State == State.Realtime)
   {
      Print("Run some logic before:: " + DateTime.Now);
      Thread.Sleep(5000); // sleeping the Instrument thread
will have adverse effects on elements outside of your
script!
      Print("Run some logic after: " + DateTime.Now);
   }
}
```

Instead, try using a Timer object if you need to delay logic execution.

Best practice

```
protected override void OnBarUpdate()
{
   if (IsFirstTickOfBar && State == State.Realtime)
   {
      // Instead of Thread.Sleep for, create a timer that
runs at the desired interval
      System.Windows.Forms.Timer timer = new
System.Windows.Forms.Timer {Interval = 5000};

      // queue the "after" logic to run when the timer
elapses
      timer.Tick += delegate
      {
         timer.Stop(); // make sure to stop the timer to
only fire ones (if desired)
         Print("Run some logic after: " + DateTime.Now);
         timer.Dispose(); // make sure to dispose of the
timer
      };

      Print("Run some logic before: " + DateTime.Now);

      timer.Start(); // start the timer immediately
following the "before" logic
   }
}
```

Miscellaneous practices

Floating-point comparison

Be aware of floating-point precision problems. It can sometimes be more reliable to check within a certain degree of tolerance, such as the TickSize.

Why: You can read more about Floating-Point Arithmetic as it applies to NinjaTrader on our support forum

Practice to avoid

```
// depending on how Value[0] was calculated, it could be
off by a degree of floating points
// where this logic below would never be true
// e.g., 2050.2499999 vs 2050.50
if (Value[0] == Close[0])
{
   // do something
}
```

Best practice

```
// you can avoid these precision issues by rewriting the
comparison to evaluate within a certain tolerance.
if (Math.Abs(Value[0] - Close[0]) < TickSize)
{
   // do something
}

// You will also see NinjaTrader developed objects use a
custom Extension Method
// double.ApproxCompare() which Returns an int based on a
Epsilon value:
if (Close[0].ApproxCompare(Value[0]) == 0)
{
   // do something
}
```

Creating user defined parameter types / enums

When creating enums for your NinjaScript objects, it is strongly suggested to define those outside the class and in a custom namespace. A reference sample providing all details could be found here.

Efficiently debugging

Extremely liberal use of Log() and Print() methods can represent a performance hit on your PC as it takes memory and time to process each one of those method calls. When running custom NinjaScript, especially when using Calculate = Calculate.OnEachTick, please be mindful of how often Log() and Print() methods are processed as it can quickly consume PC resources.

- Log() method should not be used except for critical messages as each log entry makes it to the Control Center log which stays active till the end of the day. Excessive logging can result in huge amounts of memory being allocated just to

display all the log messages which would mean less memory for NinjaTrader to do other tasks.
- Print() method can be used more liberally than the Log() method, but can still represent a performance hit if used with extremely high frequency. Consider decreasing the printing from your script if you experience slowdowns when running the script.

Debug Mode

The debug mode should only be used if you are actively debugging a script and attached to a debugger.

> **Why**: Debug Mode will compile all of the files in the custom project as a "Debug" build, which omits certain optimizations which occur in the C# compilation process. It is more efficient to use your custom objects in the default "Release" build if you are using your scripts during production.

To disable Debug Mode:

- Right mouse click in any NinjaScript Editor
- Ensure the "Debug Mode" menu item is unchecked
- Press F5 to recompile your scripts
- Your scripts will be re-built using "Release" mode

Known NinjaScript Wrappers limitations

- The NinjaScript editor detects code changes in external editors, and will compile on code changes, however code will only be automatically generated by the NinjaScript editor if it's edited within the NinjaScript editor itself (or Visual Studio)
- Wrappers cannot be generated automatically for partial and abstract classes
- Code in the Properties region of the NinjaScript object cannot be commented out with the /* */ style commenting, as it will cause issues with the wrapper generation. Code must be commented out with the // style.
- Subclassing would not allow for wrappers to be generated

11.3 Distribution

Distribution

You can distribute custom indicators and strategies to any user of NinjaTrader. The following section discusses how you can create and share your scripts. If you are a 3rd party developer, please see the Commercial Distribution section.

11.3.1 Considerations For Compiled Assemblies

Using Compiled Assemblies

Compiled assemblies (DLL's) allow you to bundle your scripts into a format that hides your proprietary code along with any supporting resources. Compiled assemblies provide distinct benefits, especially for commercially distributed code, but there are a few considerations to keep in mind. Typecasting and building resource files (sounds, images, etc.) into your assemblies must be approached differently to ensure cleanly packaged, error-free DLL's.

Using Custom enum Properties

When creating custom enum properties, it is advised to create the enum outside of your NinjaScript class, and designating it in its own fully qualified namespace. For an example, please see here. When using the enum in code, please use the fully qualified namespace as opposed to using a using directive to shorthand the expression.

Casting Types in a DLL (Using dynamic Types)

Sometimes, you may need to cast your objects to NinjaScript types, such as when iterating through the DrawObjects collection to obtain a reference to a particular Drawing Object on a chart. When running C# code which has not been compiled into an assembly, typecasting can be done normally, as in the example below:

Typecasting in code outside of a compiled assembly

```
protected override void OnBarUpdate()
{
    foreach(HorizontalLine line in DrawObjects)
    {
        // Print the tag of each Horizontal Line on the chart
        Print(String.Format("Horizontal Line {0} found.",
line.Tag));
    }
}
```

An obstacle arises with traditional typecasting in a compiled assembly, since the NinjaScript type you attempt to cast will be present in both your DLL and NinjaTrader's Custom.dll assembly. If you plan to compile your code into a DLL, you will need to use the dynamic type to avoid this conflict by dynamically assigning the type at runtime, using the guidelines below:

1. Loop through your collection using the interface type
2. Use ToString() to check the fully qualified namespace of the object in the loop
3. Cast the object to dynamic, and reference properties of that object assuming it is the expected type

Dynamic variables as an alternative to typecasting inside of a compiled assembly

```
foreach (IDrawingTool line in DrawObjects.ToList())
{
    // Use ToString().Equals() to detect the object's Type
    if
(line.ToString().Equals("NinjaTrader.NinjaScript.DrawingTools.Horiz
ontalLine"))
    {
        // Cast line as dynamic and access the object by assuming
that it is the Type we expect
        Print(String.Format("Horizontal Line {0} detected!", (line
as dynamic).Tag));
    }
}
```

The above dynamic approach will work for primitive types. For instantiating more complex types / classes though, such as adding a new PriceLevel programmatically to an existing drawing tool, Reflection would need to used.

Instantiating more complex types such as the PriceLevels class inside of a compiled assembly

```
foreach (dynamic dt in DrawObjects.ToList())
{
   if(dt.ToString().Equals("NinjaTrader.NinjaScript.DrawingTools.FibonacciRetracements"))
   {
      Type type         = dt.PriceLevels.GetType().GetGenericArguments()[0];
      Assembly assembly = type.Assembly;
      var pl            = assembly.CreateInstance(type.FullName, false, BindingFlags.CreateInstance, null, new object[] { 55.5, Brushes.Red, 2 }, new        System.Globalization.CultureInfo("en-US"), new object[] {});
      dt.PriceLevels.GetType().GetMethod("Add").Invoke(dt.PriceLevels, new object[] { pl } );
      this.ForceRefresh();
   }
}
```

Working with the dynamic type

Using dynamic variables in the technique above requires careful attention to accessing members appropriately, and thus should be avoided if you do not intend to use or distribute compiled assemblies.

- **No Intelliprompt**: Since the compiler cannot know which type you assume a dynamic variable to be, no Intelliprompt will be displayed to help search through type members. The same applies to Visual Studio's Intellisense or similar utilities.

- **No Compile Errors**: For the same reason, the compiler cannot know if you are using the variable in a way not supported by its expected type, trying to access members not present in that type, or other related errors. Thus, any such errors which would be caught by the compiler when typecasting will be missed, and will result in runtime errors instead. If a runtime error were to be triggered, the error may be more difficult to interpret.
  - Example: If you tried to access "line.tag" (improper capitalization) in the examples above, you would receive the following errors:
    - Typecasting / Compile Error: *"'NinjaTrader.NinjaScript.DrawingTools.HorizontalLine' does not contain a definition for 'tag' and no extension method accepting a first argument of type 'NinjaTrader.NinjaScript.DrawingTools.HorizontalLine' can be found (are you missing a using directive or an assembly reference?)"*
    - dynamic / Runtime Error: *"Error on calling 'OnBarUpdate' method on bar 0: 'NinjaTrader.NinjaScript.DrawingTools.DrawingTool.tag' is inaccessible due to its protection level"*

Adding XAML and Other Files Into a DLL

When exporting a compiled assembly through NinjaTrader, no additional resource files can be added. There are two ways around this. The first is to export the DLL from NinjaTrader, then open the exported .zip file, add any additional files, and re-zip the archive, but this will result in your resource files being fully accessible to end users. The second and recommended approach is to use a fully featured IDE such as Visual Studio to build your DLL's.

For more information on how to accomplish this with Visual Studio, see the "AddOn Development Environment" section of the AddOn Development Overview page. Although the page focuses on AddOn development, the sample project it provides can be used to develop other NinjaScript types, as well.

Exporting custom drawing tools as assembly / DLL

When planning to distribute your custom drawing tools via assemblies, please understand it's paramount that you implement your own Draw. method to allow the drawing tool getting called programmatically by other NinjaScript objects.

The NinjaTrader default drawing tools would implement this via a partial class, for example you would see -

Default NinjaTrader drawing tool Draw. method handling

```
public static partial class Draw
{

}
```

However since partial classes could not span across two assemblies, therefore a custom non partial Draw. method for your NinjaScript drawing tool would be needed.

Custom drawing tool Draw. method handling

```
public static class MyDrawCustom
{

}
```

Exports might not be backwards compatible

NinjaScript exports might not be backwards compatible with previous versions of NinjaTrader.

This is known to happen every time a new type (e.g. Enum) was introduced, since the newly introduced types are not known to prior releases of NinjaTrader

Typically an error message like the following would be seen:

"Error on calling 'SetState' method: Could not load type 'NinjaTrader.NinjaScript.Indicators.CumulativeDeltaType' from assembly 'NinjaTrader.Vendor, Version=8.0.12.0, Culture=neutral, PublicKeyToken=null'."

11.3.2 Import

You should only import NinjaScript Archive files (.zip) that you have obtained from a trusted source.

To import:

1. From the Control Center window select the menu Tools > Import> NinjaScript... to open the "Import" dialog window
2. Select the file you want to import
3. Press the "Import" button

11.3.3 Export

You can export NinjaScript for others to import in several formats:

- **Source files** - NinjaScript source files that can be imported and edited by others
- **Assemblies** - A compiled assembly (DLL) of NinjaScript that "hides" your source code. This can be further protected by SecureTeam's Agile.NET to prevent theft of your intellectual property.

Exporting NinjaScript as Source Files

You may want to provide other NinjaTrader users with source files of your NinjaScript in a format where they are able to view and edit them.

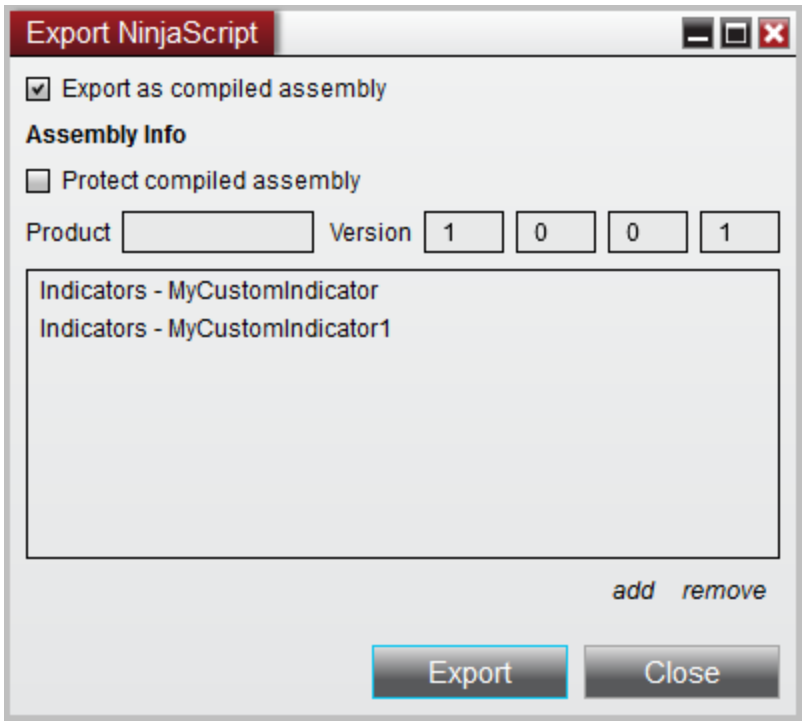
1. From the Control Center window select the menu Tools > Export > NinjaScript... to open the "Export NinjaScript" dialog window
2. Press ***"add"***
3. Use the "Type" drop down to filter available NinjaScript types
4. Select all of the files that you want to export and press the **"OK"** button
5. A list of all files that will be exported will be shown
6. Press the **"Export"** button to export the selected files

7. A file dialog will open where you can choose the location your zip export file will be created in. Per default the NinjaScript Archive File (.zip) file will be created in My Documents\<NinjaTrader Folder>\bin\Custom\ExportNinjaScript.

8. The file can be imported by another NinjaTrader application on a different PC

Exporting NinjaScript as Assembly

You may want to provide other NinjaTrader users with access to your proprietary indicators or strategies in a secure format preventing them from being able to see your proprietary source code. You can do this by exporting your NinjaScript indicators as a compiled Microsoft .NET



assembly (DLL) file.

- This is a great distribution option if your proprietary indicator or strategy files do not reference external DLL's
- If your proprietary indicator or strategy references external DLL's then its advised to create your own custom installer

1. From the Control Center window select the menu Tools > Export > NinjaScript... to open the "Export NinjaScript" dialog window

2. Select the option "Export as compiled assembly".
3. You can optionally select "Protect compiled assembly" (For information on protection see the "Protection /DLL Security page)
4. Press ***"add"***
5. Use the "Type" drop down to filter available NinjaScript types
6. Select all of the files that you want to export and press the **"OK"** button
7. A list of all files that will be exported

will be shown

8. Optionally enter information that describes the assembly in the "Product" and "Version" fields
9. Press the "Export" button to export the selected files
10. A file dialog will open where you can choose the location your zip export file will be created in. Per default the NinjaScript Archive File (.zip) file will be created in My Documents\<NinjaTrader Folder>\bin

\Custom\ExportNinjaScript.
11. The file can be imported by another NinjaTrader application on a different PC

11.3.4 Remove NinjaScript Assembly

This will allow you to remove installed NinjaScript assembly files.

To remove a NinjaScript assembly:

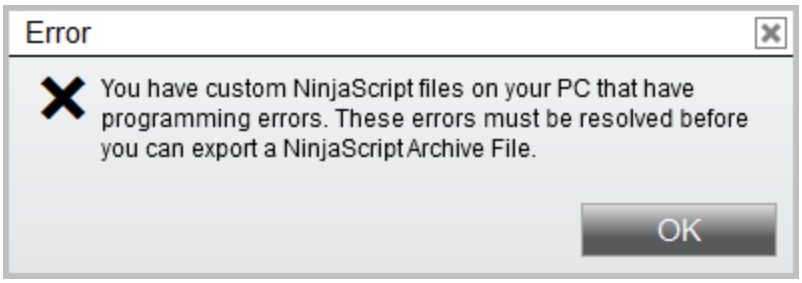
1. From the Control Center window select the menu Tools > Remove NinjaScript Assembly
2. Select the file(s) you want to remove (MultiSelect would be possible via holding Shift pressed while selecting the desired files for removal)
3. Press the "Remove NinjaScript assembly" button

> **Note:** Remove NinjaScript Assembly will not unload existing assemblies until restart, this means you should not import the same assemblies again until you have restarted NinjaTrader.

11.3.5 Export Problems

If you are having difficulties exporting NinjaScript it could be due to one of the following reasons:

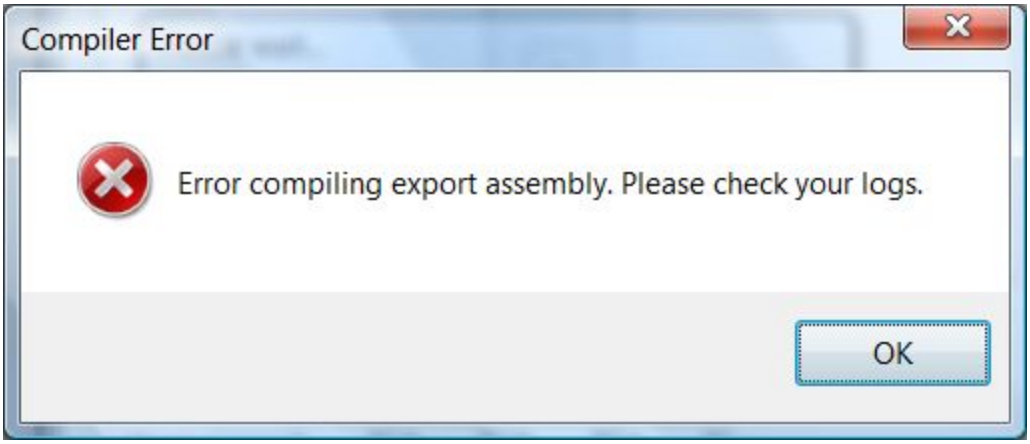
▼ NinjaScript Compile Error



If you receive the above error, you will need to compile your NinjaScript error-free before you can export. To see if your NinjaScript file is error free, open the NinjaScript Editor (Tool > Edit NinjaScript) and press F5 to compile. If you are trying to check a NinjaScript Strategy created from the Strategy Wizard you can do the same by finishing the wizard and seeing if you receive the “Strategy successfully generated” message.

If you receive any errors when compiling you will need to address them before exporting.

.NET Referencing



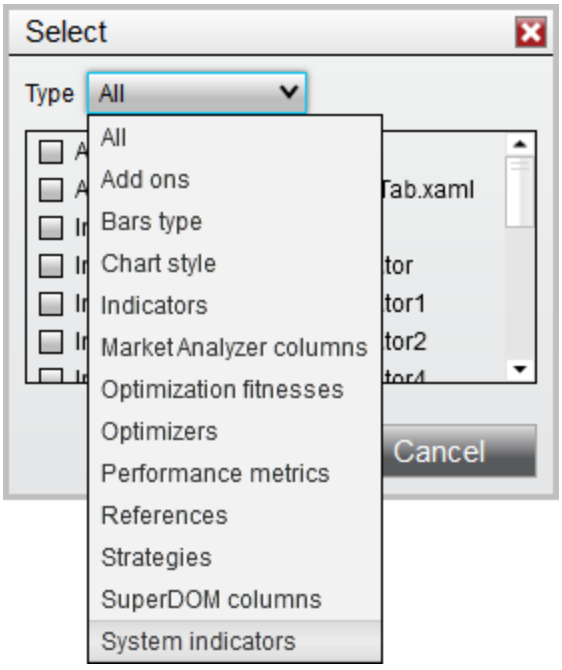
If you are able to compile without errors and still experience exporting difficulties like the one above, check to see if you receive an error similar to this in the Control Center logs:

"3/6/2014 9:25:30 AM|2|4|Error compiling export assembly: C:\Users\NinjaTrader\Documents\NinjaTrader 8\bin\Custom\Indicator\MyCustomIndicator.cs(42,18) : error CS0118: NinjaTrader.Indicator.SMA is a type but is used like a variable"

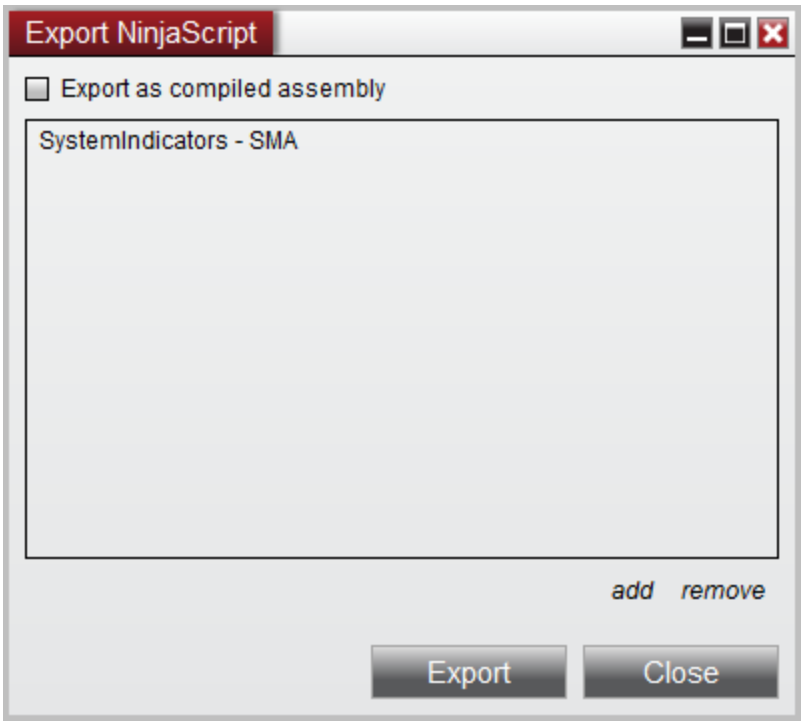
Note: This error may have a different error code and message depending on which variant of .NET you have installed. An error message indicative of this issue would include an indicator name without quotation marks.

If you experience this error, please follow this procedure:

1. Take note of which indicator is referenced by the error. In the above example, it is the SMA
2. Go to your NinjaScript Export utility. (Tools > Export > NinjaScript...)
3. After press ***"add"*** select **"System indicators"** from the **"Type"** drop down



4. Add the indicator that was referenced in the error to the export list along with your custom NinjaScript by pressing the > button



5. Press the “Export” button to create your NinjaScript Archive File. If you receive the same error again, repeat this procedure until you add all the referenced system indicators and are able to successfully export your custom NinjaScript.

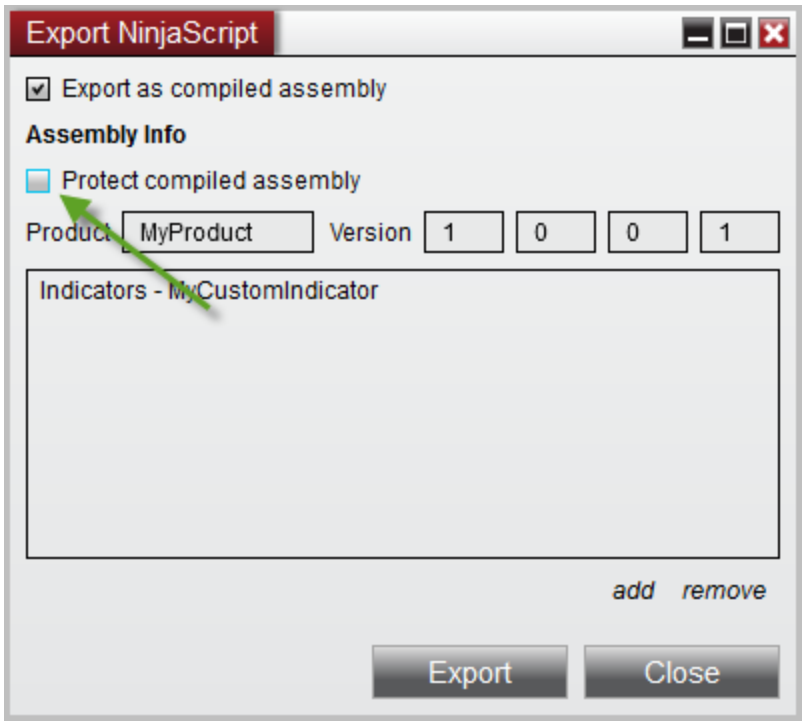
Note: If the indicator referenced in the error is another custom indicator you will need to follow the same procedure to add the custom indicator.

11.3.6 Protection/DLL Security

Although .NET DLL files are compiled which prevents users from being able to see your proprietary source code, they are still subject to decompilation and reverse engineering attempts. If you want a higher level of security, you can select the "Protect compiled assemblies" option which adds an additional layer of protection. This additional protection layer is provided by SecureTeam's Agile.NET product which has been licensed by NinjaTrader and available at a reduced price to protect NinjaTrader assemblies. This product

claims to completely stop MSIL disassembly and decompilation. We use it ourselves and are extremely happy with it.

Should you wish to use Agile.NET for protecting your NinjaScript assemblies you will first need to go here to download and purchase the product. Once installed, please run the Agile.NET standalone product once to input in the license information you should have received when you downloaded it. After that, when you use NinjaTrader's Export NinjaScript utility and select the "Protect compiled assemblies" option for export, it will automatically protect your NinjaScript assembly with Agile.NET.



Please note that this version of Agile.NET will only work for protecting NinjaScript assemblies within NinjaTrader. If you would like to protect other files outside of NinjaTrader please consider purchasing the full version of Agile.NET from SecureTeam directly here 'Agile.NET 6.0 Code Protection'. NinjaScript assemblies protected with the full version of Agile.NET will also work in NinjaTrader.

At this time we recommend using version 6.9.1.2
For clients on 8.0.28.0 or older you can continue to use 6.6.0.35

11.3.7 Commercial Distribution

Commercial Distribution Overview

As a commercial developer, you can distribute your proprietary indicators and and strategies to the growing universe of NinjaTrader users. This section contains information you should understand before distributing your work to the public.

11.3.7.1 Licensing/User Authentication

NinjaTrader provides a free vendor license management service for user authentication to qualified 3rd party developers.

The service includes the following features:

- One method call within your NinjaScript indicator or strategy's constructor will enable the authentication process
- A NinjaScript AddOn dedicated to license management (Manage license, provide free trials)
- Licenses are exclusively tied to a combination of user-defined prefix + PC machine ID value, ensuring that licenses cannot be shared
- Manage all of your individual products, or group products together for licensing
- Licenses expire based on time/date
- Create free trial periods

For more information please contact sales@ninjatrader.com or your NinjaTrader Business Development representative. Once approved, you will receive a unique Vendor ID used to manage your user licenses, a Vendor Licensing Help Guide containing information, samples, and resources to guide you through the process of managing licensing.

11.3.7.2 Best Practices for Distribution

The following are what we suggest for best practices for distribution.

Do not deploy NinjaScript Source Files

If you are a commercial vendor, you should never distribute the NinjaScript .cs source code files even if your IP is contained within an assembly or proprietary DLL. Source code files are editable by users and can result in unnecessary support issues.

Naming Conventions

Please use consistent naming convention with your indicators and strategies. We suggest adding a prefix to an indicator name. If your company name is "Hyper" you could name your indicators "HyperTrend" or "HyperOscillator" for example.

In the event that you provide NinjaScript export archives (zip files) as your means of distribution, NinjaTrader will automatically block incompatible scripts from importing so there will be no confusion by the user as to whether they are installing Version 7 or 8 scripts to their NinjaTrader installation. It is advisable to include the NinjaTrader version number in the export archive which will reduce potential support burden. For example, you could name your indicators “MyIndicator_7.zip” and “MyIndicator_8.zip”.

Clean up your resources

Always free up resources such as external windows DLL's or license management related resources. Resources should be freed within the OnStateChange() method in State.Terminate. NinjaTrader calls this method at the point at which a script is no longer used.

User Authentication Trigger

If you use a proprietary user authentication process, ensure that it is triggered within the OnStateChange() method in State.SetDefaults. This ensures that users are not forced to endure unnecessary delays on NinjaTrader start up or dialog windows that display available indicators and strategies as the windows are loaded. NinjaTrader, LLC provides a free licensing service for qualified 3rd party developers. For more information on this free service, contact your NinjaTrader Business Development representative.

User Authentication Check State

A license check should only be performed once and maintain its check state.

User Authentication Time Out

A license check should have a time out in case of internet issues, to enhance performance in this case.

Custom Installer

If you provide a custom installer, the installer should not overwrite any NinjaTrader deployed files, and you should provide an uninstall option which removes all installed files.

It is also preferred that you provide one installer that provides the user the option to install either a version 7 or version 8 compatible version of your product(s). Ensure that you only copy the correct files to the correct NinjaTrader installation folders since if you don’t it is possible that it could cause compile issues for the customer and it will be extremely difficult for all involved to isolate the cause.

These are the following folder names:

- `Documents\NinjaTrader 7\bin\Custom`
- `Documents\NinjaTrader 8\bin\Custom`

Test on Legacy Operating Systems

Some NinjaTrader customers run on older Operating Systems (such as Windows 7) and you should make sure that your indicators, custom installers and external DLLs (if any are used) properly run on these legacy operating systems.

Expose Indicator States

If your proprietary indicator acts as a trend state (green bars are bullish, red bearish) its good practice to expose the indicators's state so that consumers of your indicators can use them within their own custom indicator or strategy.

11.3.7.3 Distribution Procedure

NinjaTrader makes it easy to distribute complete packages for your clients. Not only can you distribute your indicators and strategies, but you can also seamlessly deploy your own custom assemblies, native DLLs, chart templates, and Market Analyzer templates to your clients.

Creating the distribution package

To create a distribution package, please follow the steps shown here for creating a Export file containing your NinjaScript indicators and/or strategies.

It is strongly recommended that you export your scripts as an assembly and use SecureTeam's Agile.NET. Only this process will provide you with the highest level of security possible in order to protect your intellectual property. For more information on using SecureTeam's Agile.NET please see the Protection/DLL Security section.

After you finish using the Export utility you will find the distribution package as a .zip file located in My Documents\NinjaTrader 8\bin\Custom\ExportNinjaScript. If you only wanted to distribute your NinjaScript files then providing your customers with this .zip and having them go through the Import process would install it on their machines. If you wish to add more custom files to your distribution package, please see the sections below.

Critical: It is important to let your customers know that NinjaTrader 8 indicators and strategies are NOT necessarily compatible with NinjaTrader Version 7.

Adding custom assemblies or native DLLs

1. Locate your base .zip distribution package
2. Open the .zip
3. Add to the .zip file your assemblies and/or your DLL files to the root directory of the .zip. These files cannot be behind any extra directory structures and must be directly in the root of the .zip

For custom assemblies, you will also need to add to the root of the .zip a .txt file called AdditionalReferences.txt

1. Bring up the Windows Start Menu
2. Go to the Run field and type "notepad" without the quotes and press Enter
3. In Notepad, type the name of your custom assembly and then save the file as a text file with the name "AdditionalReferences".

   Ex: If your custom assembly's name was MyCustomAssembly.dll and MyCustomAssembly.cs, in the AdditionalReferences.txt file you would type "MyCustomAssembly" without the quotes.

Note: If you have multiple custom assemblies to add you can append each of the assembly's names into the same AdditionalReferences.txt file on new lines

Adding templates

If you are distributing an indicator package, you may also want to distribute a prebuilt Chart Template that your customers can use to quickly bring up preferred settings for your chart setup. The same instructions here would work though for all other templates as well, i.e. MarketAnalyzer, DrawingTools - as long as the relative folder under templates is correctly set per the template category you're working with. The below steps run through the process for Chart templates.

1. Locate your base .zip distribution package
2. Open the .zip
3. Create a new directory called "templates" without the quotes
4. Navigate into the "templates" directory and create another new directory called "Chart"
5. Navigate into the "Chart" directory. Copy the .xml chart templates you wish to distribute from My Documents\NinjaTrader 8\templates\Chart to this directory in the .zip

Adding workspaces

If you are distributing an indicator package, you may also want to distribute a prebuilt Workspace that your customers can use to quickly bring up preferred

settings for your workspace. The below steps run through the process for workspaces.

1. Locate your base .zip distribution package
2. Open the .zip
3. Create a new directory called "workspaces" without the quotes
4. Navigate into the "workspaces" directory. Copy the .xml workspace you wish to distribute from My Documents\NinjaTrader 8\workspaces to this directory in the .zip

Adding custom resource files

You may run into the need to distribute other custom files such as pictures for buttons for use with your product as well. This can be achieved via the same approach as for the templates, as long as the resources folder is under the parent templates directory.

1. Locate your base .zip distribution package
2. Open the .zip
3. Create a new directory called "templates" without the quotes
4. Navigate into the "templates" directory and create another new directory, for example "MyResources"
5. Navigate to the directory where your files are stored. Copy the resource files you wish to distribute from this directory to your custom directory from step 4 in the .zip

Note: When modifying the .zip archives, if your zip utility application has an option for storing or recreating relative paths please be sure to turn this off as it will cause problems when importing the archive to NinjaTrader.

11.4 Editor

NinjaScript Editor Overview

The NinjaScript Editor is a powerful scripting editor that allows you to create custom indicators and strategies efficiently. The NinjaScript Editor includes powerful coding assistance and advanced debugging tools to help you custom build your indicator, strategy or any other supported NinjaScript type.

Display
- Editor Components
- NinjaScript Explorer
- NinjaScript Wizard

Errors/Debugging
- Compile Errors
- Visual Studio Debugging
- Compile Error Codes

Coding Assistance
- Intelliprompt
- Code Snippets

11.4.1 Compile Error Codes

The following error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

11.4.1.1 CS0006

See CS0234.

11.4.1.2 CS0019

The following CS0019 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

Strings cannot be compared with relational operators (<, >, <=, >=, ==, !=) to other object types. Strings can only be compared to other strings and only through the use of equality operators (==, !=).

Error Description #1

Operator '==' cannot be applied to operands of type 'string' and 'int'

```
// Erroneous Sample Code – Cannot compare a string to an integer
if ("string" == 5)

// Resolution Sample Code – Compare a string with another string
if ("string" == intValue.ToString());
```

Error Description #2
Operator '<' cannot be applied to operands of type 'string' and 'double'

```
// Erroneous Sample Code - Cannot compare a string to a double
if ("string" >= 1.2)

// Resolution Sample Code - Testing to see if the strings are not the same
if ("string" != "string2")
```

Error Description #3
Operator '>' cannot be applied to operands of type 'string' and 'string'

```
// Erroneous Sample Code - Cannot quantitatively compare a string to another string
if ("string" > "string2")

// Resolution Sample Code - Testing to see if both strings are the same
if ("string" == "string2")
```

Additional Error Descriptions
Operator '<' cannot be applied to operands of type 'string' and 'string'
Operator '<=' cannot be applied to operands of type 'string' and 'string'
Operator '>=' cannot be applied to operands of type 'string' and 'string'

11.4.1.3 CS0021

The following CS0021 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation
This is a common error when calling indicators methods. It occurs when an indicator is called without its required parameter arguments before accessing an indexed value.

To fix this error you will need to first pass to the indicator method all the necessary parameter arguments. You can do this with '()' after the indicator name. Please note that you will still need to pass an empty parameter argument list even if your indicator requires no arguments.

Error Description #1
Cannot apply indexing with [] to an expression of type 'method group'

Example #1
```
// Erroneous Sample Code - SMA is an indicator and requires parameter arguments
double value = SMA[0];

// Resolution Sample Code - SMA() properly called
double value = SMA(14)[0];
```

Example #2

```
// Erroneous Sample Code - EMA is an indicator and requires parameter arguments
double maDelta = EMA[0] - EMA[1];

// Resolution Sample Code - SMA() properly called with an overload method (one of
several variations)
double maDelta = EMA(High, 14)[0] - EMA(High, 14)[1];
```

11.4.1.4 CS0029

The following CS0029 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error can occur when you try to convert from one 'type' to another 'type'.

To fix this error, ensure that you are assigning the correct value type.

Error Description #1

Cannot implicitly convert type 'int' to 'bool'

```
// Erroneous Sample Code - 'CurrentBar' is an integer
if (CurrentBar)

// Resolution Sample Code - Compares an integer with another integer
if (CurrentBar < 1)
```

Error Description #2

Cannot implicitly convert type 'double' to 'bool'

```
// Erroneous Sample Code – Close[0] returns a double value
if (Close[0])

// Resolution Sample Code – Compares a double with another double
if (Close[0] > Close[1])
```

Error Description #3

Cannot implicitly convert type 'NinjaTrader.NinjaScript.Indicators.SMA' to 'double'

```
// Erroneous Sample Code - Incorrect since assigning an indicator to a variable of
double type
double myValue = SMA(20);

// Resolution Sample Code - Correct expression since we are accessing the current
bar's value of the SMA indicator
double myValue = SMA(20)[0];
```

11.4.1.5 CS0103

The following CS0103 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

When a variable is used before declaration, the compiler will not know what it is. This error is also commonly invoked by typos.

Please ensure that you have declared your variables prior to using them. If variables are declared or properties already exist, please check for typos.

Error Description #1

The name 'identifier' does not exist in the current context

Example #1

```
// Erroneous Sample Code - 'CurentBar' does not exist since it has been spelled
incorrectly (missing an 'r')
if (CurentBar < 10)

// Resolution Sample Code - 'CurrentBar' exists since it is spelled correctly
if (CurrentBar < 10)
```

Example #2

```
// Erroneous Sample Code - 'newVariable' is not declared
newVariable = 10;

// Resolution Sample Code - 'newVariable' is now declared as an integer
int newVariable = 10;
```

11.4.1.6 CS0200

The following CS0200 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error is most common when you try to assign values to a particular `Series<T>` index that is read-only. Instead try making your own Series<T> and assign the value there.

Error Description

Property or indexer 'NinjaTrader.NinjaScript.ISeries<double>.this[int]' cannot be assigned to -- it is read only

Example #1

```
// Erroneous Sample Code - Cannot assign values to something that is read-only
Close[0] = 25;
```

```
// Resolution Sample Code - Assigns value to a custom Series<double>
myCustomClose[0] = 25;
```

Example #2

```
// Erroneous Sample Code - Cannot reassign values to Series<double> indexed value and
cannot have an if statement based // on an assignment operator
if (Close[0] = Open[0])

// Resolution Sample Code - Properly compares two Series<double> values
if (Close[0] == Open[0])
```

11.4.1.7 CS0201

The following CS0201 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error can occur when you make a statement solely from an indicator or variable call.

You will need to do something with the value you called for the statement to be complete.

Error Description #1

Only assignment, call, increment, decrement, await and new object expressions can be used as a statement

```
// Erroneous Sample Code - Statement that does nothing
SMA(5)[0];

// Resolution Sample Code - 'currentSMA' takes on the current bar's SMA(5) value
double currentSMA = SMA(5)[0];
```

11.4.1.8 CS0234

The following CS0234 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error can occur when an imported DLL (could be a 3rd party indicator) you are referencing no longer exists / has been removed.

To resolve this the DLL must be re-imported.

To re-import a 3rd party dll:

1. Open the NinjaScript Editor via New > NinjaScript editor.

2. Right mouse click in the NinjaScript Editor main window and select the menu name "References"
3. In the "References" dialog window press the button "Add"
4. Select the 3rd party DLL

> **Warning**: Please make sure in this step to select only the 'true' DLL file needed for reference, which would not contain any X86 or X64 suffixes in its file-name, otherwise you could run into compile issues later.

Error Descriptions

The type or namespace name '<name>' could not be found (are you missing a using directive or an assembly reference?)
The type or namespace name '<name>' does not exist in the namespace 'NinjaTrader.Indicator' (are you missing an assembly reference?)

11.4.1.9 CS0246

See CS0234.

11.4.1.10 CS0428

The following CS0428 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error can occur when you miscall a method such as indicator methods.

If you are calling an indicator please ensure that you have both the parameters '()' and the indexing value '[]' set. For other methods please ensure you pass all required parameters through the parameters set '()'.

Error Description #1

Cannot convert method group 'SMA' to non-delegate type 'double'. Did you intend to invoke the method?

Example #1

```
// Erroneous Sample Code - SMA() indicator method is improperly called
double myValue = SMA;

// Resolution Sample Code - SMA() indicator method is properly called
double myValue = SMA(5)[0];
```

Example #2

```
// Erroneous Sample Code - ToString is a method and requires round brackets () to be
properly called
```

```
string str = Close[5].ToString;

// Resolution Sample Code - ToString() is properly called
string str = Close[5].ToString();
```

11.4.1.11 CS0443

The following CS0443 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error is most commonly invoked when no index value is used inside the indexing brackets.

Please ensure you place a value inside the '[]'.

Error Description #1

Syntax error, value expected

```
// Erroneous Sample Code - Missing index value
double myValue = SMA(20)[];

// Resolution Sample Code - 'myValue' takes on the current bar's SMA(20) value
double myValue = SMA(20)[0];
```

11.4.1.12 CS1002

The following CS1002 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error can be invoked when statements are not ended properly.

All statement lines must be closed with a semicolon.

Error Description #1

; expected

```
// Erroneous Sample Code - Statement is not closed
double myValue = SMA(20)[0]

// Resolution Sample Code - Statement is closed
double myValue = SMA(20)[0];
```

11.4.1.13 CS1061

The following CS1061 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error's code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error can occur when you try to use a method or access an exposed property that does not exist for your particular object.

Please check the methods and exposed property available for your particular object.

Error Description #1

'NinjaTrader.Indicator.CurrentDayOHL' does not contain a definition for 'CurentOpen'

```
// Erroneous Sample Code - CurrentDayOHL()’s property is 'CurrentOpen' not
'CurentOpen' (typo)
double value = CurrentDayOHL().CurentOpen[0];

// Resolution Sample Code - 'CurrentOpen' property available
double value = CurrentDayOHL().CurrentOpen[0];
```

11.4.1.14 CS1501

The following CS1501 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error can occur when you use use an overload (method parameter signature) that does not exist. This could be because you are passing in 3 arguments when the method only requires 2.

You can cycle through the available overloads with the use of the up and down arrows on the Intelliprompt when you call an indicator method or any other method.

Error Description #1

No overload for method 'SMA' takes '0' arguments

Example #1

```
// Erroneous Sample Code - SMA() does not contain an overload that has 3 arguments
double myValue = SMA(Close, 5, 2)[0];

// Resolution Sample Code - SMA() has an overload consisting of 2 arguments
double myValue = SMA(Close, 5)[0];
```

Example #2

```
// Erroneous Sample Code - EMA() does not contain an overload that has 0 arguments
double myValue = EMA()[0];

// Resolution Sample Code - EMA() has an overload consisting of 1 argument
double myValue = EMA(5)[0];
```

11.4.1.15 CS1502

The following CS1502 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error can occur when you pass in incorrect parameter object types into a method such as an indicator.

Please check the overload methods for the proper parameter object types and pass in the proper object. You can check the overload methods with NinjaScript editor's Intelliprompt when you call a method.

Error Description #1

The best overloaded method match for 'NinjaTrader.NinjaScript.StrategyBase.SetStopLoss(CalculationMode, double)' has some invalid arguments

```
// Erroneous Sample Code - Close is a Series<double> object type and is not a valid
value to the SetStopLoss() method
SetStopLoss(CalculationMode.Price, Close);

// Resolution Sample Code - The SetStopLoss() method takes a double value so pass in
Close[0]
SetStopLoss(CalculationMode.Price, Close[0]);
```

Error Description #2

The best overloaded method match for 'NinjaTrader.Indicator.Indicator.SMA(NinjaTrader.NinjaScript.ISeries<double>, int)' has some invalid arguments

```
// Erroneous Sample Code - Using an integer when the first parameter should be a
Series<double>
double myValue = SMA(5, 5);

// Resolution Sample Code - 'myValue' will take the value of the current bar's SMA
double myValue = SMA(Close, 5)[0];
```

11.4.1.16 CS1503

The following CS1503 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error can occur when you try to assign a value to a Series<T> that is not of the correct value type.

Series<double> objects can only contain double values. Series<bool> objects can only contain bool values. Etc.

Error Description #1

Cannot implicity convert type from 'string' to 'double'

```
// Erroneous Sample Code - Cannot pass in a string to a Series<double>
Value[0] = "Close[0]";

// Resolution Sample Code - Sets Series<double> to the current bar's Close value
Value[0] = Close[0];
```

Error Description #2

Cannot implicitly convert type 'NinjaTrader.NinjaScript.Indicators.SMA' to 'double'

```
// Erroneous Sample Code - Cannot pass in a Series<double> object to a Series<double>
Set() method
Values[0] = SMA(20);

// Resolution Sample Code - Sets Series<double> to the current bar's SMA(20) value
Values[0] = SMA(20)[0];
```

11.4.1.17 CS1513

The following CS1513 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

This error is most common with chaining if-else or loop statements.

Please check all code segments and statements are closed. Every opening curly brace '{' needs a matching closing curly brace '}' .

Error Description #1

} expected

```
// Erroneous Sample Code - If statement is not closed
if (CurrentBar < 1)
{
// Do something
<--- Missing closing curly brace
// Resolution Sample Code - If statement is closed
if (CurrentBar < 1)
{
// Do something
}
```

11.4.1.18 CS1525

The following CS1525 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

Error Code Explanation

The compiler detected an invalid character in an expression.

Error Description #1

{ expected

```
// Erroneous Sample Code - If statement is not opened
protected override void OnBarUpdate()
{
    if(IsFirstTickOfBar)
}

// Resolution Sample Code - If statement is open and closed
protected override void OnBarUpdate()
{
   if (IsFirstTickOfBar)
   {
      // do something
   }
}
```

11.4.1.19 NoDoc

Unfortunately we do not have NinjaScript context based Help information on this specific error code. You can check the Microsoft MSDN site section on error codes for futher information.

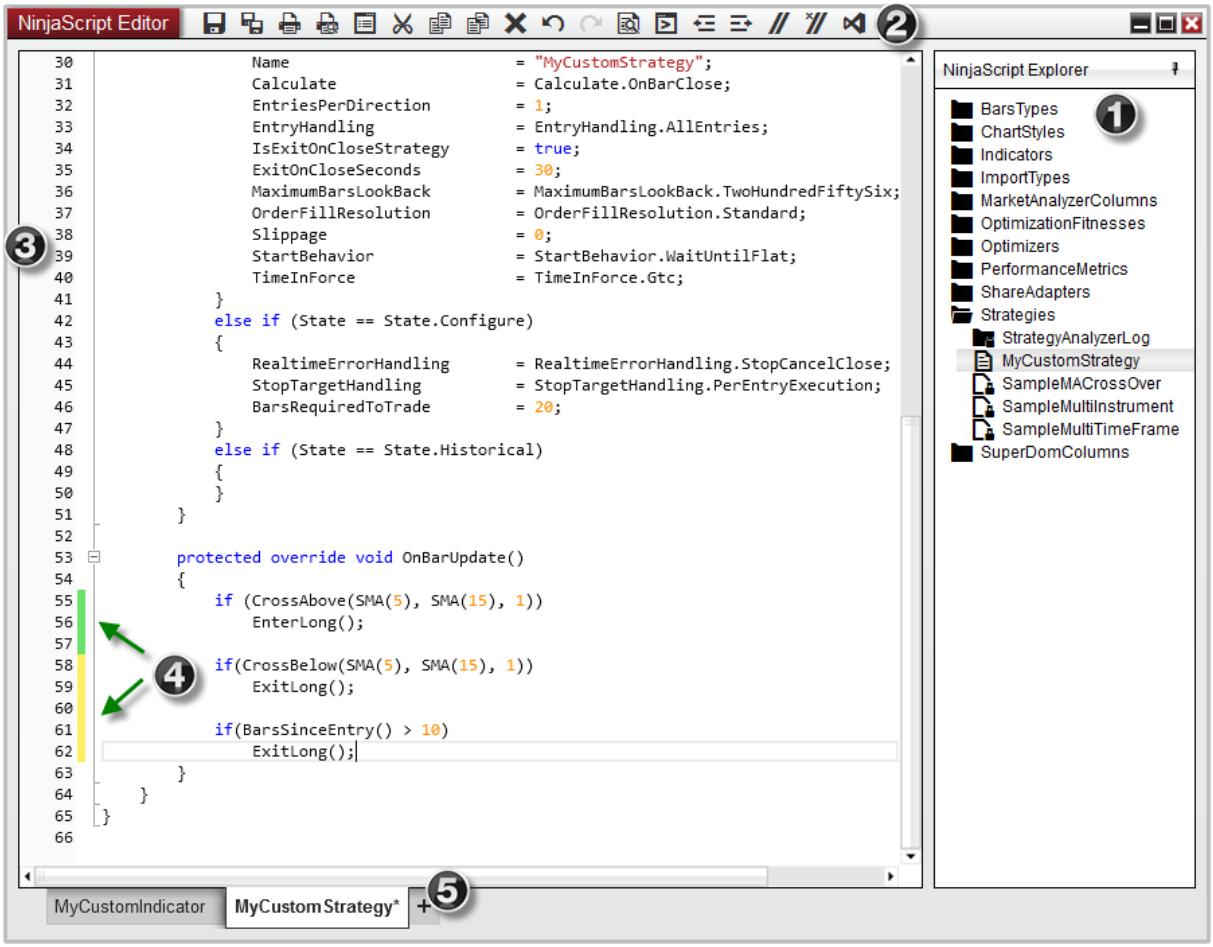
11.4.2 NinjaScript Editor Components

Overview

The NinjaScript Editor is a powerful scripting editor that allows you to create custom indicators, strategies, and any other custom NinjaScript types used to enhance the NinjaTrader platform.

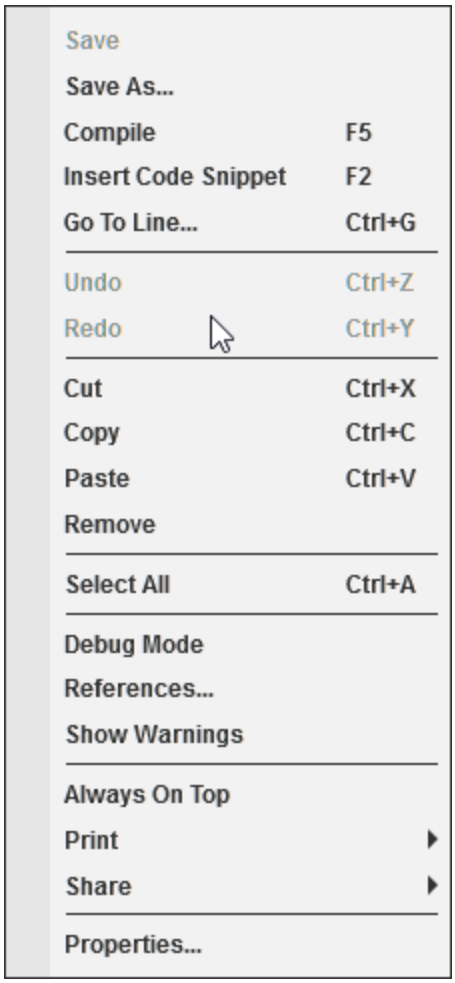
The NinjaScript Editor can be opened by selecting the **New** menu from the NinjaTrader Control Center. Then left mouse click on the menu item **NinjaScript Editor**

1. NinjaScript Explorer - Displays files, folders, and allows for additional file management
2. Tool bar - Moving your mouse over each icon will display the function of the icon button
3. Line numbers
4. Line modification marking - Yellow flags indicate unsaved line modifications where green flags indicate saved modifications
5. Tabs for creating new scripts via the NinjaScript wizard and working on multiple scripts.



Context Menus

Context menus can be opened by right-clicking in the NinjaScript Editor.

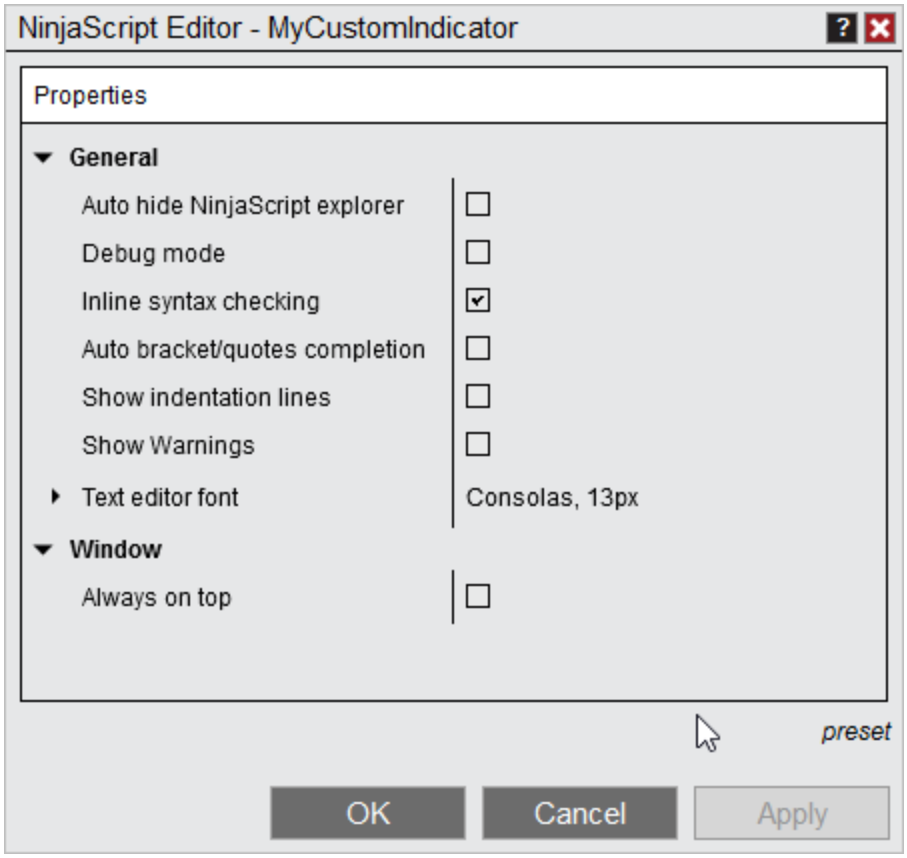


Context Menu Items	
Save	Saves pending changes to the currently open NinjaScript
Save As	Creates a copy of the script and attempts to rename the class name so the new script is unique
Insert Code Snippet	Inserts a code snippet (see Code Snippets for more information)
Go To Line...	Moves the cursor to the line of code specified.

Undo	Undoes the last modification
Redo	Applies the modification that was last Undone
Cut	Removes selected text and copies to clipboard
Copy	Copies selected text to clipboard
Paste	Pastes the text saved in the clipboard
Remove	Removes the selected text
Select All	Selects all text in the Code Editor
Debug Mode	Sets if a debug dll should be generated on compilation (see Visual Studio Debugging for more information)
References...	Opens the list of dll references used by NinjaTrader. This includes dll's used by NinjaTrader and dll's installed with custom Add On's.
Show Warnings	Enables Warning messages to be seen alongside compile errors
Always On Top	Sets the NinjaScript Editor to viewed on top of other windows
Print	Allows printing the content of this window (see Printing Content for more information)
Share	Allows sharing the content of this window (see Sharing Content for more information)

Properties	Opens the Properties menu (see below)

Properties and Definitions



General	
Auto hide NinjaScript explorer	Sets if the NinjaScript explorer should be collapsed by default
Debug mode	Sets if a debug dll should be generated on compilation (see Visual Studio Debugging for more information)
Inline syntax checking	Sets if errors and warnings should be detected as code is

	written (without needing to compile)
Auto bracket completion	Sets if opening characters should automatically bed appended closing characters. Works for (parentheses), [brackets], {braces}, <angled brackets>
Show indentation lines	Displays vertical lines for code formatting
Show Warnings	Sets if code warnings should be show on compilation.
Font	Sets the font options
Window	
Always on top	Sets if the window will be always on top of other windows.

11.4.3 NinjaScript Explorer

The **NinjaScript Explorer** provides a **Folder** view of all the supported NinjaScript categories that can be developed in NinjaTrader.

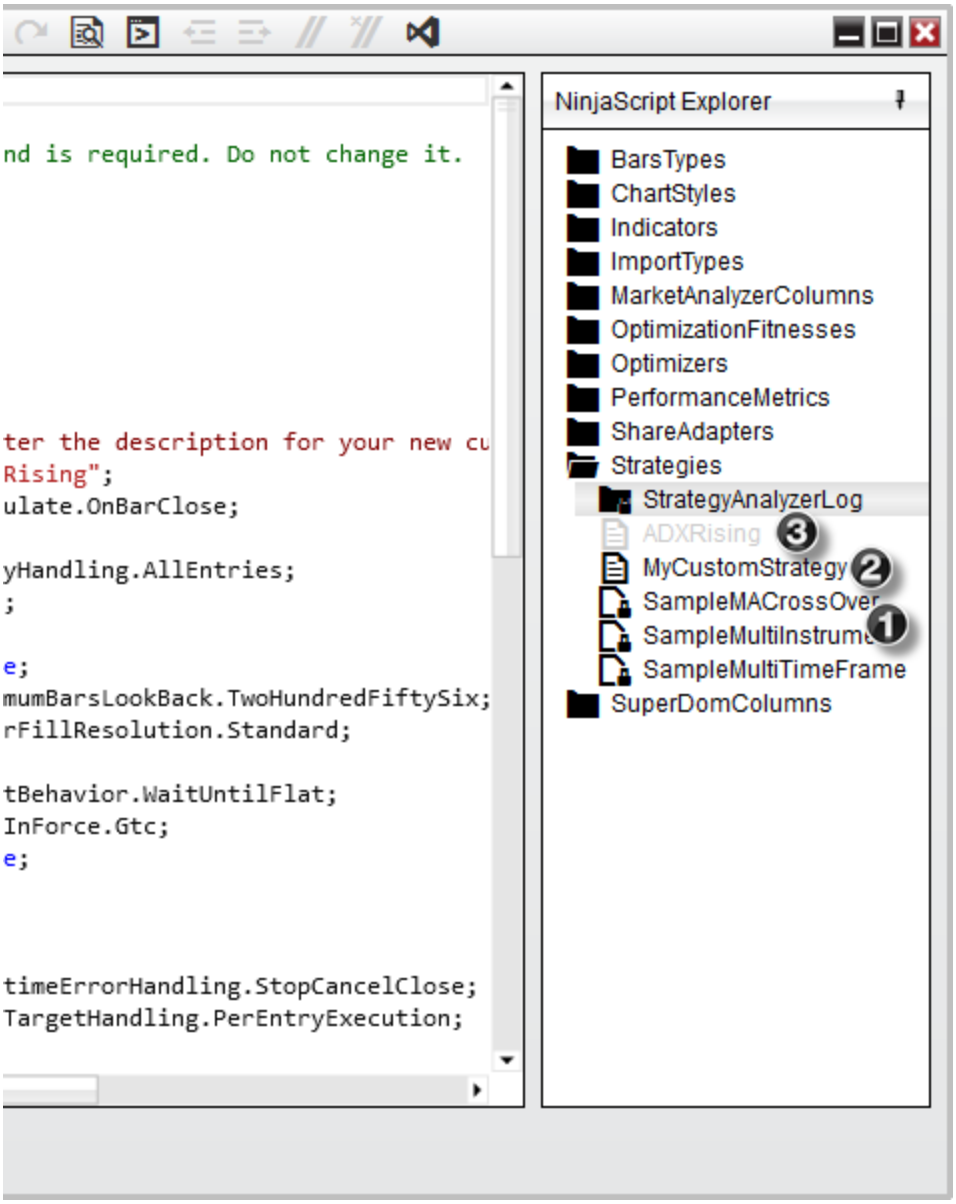
▼ Understanding the NinjaScript Explorer display

Folder Displays

The **NinjaScript Explorer** will organize each script installed on your system by type of NinjaScript object (Indicator, Strategy, SuperDOM Column, etc). Each folder will display the following scripts under each category:

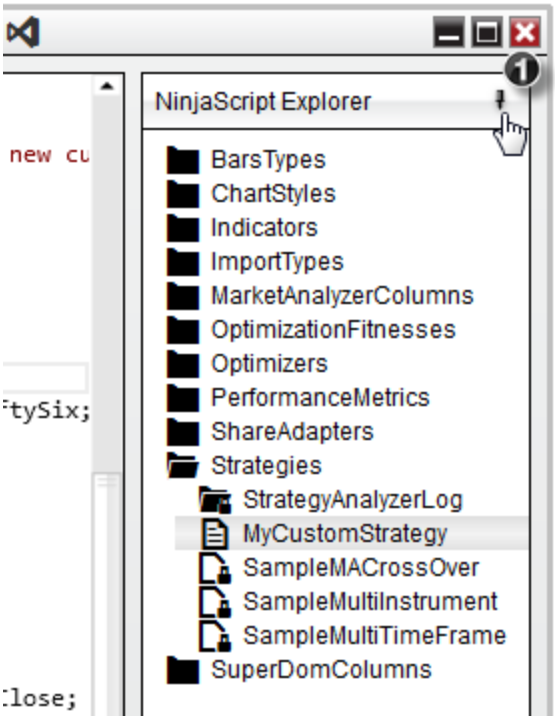
1. Locked scripts	Pre-built system scripts which come installed with NinjaTrader which can be viewed as read-only and are required for compilation (of course you can safe a custom copy of those to modify)

2. Custom scripts	Any script imported, or under development, which can be modified
3. Ignored custom scripts	Custom scripts which have been excluded from compilation (see the "*Excluding a script from compilation*" section below for more information)



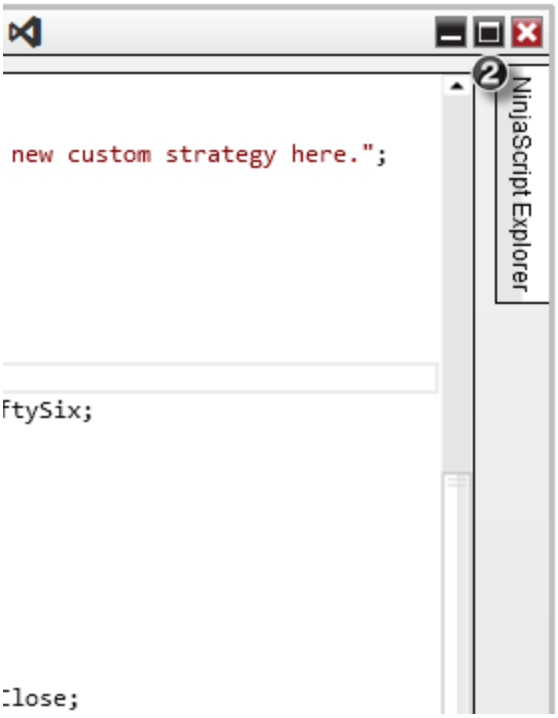
Pinning the NinjaScript Explorer

1. By default the NinjaScript Explorer will be "pinned" to the right side of the NinjaScript editor, however it can be collapsed out of view by pressing the pin icon located at the top right of the explorer window.



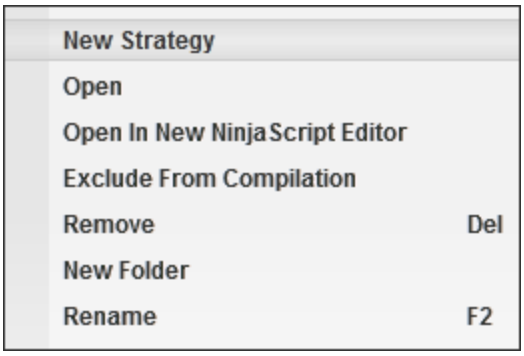
2. Once the NinjaScript Explorer is collapsed, you can quickly bring it back in view simply by selecting the NinjaTrader Explorer tab located on the right side.

Selecting the pin icon again will re-pin the NinjaScript Explorer to the NinjaScript Editor.



Right Click Menu

Right clicking on an individual folder or script will give you a number of different menu items to help with the management of your custom scripts.



New	Opens the NinjaScript Wizard for the relevant object type.
Open	Opens the selected script in a new tab in the current NinjaScript Editor window

Open In New NinjaScript Editor	Opens the selected script(s) in a new NinjaScript Editor window
Exclude From Compilation	Prevents the selected script(s) from being compiled (see the "*Excluding a script from compilation*" section below for more information)
Remove	Removes the current file or folder from the system
New Folder	Creates a new custom folder to organize your scripts
Rename	Renames the current selected file or folder

Managing scripts and folders

Opening an existing Script

There are two ways to open a script:

1. Double left mouse click on the script you wish to view or edit in the current window
2. Right mouse click on the script and select **open** to view or edit the script as a tab the current window, or select **Open in NinjaScript Editor** to open the script as a tab in a new window

Creating new scripts

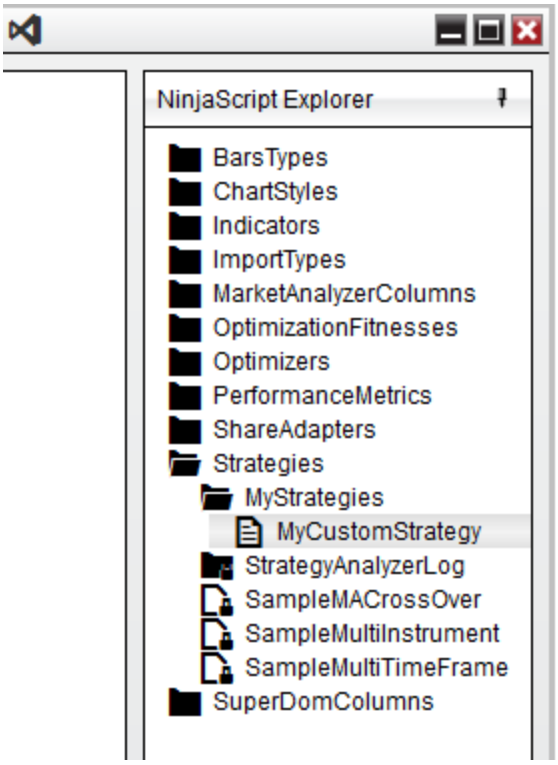
Right clicking on a NinjaScript category and selecting **New...** will open the **NinjaScript wizard** allowing you to create new custom scripts.

Please see the Help Topic on the NinjaScript Wizard for more information.

Creating custom folders

The NinjaScript Explorer gives you the flexibility to relocate and organize your custom scripts in a number of custom user defined folders.

- To create a new folder, simply right click on the NinjaScript folder category you wish to organize, select **New Folder,** and use your keyboard to type a user defined name to identify the folder.



Once you have created your new folder, using your mouse you can drag and drop any custom scripts of it's category under this folder.

Notes:
1. You cannot relocate a locked system script.
2. You can only relocate a custom script if it is closed from the NinjaScript Editor.
3. You can only relocate a script to a folder under it's own folder category (i.e., custom strategies can only be placed under the strategy folder, it would not be possible to move it to an indicator folder)
4. If you move a child script that is called by a parent, please be sure to update the references to the child as well, as the new folder you assigned will automatically move the child to a new namespace

Renaming scripts and folders

There are two methods for renaming custom scripts:

1. Right mouse click on the script from the NinjaScript explorer and select **Rename**.
2. Select the desired script and press the F2 key on your keyboard

Renaming a script will automatically rename all relevant class names and all other required components.

Notes:
1. You cannot rename a locked system script or folder.
2. You can only rename a custom script when it is closed.
3. You can only rename a folder if all of the scripts contained are closed.

Removing scripts and folders

There are two methods for removing custom scripts from your system

1. Right mouse click on the script from the NinjaScript explorer and select **Remove**
2. Select the desired script and press the DEL key on your keyboard

Removing a script will completely delete the script from your system. This action cannot be undone.

Notes:
1. You cannot delete a locked system script or folder.
2. Removing a custom folder will delete all of the scripts contained within

Understanding Folders in the NinjaScript Editor and the File System

When you create a folder in the NinjaScript Editor, it will also be created in the file system on your PC. For example, if you were to create a sub-folder named "MyScripts" in the existing "Indicators" folder, a sub-folder would also be created in the Documents\NinjaTrader 8\bin\Custom\Indicators folder. Once a sub-folder is created, scripts can be created or moved in that folder using the same processes outlined above.

Warning: Changes to Sub-folders directly through the file system will **NOT** be reflected in the NinjaScript Editor. Creating and editing folders must be performed within the NinjaScript Editor.

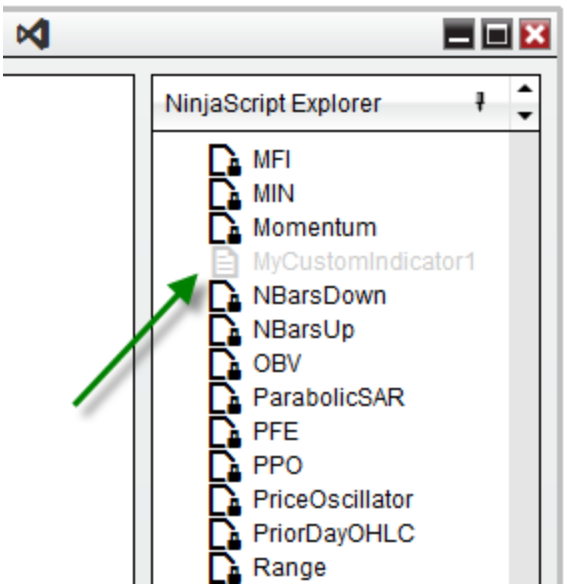
Excluding a script from compilation

Ignoring a script

There may be situations where you have a custom script installed on your system that is preventing other scripts from compiling due to errors. The reason for this is that NinjaTrader will compile ALL custom NinjaScript files into a single DLL for performance reasons. If you find you have installed a script that is giving you errors that you cannot resolve, or you're currently in the middle of developing a script which is unable to compile, you can easily ignore these files from the compiler from the NinjaScript editor.

- To ignore a script, right click on script name and select **Exclude From Compilation**

When a script is ignored, it will be faded from the NinjaScript explorer to indicate that it will not be compiled.



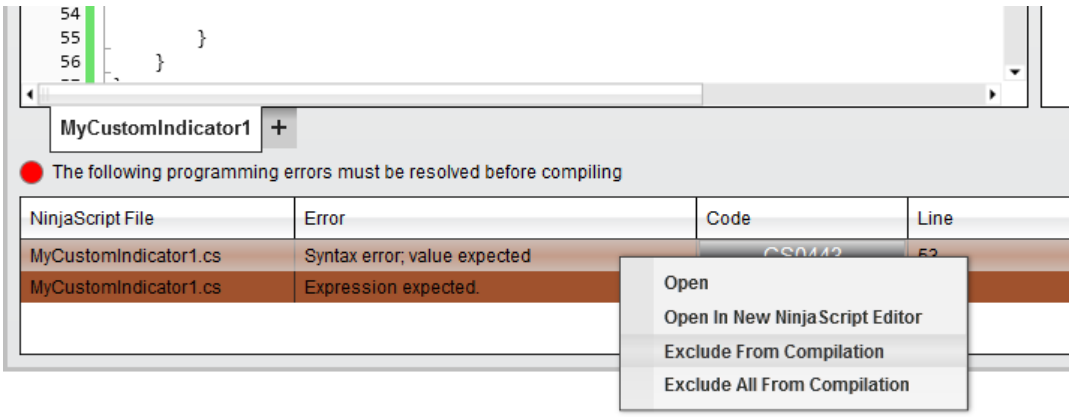
To include this script for the next compilation, simply right click on the script from the NinjaScript Explorer and uncheck **Exclude From Compilation**

Note: You cannot excluded a locked system script or folder

Tip: You will also find an option to exclude scripts from compilation by right

clicking on the listed of errors generated at the bottom of the NinjaScript editor

- Selecting **Exclude From Compilation** will ignore only the NinjaScript file selected
- Selecting **Exclude All From Compilation** will exclude all the NinjaScript files currently with errors



11.4.4 NinjaScript Wizard

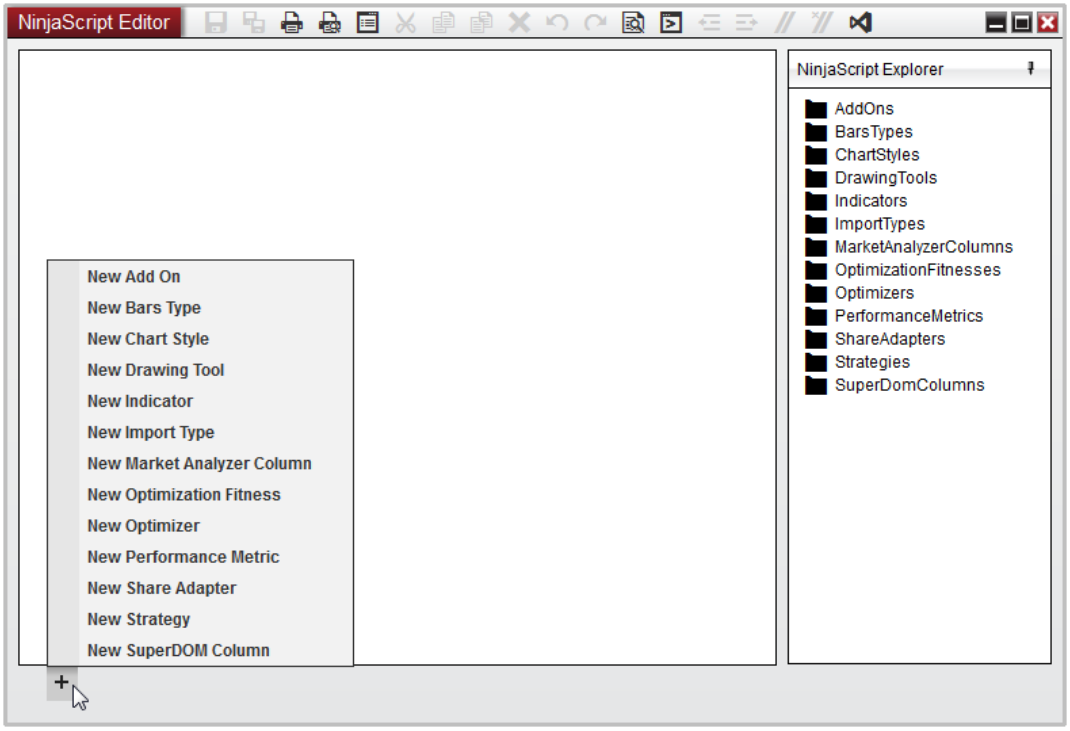
The **NinjaScript Wizard** is used to generate the minimum code to get started programming any supported NinjaScript type. This wizard will allow you to define any **default properties**, add **custom input parameters**, add **additional data series**, and add any relevant **event methods**. There are a number of different properties and options available in the **NinjaScript Wizard** depending on the type of NinjaScript object you are creating.

The information on this page is to be used as a standard overview of the various components of the **NinjaScript Wizard**. For more information on NinjaScript methods and properties, please see the NinjaScript Language Reference section of our Help Guide.

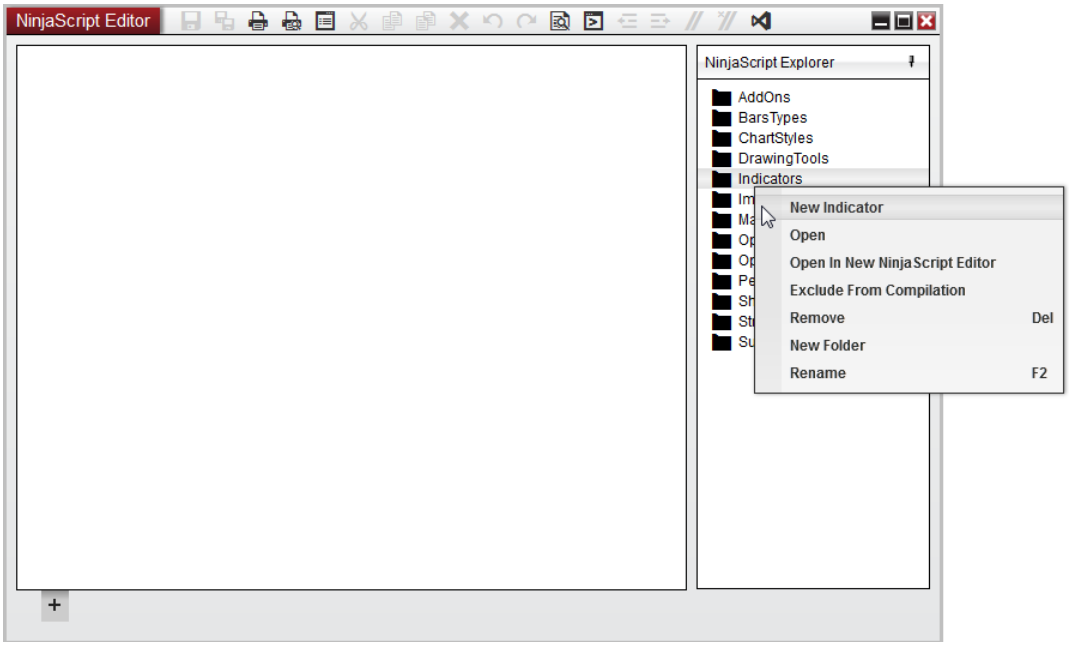
Opening the NinjaScript Wizard

Creating a new NinjaScript file

The **NinjaScript Wizard** can be opened from the **NinjaScript Editor** by selecting the + symbol on the tab row, and then selecting the NinjaScript object type you wish to develop.

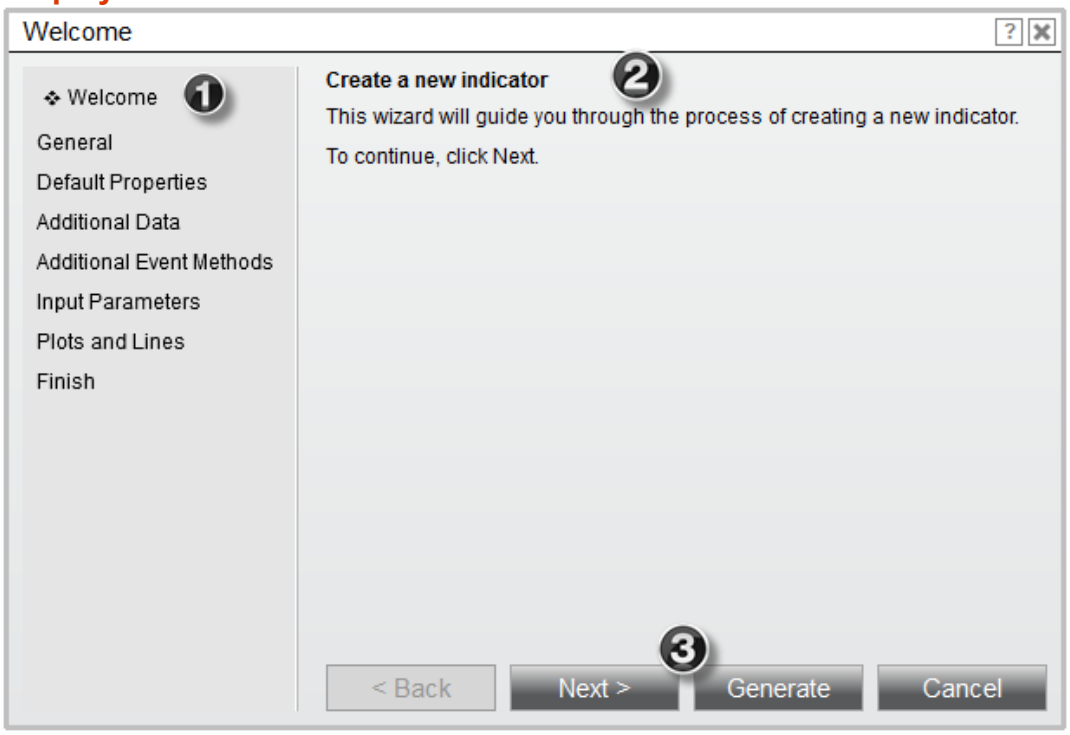


You can also right click on any of the NinjaScript categories listed in the NinjaScript Explorer and select "**New...**"



Understand the NinjaScript Wizard Display

Display Overview



1. Wizard Navigation Menu	Used to navigate to various pages of the wizard. You can skip ahead or return to any page in the wizard at any time.
2. Wizard Screen	Displays relevant information pertaining to the step of wizard you have navigated to and will provide instructions to help you define your script at various stages.
3. Wizard Controls	Buttons used to perform various actions pertaining to the script that is being created. Selecting **Generate** at any time will exit the wizard and open your script in the NinjaScript Code Editor (Note: You cannot return back to the NinjaScript Wizard once the code is generated).

Understanding the Wizard Screens

Optional Pages

The **NinjaScript Wizard** has a number of different pages available used to define various steps of your custom script. Please note that the table below describes ALL of the pages available from the **Wizard**, but does not imply that these steps will be available for the script you are currently creating.

Welcome	The first step of the **Wizard**, used to identify which type of object is being created
General	Used to define a name and description to identify the NinjaScript file
Default Properties	Sets various properties and start behavior for the script being created
Additional Data	Used to optionally add additional data series such as minute, tick, etc or even custom series you may plan on calculating programmatically
Additional Event Methods	Optionally add additional event methods to your custom script, such as OnMarketData, OnMarketDepth, etc
Input Parameters	Used to define any public properties that may be used in your script
Plots and Lines	Optionally add visual plots or lines to your script for charting purposes
Finish	Last page of the **Wizard**, gives you a chance to go back and review each page if desired before finishing generating the script.

11.4.5 Code Snippets

Code Snippets can provide you with useful code templates to speed up your coding process.

▼ Understanding Code Snippet shortcuts

You can quickly add commonly used methods and code structures

via

- Short cut characters
- Clicking on your right mouse button and selecting the menu name "**Insert Code Snippet**"
- Pressing the F2 key on your keyboard

▼ How to use Code Snippet shortcuts via the keyboard

Using the keyboard

Enter the text in the left column and press the "Tab" key within the NinjaScript Editor.

Current Bar Values

cb	CurrentBar
o	Open[0]
h	High[0]
l	Low[0]
v	Volume[0]
i	Input[0]

Previous Bar Values

c1	Close[1]
o1	Open[1]
h1	High[1]
l1	Low[1]
v1	Volume[1]

i1	Input[1]

Indicator Plotting

line	`AddLine(new Stroke(Brushes.Blue, 1), 0, "Line");`
plot	`AddPlot(new Stroke(Brushes.Blue, 1), PlotStyle.Line, "Plot");`

Arithmetic

abs	Math.Abs(value)
min	Math.Min(value1, value2)
max	Math.Max(value1, value2)

Event Handler Callback Methods

account	`protected override void OnAccountItemUpdate(Account account, AccountItem accountItem, double value)`
	`{`
	
	`}`

trade	`protected override void OnAddTrade(Cbi.Trade trade)`
	`{`
	
	`}`

barschange	`public override void OnBarsChanged()`
	`{`
	
	`}`

min max	`public override void OnCalculateMinMax()`
	`{`
	`  // It is important to set MinValue and MaxValue to the min/max Y values your drawing tool uses if you want it to support auto scale`
	`}`

calc perf	`protected override void OnCalculatePerformanceValue(StrategyBase strategy)`
	`{`
	
	`}`

connection	`protected override void OnConnectionStatusUpdate(ConnectionStatus orderStatus, ConnectionStatus priceStatus)`
	`{`
	
	`}`

data point	`protected override void OnDataPoint(Bars bars, double open, double high,`
	`  double low, double close, DateTime time,`
	`  long volume, bool isBar, double bid, double ask)`
	`  {`
	
	`  }`

execution	`protected override void OnExecutionUpdate(Execution execution, string executionId, double price,`
	`  int quantity, MarketPosition marketPosition, string orderId, DateTime time)`
	`  {`
	
	`  }`

fundamental	`protected override void OnFundamentalData(FundamentalDataEventArgs fundamentalDataUpdate)`
	`{`
	
	`}`

data	`protected override void OnMarketData(MarketDataEventArgs marketDataUpdate)`
	`{`
	
	`}`

depth	`protected override void OnMarketDepth(MarketDepthEventArgs marketDepthUpdate)`
	`{`
	
	`}`

mergeperf	`protected override void OnMergePerformanceMetric(PerformanceMetricBase merge)`
	`{`
	
	`}`

moused	`public override void OnMouseDown(ChartControl chartControl, ChartPanel chartPanel, ChartScale chartScale, ChartAnchor dataPoint)`
	`{`
	
	`}`

mousem	`public override void OnMouseMove(ChartControl chartControl, ChartPanel chartPanel, ChartScale chartScale, ChartAnchor dataPoint)`
	`{`
	
	`}`

mouseu	`public override void OnMouseUp(ChartControl chartControl, ChartPanel chartPanel, ChartScale chartScale, ChartAnchor dataPoint)`
	`{`
	
	`}`

optimize	`protected override void OnOptimize()`
	`{`
	
	`}`

orderT	protected override void OnOrderTrace(DateTime timestamp, string message)
	{
	
	}

orderU	protected override void OnOrderUpdate(Order order, double limitPrice, double stopPrice, int quantity, int filled, double averageFillPrice, OrderState orderState, DateTime time, ErrorCode error, string nativeError)
	{
	
	}

position	protected override void OnPositionUpdate(Position position, double averagePrice, int quantity, MarketPosition marketPosition)
	{
	
	}

render	protected override void OnRender(ChartControl chartControl, ChartScale chartScale)
	{
	
	}

windowC	protected override void OnWindowCreated(Window window)
	{
	
	}

windowD	protected override void OnWindowDestroyed(Window window)
	{
	
	}

Control Statements

if	`if (expression)`
	`{`
	
	`}`
	`else`
	`{`
	
	`}`

for	`for (int index = 0; index < count; index++)`
	`{`
	
	`}`

switch	`switch (expression)`
	`{`
	`  case value1:`
	
	`  break;`
	`  case value2:`
	
	`  break;`
	`  default:`
	
	`  break;`
	`}`

Drawing

Shortcut	Method Signature
dap	`Draw.AndrewsPitchfork(this, "MyAndrewsPitchfork", false, 10, Close[10], 5, High[5], 0, Low[5], Brushes.Blue, DashStyleHelper.Solid, 1);`
da	`Draw.Arc(this, "MyDrawArc", false, 10, Close[10], 0, Close[0], Brushes.LimeGreen, DashStyleHelper.Dot, 2);`

dd	`Draw.ArrowDown(this, "MyArrowDown", false, 0, High[0], Brushes.Red);`
du	`Draw.ArrowUp(this, "MyArrowUp", false, 0, Low[0], Brushes.Red);`
ddi	`Draw.Diamond(this, "MyDiamond", false, 0, High[0] + 2 * TickSize, Brushes.Blue);`
dt	`Draw.Dot(this, "MyDot", false, 0,  High[0] + 2 * TickSize, Brushes.Blue);`
de	`Draw.Ellipse(this, "MyEllipse", 10, Low[10], 0, High[0], Brushes.Blue);`
di	`Draw.ExtendedLine(this, "MyExtendedLine", 10, Close[10], 0, Close[0], Brushes.Blue);`
dfc	`Draw.FibonacciCircle(this, "MyFibonacciCircle", true,  10, Close[10], 0, Close[0]);`
dfe	`Draw.FibonacciExtensions(this, "MyFibonacciExtensions", true, 15, Close[15], 10, Close[10], 5, Close[5]);`
dfr	`Draw.FibonacciRetracements(this, "MyFibonacciRetracements", false, 10, Close[10], 0, Close[0]);`
dft	`Draw.FibonacciTimeExtensions(this, "MyFibonacciTimeExtensions", false, 10, Close[10], 0, Close[0]);`
dg	`Draw.GannFan(this, "MyGannFan", true, 10, Close[10]);`
dh	`Draw.HorizontalLine(this, "MyHorizontalLine", Close[0], Brushes.Blue);`
dl	`Draw.Line(this, "MyLine", 10, Close[10], 0, Close[0], Brushes.Blue);`

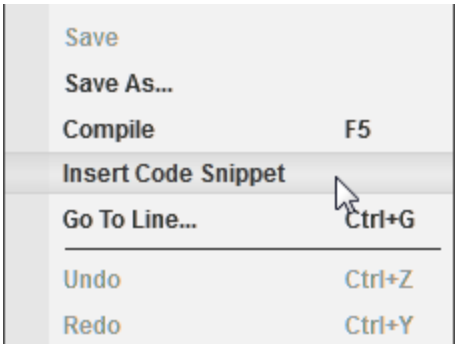
dy	`Draw.Ray(this, "MyRay", 10, Close[10], 0, Close[0], Brushes.Blue);`
dr	`Draw.Rectangle(this, "MyRectangle", 10, Low[10], 0, High[0], Brushes.Blue);`
dre	`Draw.Region(this, "MyRegion", CurrentBar, 0, Bollinger(2, 14).Upper, Bollinger(2, 14).Lower, Brushes.Green, Brushes.Blue, 50);`
drx	`Draw.RegionHighlightX(this, "MyRegionHighlightX", 10, 0, Brushes.Blue);`
dry	`Draw.RegionHighlightY(this, "MyRegionHighlightY", High[0], Low[0], Brushes.Blue, Brushes.Green, 20);`
drr	`Draw.RiskReward(this, "MyRiskReward", false, 0, High[0], 10, Low[0], 2, true);`
dru	`Draw.Ruler(this, "tag1", true, 4, Low[4], 3, High[3], 1, Low[1]);`
ds	`Draw.Square(this, "MySquare", false, 0, High[0] + 2 * TickSize, Brushes.Blue);`
dx	`Draw.Text(this, "MyText", "Sample text ", 0, High[0] + 2 * TickSize, Brushes.Blue);`
dxf	`Draw.TextFixed(this, "MyTextFixed", "Text to draw", TextPosition.TopRight);`
dtc	`Draw.TrendChannel(this, "TrendChannel", true, 10, Low[10], 0, High[0], 10, High[10] + 5 * TickSize);`
dtd	`Draw.TriangleDown(this, "MyTriangleDown", false, 0, High[0] + 2 * TickSize, Brushes.Red);`
dtu	`Draw.TriangleUp(this, "MyTriangleUp", false, 0, Low[0] - 2 * TickSize, Brushes.Blue);`

dv	`Draw.VerticalLine(this, "MyVerticalLine", 0, Brushes.Blue);`

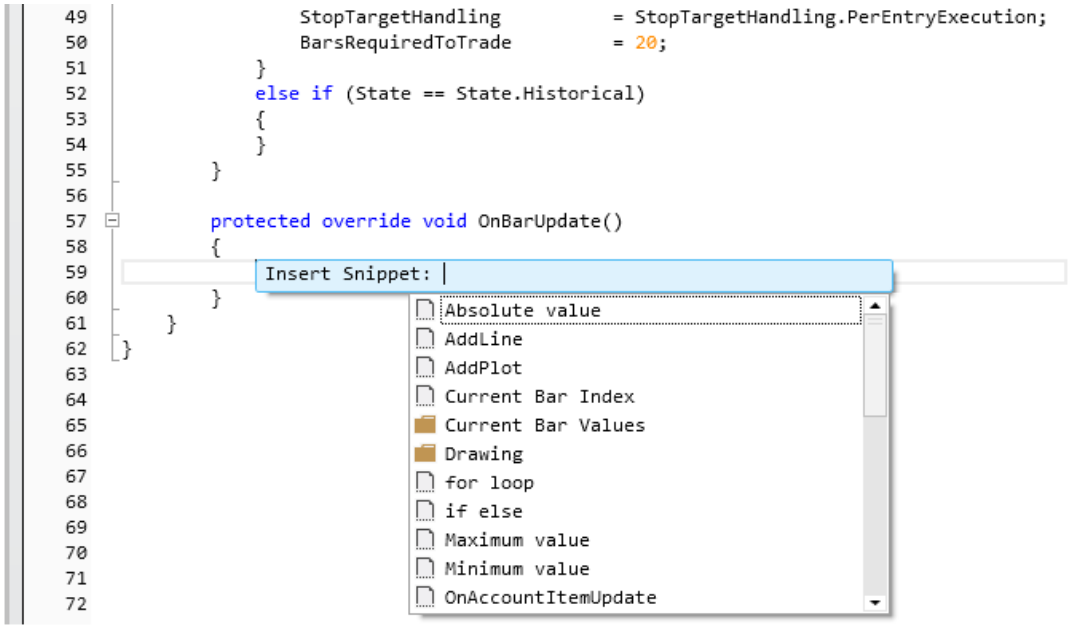
How to insert Code Snippets via the mouse or F2 key

Via mouse or pressing the F2 key

1. Right mouse click in the NinjaScript Editor and select the menu name "**Insert Code Snippet**"



2. A menu will display all available code snippets.



11.4.6 Compile Errors

When compiling a custom indicator or strategy it is possible and likely that you will generate compile errors.

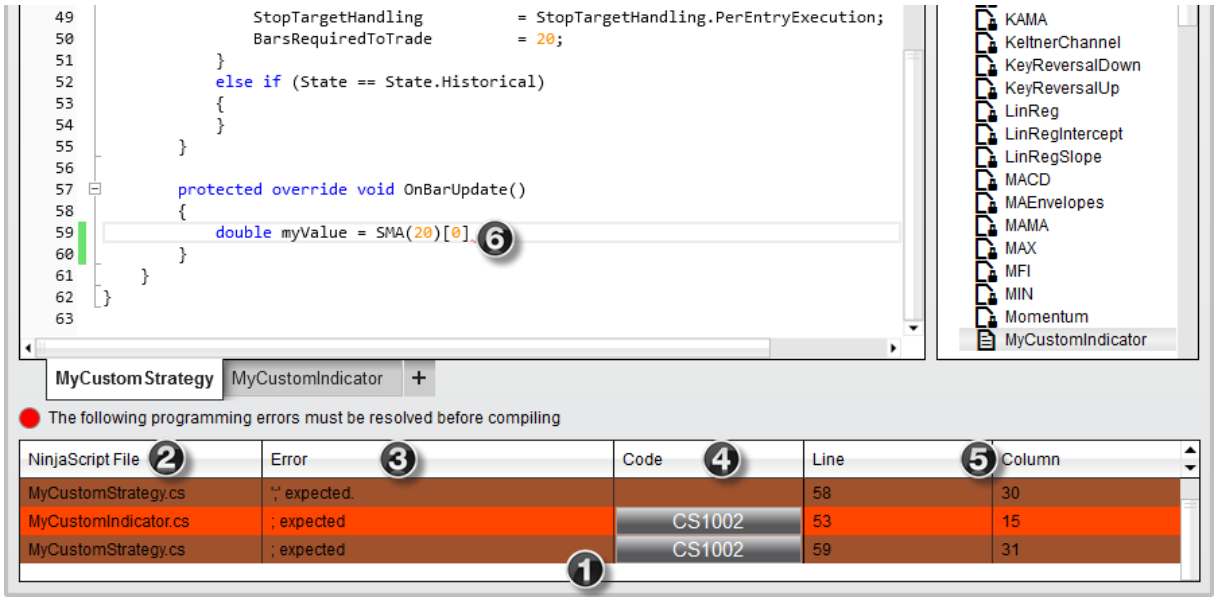
- NinjaTrader will compile ALL NinjaScript files NOT only the file you are working on
- A list of compile errors for all files will be displayed in the lower portion of the NinjaScript Editor
- Double click on an error to load the problem file and highlight the problem area
- Click on the error code to bring up Help Documentation on a specific error
- Right click on the error to exclude the problem file from compilation (see the section on Excluding a script from compilation for more information)

The image below illustrates a compile error

1. Section where compile errors are displayed. Errors in the current loaded file are color coded a light color while errors in other files have a darker color code.
2. The file that contains the error
3. A description of the error
4. A error code link that will open the Help Guide with any relevant error code information
5. Line number and column number of the error
6. Error is underlined with a red wavy line

The error highlighted by icon (6) below shows that the expression is not closed with a semicolon. The expression should be:

```
double myValue = SMA(20)[0];
```



11.4.7 Intelliprompt

What is Intelliprompt?

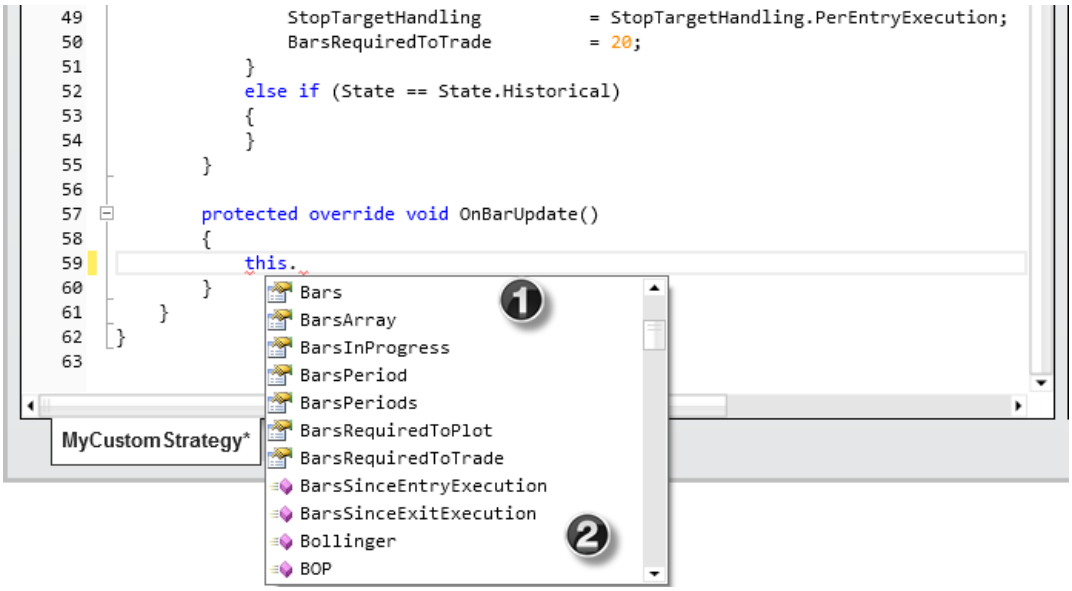
Intelliprompt is a form of automated autocompletion popularized by the Microsoft Visual Studio Integrated Development Environment. It also serves as documentation and disambiguation for variable names, functions and methods. Intelliprompt is built into the NinjaScript Editor resulting in an efficient environment to code your custom indicators and strategies.

How to access the Intelliprompt list box

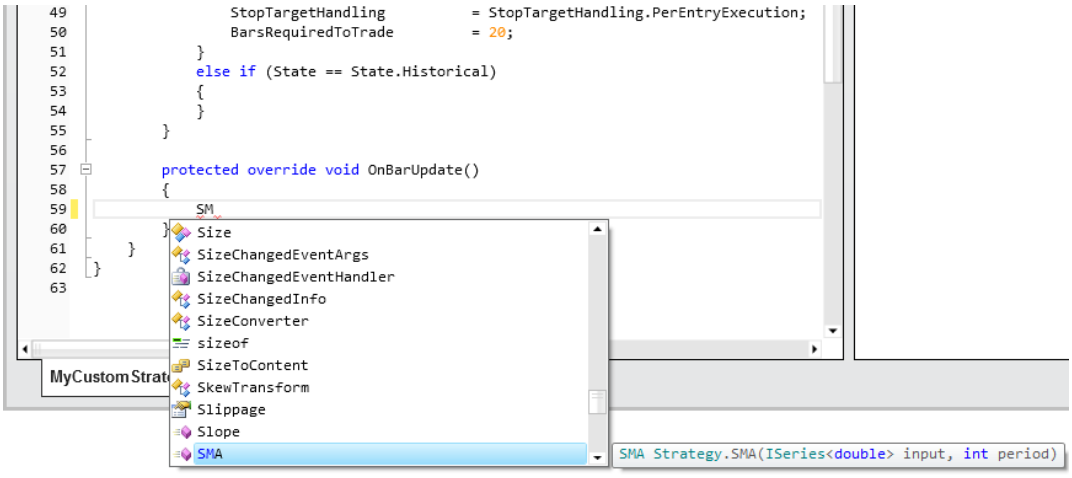
Within the NinjaScript Editor you can type "this." to bring up the Intelliprompt list box. The list box contains all methods (functions) and properties available for use. You can select a method or property by simply selecting it via your mouse, or scrolling with your up or down arrow key. Pressing either the "Tab" or "Enter" key will automatically insert the code into the NinjaScript Editor. While in the list box, you can press any letter key to rapidly scroll down to the next property or method beginning with the letter of the key you pressed.

In the image below:

1. A property
2. A method



If you know that you want to access the Simple Moving Average indicator method which is SMA(), and you think it starts with "SM" enter "SM" and press CTRL-Space Bar which would display the Intelliprompt list box below.



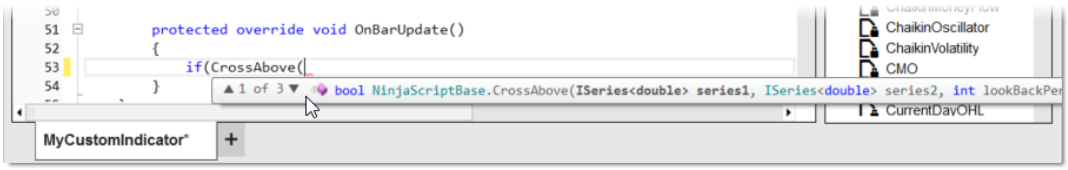
Pressing CTRL + space bar after any text will always either

- Bring up the Intelliprompt list box with related methods and properties
- Automatically insert code if the text can uniquely identify a method or property
- More keyboard shortcuts could be reviewed under this link.

Understanding Method Description and Signatures

When selecting a method

1. Type in "(" to display the method description and signature
2. A light yellow colored frame will appear with the method description and available signatures
3. In the image below you will see "1 of 3" which means that we are looking at the first of three available method signatures. You can scroll through all available signatures by pressing on the arrow up and down keys.



What is a method signature?

A method signature is a common term used in object-orientated programming to uniquely identify a method. This usually includes the method name, the number and type of its parameters and its return type.
From the image above, the DMI() method represents the Dynamic Momentum Index indicator has two method signatures:

```
DMI(int period)
DMI(IDataSeries inputData, int period)
```

11.4.8 Output

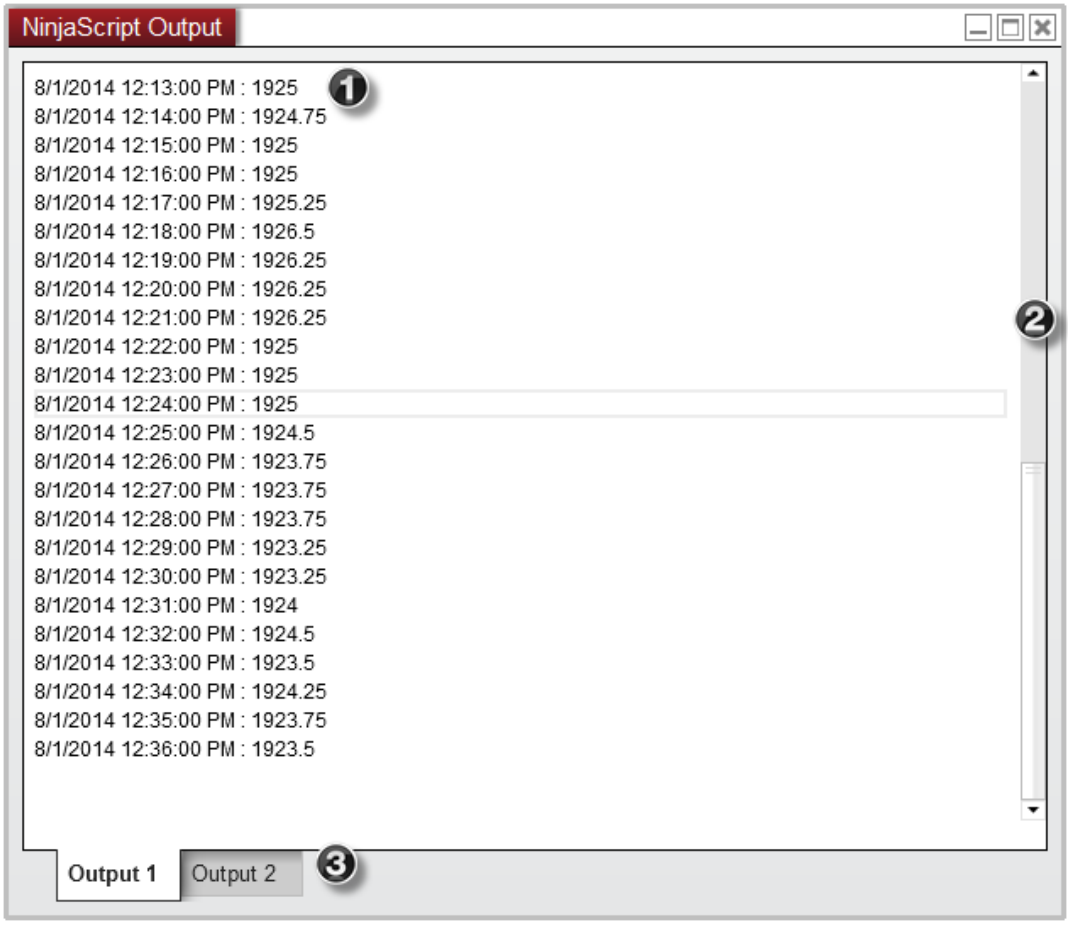
The NinjaScript Output is a powerful debugging tool which can be used to further analyze valuable information generated by your NinjaScript files. The **Output** window will only display data when other debugging methods such as the Print() or TraceOrders (for strategies) have been configured in a custom script.

You can open the NinjaScript Output window by going to the New menu, and selecting **NinjaScript Output**

Understanding the Output window display

Display Overview

1. Output table	The main component of the Output window, will display any Print or Information message sent from a script
2. Scrollbar	Used to navigate up/down on the output window
3. Output tabs	Two tabs available allowing you to separate the Print information for separate scripts.
4. Line highlight	Left clicking on a line will highlight a particular point of interest and will remain highlighted as the Output window updates or is scrolled up and down



Right click menu

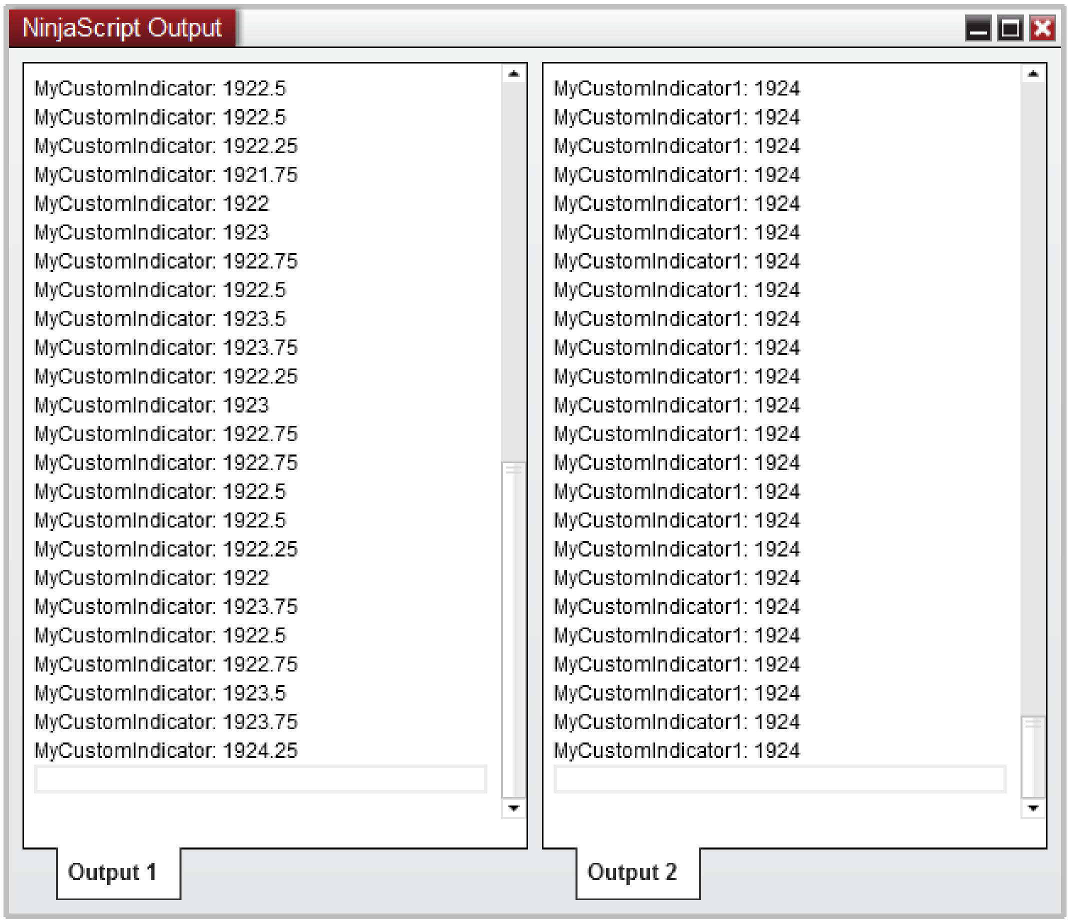
Clear	Clears the current content of the select Output window tab
Find...	Searches for a term in the Output window
Save As...	Saves the current content of the Output window in a text file
Always On Top	Sets the window to always be on top of other windows
Dual View	Enables/Disables the splitting of the Output tabs between

	the window allowing you to view both tabs at simultaneously
Synchronize Vertical Scrolling	When enabled, both tabs will scroll up/down at the same time and pace
NinjaScript Utilization Monitor	Opens the NinjaScript Utilization Monitor window
Print	Displays options for printing the current window content to your printer
Share	Displays the Share options
Properties	Sets the Output window properties

Understanding the dual tab view

Dual view

You can optionally split the **Output window** tab's into a **dual view** which will allow you to view both outputs windows at the same time. To enable this feature, simply right click on the Output window and select **Dual view**



In the image above, we have enabled the dual view mode where we can see the output from two separate indicators. **MyCustomIndicator** is programmed to print to the **Output 1** tab, while **MyCustomIndicator1** is programmed to print to the **Output** 2 tab. (Please see the Help Guide article on the PrintTo() method for more information on programming a custom script to print to a second output tab)

Synchronized Scrolling

While the Output window is in **Dual view** mode, each output window will have an independent scroll bar which allows you to navigate each output tab separately. However if desired, you can synchronize the vertical scrolling between these two windows which will allow you to easily compare the output from two difference scripts where both tabs will scroll up/down equally at the same time.

To enable this feature, right click on the Output window and select the **Synchronized Vertical Scrolling** menu item.

Searching and highlighting

Using the Find Tool

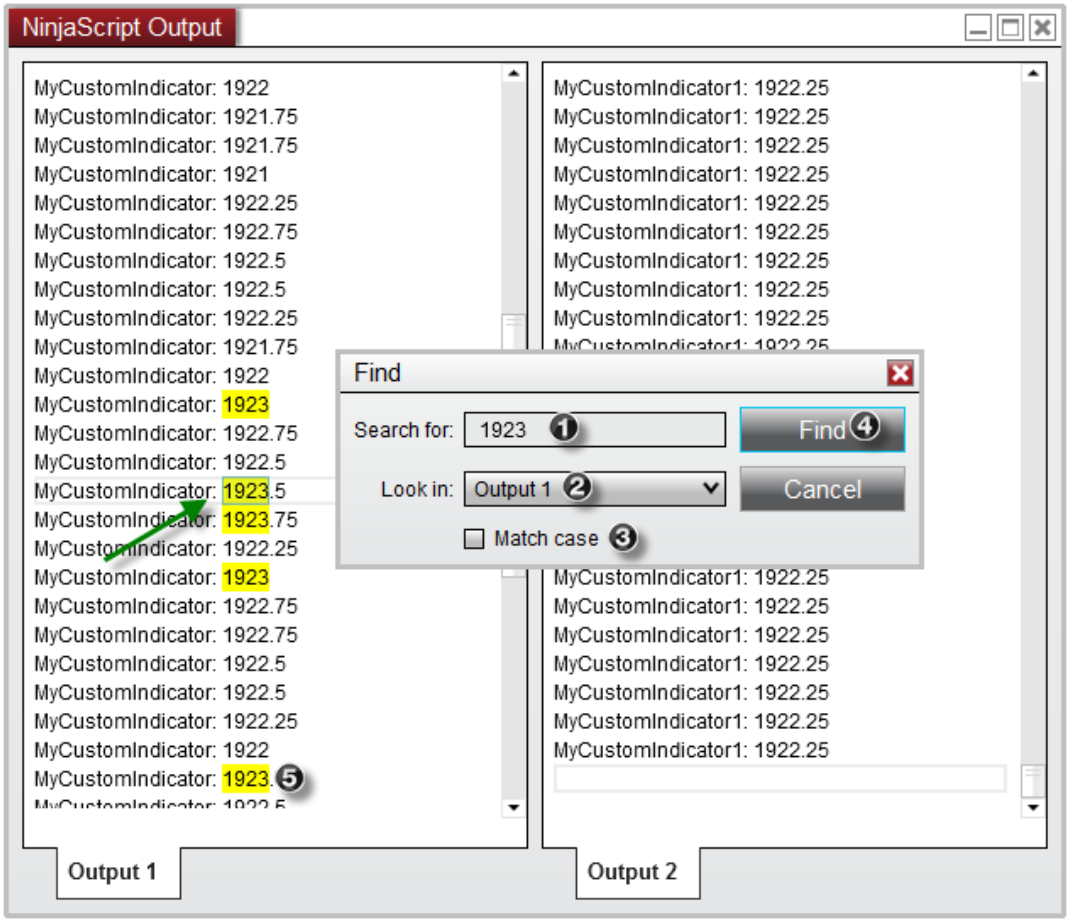
If you would like to search for a specific value or text displayed in your Output window, you can use the **Find** tool to both highlight and navigate any terms that match your search.

To bring up the Find menu, right click on the Output window and select **Find** (or use CTRL + F as a keyboard shortcut).

To search for a specific term:

1. Enter the text/value you wish to search for
2. Specify which **Output tab** you would like to search
3. Optionally check **Match case** to only look for terms which contain the exact text case of your term (i.e., Close would not be the same as close)
4. Select the **Find** button which will navigate to and highlight the next matching term (indicated by the green arrow in the image below)
5. The search will also highlight any other matches in the output window that match the search

Selecting the Find button again will continue to search through the Output window and will highlight the next match.



Tip: Without the **Find** tool, you can also highlight terms simply by double clicking on the text in the output window. Doing so will automatically search the highlighted term and highlight all results.

Clearing and saving output information

Clearing Output Information

After some time, you may feel the need to erase all the current information in the current output tab. To do so, simply right click on the current output tab and select "**Clear**".

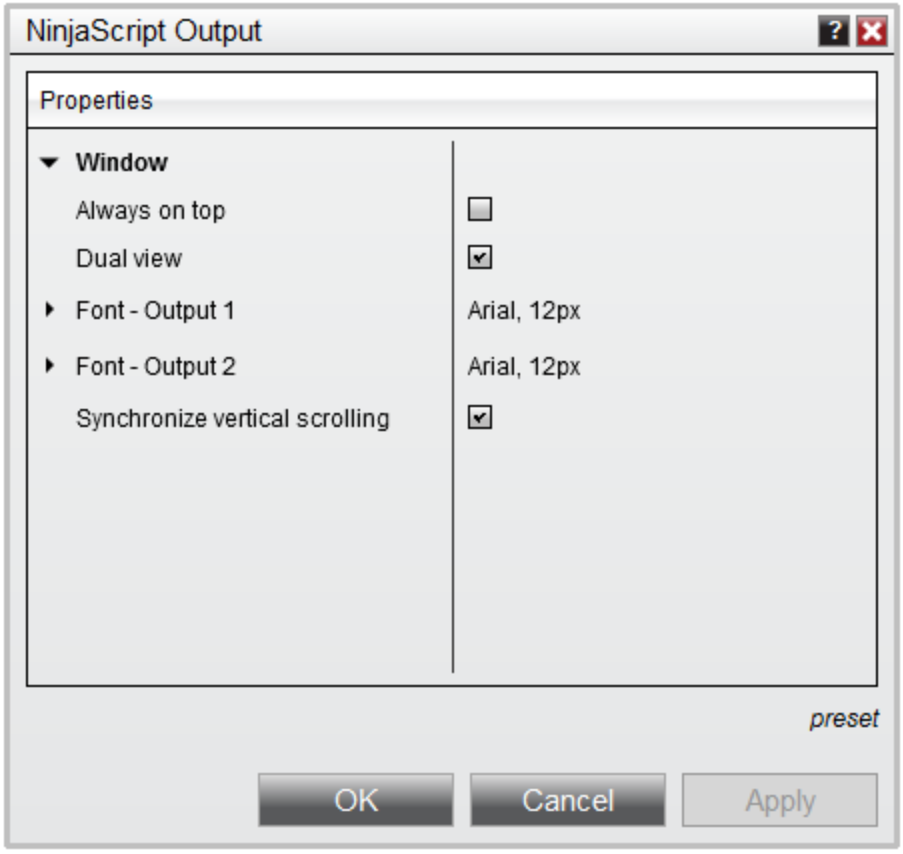
Tip: You can also use the ClearOutputWindow() method in directly your script to automatically clear the output content at a specific event or interval

Saving Output Information

If you would like to save the current results of your output, you can right mouse click on the desired output tab and select "**Save As**". Doing so will provide you with a **Save As** dialog window which will allow you to save your output in a Text (.txt) file at any location on your computer.

Output window properties

The following properties are available for configuration within the **NinjaScript Output** properties window:



Window	
Always on top	Sets the Output window to be on top of other windows
Dual view	Enables/Disables the splitting of the Output tabs between

	the window allowing you to view both tabs at simultaneously
Font - Output 1	Sets the font display for the Output 1 tab
Font - Output 2	Sets the font display for the Output 2 tab
Synchronize vertical scrolling	Enables/Disables where both tabs will scroll up/down at the same time and pace

NinjaScript Utilization Monitor

The NinjaScript Utilization monitor is opened via a right click in the NinjaScript Output window and will be mainly used as a diagnostic tool for performance issues.

It will track any NinjaScript objects total resource time from the moment the window was opened:

- NinjaScript Utilization monitor window will not be saved in any workspace file
- NinjaScript Utilization monitor window works across all workspaces using a single window instance
- NinjaScript Utilization monitor window will work even if hidden / non-visible

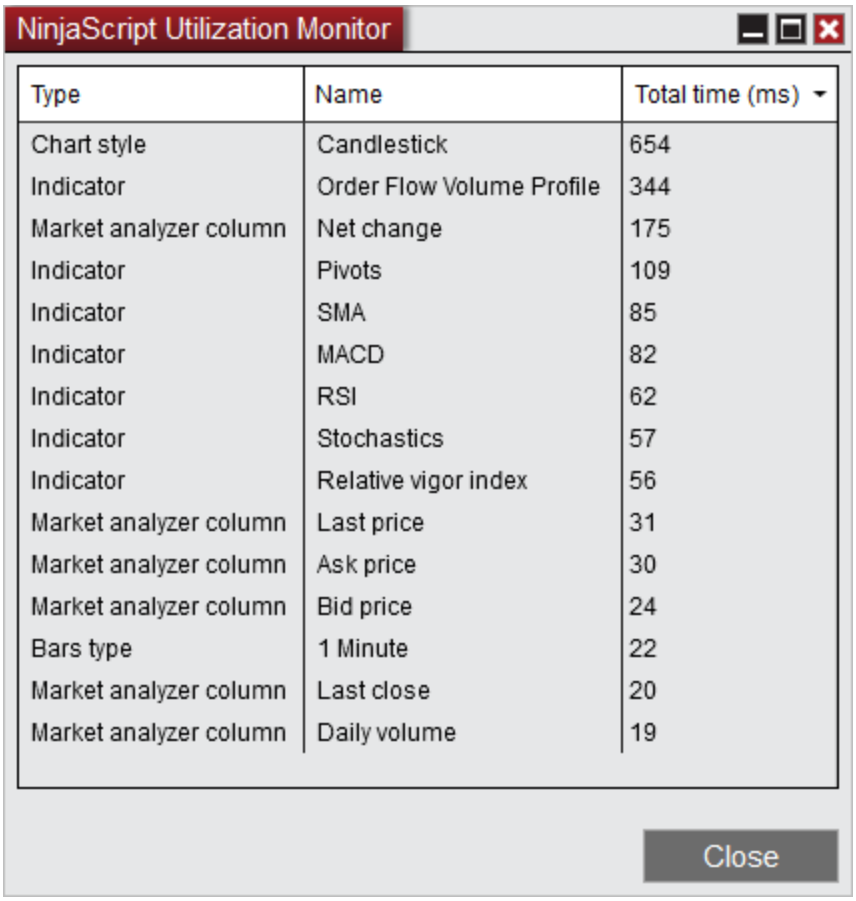
When using it as debugging aid, it's recommended to focus on the top resource using NinjaScript's, while absolute total time is negligible.

Also it's important to understand that a resource heavy NinjaScript could be meaning:

a) the NinjaScript may not be coded as efficiently as possible and would be worthwhile to review if everything has been done to achieve optimal performance
b) it could be doing intense / steady calculations by design and a higher than average resource use therefore could likely not be avoided.

It should be thought of a gauge to see where likely performance / code optimization time is likely most wisely spend if the overall performance footprint is to be reduced.

Our support team is trained with this process and is available to assist.



11.4.9 Visual Studio Debugging

You can debug your NinjaScript objects using Microsoft Visual Studio. NinjaScript objects are compiled into a single DLL, named "NinjaTrader.Custom.dll." When debugging, a special debug DLL is created for temporary use, with the same name as the release version.

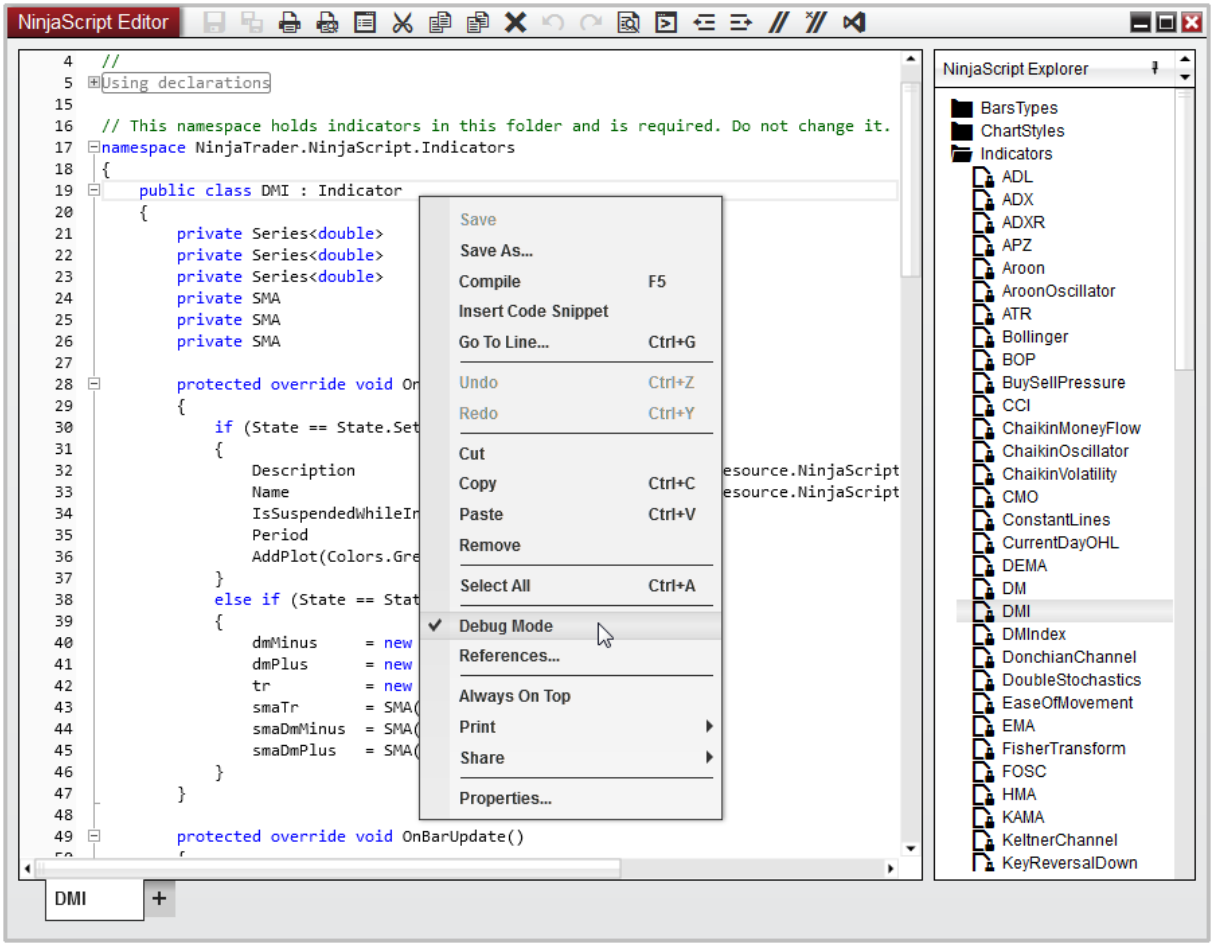
Notes:

- Using the debug DLL can incur a runtime performance impact, so it is recommended to disable Visual Studio debugging and re-compile your scripts when finished. This will replace the debug DLL with the release version.

- The Visual Studio button will work with Visual Studio 2019 or 2022 - if multiple versions are installed, it will start the highest one.

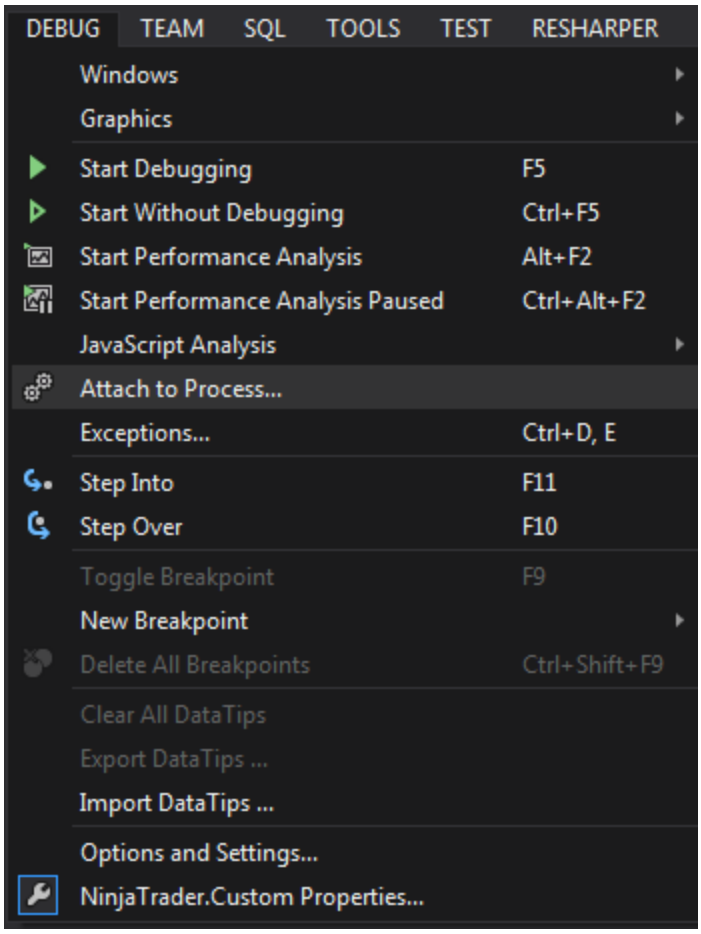
Using Visual Studio Debugging

1. In the NinjaScript Editor, enable "Debug Mode" via the right-click menu, as seen in the image below. After this, compile your scripts to create the debug DLL.

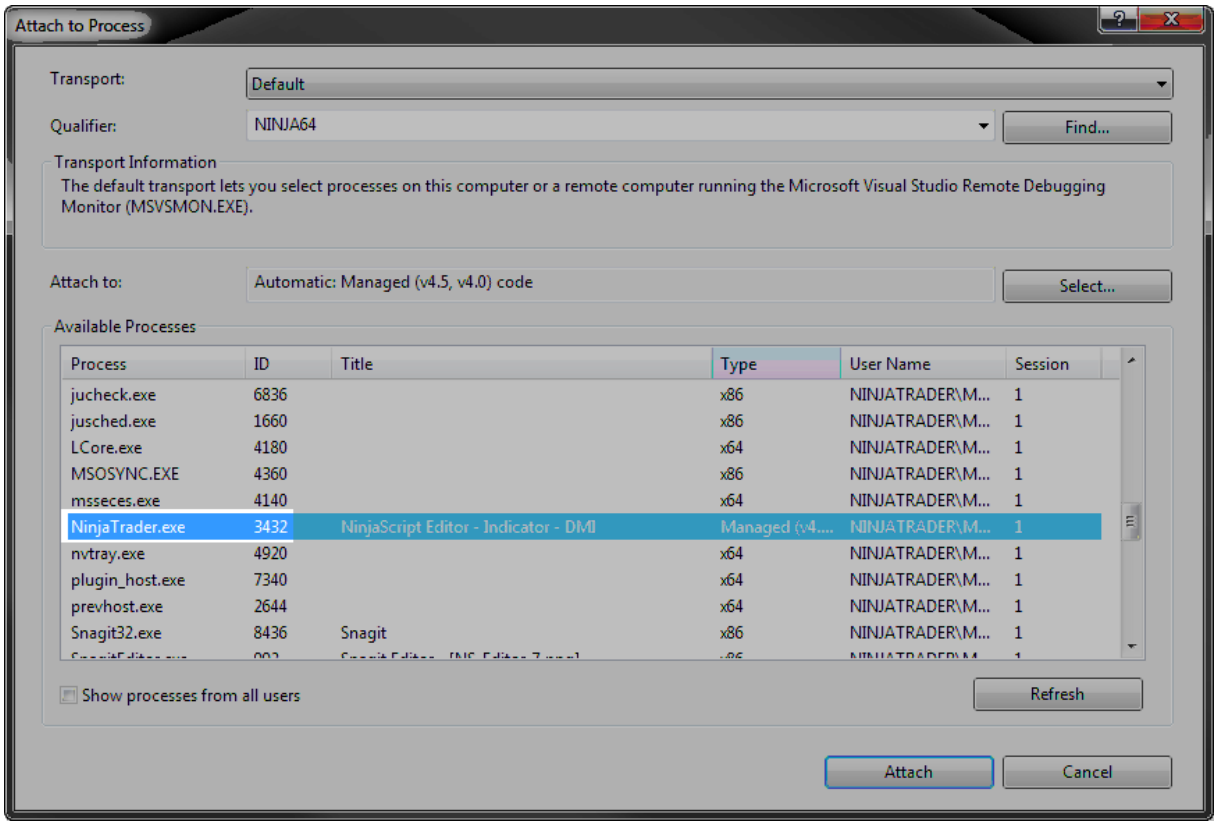


2. From the NinjaScript Editor, click on the Visual Studio icon from the tool bar, which will automatically load the NinjaTrader.Custom project with your installed version of Visual Studio.

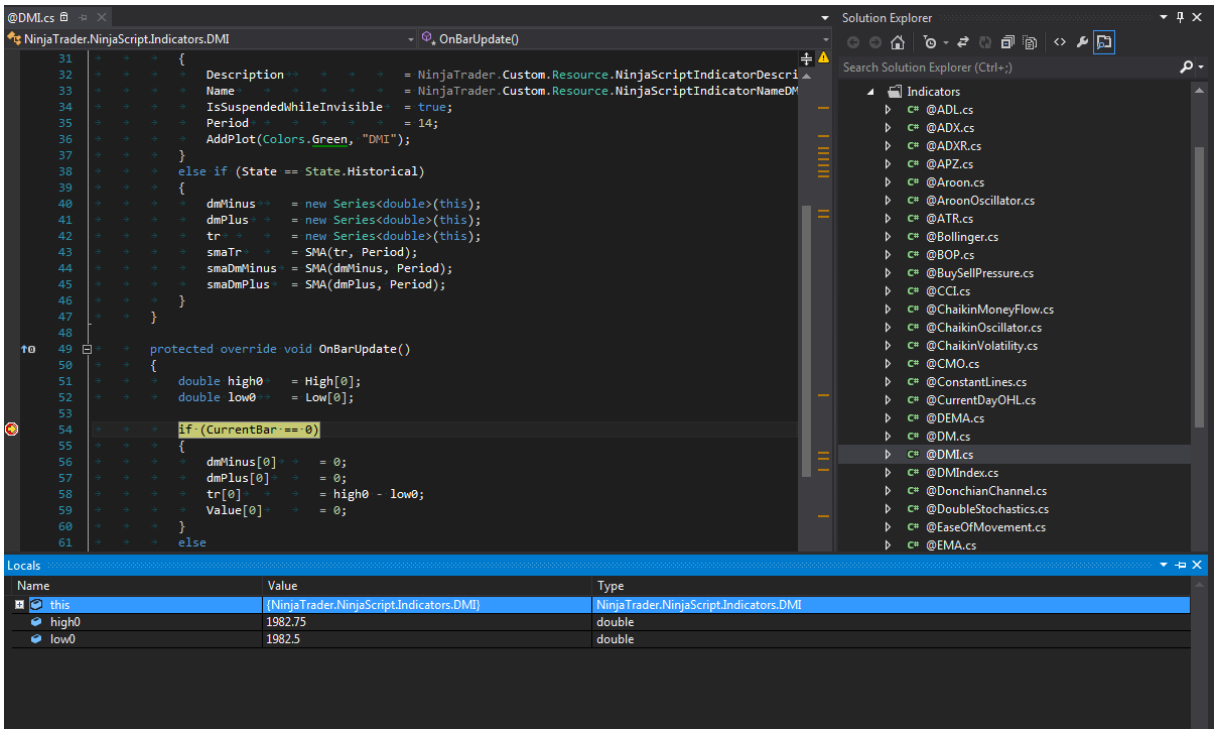
3. In Visual Studio, select Debug, then select **Attach to Process**



4. Select NinjaTrader from the list of processes, then select **Attach.** Be sure the "Attach to" field is set to "Automatic: Managed code" or "Managed code".



4. Open the NinjaScript source file within Microsoft Visual Studio and set your break point(s)



5. Run your NinjaScript object in NinjaTrader and it should stop at your break points and all the debugging tools and information should be available to inspect the current state of the code.

Tip: You can also use Visual Studio as editor for your NinjaScript files - for that open the project as in step 2 above and then use Visual Studio for editing and once done **save the file** (don't run or build the solution then in Visual Studio), preferably with the NinjaScript editor opened still at the same time, so changes would be auto compiled in then.

11.4.10 Editor Keyboard Shortcuts

The NinjaScript Editor includes a range of keyboard shortcuts not available in other areas of the platform. Below is a list of available shortcuts and the actions they perform:

Ctrl + C, Ctrl + Insert	Copy to Clipboard
Ctrl + X, Shift + Delete	Cut to Clipboard
Ctrl + L	Cut line to Clipboard
Ctrl + V, Shift + Insert	Paste from Clipboard
Ctrl + Y, Ctrl + Shift + Z	Redo action
Ctrl + Z	Undo action
Ctrl + Backspace	Backspace to previous word
Ctrl + Shift + L	Delete line
Ctrl + Delete	Delete to next word
Ctrl + Enter	Open line above

Ctrl + Shift + Enter	Open line below
Ctrl + Space	Intelliprompt auto-complete
Ctrl + Shift + Space	Intelliprompt show parameters
Ctrl + T	Transpose characters
Ctrl + Shift + T	Transpose words
Shift + Alt + T	Transpose lines
Ctrl + Shift + U	Make uppercase
Shift + Tab	Remove tab indent
Alt + Up	Move selected lines up
Alt + Down	Move selected lines down
Ctrl + Left	Move to previous word
Ctrl + Right	Move to next word
Ctrl + Home	Move to document start
Ctrl + End	Move to document end
Ctrl + PageUp	Move to visible top of document
Ctrl + PageDown	Move to visible bottom of document
Ctrl +]	Move to matching bracket

Ctrl + Down	Scroll down
Ctrl + Up	Scroll up
Shift + PageUp	Select all above
Shift + PageDown	Select all below
Ctrl + Shift + PageUp	Select visible area above
Ctrl + Shift + PageDown	Select visible area below
Ctrl + Shift + W	Select word
Ctrl + Shift +]	Select up to matching bracket
Shift + Alt + Arrow Keys	Expand/contract selection region

11.5 Educational Resources

Education Resources

The following pages contain valuable resources for developing your custom NinjaScript objects within NinjaTrader. Continuing education and resources can be found on the NinjaTrader Support Forum.

Development

› AddOn Development Overview
› Considerations For Compiled Assemblies
› Developing for Tick Replay
› Historical Order Backfill Logic
› Multi-Threading Consideration for NinjaScript
› Multi-Time Frame & Instruments
› Understanding the lifecycle of your NinjaScript objects
› Using 3rd Party Indicators
› Using ATM Strategies
› Using BitmapImage Objects with Buttons
› Using Historical Bid/Ask Series
› Using Images and Geometry with Custom Icons
› Working with Brushes
› Working with Pixel Coordinates
› Working with Price Series

Reference

› Reference Samples
› Tips
› C# Method (Functions) Reference

Tutorials

> Basic Programming Concepts
> Developing Indicators
> Developing Strategies

11.5.1 AddOn Development Overview

AddOn Development Basics

The NinjaScript AddOn framework provides functionality reaching across the NinjaTrader platform while granting access to certain core methods and properties not contained within the NinjaScript namespace. In addition to creating your own independent window or modifying

the user interface and functionality of existing NinjaTrader windows (charts, etc.), AddOns can also subscribe to live market data, access account information, and more.

Note: Most of the topics covered on this page and its sub-pages can be seen in a fully functional example of the **AddOn Framework** accessible on this page. There are two versions deployed when you can access depending on your desired development environment. The pros and cons of each approach are described in the following section. In either version, the heavily commented code in the example can supplement the information on these pages to provide deeper insight.

NinjaScript Editor Development Environment (NinjaScript Basic)

The NinjaScript Editor can be used to create and write custom AddOns in C#

Pros

- Use the familiar NinjaScript editor (if you are uncomfortable with Visual Studio)
- Changes to the AddOn are reflected immediately upon NS Editor Compile and does not require restart

Cons

- If you wish to design a custom NTWindow, XAML files cannot be edited in the NinjaScript editor.
- NinjaScript editor lacks support of common development and debugging tools available in an IDE's like Visual Studio

Below is a NinjaScript Editor compatible zip file (which also contains a XAML file)

- Download AddOn Framework NinjaScript Basic file to your desktop
- From the Control Center window select the menu Tools > Import > NinjaScript
- Select the downloaded file

Once imported, the AddOn can be launched via the New menu in the Control Center

AddOn Development Environment (Visual Studio Advanced)

Since AddOns can include multiple classes, unique user interfaces, and various file types (XAML, sounds, etc.), the recommended development environment for AddOns differs from other NinjaScript Types. Following the guidelines below to set up an AddOn development environment can help to streamline the process.

Pros

- Use Visual Studio or a comparable IDE to create a solution linking all project files together
- Use your IDE to build a DLL, rather than exporting through NinjaTrader
- This will allow you to bundle XAML and other files into the DLL

- Set a post-build event to place the DLL into the appropriate folder (NinjaTrader 8/bin/Custom)
- Set a Debug Start Action to launch NinjaTrader

Cons

- NinjaTrader needs to be restarted in order to re-load the compiled DLL after changes

If you use this setup and build a DLL with your IDE, the IDE will automatically place it where it needs to be and immediately launch the platform for testing any changes.

Below is a complete Visual Studio project with this setup in place. Simply unzip the contents of this archive to your desired location, then open the "NinjaTraderAddOnProject.sln" solution in Visual Studio.

Download Visual Studio Solution for AddOn Development

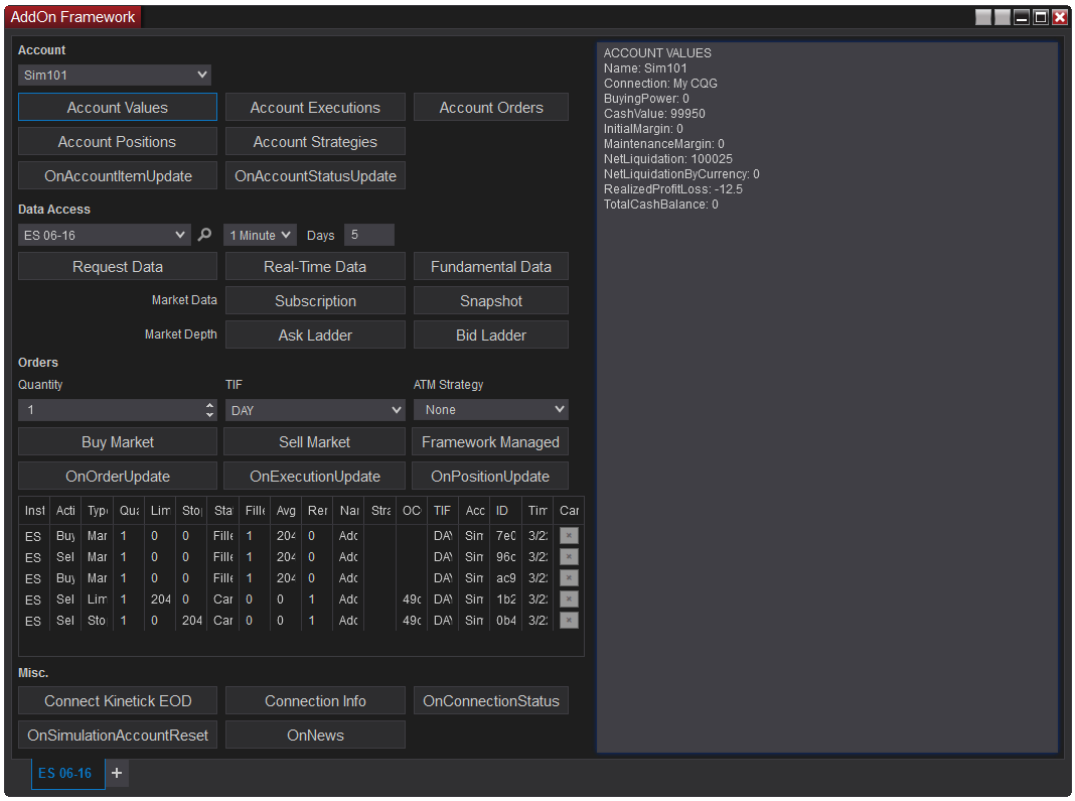
Notes:

- This Visual Studio solution cannot be imported into NinjaTrader. It must be opened in Visual Studio.
- The default behavior of the project file uses the following path in the Start Action: C:\Program Files\NinjaTrader 8\bin\NinjaTrader.exe. If you have installed NinjaTrader in a different directory, you will need to adjust the file path accordingly.
- The solution targets .NET 4.8 with NinjaTrader Release R23 or higher, if you open it in a lesser .NET version Visual Studio will prompt you to download the required higher version SDK.

Creating Your Own AddOn Window

NinjaScript developers can utilize the AddOn framework to create free-standing, independent windows to provide custom functionality. Helper classes are available in the framework to instantiate windows styled the same as pre-built NinjaTrader windows, including familiar functionality such as window linking, the tabbed interface, and the ability to save the window and its state in workspaces. In addition, general WPF user interface elements and XAML can be used to style and modify windows using the .NET framework.

For a detailed walkthrough of creating your own window using NinjaScript helper classes, see the Creating Your Own AddOn Window page.

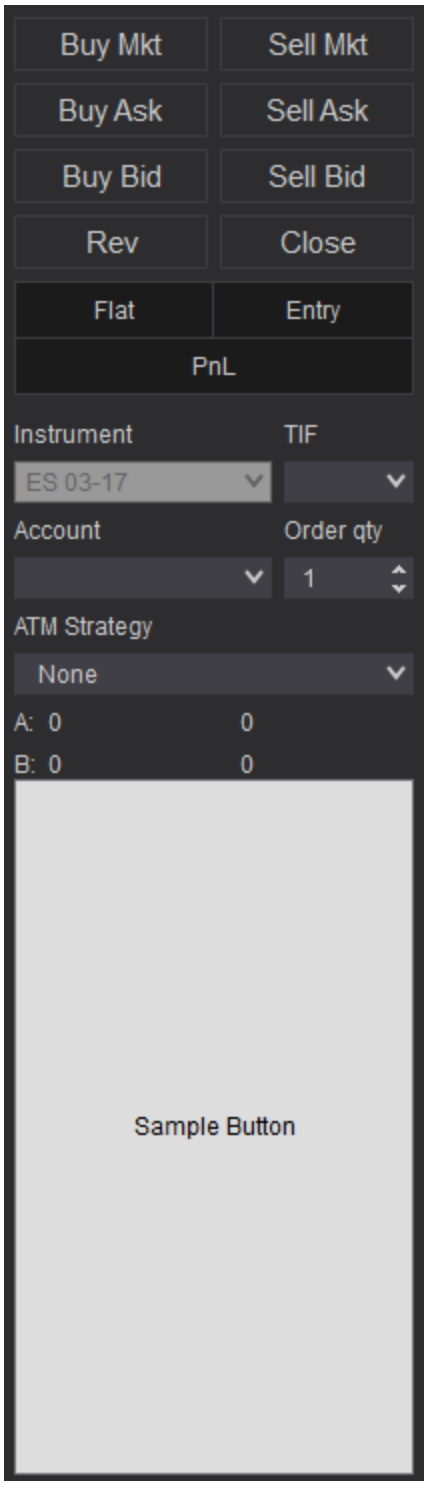


The image above shows a completely new window created by a custom AddOn.

Other Uses for an AddOn

An AddOn does not require its own window to function. It can instead be used to accomplish non-UI-driven functionality across the platform, such as monitoring market data or accessing account, position, and order information. AddOns can also be used to add functionality or interface elements to other NinjaTrader windows, such as charts.

For detailed information on other common uses of an AddOn, see the Other Uses for an AddOn page.



In the image above, the custom "Sample button" button has been drawn on a chart window using an AddOn.

11.5.1.1 Developing Add Ons

Add Ons Overview

Add Ons are incredibly powerful NinjaScript objects that let you create unprecedented tools which are seamlessly integrated (visually and functionally) into NinjaTrader. Experienced programmers can leverage the information available through the framework to create exciting new windows and utilities that can give users an incredible edge over the markets.

How to make Add Ons

The process to make an Add On is fairly simple once the structure is understood. A few questions should be answered to determine how to build your Add On:

1. Where should the entry point for the Add On be? E.g. Should it be launched from the Control Center menus? Should it be launched from a Chart?
2. Should the Add On leverage the tab functionality available in NinjaTrader?
3. Should the Add On leverage the window linking functionality available in NinjaTrader?
4. Should the Add On be persisted in NinjaTrader workspaces?

Once the functionality of your Add On is determined you can use the following building blocks to create your Add On:

AddOnBase	This is where you create the entry point for the Add On.
NTWindow	This is where you define the parent window container for your Add On. Tabs would reside within this parent window should you choose. This is also where workspace persistence would be created.
NTTabPage	This is where you define the content of each tab that resides inside NTWindow. This is also where you create the window linking functionality.
Class implementing the INTTabFactory interface	This is necessary to ensure proper tab functionality like adding, removing, moving tabs around in your NTWindow.

The general flow goes from AddOnBase > NTWindow > INTTabFactory > NTTabPage. AddOnBase determines the user entry point and then creates the event handler to create the NTWindow. NTWindow calls the tab factory which then brings in the NTTabPage content in the form of tabs into NTWindow.

11.5.1.2 Creating Your Own AddOn Window

The NTWindow Class

The NTWindow class allows you to quickly build windows using the same style and skin as other windows in NinjaTrader. An NTWindow does not contain user-interface functionality, but rather serves as a container for instances of NTTabPage, which will contain controls and functionality for the window.

```
/* This is where we define our AddOn window. The actual content is
contained inside the tabs of the window defined in a custom class
inheriting from NTTabPage.
    We must create a new window class which inherits from
Tools.NTWindow for styling and implements the IWorkspacePersistence
interface for the ability to save/restore from workspaces.*/
public class AddOnFrameworkWindow : NTWindow, IWorkspacePersistence
{
    public AddOnFrameworkWindow()
    {
        // set Caption property (not Title), since Title is managed
internally to properly combine selected Tab Header and Caption for
display in the windows taskbar
        // This is the name displayed in the top-left of the window
        Caption = "AddOn Framework";

        // Set the initial dimensions of the window
        Width   = 1085;
        Height  = 900;
    }
}
```

Using TabControl for Tab Functionality

After declaring an NTWindow, you can enable tab functionality on it (creating new tabs, copying tabs, etc.). The process for implementing tab functionality must be done within the constructor for your NTWindow, using the following process:

1. Instantiate a new TabControl object
2. Call helper methods of the TabControlManager class, passing in your TabControl object as an argument, to enable specific functionality
3. Use the same approach as #2 to set an NTTabFactory for your TabControl (see below for more information)
4. Set the Content property of your NTWindow to your TabControl

```
public class AddOnFrameworkWindow : NTWindow, IWorkspacePersistence
{
    public AddOnFrameworkWindow()
    {
        ...

        // TabControl should be created for window content if tab
features are wanted
        TabControl tc = new TabControl();

        // Attached properties defined in TabControlManager class
should be set to achieve tab moving, adding/removing tabs
        TabControlManager.SetIsMovable(tc, true);
        TabControlManager.SetCanAddTabs(tc, true);
        TabControlManager.SetCanRemoveTabs(tc, true);

        // if ability to add new tabs is desired, TabControl has to
have attached property "Factory" set.
        TabControlManager.SetFactory(tc, new
AddOnFrameworkWindowFactory());

        Content = tc;
    }
}
```

Note the instantiation of a new AddOnFrameworkWindowFactory in the example above. In this example, AddOnFrameworkWindowFactory is a custom class implementing the INTTabFactory interface. Within this class, the CreateParentWindow() and CreateTabPage() methods contained in INTTabFactory are hidden, as seen below:

```
/* Class which implements Tools.INTTabFactory must be created and
set as an attached property for TabControl
in order to use tab page add/remove/move/duplicate functionality */
public class AddOnFrameworkWindowFactory : INTTabFactory
{
    // INTTabFactory member. Required to create parent window
    public NTWindow CreateParentWindow()
    {
        return new AddOnFrameworkWindow();
    }

    // INTTabFactory member. Required to create tabs
    public NTTabPage CreateTabPage(string typeName, bool isTrue)
    {
        return new NinjaTraderAddOnProject.AddOnPage();
    }
}
```

Note: Take note of the instantiation of the AddOnPage class in the example above. In our example, AddOnPage is a XAML-defined class. Thus, when CreateTabPage() is called on an instance of AddOnFrameworkWindowFactory, it instantiates our XAML-defined user interface. See below for more information on defining user interfaces in XAML.

Creating an NTTabPage within an NTWindow

With an NTWindow defined and a TabControl set up, the next step is to instantiate an NTTabPage and add it to your TabControl. The first step is to define a class inheriting NTTabPage and implementing the IInstrumentProvider and IIntervalProvider interfaces to set up window-linking functionality.

```
/* This is where we define the actual content of the tabs for our
AddOn window.
    Note: Class derived from Tools.NTTabPage has to be created if
instrument link or interval link functionality is desired.
    Tools.IInstrumentProvider and/or Tools.IIntervalProvider
interface(s) should be implemented.
    Also NTTabPage provides additional functionality for properly
naming tab headers using properties and variables such as
@FUNCTION, @INSTRUMENT, etc. */
public class AddOnFrameworkTab : NTTabPage,
NinjaTrader.Gui.Tools.IInstrumentProvider,
NinjaTrader.Gui.Tools.IIntervalProvider
{
    public AddOnFrameworkTab()
    {
        AddOnFrameworkWindowFactory myAddOnFrameworkWindowFactory =
new AddOnFrameworkWindowFactory();

        Content =
myAddOnFrameworkWindowFactory.CreateTabPage("AddOnPage",true);
    }
}
```

With this class defined, the next step is to add it to your TabControl. You can do this via the AddNTTabPage() helper method contained in your TabControl object:

```
public class AddOnFrameworkWindow : NTWindow, IWorkspacePersistence
{
    public AddOnFrameworkWindow()
    {
        ...

        /* In order to have link buttons functionality, tab control
items must be derived from Tools.NTTabPage
        They can be added using extension method
AddNTTabPage(NTTabPage page) */
        tc.AddNTTabPage(new AddOnFrameworkTab());
    }
}
```

Setting Up Workspace Persistence

The last step in setting up the foundation for your custom window is to configure it to be saved and restored in NinjaTrader workspaces.

1. Hide the WorkspaceOptions property of the implemented IWorkspacePersistence interface
2. Use a delegate to set the WorkspaceOptions property to a new instance of the WorkspaceOptions class inside the NTWindow's constructor
3. Hide the Restore() method of IWorkspacePersistence to call the static RestoreFromXElement() method on the MainTabControl property
4. Hide the Save() method of IWorkspacePersistence to call the static SaveToXElement method in the same way

```
public class AddOnFrameworkWindow : NTWindow, IWorkspacePersistence
{
    public AddOnFrameworkWindow()
    {
        ...

        // WorkspaceOptions property must be set
        Loaded += (o, e) =>
        {
            if (WorkspaceOptions == null)
                WorkspaceOptions = new
WorkspaceOptions("AddOnFramework-" + Guid.NewGuid().ToString("N"),
this);
        };
    }

    // IWorkspacePersistence member. Required for restoring window
from workspace
    public void Restore(XDocument document, XElement element)
    {
        if (MainTabControl != null)
            MainTabControl.RestoreFromXElement(element);
    }

    // IWorkspacePersistence member. Required for saving window to
workspace
    public void Save(XDocument document, XElement element)
    {
        if (MainTabControl != null)
            MainTabControl.SaveToXElement(element);
    }

    // IWorkspacePersistence member
    public WorkspaceOptions WorkspaceOptions { get; set; }
}
```

Using XAML to Define Window Layout

There are two options available for laying out the user interface in your NTTabPage. The first is to use XAML, a markup language commonly used to define graphical interfaces in WPF applications. The process of pairing a XAML file with your C# classes is straightforward; simply create your XAML class in it's own file within your project, and it can be packaged together with your C# code in a DLL.

Example of creating a two-column grid in XAML

```
<Grid Background="Transparent">
    <!-- Define our layout with two columns. Rows can then be
assigned to columns -->
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="55*"/>
        <ColumnDefinition Width="45*"/>
    </Grid.ColumnDefinitions>
</Grid>
```

Using C# to Define Window Layout

You are not required to use XAML for window layout. You can code everything in C# if you choose. Defining user interface elements in C# is more verbose than XAML, but all of the same functionality is available. The example below shows the C# equivalent of the XAML code in the prior section.

Example of creating a two-column grid in C#

```
Grid grid = new Grid();
grid.Background = new SolidColorBrush(Colors.Transparent);

ColumnDefinition col1 = new ColumnDefinition();
col1.Width = newGridLength(55);

ColumnDefinition col2 = new ColumnDefinition();
col2.Width = newGridLength(45);

grid.ColumnDefinitions.Add(col1);
grid.ColumnDefinitions.Add(col2);
```

Launching Your Window From the Control Center

Once your window is set up and laid out, you will need a way to launch it from the Control Center. This can be done by adding a new item into one of the Control Center's menus (most commonly the **New** menu). This can be accomplished in four steps:

1. Obtain a reference to the Control Center menu in question
2. Instantiate an NTMenuItem

3. Add your NTMenuItem into the menu
4. Attach you NTMenuItem's Click event to a custom event handler
5. Use your custom event handler to launch your NTWindow

```
// Will be called as a new NTWindow is created. It will be called
in the thread of that window
protected override void OnWindowCreated(Window window)
{

    // We want to place our AddOn in the Control Center's menus
    ControlCenter cc = window as ControlCenter;
    if (cc == null)
        return;

    /* Determine we want to place our AddOn in the Control Center's
"New" menu
    Other menus can be accessed via the control's "Automation ID".
For example: toolsMenuItem, workspacesMenuItem,
connectionsMenuItem, helpMenuItem. */
    existingMenuItemInControlCenter =
cc.FindFirst("ControlCenterMenuItemNew") as NTMenuItem;
    if (existingMenuItemInControlCenter == null)
        return;

    // 'Header' sets the name of our AddOn seen in the menu
structure
    addOnFrameworkMenuItem = new NTMenuItem { Header = "AddOn
Framework", Style =
Application.Current.TryFindResource("MainMenuItem") as Style };

    // Add our AddOn into the "New" menu
    existingMenuItemInControlCenter.Items.Add(addOnFrameworkMenuIte
m);

    // Subscribe to the event for when the user presses our AddOn's
menu item
    addOnFrameworkMenuItem.Click += OnMenuItemClick;
}

// Open our AddOn's window when the menu item is clicked on
private void OnMenuItemClick(object sender, RoutedEventArgs e)
{
    Core.Globals.RandomDispatcher.BeginInvoke(new Action(() => new
AddOnFrameworkWindow().Show()));
}
```

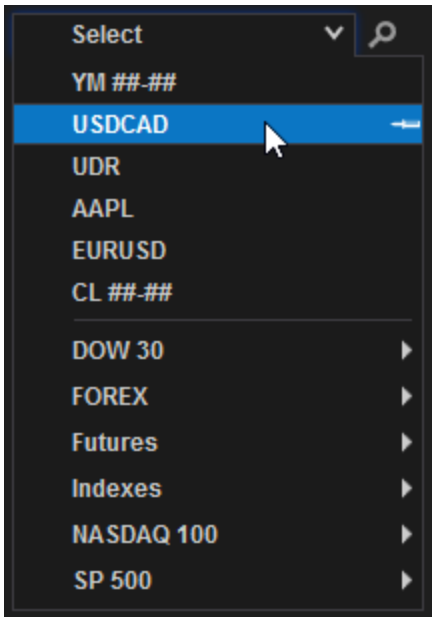
As always, it is important to unsubscribe from event handlers and dispose of unused resources when they are no longer needed. The OnWindowDestroyed() method can be used to clean up our work from the examples above:

```
// Will be called as a new NTWindow is destroyed. It will be called
in the thread of that window
protected override void OnWindowDestroyed(Window window)
{
    if (addOnFrameworkMenuItem != null && window is ControlCenter)
    {
        if (existingMenuItemInControlCenter != null &&
existingMenuItemInControlCenter.Items.Contains(addOnFrameworkMenuIt
em))
            existingMenuItemInControlCenter.Items.Remove(addOnFrame
workMenuItem);

        addOnFrameworkMenuItem.Click -= OnMenuItemClick;
        addOnFrameworkMenuItem = null;
    }
}
```

Adding NinjaTrader Custom Controls

User-interface controls, such as buttons, text fields, and dropdown menus can be defined via XAML (or C#), then behavior and functionality of those controls can be set via C# along with the core logic of your AddOn. In addition to the standard WPF controls, the NinjaScript AddOn framework provides access to each of the custom NinjaTrader controls that can be found throughout the platform. Below is a list of the most commonly used NinjaTrader controls, along with examples of defining these controls in XAML and adding functionality to them in C#:

1. The Instrument Selector



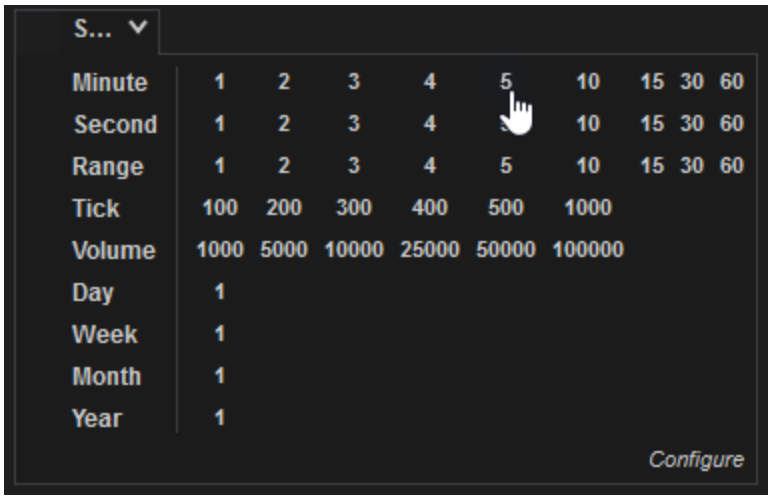
XAML - Instrument Selector Definition

```
<t:InstrumentSelector   x:Name="instrumentSelector" Grid.Row="6"
Grid.Column="0" LastUsedGroup="AddOnFramework"
InstrumentChanged="OnInstrumentChanged">
    <t:InstrumentSelector.Margin>
        <Thickness Left="{StaticResource MarginBase}"
Top="{StaticResource PaddingColumn}" Bottom="0"/>
    </t:InstrumentSelector.Margin>
</t:InstrumentSelector>
```

C# - Using the Instrument Selector

```
private InstrumentSelector instrumentSelector;

...

// Find instrument selector and attach event handler
instrumentSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
 "instrumentSelector") as InstrumentSelector;
if (instrumentSelector != null)
    instrumentSelector.InstrumentChanged += OnInstrumentChanged;
```

2. The Interval Selector



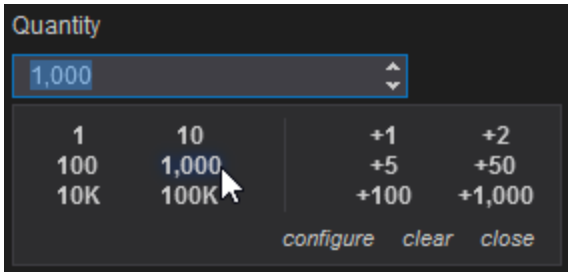
XAML - Interval Selector Definition

```
<t:IntervalSelector      x:Name="intervalSelector" Grid.Column="0"
HorizontalAlignment="Left" IntervalChanged="OnIntervalChanged">
    <t:IntervalSelector.Margin>
        <Thickness Left="{StaticResource MarginBase}"
Top="{StaticResource PaddingColumn}" Bottom="0"/>
    </t:IntervalSelector.Margin>
</t:IntervalSelector>
```

C# - Using the Interval Selector

```
private IntervalSelector intervalSelector;

...

// Find interval selector and attach event handler
intervalSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"intervalSelector") as IntervalSelector;
if (intervalSelector != null)
    intervalSelector.IntervalChanged += OnIntervalChanged;
```

3. The Quantity Up/Down Selector



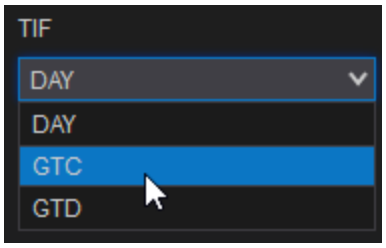
XAML - Quantity Up/Down Selector Definition

```
<t:QuantityUpDown x:Name="qudSelector"  Value="1" Grid.Row="12"
Grid.Column="0">
    <t:QuantityUpDown.Margin>
        <Thickness Left="{StaticResource MarginBase}"
Top="{StaticResource MarginControl}" Bottom="0" />
    </t:QuantityUpDown.Margin>
</t:QuantityUpDown>
```

C# - Using the Quantity Up/Down Selector

```
private QuantityUpDown qudSelector;

...

// Find Quanity selector
qudSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"qudSelector") as QuantityUpDown;
```

4. The Time-in-Force Selector



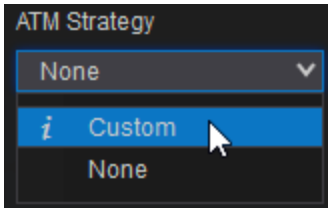
XAML - Time-in-Force Selector Definition

```
<t:TifSelector x:Name="tifSelector"     Grid.Row="12"
Grid.Column="1">
    <t:TifSelector.Margin>
        <Thickness Left="{StaticResource MarginButtonLeft}"
Top="{StaticResource MarginControl}" Right="0" Bottom="0" />
    </t:TifSelector.Margin>
</t:TifSelector>
```

C# - Using the Time-in-Force Selector

```
private TifSelector tifSelector;

...

// Find TIF selector
tifSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"tifSelector") as TifSelector;

// Be sure to bind our account selector to our TIF selector to
ensure proper functionality
tifSelector.SetBinding(TifSelector.AccountProperty, new Binding
{ Source = accountSelector, Path = new
PropertyPath("SelectedAccount") });

// When our TIF selector's selection changes
tifSelector.SelectionChanged += (o, args) =>
{
    // Change the selected TIF in the ATM strategy too
    if (atmStrategySelector.SelectedAtmStrategy != null)
    {
        atmStrategySelector.SelectedAtmStrategy.TimeInForce =
tifSelector.SelectedTif;
    }
};
```

5. The ATM Strategy Selector



XAML - ATM Strategy Selector Definition

```
<AtmStrategy:AtmStrategySelector x:Name="atmStrategySelector"
LinkedQuantity="{Binding ElementName=qudSelector, Path=Value,
Mode=OneWay}" Grid.Row="12" Grid.Column="2">
    <AtmStrategy:AtmStrategySelector.Margin>
        <Thickness Left="{StaticResource MarginButtonLeft}"
Top="{StaticResource MarginControl}" Right="{StaticResource
MarginBase}" Bottom="0" />
    </AtmStrategy:AtmStrategySelector.Margin>
</AtmStrategy:AtmStrategySelector>
```

C# - Using the ATM Strategy Selector

```
private AtmStrategy.AtmStrategySelector atmStrategySelector;

...

// Find ATM Strategy selector and attach event handler
atmStrategySelector =
LogicalTreeHelper.FindLogicalNode(pageContent,
"atmStrategySelector") as AtmStrategy.AtmStrategySelector;
atmStrategySelector.Id = Guid.NewGuid().ToString("N");
if (atmStrategySelector != null)
    atmStrategySelector.CustomPropertiesChanged +=
OnAtmCustomPropertiesChanged;

// Be sure to bind our account selector to our ATM strategy
selector to ensure proper functionality
atmStrategySelector.SetBinding(AtmStrategy.AtmStrategySelector.Acco
untProperty, new Binding { Source = accountSelector, Path = new
PropertyPath("SelectedAccount") });

// When our ATM selector's selection changes
atmStrategySelector.SelectionChanged += (o, args) =>
{
    if (atmStrategySelector.SelectedItem == null)
        return;
    if (args.AddedItems.Count > 0)
    {
        // Change the selected TIF in our TIF selector too
        NinjaTrader.NinjaScript.AtmStrategy selectedAtmStrategy =
args.AddedItems[0] as NinjaTrader.NinjaScript.AtmStrategy;
        if (selectedAtmStrategy != null)
        {
            tifSelector.SelectedTif =
selectedAtmStrategy.TimeInForce;
        }
    }
};
```

Linking with Other Windows

If you utilize NinjaTrader controls to allow selection of instruments or intervals, you can add instrument or interval linking functionality to your window. The PropagateInstrumentChange() and PropagateIntervalChange() methods can be used to accomplish this. To call PropagateIntervalChange(), use the process below:

1. Hide the Instrument property of the IInstrumentProvider interface, which your NTTabPage inheriting class should be implementing

2. Call PropagateInstrumentChange() within the setter for the hidden Instrument property

```
// IInstrumentProvider member. Required if you want to use the
instrument link mechanism on this window.
public Cbi.Instrument Instrument
{
    get { return instrument; }
    set
    {
        // Send instrument to other windows linked to the same
color
        PropagateInstrumentChange(value);
    }
}
```

In a real-world scenario, you would most likely use an instrument selector to call the setter for the Instrument property. Thus, when a user toggled the instrument selector, PropagateInstrumentChange() would be called in addition to any other logic you put in place. In the same way, you can use an interval selector to push changes to the Interval Linking feature. In this case, you can attach a custom event handler to an interval selector's IntervalChanged event, then call PropagateIntervalChange() within that event handler:

```
...

// Find an interval selector that we've added to our UI, and attach
a custom event handler
intervalSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"intervalSelector") as IntervalSelector;
if (intervalSelector != null)
    intervalSelector.IntervalChanged += OnIntervalChanged;

...

// This method is fired when our interval selector changes
intervals
private void OnIntervalChanged(object sender, BarsPeriodEventArgs
args)
{
    if (args.BarsPeriod == null)
        return;

    PropagateIntervalChange(args.BarsPeriod);
}
```

11.5.1.3 Other Uses for an Addon

Modifying Existing NinjaTrader Windows

To modify an existing type of NinjaTrader window (for example, to add a button to all charts), you will first need to obtain a reference to each individual window of that type that is open. This can be done by overriding the OnWindowCreated() method, then declaring an object of the Type of the window you are looking for, and finally assigning the object a reference to the Window passed into the method:

```
// OnWindowCreated() will be called any time a new NTWindow is
created. It will be called in the thread of that window
protected override void OnWindowCreated(Window window)
{
    // Declare a Chart object and instantiate it to the Window
passed into the method
    Gui.Chart.Chart myChart = window as Gui.Chart.Chart;

    // Use this check to return if the calling Window is not of the
Type you are looking for
    if (myChart == null)
        return;
}
```

If you are unsure of the Type name for a particular type of window, you can open an instance of that window then run the code below, which will print the Type to the Output Window:

```
protected override void OnWindowCreated(Window window)
{
    // Print the Type of any open windows, for future reference
    Print(window.ToString());
}
```

Once you've obtained a reference to a window, you can then directly manipulate the WPF grids, controls, and other elements to customize its user interface or functionality. For example, if your goal was to add a new button to Chart Trader on all charts, you could use your reference to Chart objects to first locate their attached Chart Trader instances, then place a custom-defined button directly into the WPF grid used to lay out buttons in Chart Trader. Since this code would run within OnWindowCreated(), it would be applied to every Chart Trader instance that is open. You would not be changing the format used to create Chart Traders in the first place, but would rather be detecting every open instance and adding the buttons into them. This is an important distinction to make, because this approach requires that you also remove the elements you've added when each window is destroyed.

```
// Declare a Chart, ChartTrader, and UI elements to add to Chart
Trader
Gui.Chart.Chart myChart;
Gui.Chart.ChartTrader chartTrader;
Button sampleButton;
Grid myGrid;
Grid mainGrid;

protected override void OnWindowCreated(Window window)
{
    // Instantiate myChart by assigning a reference to the calling
Window
    myChart = window as Gui.Chart.Chart;

    if (myChart == null)
    {
        return;
    }

    //find chart trader from myChart's Chart Control by its
Automation ID: "ChartWindowChartTrader"
    chartTrader =
Window.GetWindow(myChart.ActiveChartControl.Parent).FindFirst("Char
tWindowChartTraderControl") as Gui.Chart.ChartTrader;

    if (chartTrader == null)
    {
        return;
    }

    // Instantiate sampleButton
    sampleButton = new Button
    {
        Content = "Sample Button",
        Style =
System.Windows.Application.Current.TryFindResource("Button") as
Style
    };

    // Attach a custom event handler to the .Click event
    sampleButton.Click += SampleButton_Click;

    // Set a custom AutomationId for the button, so that it can be
referenced elsewhere the same way we found Chart Trader
    System.Windows.Automation.AutomationProperties.SetAutomationId(
sampleButton, "SampleButton");

    //this is the main chart trader grid where the default buttons
and controls reside
    mainGrid = chartTrader.FindName("grdMain") as Grid;

    // Return if Chart Trader is null
    if (mainGrid == null)
```

Since we are dynamically adding elements to open windows, it is important to clean up any unused resources and detach any event handlers when the affected windows are destroyed. You can use the same approach as shown above to obtain a reference to each affected window within the OnWindowDestroyed() method:

```
protected override void OnWindowDestroyed(Window window)
{
    // Return if there is no button, or if the destroyed window is
not a chart
    if(sampleButton == null || !(window is Gui.Chart.Chart))
    {
        return;
    }

    // Detach the event handler from the .Click event, remove the
grid, and nullify the button
    sampleButton.Click -= SampleButton_Click;
    mainGrid.Children.Remove(myGrid);
    sampleButton = null;
}
```

Below is another example of adding elements into chart windows. In this example, we add a new panel to the top of all chart windows, then take all existing chart content and move it into a row beneath the panel we've just added:

```
protected override void OnWindowCreated(Window window)
{
    // Obtain a reference to any chart that triggered
OnWindowCreated
    Chart Window = window as Chart;

    // Instantiate a grid to hold a reference to the content of the
chart window
    Grid mainWindowGrid = Window.Content as Grid;

    // Add existing row definition for existing row if it is not
present
    if (mainWindowGrid.RowDefinitions.Count == 0)
    {
        mainWindowGrid.RowDefinitions.Add(new RowDefinition());
    }

    // Instantiate a RowDefinition and set its height
    RowDefinition row = new RowDefinition();
    row.Height = new GridLength(PanelLength);

    // Insert the new row into the chart's main window grid
    mainWindowGrid.RowDefinitions.Insert(0, row);

    //Move Existing Elements down one row, since our new content
will take the top row
    foreach (UIElement element in mainWindowGrid.Children)
    {
        element.SetValue(Grid.RowProperty, (int)
element.GetValue(Grid.RowProperty) +  1);
    }

    //Create the Top Panel grid and add it to our newly defined row
    Grid Panel = new Grid();
    Panel.SetValue(Grid.RowProperty, 0);
    mainWindowGrid.Children.Add(Panel);

    //Create a sample text block and add it to the Top/Bottom Panel
Grid.
    TextBlock TextBlock = new TextBlock();
    TextBlock.Text = PanelDirection.ToString() + " Panel (" +
PanelLocation.ToString() + ") Sample Text Block";
    TextBlock.Foreground = Brushes.Red;
    TextBlock.SetValue(Grid.RowProperty, 0);
    Panel.Children.Add(TextBlock);
}
```

Accessing Account Data

From time to time, you may need to access certain global data, such as account values, order states, position info, etc. In these cases, you can subscribe to an appropriate event using a custom event handler method. Below is a list of a few such events which can be captured:

<Account>.AccountItemUpdate	Triggers on account item updates
<Account>.ExecutionUpdate	Triggers on any execution
<Account>.OrderUpdate	Triggers on any order state changes
<Account>.PositionUpdate	Triggers on any position updates

```
// Custom Subscribe() method to refresh subscriptions
private void Subscribe()
{
    if (myAccount != null)
    {
        // Unsubscribe to any prior account subscriptions
        myAccount.AccountItemUpdate -= OnAccountItemUpdate;
        myAccount.ExecutionUpdate -= OnExecutionUpdate;
        myAccount.OrderUpdate -= OnOrderUpdate;
        myAccount.PositionUpdate -= OnPositionUpdate;

        // Subscribe to new account subscriptions
        myAccount.AccountItemUpdate   += OnAccountItemUpdate;
        myAccount.ExecutionUpdate     += OnExecutionUpdate;
        myAccount.OrderUpdate         += OnOrderUpdate;
        myAccount.PositionUpdate      += OnPositionUpdate;
    }
}

private void OnAccountItemUpdate(object sender,
AccountItemEventArgs e)
{
    // Handle account item updates
}

private void OnExecutionUpdate(object sender, AccountItemEventArgs
e)
{
    // Handle execution updates
}

private void OnOrderUpdate(object sender, AccountItemEventArgs e)
{
    // Handle order updates
}

private void OnPositionUpdate(object sender, AccountItemEventArgs
e)
{
    // Handle position updates
}
```

Accessing Market Data

Market data can be accessed via a BarsRequest object, which can provide real-time or snapshot data for use by your classes. A BarsRequest object can be loaded with a series of bar data without the need to actually draw bars on a chart. The BarsRequest object can then

be accessed via the BarsUpdateEventArgs object passed into your event handler via the BarsRequest's Update method. The process for using a BarsRequest is as follows:

1. Instantiate an Instrument object
2. Instantiate and parameterize a BarsRequest object
3. Hook the BarsRequest's Update event to a custom event handler
4. Call the BarsRequest's Request() method
5. Access bars data directly from the BarsRequest object within your event handler method

```
// Custom method to perform a BarsRequest
private NinjaTrader.Data.BarsRequest DoBarsRequest(Instrument
instrument, int lookBackPeriod)
{
    // Declare a BarsRequest object
    NinjaTrader.Data.BarsRequest barsRequest;

    // Request x number of days back of data.
    barsRequest = new NinjaTrader.Data.BarsRequest(instrument,
DateTime.Now.AddDays(-lookBackPeriod), DateTime.Now);

    // If you wish to request x number of bars back instead you can
use this signature:
    // barsRequest = new NinjaTrader.Data.BarsRequest(instrument,
lookBackPeriod);


    // Parameterize the request
    barsRequest.BarsPeriod  = new NinjaTrader.Data.BarsPeriod
{ BarsPeriodType = BarsPeriodType.Minute, Value = 60 };
    barsRequest.TradingHours     =
NinjaTrader.Data.TradingHours.Get("Default 24 x 7");

    // Additional parameters which could be set
    // barsRequest.IsDividendAdjusted      = true;
    // barsRequest.IsResetOnNewTradingDay   = false;
    // barsRequest.IsSplitAdjusted         = true;
    // barsRequest.LookupPolicy            =
LookupPolicies.Provider;
    // barsRequest.MergePolicy            = MergePolicy.DoNotMerge;

    // Attach event handler for real-time events if you want to
process real-time data
    barsRequest.Update      += MyOnBarUpdate;

    // Call the Request method on the BarsRequest object to request
the bars
    barsRequest.Request(new Action<NinjaTrader.Data.BarsRequest,
ErrorCode, string>((bars, errorCode, errorMessage) =>
    {
        Dispatcher.InvokeAsync(new Action(() =>
        {
            if (errorCode != ErrorCode.NoError)
            {
                // Handle any errors in requesting bars here
                outputBox.Text = string.Format("Error on requesting
bars: {0}, {1}", errorCode, errorMessage);
                return;
            }
        }));
    }));

    // Return the Bars Request to any callers of this method
```

11.5.2 C# Method (Functions) Reference

Native Methods

The Microsoft .NET environment has a rich class library that you can access when developing custom indicators and strategies. There is a plethora of information available online and in print that details class libraries in great depth. Below are quick links to the Microsoft Developers Network for some of the basic classes whose functionality you may harness when developing in NinjaScript.

Complete list of classes in the Microsoft .NET environment.

MSDN (Microsoft Developers Network) C# Language Reference
Keywords
Operators
Arrays

System.Math

Provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions.
Full list of member of the System.Math class.

```
// Example of the Max method of the System.Math class
int myInteger = Math.Max(10, 20);
Print("The larger value between 10 and 20 is " +
myInteger.ToString());
```

System.DateTime

Represents an instant in time, typically expressed as a data and time of day.
Full list of members of the Sytem.DateTime structure.

```
// Example of the Now property member of the System.DateTime
structure
DateTime startTime = DateTime.Now;
Print("Time elapsed is " +
DateTime.Now.Subtract(startTime).TotalMilliseconds.ToString() + "
milliseconds.");
```

System.String

Represents text; that is, a series of unicode characters.
Full list of members of the System.String class.

```
// Example of the ToUpper() method of the System.String class
string myString = "ninjatrader";
Print("The following word is in uppercase " +
myString.ToUpper()););
```

11.5.3 Developing for Tick Replay

Tick Replay is used to playback 1 tick historical data to build the bars as if they had been build live, this means that tick data will be thrown as Market Data events in historical and subsequently OnMarketData and OnBarUpdate events will be called as if it was live. This provides more granular tick related information and can be helpful if you need to know the most recent last price, last volume, best ask price, or best bid price that occurred on historical data during the bar. An indicator or strategy running Tick Replay needs to have been specifically designed to take advantage of Tick Replay. In general, this means adding additional logic to the OnMarketData() event handler, however, Tick Replay can also be used to call OnBarUpdate() "OnEachTick" or "OnPriceChange" during historical calculations.

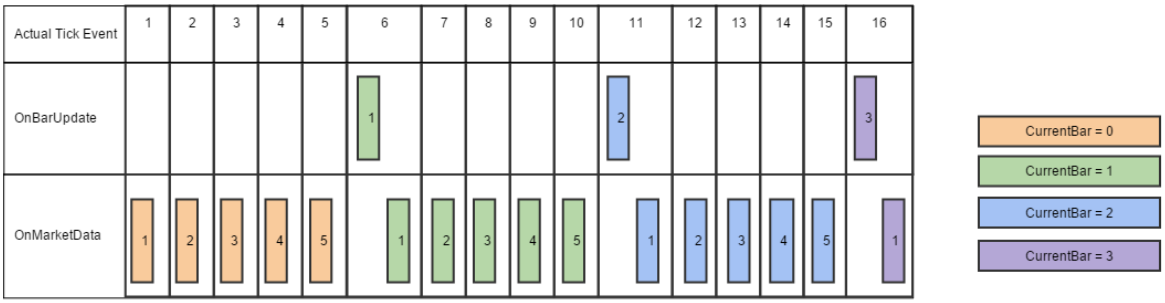
How to Enable Tick Replay

To enable tick replay it **must** be manually enabled on the primary Data Series and the option to allow this mode is hidden by default. The option to allow for Tick Replay is located in **Tools** > **Options** > **Market Data** > "**Show Tick Replay**". The reason why it is hidden by default is that the tick replay engine utilizes 1 tick data to build historical bars. TickReplay can generate thousands of events per bar and may take an excessive amount of time to load. It is recommended to optimize your indicators that you plan to calculate on such data by only running them in Calculate On Bar Close mode or reducing the amount of data to load to the minimum amount of data required. Since bars are built with tick data you will only be able to build bars back as far as your historical data provider allows download of tick data.

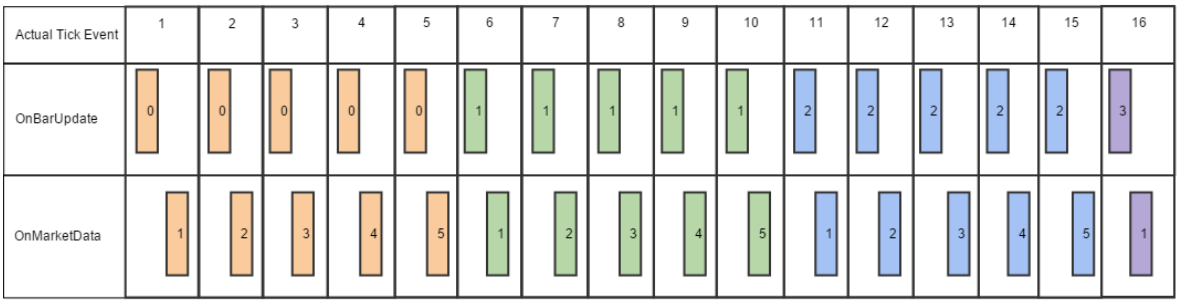
How the Tick Replay Engine Works

Tick Replay guarantees an exact sequence of stored events are played back for both the OnBarUpdate and OnMarketData events. This mode also ensures the **OnMarketData** event is called after every **OnBarUpdate** event used to build the current bar. Consider the following examples with Tick Replay enabled on a 5-tick input series, each box is when each event occurs during Tick Replay simulation.

Sequence of Events using Calculate.OnBarClose for a 5 Tick Bar



Sequence of Events using Calculate.OnEachTick for a 5 Tick Bar in Realtime or Tick Replay



As you can see from the table above, the **Calculate** setting will have a varying degree of impact on how your indicator or strategies **OnBarUpdate** event is raised. This process repeats for every historical bar on the chart and would continue as the indicator or strategy transitions to real-time data.

Accessing the current best bid and ask at the time of a trade

NinjaTrader stores the best bid price and best ask price as the last trade occurs during the MarketDataType.Last event and provides it per the table below:

`marketDataUpdate.Price`	The current market data price of the last trade event
`marketDataUpdate.Ask`	The current asking price at the time of the last trade event
`marketDataUpdate.Bid`	The current bidding price at the time of the last trade event
`marketDataUpdate.Volume`	The current market data volume of the last trade event

marketDataUpdate.Time	The current time of the last trade event

An example below shows how to access historical Bid and Ask prices with Tick Replay

Accessing the current best bid and ask at the time of a trade

```
protected override void OnMarketData(MarketDataEventArgs
marketDataUpdate)
{
   // TickReplay events only occur on the "Last" market data type
   if (marketDataUpdate.MarketDataType == MarketDataType.Last)
   {
      if (marketDataUpdate.Price >= marketDataUpdate.Ask)
      {
         Print(marketDataUpdate.Volume + " contracts traded at
asking price " + marketDataUpdate.Ask);
      }

      else if (marketDataUpdate.Price <= marketDataUpdate.Bid)
      {
         Print(marketDataUpdate.Volume + " Contracts Traded at
bidding price " + marketDataUpdate.Bid);
      }
   }
}
```

Calling a Tick Replay indicator from another Indicator or Strategy

A hosting indicator or strategy must be aware of the requirement to run through another indicator's historical Tick Replay data before it reaches State.Historical. To achieve desired results, you either need to store the reference in **State.DataLoaded** or (for a strategy) you can call AddChartIndicator(). Either approach ensures that the hosting indicator or strategy is aware of the requirements to process Tick Replay during its **State.Historical** mode and helps to ensure that the hosted indicator calculates as designed up to its current bar using Tick Replay. Please see the example below.

Calling a Tick Replay indicator from another Indicator or Strategy

```
TickReplayIndicator myTickReplayIndicator = null;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        Name    = "TestHost";
    }
    else if (State == State.DataLoaded)
    {
        // Store a reference to the Tick Replay indicator before
State.Historical
        // Doing so ensures the hosted indicator will run through
Tick Replay
        myTickReplayIndicator = TickReplayIndicator();

        // For a strategy, you can just call
AddChartIndicator(TickReplayIndicator());
        // However this also adds a copy of the indicator to the
chart, which may or may not be desired
        // For calculation purposes only, storing the reference
should all that needs to be required.
    }
}

protected override void OnBarUpdate()
{
    // Access the stored reference which calculates through
    // historical Tick Replay data and print the value as expected
    Print(myTickReplayIndicator[0]);
}
```

Notes:

1. Tick Replay was **NOT** designed to provide accuracy in backtesting concerning order fills and execution and should **NOT** be used to expect the exact sequence of executions as running a strategy on live data. For greater order-fill resolution and accuracy in strategy backtesting, you can use the High Fill Resolution in the Strategy Analyzer. Furthermore you cannot combine both Tick Replay and High Order Fill resolution.

2. If the data provided has no bid/ask data tied to the last tick data, NinjaTrader substitutes the bid/ask data for consistent user experience purposes (i.e., Bid = Last price, Ask = Bid + 1 tick). For a list of providers who support tick replay, please see the table from Understanding the data provided by your connectivity provider. Only bid and ask price is made available, bid and ask volume is **NOT** available.
3. Tick Replay **ONLY** replays the Last market data event, and only stores the best inside bid/ask price at the time of the last trade event. You can think of this as the equivalent of the bid/ask price at the time a trade was reported. As such, historical bid/ask market data events (i..e, bid/ask volume) **DO NOT** work with Tick Replay. To obtain those values, you need to use a historical bid/ask series separately from Tick Replay through OnBarUpdate()
4. Tick Replay data is accessed via the MarketDataEventArgs object passed into OnMarketData() events, rather than attempting to access it via GetCurrentAsk() and GetCurrentBid(), which are methods designed to function on real-time data only
5. Due to the nature of how some unique bars build, Tick Replay is **NOT** available for all bar types. For example, the default **Renko** and **LineBreak** bars which use RemoveLastBar() are not compatible with Tick Replay. Other custom bar types which use similar methods encounter the same limitation
6. Tick Replay is forced for all series loaded, and there is **NOT** any method to reduce the number of calculations on a per series basis. In other words, you cannot mix and match tick replay series with non-tick replay series
7. Tick Replay was only **ONLY** designed to work with **MarketDataType.Last**. A TickReplay indicator or strategy should **NOT** be mixed with a **MarketDataType.Ask** or **MarketDataType.Bid** series
8. Tick Replay is not compatible with most Multi-Time Frame / Multi Instrument indicators, as there could be series synchronization issues leading to unexpected results.

11.5.4 Developing Indicators

Indicators are the building blocks of any automated trading system. NinjaScript allows you to develop custom indicators quickly. A few key points are:

- Custom indicators are compiled and run natively within the NinjaTrader application, providing the highest performance possible
- Indicator values are calculated at the current bar, which ensures that you do not accidentally include future data in your calculations
- You can retain calculations between bar updates

- You can retain and share calculation values between bar updates and across indicators

Custom indicator development follows a logical progression.

Wizard
The wizard allows you to define your overall indicator parameters which include name, properties, inputs, plots and oscillator lines. The wizard will then generate the necessary NinjaScript code and open up the NinjaScript Editor.

OnStateChange() Method
The OnStateChange() method is called once before any initial calculation triggered by an update bar event. This method is used to configure the indicators plots, lines and properties. The wizard will generate the required NinjaScript code for this method for most cases.

OnBarUpdate() Method
The OnBarUpdate() method is called with either with each incoming tick or on the close of each bar, depending on how you deploy the indicator at run time. Your core indicator calculation logic is contained within this method.

Debug
The NinjaScript Editor will perform both syntax and semantic checks and list any errors at the bottom of the window. If there are logic problems with your indicator, they will be listed in the Log tab of the NinjaScript Control Center during run time. You can use the Print() method within your script to help debug your code. Output will be sent to the **NinjaScript Output** window.

Compilation
Once the coding effort is completed, you must then compile the indicator (several second process) directly from the NinjaScript Editor.

Usage
The completed indicator is now available through any window that can use an indicator, such as a Chart.

Tutorial Descriptions
All internal NinjaTrader indicators come with full source code and can be viewed within the NinjaScript Editor. Please review the tutorials within this section for detailed walk throughs of custom indicator development.

> Level 1 - Demonstrating the use of price variables
> Level 2 - Demonstrating the use of indicator on indicator

- Level 3 - Demonstrating the use of a "for" loop to build a simple moving average indicator
- Level 4 - Demonstrating the use of Indicator Series objects to retain historical custom calculations data series
- Level 5 - Demonstrating the use of custom plot coloring based on threshold values
- Level 6 - Demonstrating the use of custom of drawing using bar color, back color and line colors

11.5.4.1 Advanced - Custom Drawing

Custom Drawing Overview

In this advanced tutorial, we are going to build a custom indicator which is a variation on the CCI, to show different drawing options for bar color, line color, and background color.

- Set Up
- Entering Calculation Logic
- Compiling
- Using

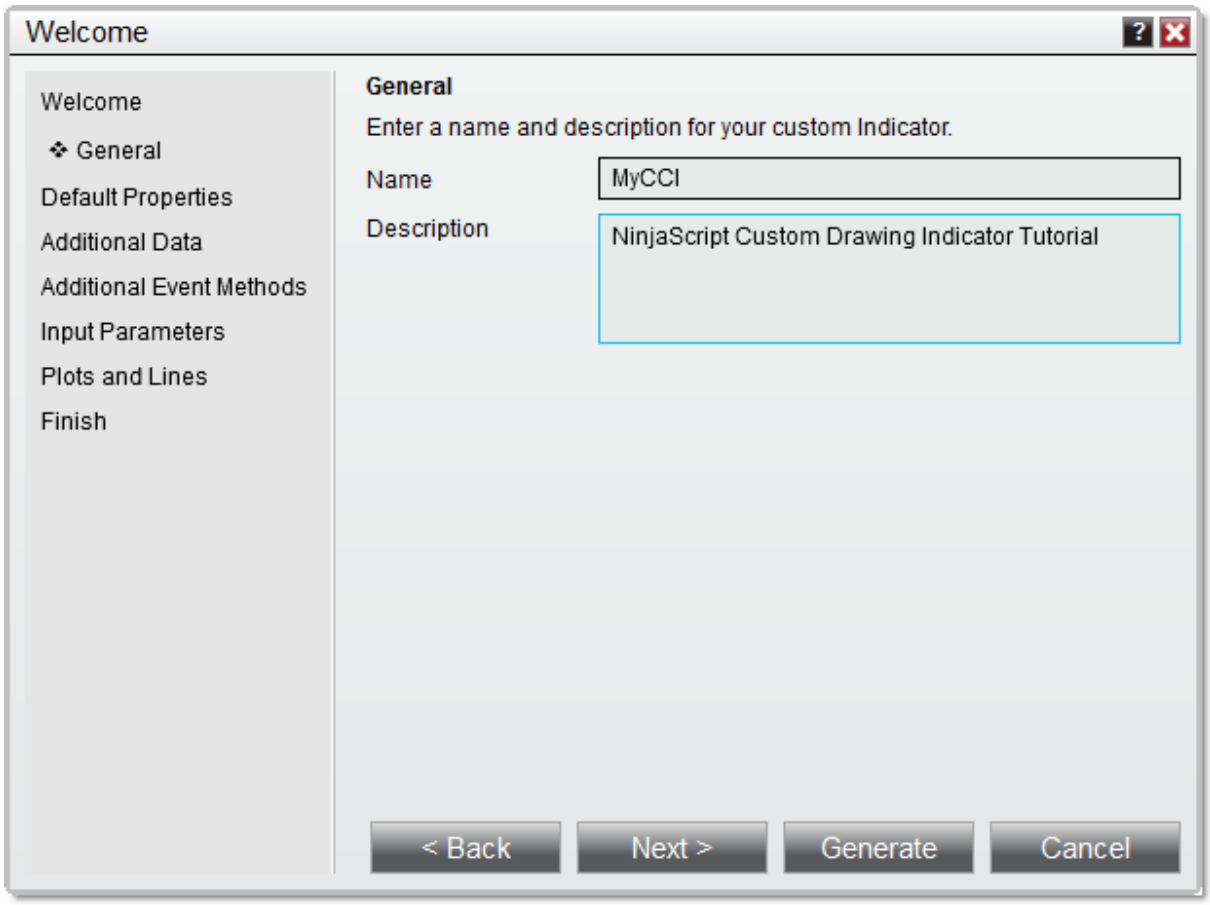
11.5.4.1.1 Set Up

The first step in creating a custom indicator is to use the custom indicator wizard. The wizard will generate the required NinjaScript code that will serve as the foundation for your custom indicator.

1. Within the NinjaTrader Control Center, select the **New** menu, then select the **NinjaScript Editor** menu item.
2. Right mouse click the "Indicators" folder in the **NinjaScript Explorer** section, then select the **New Indicator** menu item to open the **New Indicator Wizard**.

Defining Indicator Properties and Name

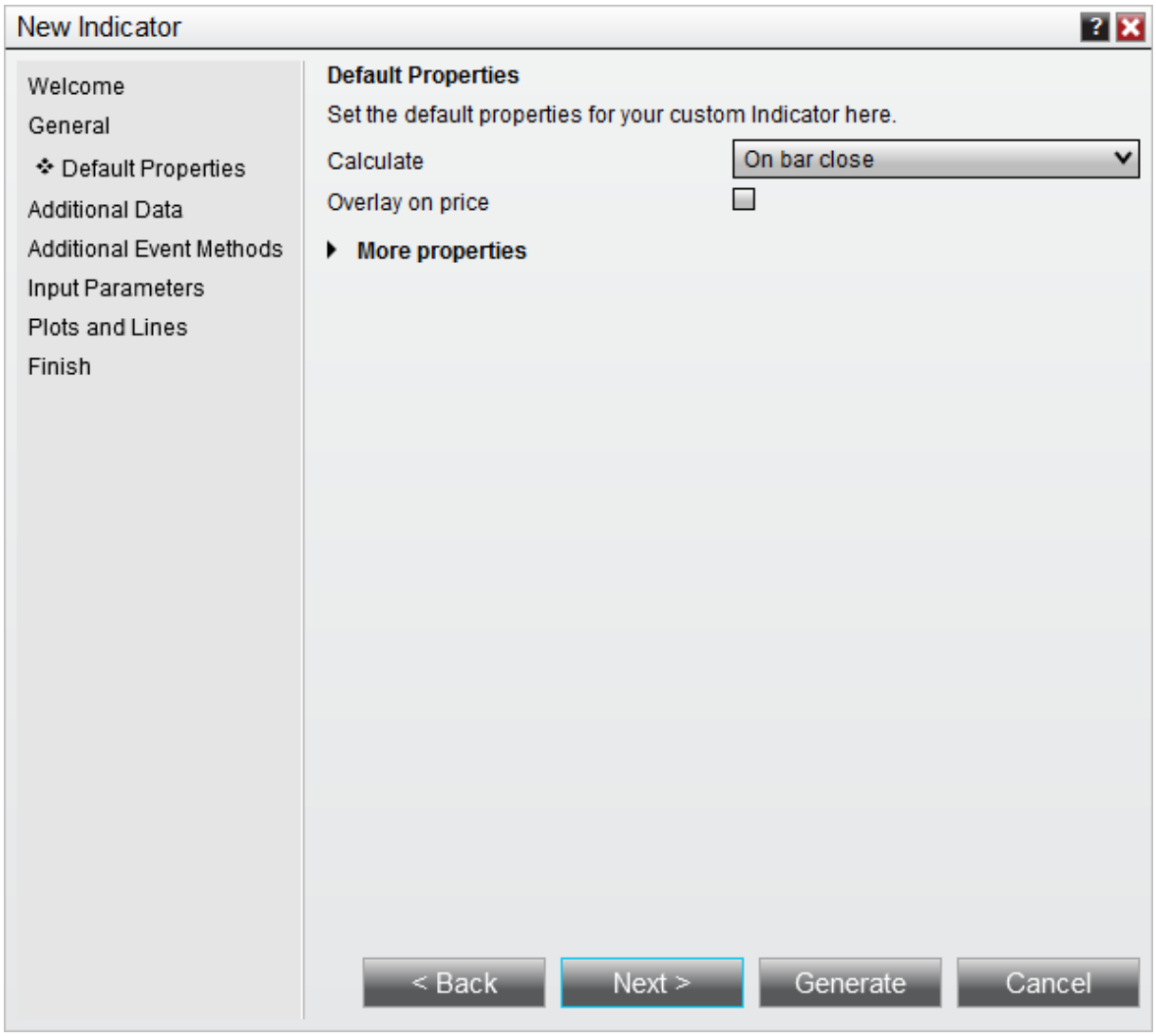
First you will define your indicator's name and several indicator properties. Begin by clicking the **Next >** button on the first page of the wizard to view the page shown below.



3. Enter the information as shown above
4. Click the **Next >** button

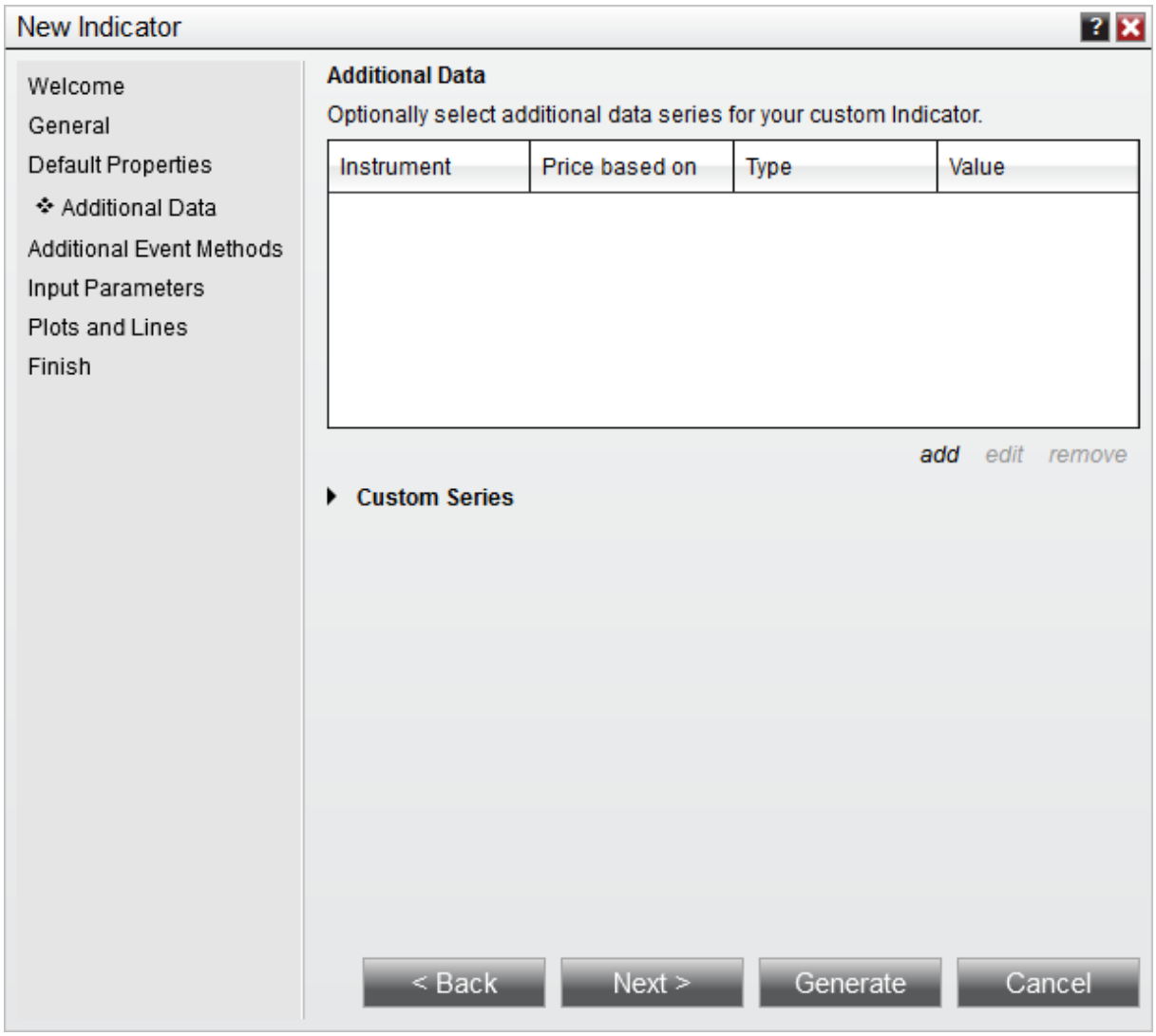
Setting Default Properties

The next page will allow you to set defaults for basic properties related to your indicator, including it's Calculate and Overlay settings. Click the **More Properties** button to expose additional properties. For this tutorial, we will not change any basic properties' defaults, and instead will leave them all set to the values shown below:



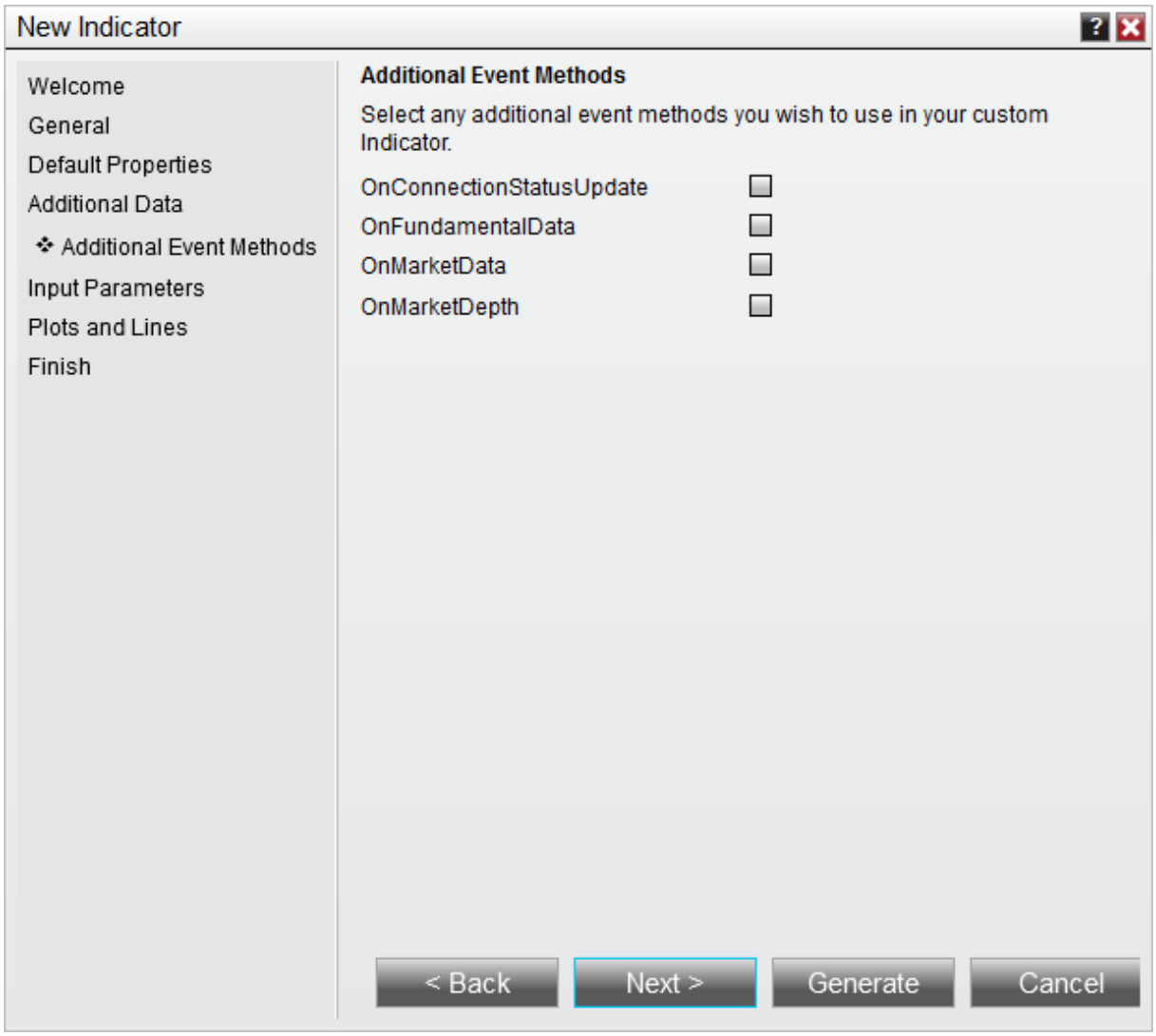
Adding Additional Data

The next page will allow you to configure one or more additional Bars objects for use by the indicator. For our purposes, we will leave this page blank and move forward by clicking the **Next >** button.



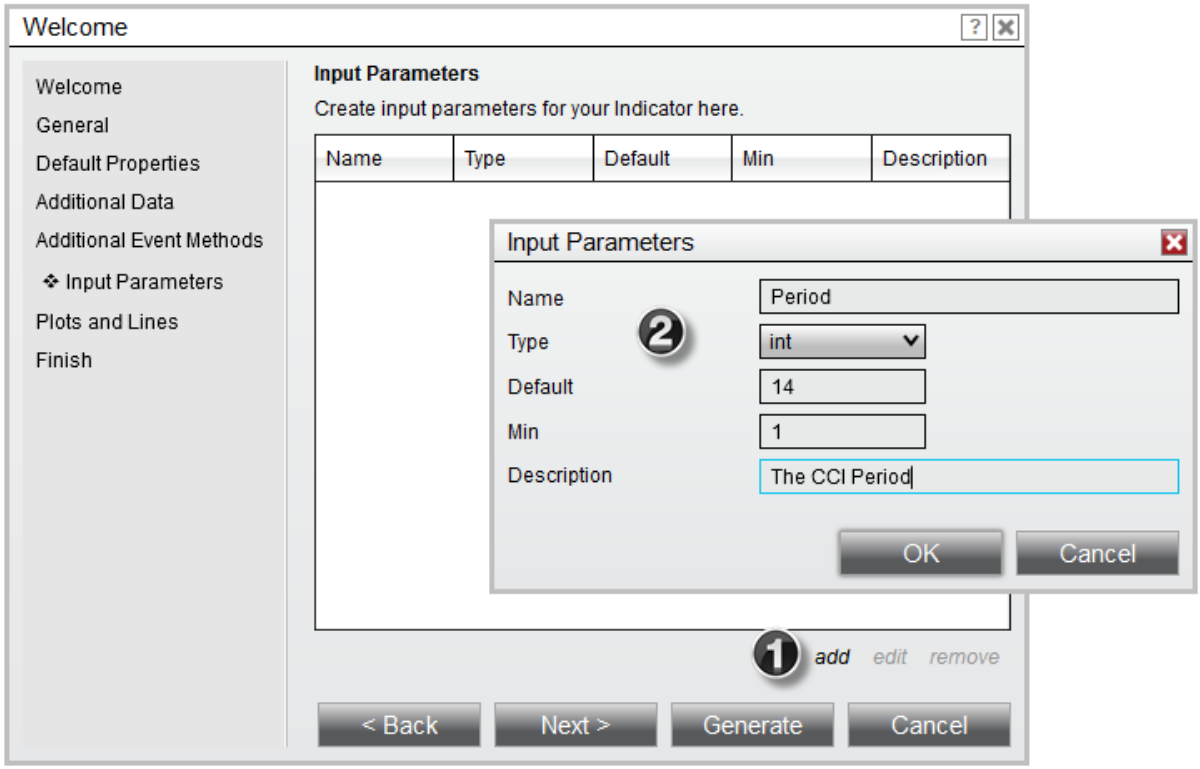
Adding Event Methods

The next page will allow you to pre-populate certain event methods into the NinjaScript code generated by the wizard. For our purposes, we will leave all of the checkboxes corresponding to different event methods unchecked, and will move on by clicking the **Next >** button.



Defining Input Parameters

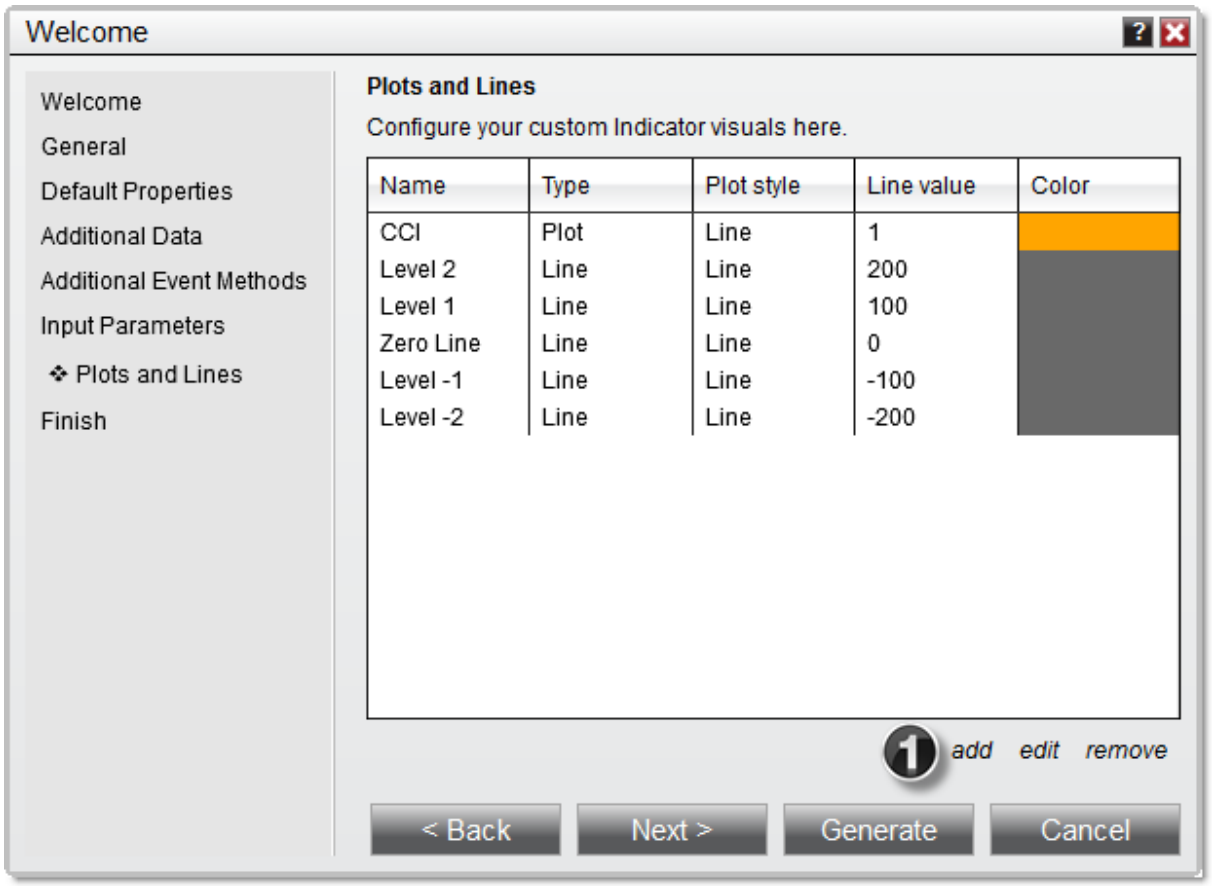
The next page will allow us to configure user input parameters for the indicator. For our custom CCI indicator, we will create a single input parameter which can be changed by users in the Indicators window when applying or editing the indicator. This input parameter will determine the CCI's period. We will select `int` as the Type, since integers are the most efficient native data types to be used for positive whole numbers, like those used to specify a number of bars to look back (a period). We will enter a "Default" value of "14" for the period, and a "Min" value of 1, to ensure that users do not enter zero or lower.



1. Clicking the **add** button on the "Input Parameters" page brings up the **Input Parameters** dialogue
2. The **Input Parameters** dialogue can be used to define user inputs

Defining Plots and lines

The next page will allow us to define plots and static lines for the indicator. For the CCI, we will define a single plot, called "CCI," and define five lines to draw in the indicator panel. For each item, first click the add button, then use the Plots and Lines dialogue to configure each item as seen below.



1. The add button will allow you to configure plots and lines for the indicator.

After this, click the **Finish** button, and the **Indicator Wizard** will generate a basic code structure implementing the parameters that you have set. You are now ready to move on to entering calculation logic in your code.

11.5.4.1.2 Entering Calculation Logic

The OnBarUpdate() method is called for each incoming tick, or on the close of a bar (if enabled) when performing real-time calculations, and is called on each bar of a Bars object when re-calculating the indicator (For example, an indicator would be re-calculated when adding it to an existing chart that has existing price data displayed). This is the main method called for indicator calculation, and we will calculate the CCI value and set the conditions used to draw the CCI plot within this method.

The OnStateChange() method is called once before any bar data is loaded, and is used to configure the indicator (among other things).

Initializing the Indicator

The code below is automatically generated by the wizard and added to the `OnStateChange()`

method, within `State.SetDefaults`. It configures the indicator for one plot and five lines, and sets the parameters entered in the wizard:

```
AddPlot(Brushes.Orange, "MyPlot");
AddLine(Brushes.DimGray, 200, "Level 2");
AddLine(Brushes.DimGray, 100, "Level 1");
AddLine(Brushes.DimGray, 0, "Zero Line");
AddLine(Brushes.DimGray, -100, "Level -1");
AddLine(Brushes.DimGray, -200, "Level -2");
```

To change the visual properties of the Zero Line, replace the fourth line in the code above with the line below. This will change the color to black and the line style to "dash:"

```
AddLine(new Stroke(Brushes.Black, DashStyleHelper.Dash, 2), 0,
"Zero Line");
```

The code above uses an alternative method overload (an alternative set of arguments passed in to the `AddLine()` method), in order to pass in a Stroke object rather than a Brush. With a `Stroke`, not only can we still specify a `Brush`, but we have additional options to change the dash style (via `DashStyleHelper`) and the line width. After this change, your configured lines and plots should look like this:

```
AddPlot(Brushes.Orange, "MyCCI_Plot");
AddLine(Brushes.DimGray, 200, "Level 2");
AddLine(Brushes.DimGray, 100, "Level 1");
AddLine(new Stroke(Brushes.Black, DashStyleHelper.Dash, 2), 0,
"Zero Line");
AddLine(Brushes.DimGray, -100, "Level -1");
AddLine(Brushes.DimGray, -200, "Level -2");
```

Adding Core Indicator Logic

Since this tutorial is meant to cover custom drawing and manually changing properties within an indicator, we will not go too in-depth into the core calculation logic for this custom CCI. Instead, we will copy and paste the core calculation logic from the @CCI indicator already built-in to NinjaTrader.

The @CCI indicator uses an SMA object in its calculations. To add this, copy the line below from @CCI into your custom CCI, directly below the class declaration:

```
private SMA sma;
```

Next, copy the following initialization for the SMA object into the `OnStateChange()` method, within `State.DataLoaded`:

```
sma  = SMA(Typical, Period);
```

Next, copy the core calculation logic from @CCI into the `OnBarUpdate()` method of your custom indicator:

```
if (CurrentBar == 0)
    Value[0] = 0;
else
{
    double mean = 0;
    double sma0 = sma[0];

    for (int idx = Math.Min(CurrentBar, Period - 1); idx >= 0;
idx--)
        mean += Math.Abs(Typical[idx] - sma0);

    Value[0] = (Typical[0] - sma0) / (mean.ApproxCompare(0) == 0 ?
1 : (0.015 * (mean / Math.Min(Period, CurrentBar + 1))));
}
```

The code for your MyCCI class should now look as follows (in addition to the `using` statements and class declaration) :

```
public class MyCCI : Indicator
{
    private SMA sma;

    protected override void OnStateChange()
    {
        if (State == State.SetDefaults)
        {
            Description                 = @"NinjaScript Custom
Drawing Indicator Tutorial";
            Name                        = "MyCCI";
            Calculate                   = Calculate.OnBarClose;
            IsOverlay                   = false;
            DisplayInDataBox            = true;
            DrawOnPricePanel            = true;
            DrawHorizontalGridLines     = true;
            DrawVerticalGridLines       = true;
            PaintPriceMarkers           = true;
            ScaleJustification          =
NinjaTrader.Gui.Chart.ScaleJustification.Right;
            //Disable this property if your indicator requires
custom values that cumulate with each new market data event.
            //See Help Guide for additional information.
            IsSuspendedWhileInactive    = true;
            Period                      = 14;
            AddPlot(Brushes.Orange, "MyPlot");
            AddLine(Brushes.DimGray, 200, "Level 2");
            AddLine(Brushes.DimGray, 100, "Level 1");
            AddLine(new Stroke(Brushes.Black,
DashStyleHelper.Dash,2), 0, "Zero Line");
            AddLine(Brushes.DimGray, -100, "Level -1");
            AddLine(Brushes.DimGray, -200, "Level -2");

        }
        else if (State == State.DataLoaded)
        {
            sma = SMA(Typical, Period);
        }
    }

    protected override void OnBarUpdate()
    {
        if (CurrentBar == 0)
            Value[0] = 0;
        else
        {
            double mean = 0;
            double sma0 = sma[0];

            for (int idx = Math.Min(CurrentBar, Period - 1); idx >=
 0; idx--)
                mean += Math.Abs(Typical[idx] - sma0);
```

Custom Drawing

Add the following code into the `OnBarUpdate()` method, directly beneath the core calculation logic:

```
// if the plot value is greater than 100, paint the plot green at
that bar index
if (Value[0] > 100)
    PlotBrushes[0][0] = Brushes.Green;

// if the plot value is less than -100, paint the plot red at that
bar index
if (Value[0] < -100)
    PlotBrushes[0][0] = Brushes.Red;

// if the plot value is between 100 and -100, paint the plot orange
at that bar index
if (Value[0] >= -100 && Value[0] <= 100)
    PlotBrushes[0][0] = Brushes.Orange;
```

This will conditionally change the color of the CCI plot (referenced by `Values[0]`) based on its value. By using `PlotBrushes[0][0]`, we are specifying that we wish to change the color of the first plot in the collection at a specific bar index (the current bar index each time the condition is triggered), and we wish for the plot the remain that color at that index, even if the plot value changes in the future. If instead we wished to change the entire plot color, we could use `Plots[0].Brush`.

`PlotBrushes` holds a collection of brushes used for the various plots in the indicator. In addition to this, there are several other collections that serve similar purposes, which can be used in the same way. Some examples of these collections are below:

BackBrushes	A collection of Brushes used for chart background color at specific bar indexes
BarBrushes	A collection of Brushes used to paint bars at specific indexes
CandleOutlineBrushes	A collection of Brushes used to paint candle outlines at specific indexes

Now that everything is in place, your class code should look as below. You are now ready to compile the indicator and configure it on a chart.

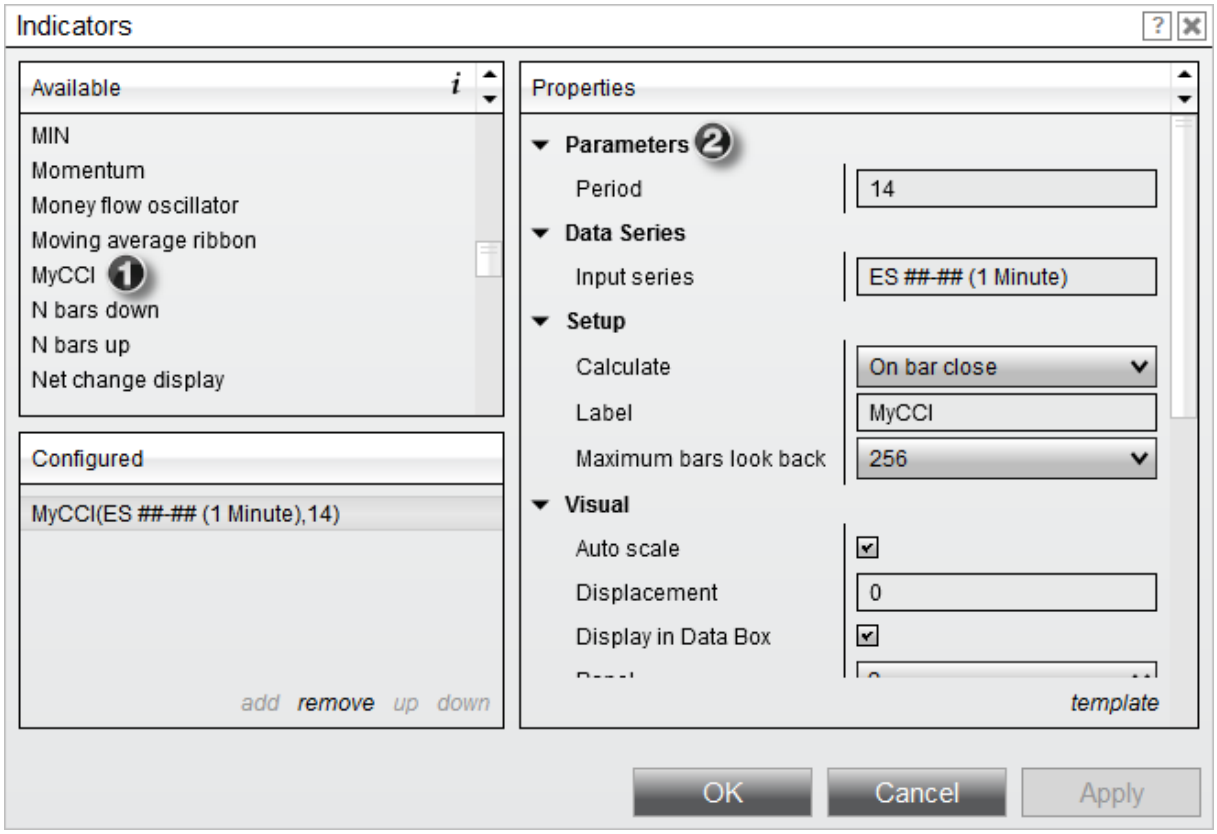
```
public class MyCCI : Indicator
{
   private SMA sma;

   protected override void OnStateChange()
   {
      if (State == State.SetDefaults)
      {
         Description                              = @"NinjaScript
Custom Drawing Indicator Tutorial";
         Name                                     = "MyCCI";
         Calculate                                =
Calculate.OnBarClose;
         IsOverlay                                = false;
         DisplayInDataBox                         = true;
         DrawOnPricePanel                         = true;
         DrawHorizontalGridLines                  = true;
         DrawVerticalGridLines                    = true;
         PaintPriceMarkers                        = true;
         ScaleJustification                       =
NinjaTrader.Gui.Chart.ScaleJustification.Right;
         //Disable this property if your indicator requires custom
values that cumulate with each new market data event.
         //See Help Guide for additional information.
         IsSuspendedWhileInactive                 = true;
         Period                                   = 14;
         AddPlot(Brushes.Orange, "MyPlot");
         AddLine(Brushes.DimGray, 200, "Level 2");
         AddLine(Brushes.DimGray, 100, "Level 1");
         AddLine(new Stroke(Brushes.Black, DashStyleHelper.Dash,
2), 0, "Zero Line");
         AddLine(Brushes.DimGray, -100, "Level -1");
         AddLine(Brushes.DimGray, -200, "Level -2");
      }
      else if (State == State.DataLoaded)
      {
         sma = SMA(Typical, Period);
      }
   }

   protected override void OnBarUpdate()
   {
      if (CurrentBar == 0)
          Value[0] = 0;
      else
      {
          double mean = 0;
          double sma0 = sma[0];

          for (int idx = Math.Min(CurrentBar, Period - 1); idx >=
0; idx--)
              mean += Math.Abs(Typical[idx] - sma0);
```

11.5.4.1.3 Compiling

The indicator code is now complete and needs to be compiled. You can compile this indicator by selecting the **Compile** menu item from within the NinjaScript Editor Right Click, by clicking the Compile icon on the toolbar at the top of the window, or by pressing the F5 key on your keyboard. It is important to understand that this process makes the indicator ready for real-time use natively within NinjaTrader. It does not run as interpreted code, as many other applications do, but rather as a C# assembly. This provides you with the highest performance possible. If there are any errors reported during compiling, the error messages will be displayed at the bottom of the NinjaScript Editor.

11.5.4.1.4 Using

Your indicator is now ready for use and will be listed in the Indicator Dialog window.



1) The indicator can now be found in the "Available" section of the **Indicators** window
2) Once added to the "Configured" section, our user-defined inputs can be entered along with standard indicator properties.

Once applied to a chart, the indicator should look something like the image below.

11.5.4.2 Advanced - Custom Plot Colors via Thresholds

Custom Plot Colors via Thresholds Overview

In this advanced level tutorial we are going to build a custom indicator which is a ROC variation and paints one color above the zero band and another below. This indicator will show you how to use the concept of plot thresholds.

- Set Up
- Entering Calculation Logic
- Compiling
- Using

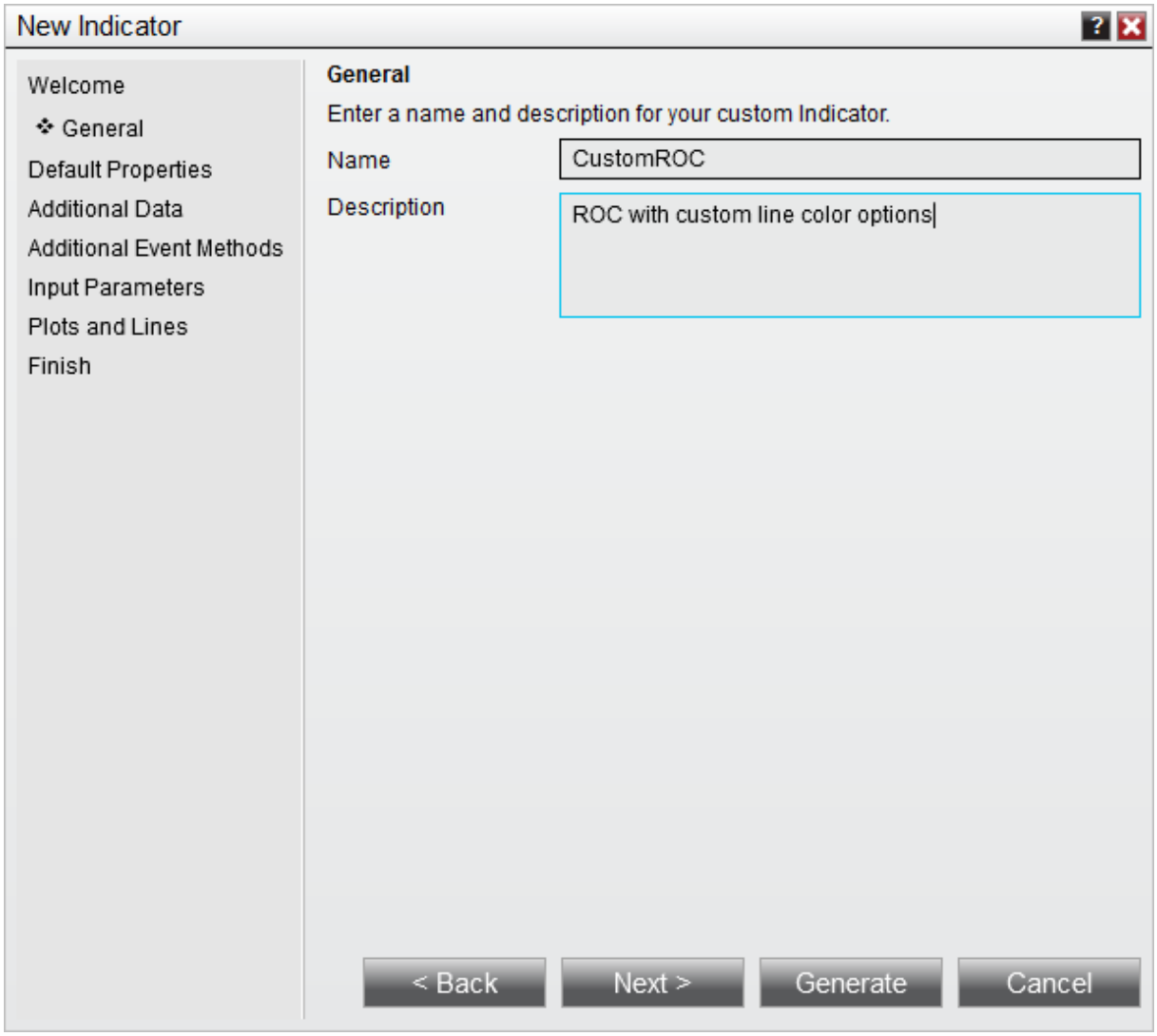
11.5.4.2.1 Set Up

The first step in creating a custom indicator is to use the custom indicator wizard. The wizard will generate the required NinjaScript code that will serve as the foundation for your custom indicator.

1. Within the NinjaTrader Control Center, select the **New** menu, then select the **NinjaScript Editor** menu item.
2. Right mouse click the "Indicators" folder in the **NinjaScript Explorer** section, then select the **New Indicator** menu item to open the **New Indicator Wizard**.

Defining Indicator Properties and Name

First you will define your indicator's name and several indicator properties. Begin by clicking the **Next >** button on the first page of the wizard to view the page shown below.

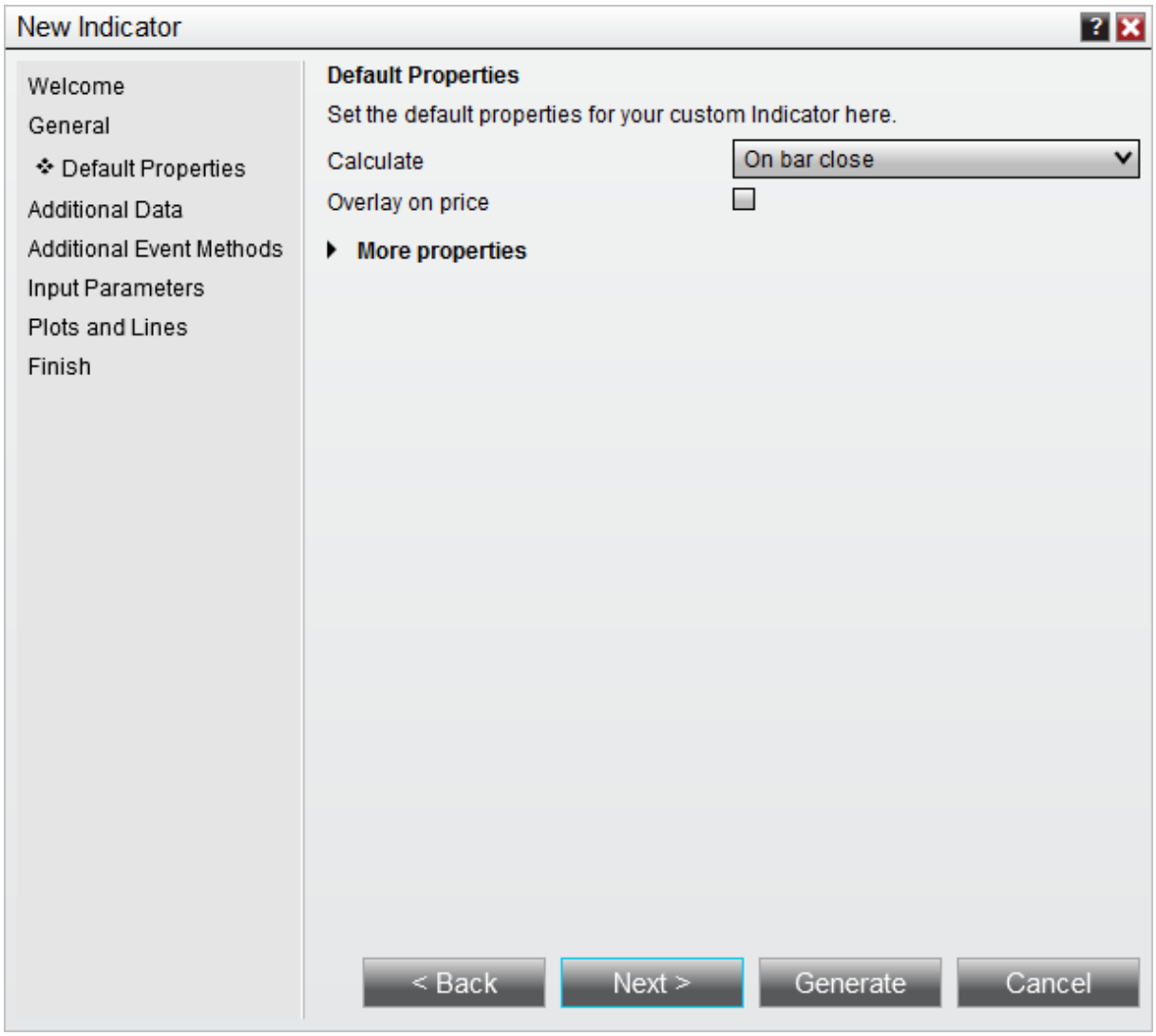


3. Enter the information as shown above
4. Click the **Next >** button

Setting Default Properties

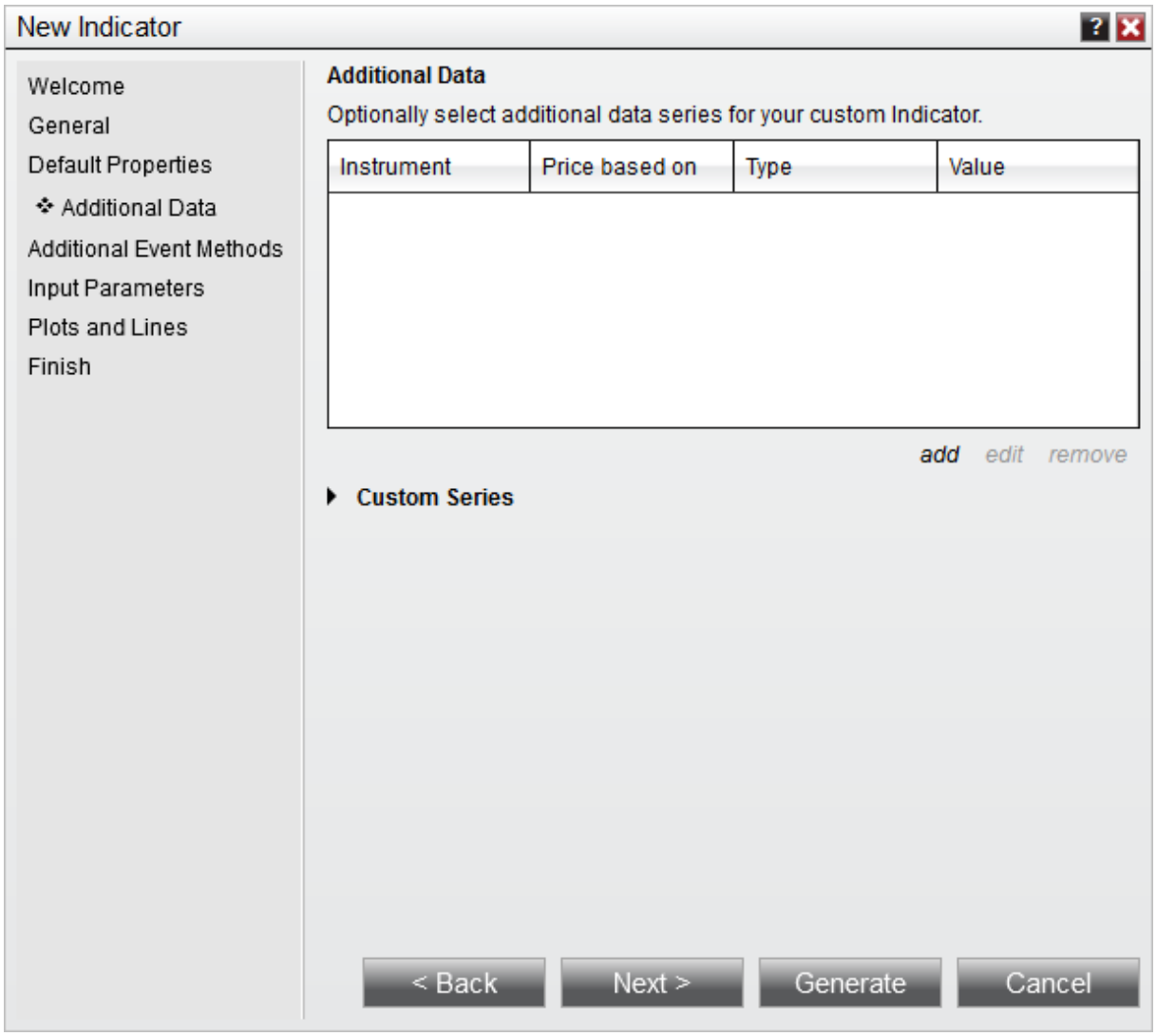
The next page will allow you to set defaults for basic properties related to your indicator, including it's Calculate and Overlay settings. Click the **More Properties** button to expose

additional properties. For this tutorial, we will not change any basic properties' defaults, and instead will leave them all set to the values shown below and move forward by clicking the **Next >** button.:



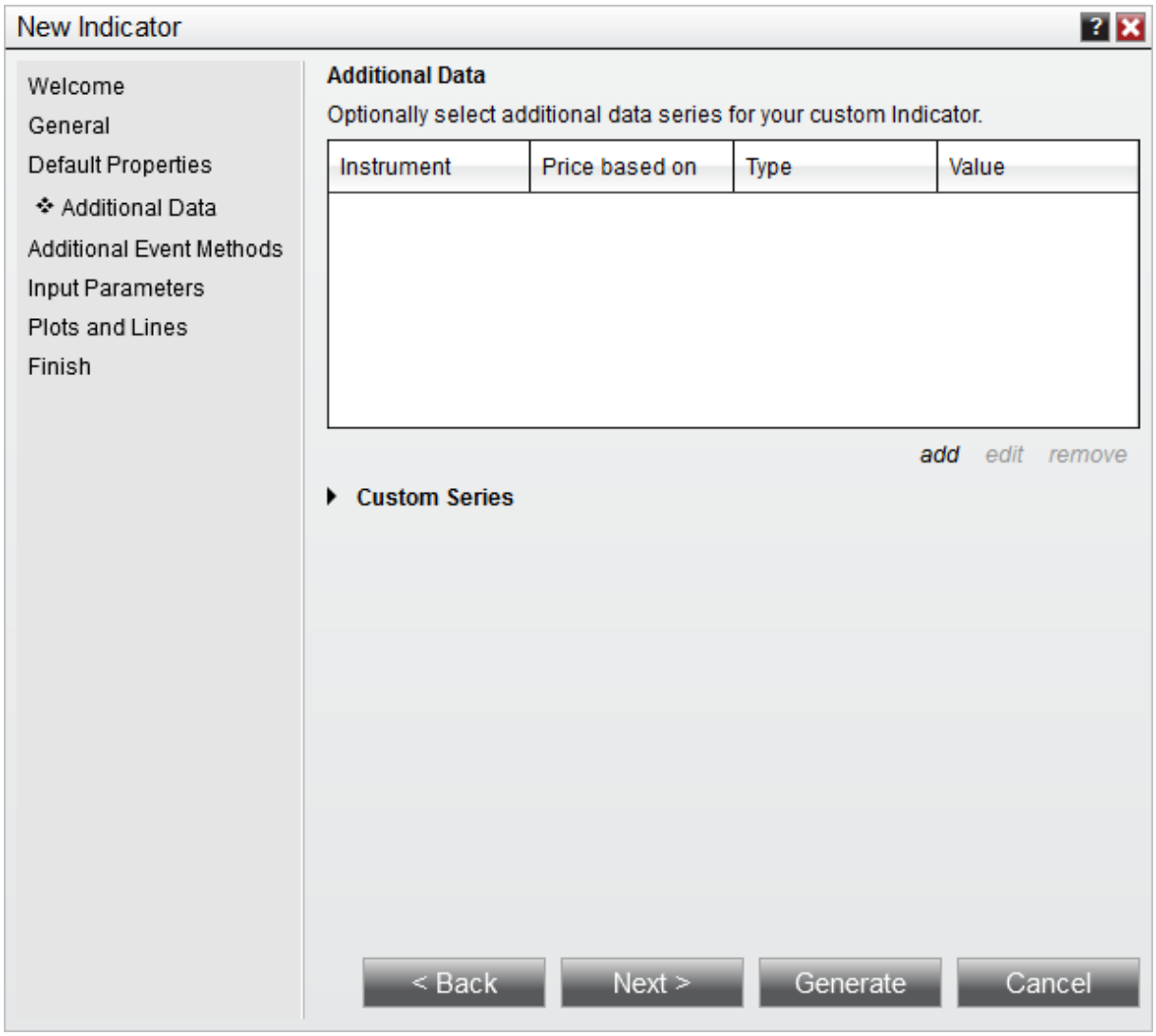
Adding Additional Data

The next page will allow you to configure one or more additional Bars objects for use by the indicator. For our purposes, we will leave this page blank and move forward by clicking the **Next >** button.



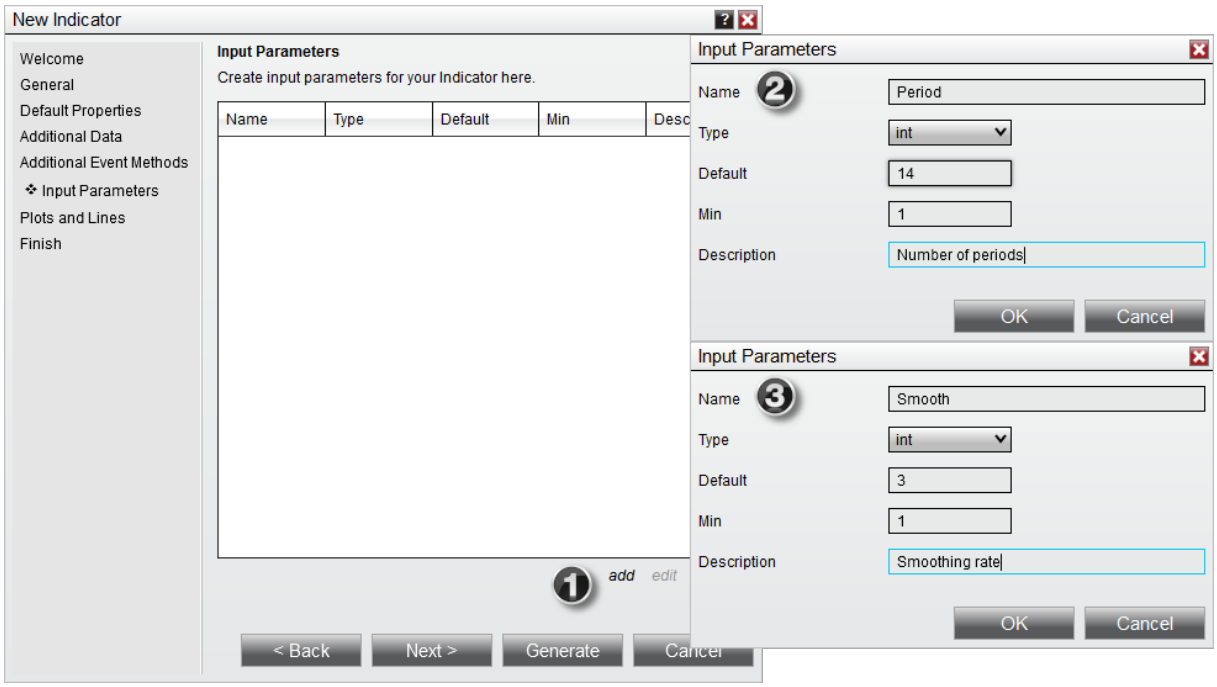
Adding Event Methods

The next page will allow you to pre-populate certain event methods into the NinjaScript code generated by the wizard. For our purposes, we will leave all of the checkboxes corresponding to different event methods unchecked, and will move on by clicking the **Next >** button.



Defining Input Parameters

The next page will allow us to configure user input parameters for the indicator. For our custom indicator, our eventual goal will be to create a simple plot that follows either above or below the bars based upon the Close price of a specified bar compared to the preceding bar. To allow for the variable selection of a number of bars ago, we will create one input parameter and call it "Periods." This variable will then be used to determine the number of bars used in the plot calculation.

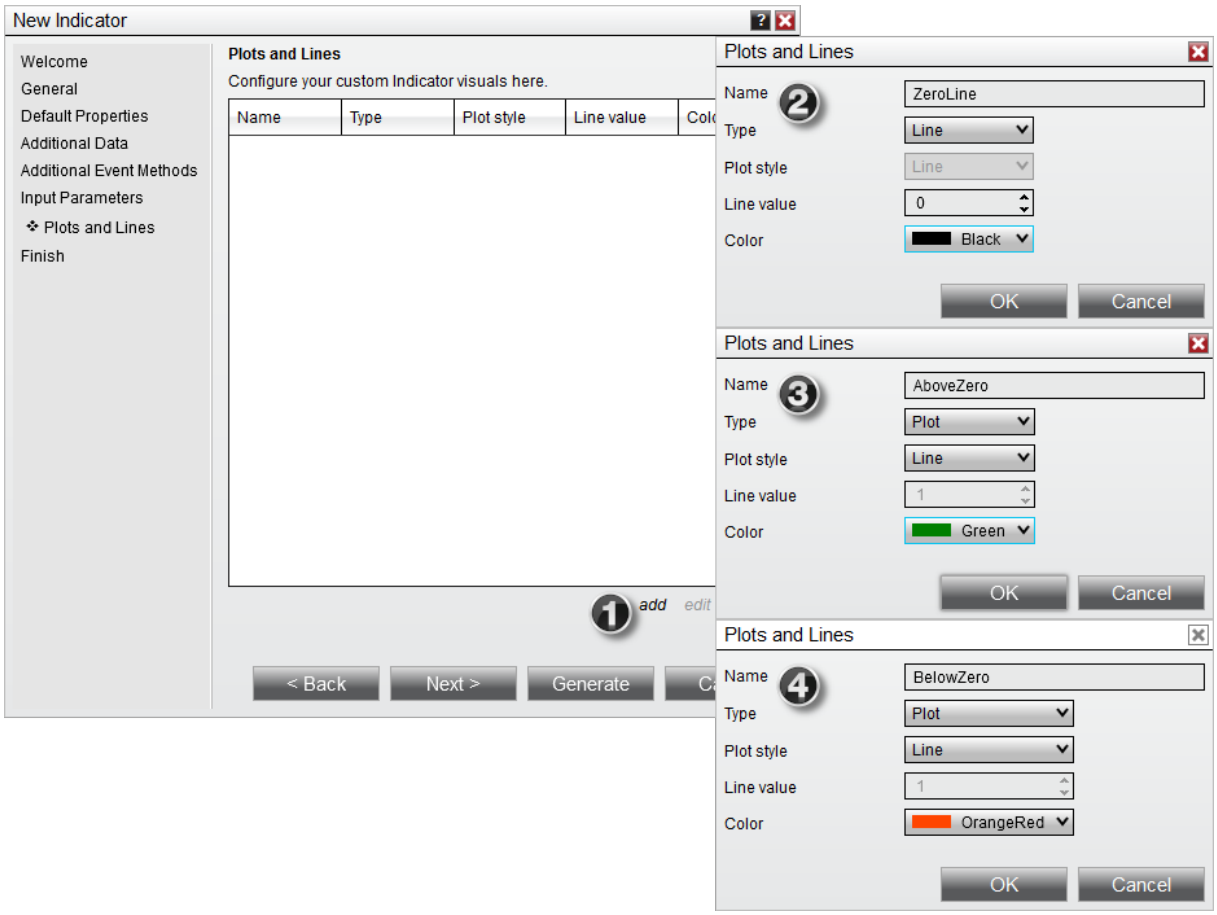


1. Clicking the **add** button on the "Input Parameters" page brings up the **Input Parameters** dialogue
2. The **Input Parameters** dialogue can be used to define user inputs
3. Click the **add** button again on the "Input Parameters" page and enter the information detailed in the **Input Parameters** dialogue marked 3

We specify a default value of 10, which will refer to 10 bars in the calculation. We also specify a minimum value of 1 to ensure that we cannot enter a 0 or negative number for `Periods`.

Defining Plots and lines

The next page will allow us to define plots and static lines for the indicator. For this indicator, we will define 1 line and 2 plots and a line, called "Zero."



1. Clicking the **add** button on the "Plots and Lines" page brings up the **Plots and Lines** dialogue
2. The **Plots and Lines** dialogue can be used to define the ZeroLine
3. Click the **add** button again on the "Plots and Lines" page and enter the information detailed in the Plots and Lines window marked 3 to add the AboveZero plot
4. Click the **add** button once more on the "Plots and Lines" page and enter the information detailed in the Plots and Lines window marked 4 to add the BelowZero plot

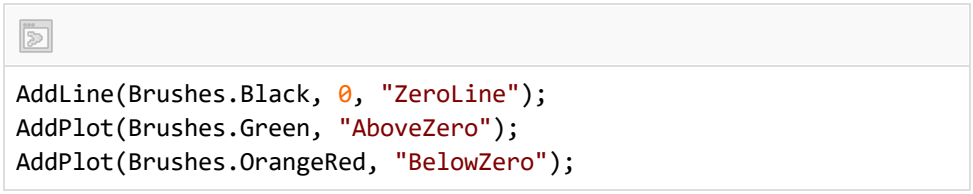
After this, click the **Finish** button, and the **Indicator Wizard** will generate a basic code structure implementing the parameters that you have set. You are now ready to move on to entering calculation logic in your code.

11.5.4.2.2 Entering Calculation Logic

The OnBarUpdate() method is called for each incoming tick or on the close of a bar (user defined) when performing real-time calculations and is called on each bar of a data series when re-calculating the indicator. For example, an indicator would be re-calculated when adding it to an existing chart that has existing price data displayed. Therefore, this is the main method called for indicator calculation and we will use this method to enter the script that will calculate the ROC value.

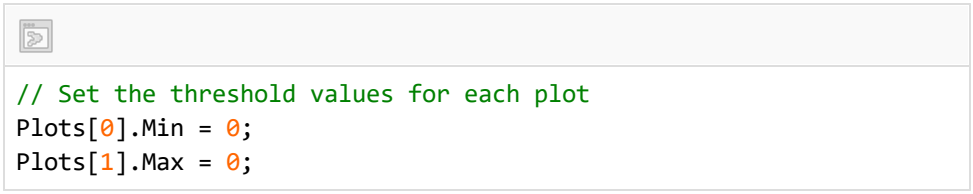
Setting Plot Thresholds

The OnStateChange() method is called once before any bar data is loaded and is used to configure the indicator. The code below is automatically generated by the wizard and added to the OnStateChange() method. It configures the indicator for two plots and one line and sets the parameters.



```
AddLine(Brushes.Black, 0, "ZeroLine");
AddPlot(Brushes.Green, "AboveZero");
AddPlot(Brushes.OrangeRed, "BelowZero");
```

Enter the following code in the OnStateChange() method and below the wizard generated code:



```
// Set the threshold values for each plot
Plots[0].Min = 0;
Plots[1].Max = 0;
```

The concept of setting threshold values is to set when and when not to paint a plot on the chart. In this indicator, we have an "AboveZero" plot with a default color of green which we only want to see when the value of ROC is above zero and a "BelowZero" plot with a default color of OrangeRed which we only want to see when the value of ROC is below zero. In order to make that happen we have to set the threshold values of each plot.

```
Plots[0].Min = 0;
```

This statement says, in the collection of Plot objects, take the first one (`Plots[0]`) and set its minimum value to zero. This means any value below zero will not display.

```
Plots[1].Max = 0;
```

This statement says, in the collection of Plot objects, take the second one (`Plots[1]`) and set its maximum value to zero. This means any value above zero will not display.

We now have a simple plot switching mechanism that displays the correct colored line depending on if the value of ROC is above or below zero. In fact, you can take this concept a little bit farther. You can even set different plots style (bar, dot etc..) depending on threshold values.

A quick word about collections. Collections are objects that store a collection of objects, kind of like a container. In this case we are working with a collection of plots. In the above wizard generated code you will notice that we are adding new plots to the "Plots" collection.

"AboveZero" was added first and then "BelowZero". This means that we can reference the "AboveZero" plot object through Plots[0]. The reason we don't pass in a value of 1 is because collections are zero based indexes. This means the first item has an index of 0, the second time an index of 1 and so forth.

Completing the Indicator

This indicator is actually quite simple in its implementation. The last thing we need to do is add the calculation code and set the value of ROC to both our plot lines.
Replace the wizard generated code with the following code into the OnBarUpdate() method in the NinjaScript Editor:

```
// Are there enough bars
if (CurrentBar < Period) return;

// Set the plot values
AboveZero[0] = SMA(ROC(Period), Smooth)[0];
BelowZero[0] = SMA(ROC(Period), Smooth)[0];
```

The calculation first checks to ensure there are enough bars to complete the calculation and then sets both plot lines to the ROC value.

The class code in your editor should look identical to the image below. You are now ready to compile the indicator and configure it on a chart.

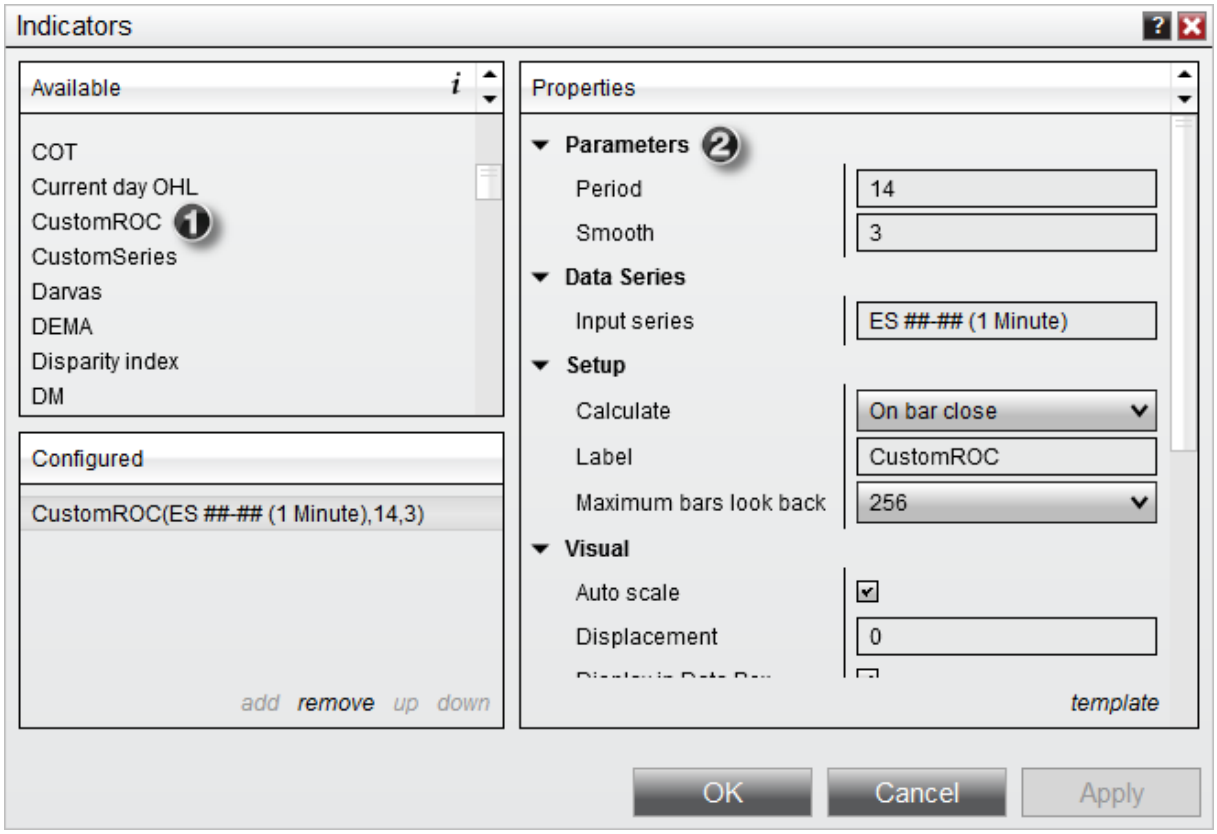
```
public class CustomROC : Indicator
{
   protected override void OnStateChange()
   {
      if (State == State.SetDefaults)
      {
         Description                                 = @"ROC with custom
line color options";
         Name                                        = "CustomROC";
         Calculate                                   =
Calculate.OnBarClose;
         IsOverlay                                   = false;
         DisplayInDataBox                            = true;
         DrawOnPricePanel                            = true;
         DrawHorizontalGridLines                     = true;
         DrawVerticalGridLines                       = true;
         PaintPriceMarkers                           = true;
         ScaleJustification                          =
NinjaTrader.Gui.Chart.ScaleJustification.Right;
         //Disable this property if your indicator requires custom
values that cumulate with each new market data event.
         //See Help Guide for additional information.
         IsSuspendedWhileInactive                    = true;
         Period                                      = 14;
         Smooth                                      = 3;
         AddLine(Brushes.Black, 0, "ZeroLine");
         AddPlot(Brushes.Green, "AboveZero");
         AddPlot(Brushes.OrangeRed, "BelowZero");
         Plots[0].Min = 0;
         Plots[1].Max = 0;
      }
      else if (State == State.Configure)
      {
      }
   }

   protected override void OnBarUpdate()
   {
      // Are there enough bars
      if (CurrentBar < Period) return;

      // Set the plot values
      AboveZero[0] = SMA(ROC(Period), Smooth)[0];
      BelowZero[0] = SMA(ROC(Period), Smooth)[0];
   }

   #region Properties
   [NinjaScriptProperty]
   [Range(1, int.MaxValue)]
   [Display(Name="Period", Description="Number of periods",
Order=1, GroupName="Parameters")]
   public int Period
   { get; set; }
```

11.5.4.2.3 Compiling

The indicator code is now complete and needs to be compiled. You can compile this indicator from within the NinjaScript Editor right mouse button menu "Compile" menu or simply press the F5 key. It is important to understand that this process makes the indicator ready for real-time use and will run natively within NinjaTrader directly. It does not run interpreted as many other applications do. This provides you with the highest performance possible. If there are any errors reported during compiling, the error messages will be displayed at the bottom of the NinjaScript Editor.

11.5.4.2.4 Using

Your indicator is now ready for use and will be listed in the Indicator Dialog window.



1) The indicator can now be found in the "Available" section of the **Indicators** window
2) Once added to the "Configured" section, our user-defined inputs can be entered along with standard indicator properties.

Once applied to a chart, the indicator should look something like the image below.

11.5.4.3 Intermediate - Historical Custom Data Series

Historical Custom Series<T> Overview

In this intermediate level tutorial we are going to build a custom indicator that stores intermediary calculations without the use of plots. This indicator will show you how to use a Series<T> object.

- Set Up
- Entering Calculation Logic
- Compiling
- Using

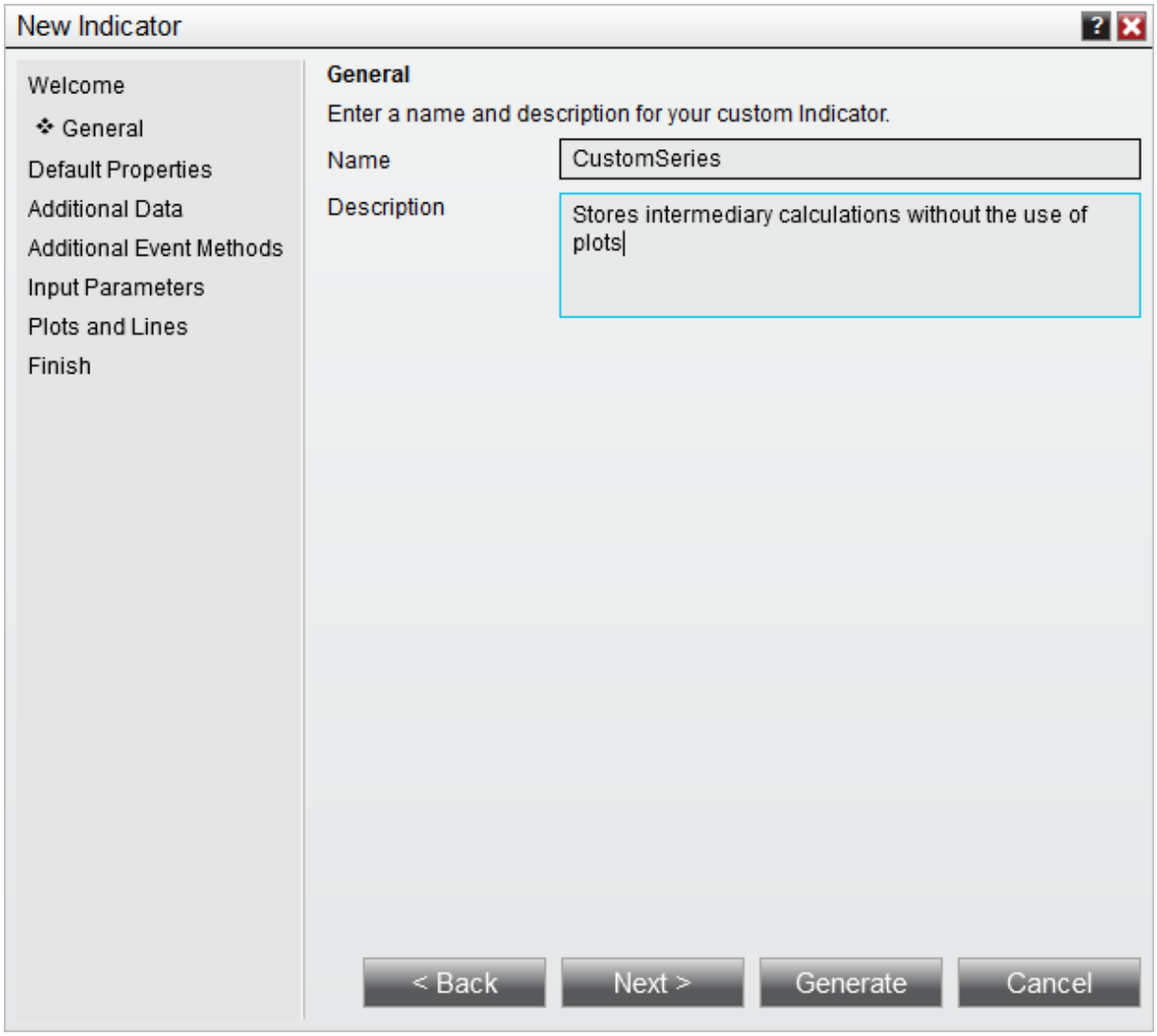
11.5.4.3.1 Set Up

The first step in creating a custom indicator is to use the custom indicator wizard. The wizard will generate the required NinjaScript code that will serve as the foundation for your custom indicator.

1. Within the NinjaTrader Control Center, select the **New** menu, then select the **NinjaScript Editor** menu item.
2. Right mouse click the "Indicators" folder in the **NinjaScript Explorer** section, then select the **New Indicator** menu item to open the **New Indicator Wizard**.

Defining Indicator Properties and Name

First you will define your indicator's name and several indicator properties. Begin by clicking the **Next >** button on the first page of the wizard to view the page shown below.

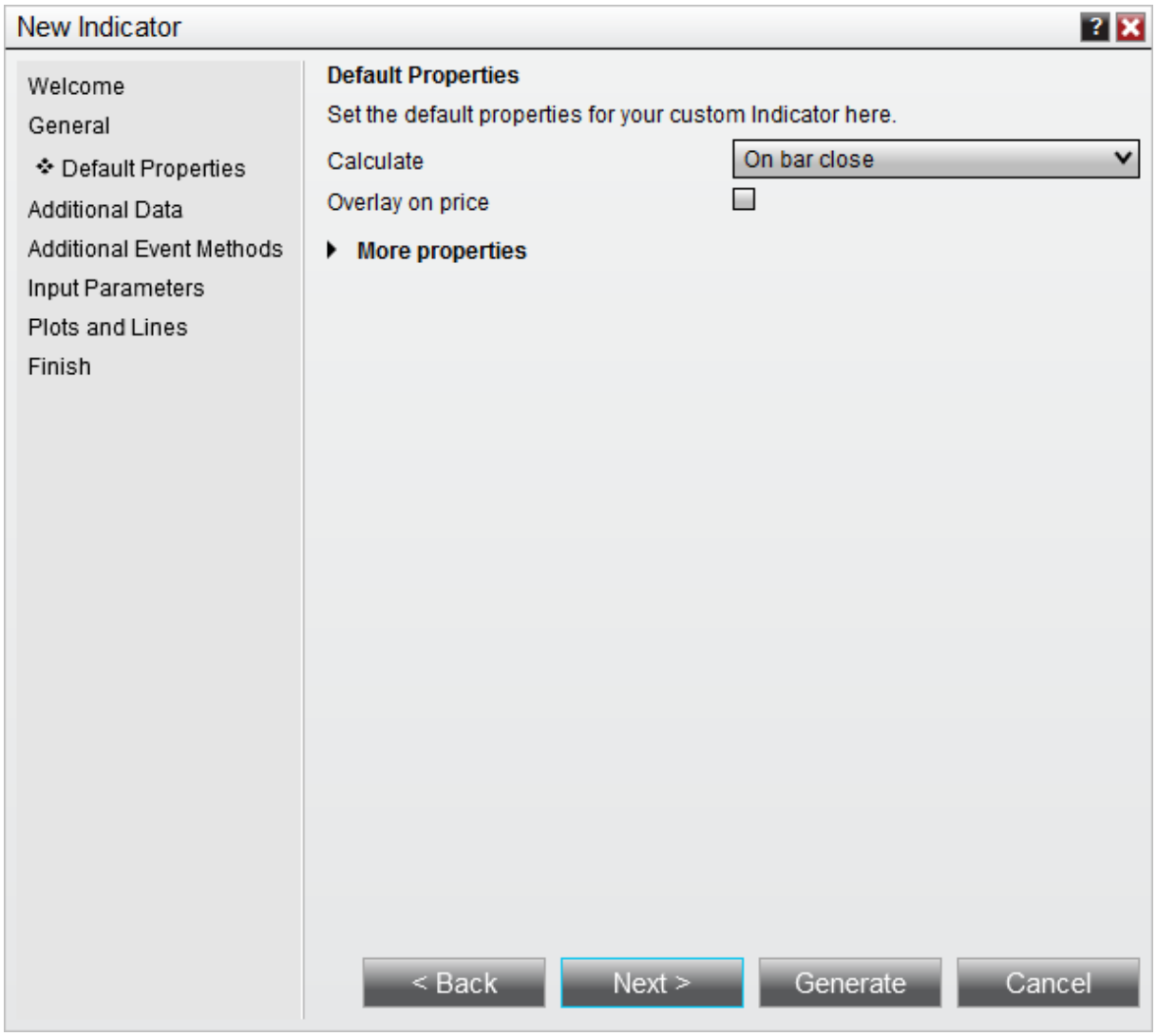


3. Enter the information as shown above
4. Click the **Next >** button

Setting Default Properties

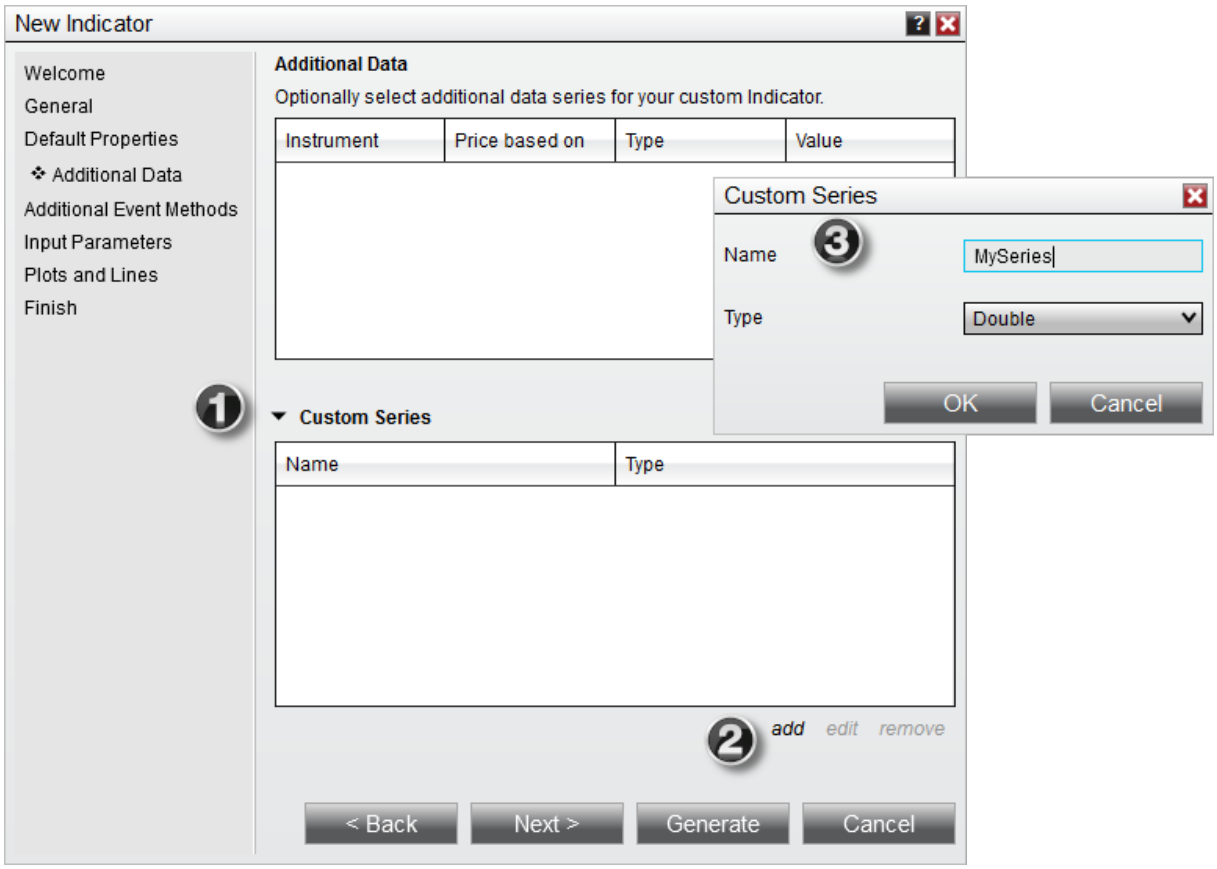
The next page will allow you to set defaults for basic properties related to your indicator, including it's Calculate and Overlay settings. Click the **More Properties** button to expose

additional properties. For this tutorial, we will not change any basic properties' defaults, and instead will leave them all set to the values shown below and move forward by clicking the **Next >** button.:



Adding Additional Data

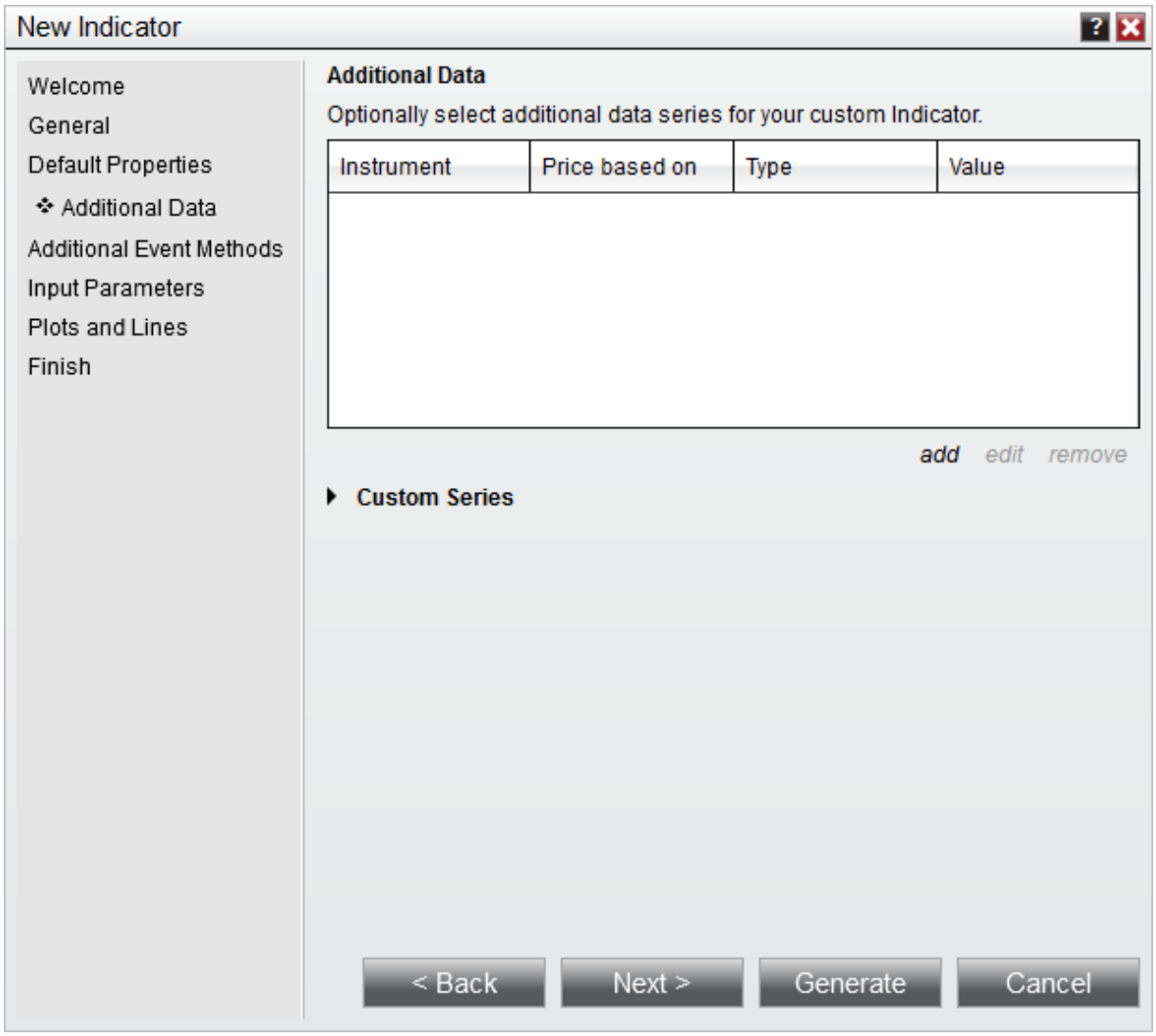
The next page will allow you to configure one or more additional Bars objects for use by the indicator. For our purposes, we will leave this page blank and move forward by clicking the **Next >** button.



1. Here we will click the arrow or the bold **Custom Series** text to be able to use the wizard to add our custom Series<T> object
2. Once the menu is expanded, we can click the **add** button to add a Series<T>
3. We will then enter the information above, and select **Double** as the variable type for the Series<T>

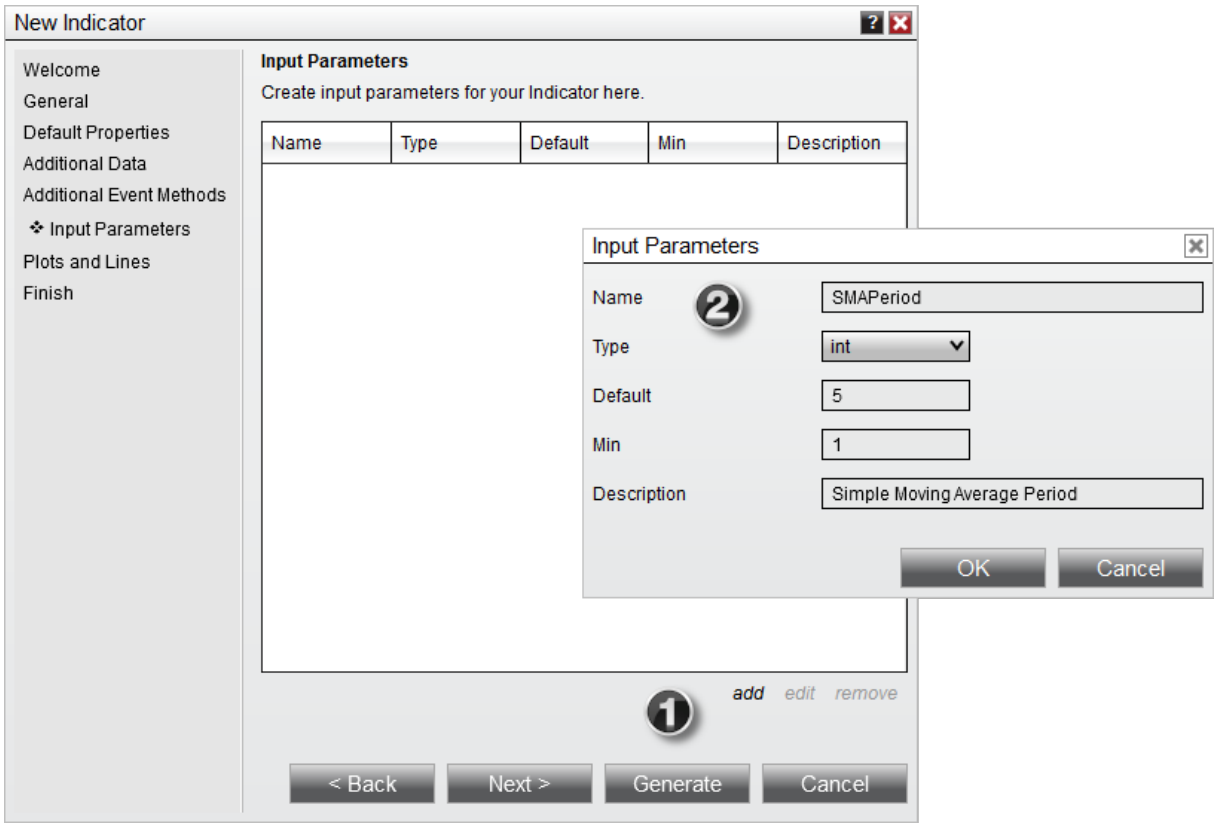
Adding Event Methods

The next page will allow you to pre-populate certain event methods into the NinjaScript code generated by the wizard. For our purposes, we will leave all of the checkboxes corresponding to different event methods unchecked, and will move on by clicking the **Next >** button.



Defining Input Parameters

The next page will allow us to configure user input parameters for the indicator. For our custom indicator, our eventual goal will be to create a simple plot that follows either above or below the bars based upon the Close price of a specified bar compared to the preceding bar. To allow for the variable selection of a number of bars ago, we will create one input parameter and call it "Periods." This variable will then be used to determine the number of bars used in the plot calculation.

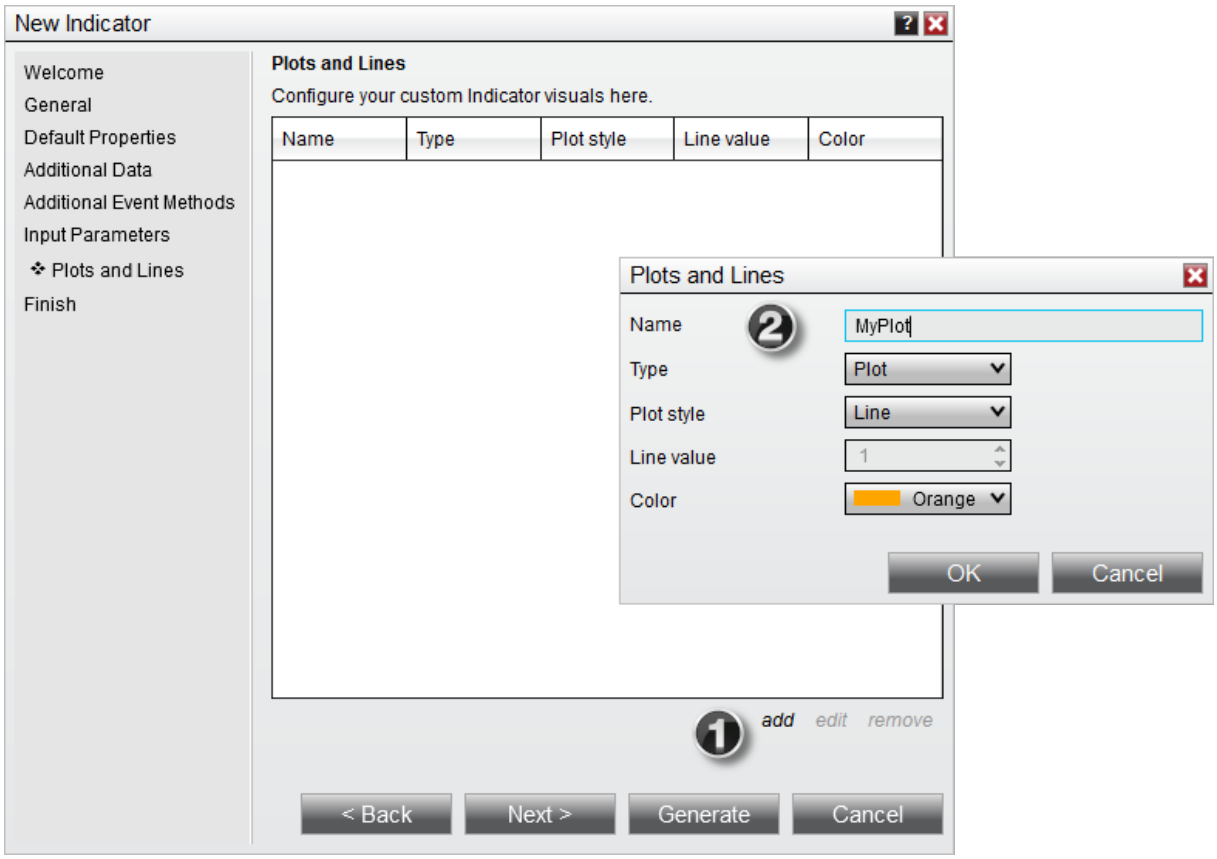


1. Clicking the **add** button on the "Input Parameters" page brings up the **Input Parameters** dialogue
2. The **Input Parameters** dialogue can be used to define user inputs

We specify a default value of 10, which will refer to 10 bars in the calculation. We also specify a minimum value of 1 to ensure that we cannot enter a 0 or negative number for `Periods`.

Defining Plots and lines

The next page will allow us to define plots and static lines for the indicator. For this indicator, we will define a single plot, called "MyPlot."



1. Clicking the **add** button on the "Plots and Lines" page brings up the **Plots and Lines** dialogue
2. The **Plots and Lines** dialogue can be used to define the plot

After this, click the **Finish** button, and the **Indicator Wizard** will generate a basic code structure implementing the parameters that you have set. You are now ready to move on to entering calculation logic in your code.

11.5.4.3.2 Entering Calculation Logic

The OnBarUpdate() method is called for each incoming tick or on the close of a bar (user defined) when performing real-time calculations and is called on each bar of a data series when re-calculating the indicator. For example, an indicator would be re-calculated when adding it to an existing chart that has existing price data displayed. Therefore, this is the main method called for indicator calculation and we will use this method to enter the script that will do our calculations.

Creating the Series<double> Object

This has already been added by the New Indicator wizard, but let's take note of the code added that creates our `Series<double>`

1. Declare a variable ("MySeries" used in this example) of type Series<double> that will hold a Series<double> object within the region "Variables"
2. Create a new Series<double> object and assigning it to the MySeries variable within the OnStateChange() method

```
private Series<double> MySeries;

protected override void OnStateChange()
{
  if (State == State.SetDefaults)
  {
    ...
  }
  else if (State == State.DataLoaded)
  {
    MySeries = new Series<double>(this);
  }
}
```

Storing calculations in the DataSeries object

Now that we have our Series<double> object we can store double values into it. For this example we will store a simple Close minus Open calculation.
Enter the following code into the OnBarUpdate() method:

```
// Calculate the range of the current bar and set the value
MySeries[0] = Close[0] - Open[0];
```

The value of a Series<T> object will be aligned with the current bar. This means that all Series<T> objects will be synced with the CurrentBar index. It allows you to store a double value that corresponds with every bar.

Using Series<T> values

With our new Series<double> object we can continue with further calculations easily. We can now use our Series<double> object as input to an indicator method such as SMA or instead of always writing Close[0] - Open[0] we can substitute our Series<double> object instead as per the example below.

To plot our final calculation we will store the calculation in our plot called 'MyPlot.' In the OnBarUpdate() method add the following code snippet:

```
// Add the bar's range to the SMA value
MyPlot[0] = SMA(SMAPeriod)[0] + MySeries[0];
```

Here we assign the SMA + `Series<double>` value to the property that represents the plot data using the "=" assignment operator. We have just finished coding our CustomSeries example. The class code in your editor should look identical to the below. You are now ready to compile the indicator and configure it on a chart.

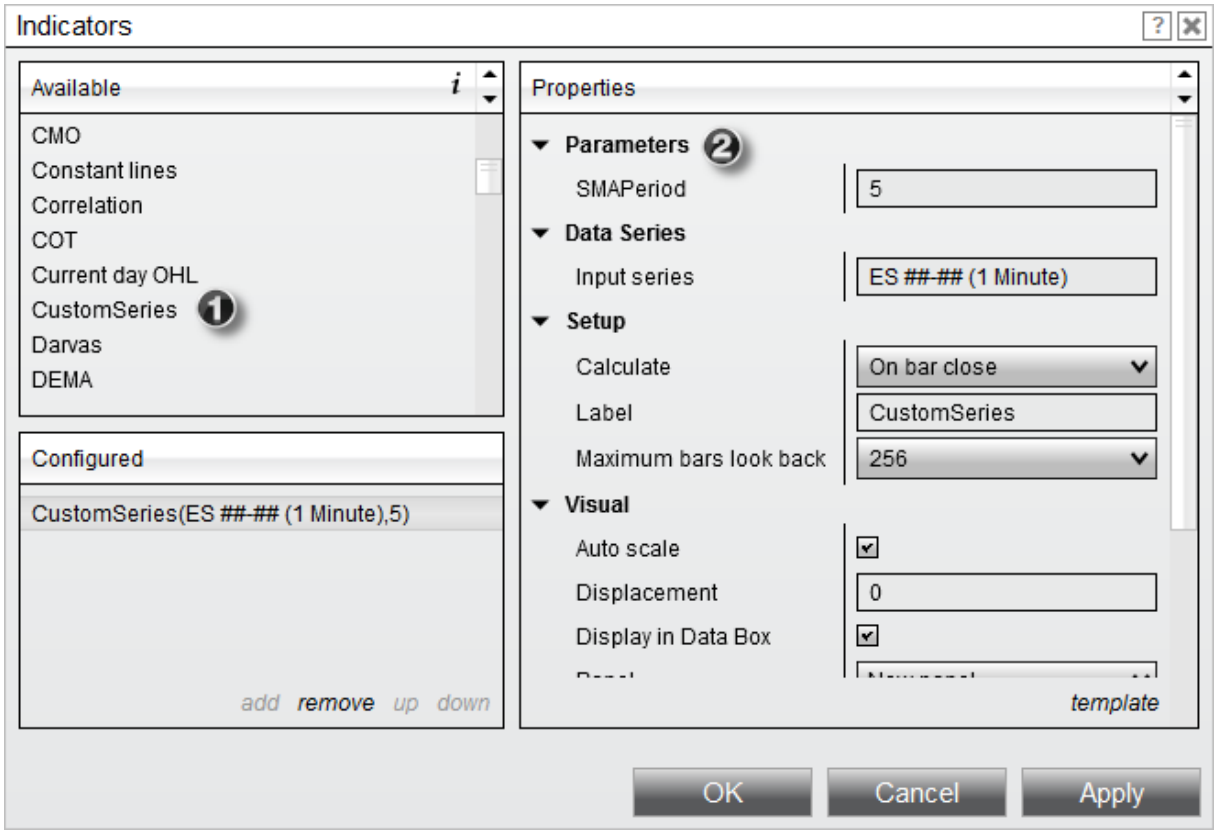
```
public class CustomSeries : Indicator
{
   private Series<double> MySeries;

   protected override void OnStateChange()
   {
      if (State == State.SetDefaults)
      {
         Description                              = @"Stores
intermediary calculations without the use of plots";
         Name                                     = "CustomSeries";
         Calculate                                =
Calculate.OnBarClose;
         IsOverlay                                = false;
         DisplayInDataBox                         = true;
         DrawOnPricePanel                         = true;
         DrawHorizontalGridLines                  = true;
         DrawVerticalGridLines                    = true;
         PaintPriceMarkers                        = true;
         ScaleJustification                       =
NinjaTrader.Gui.Chart.ScaleJustification.Right;
         //Disable this property if your indicator requires custom
values that cumulate with each new market data event.
         //See Help Guide for additional information.
         IsSuspendedWhileInactive                 = true;
         SMAPeriod                                = 5;
         AddPlot(Brushes.Orange, "MyPlot");
      }
      else if (State == State.Configure)
      {
      }
      else if (State == State.DataLoaded)
      {
         MySeries = new Series<double>(this);
      }
   }

   protected override void OnBarUpdate()
   {
      // Calculate the range of the current bar and set the value
      MySeries[0] = Close[0] - Open[0];

      // Add the bar's range to the SMA value
      MyPlot[0] = SMA(SMAPeriod)[0] + MySeries[0];
   }

   #region Properties
   [NinjaScriptProperty]
   [Range(1, int.MaxValue)]
   [Display(Name="SMAPeriod", Description="Simple Moving Average
Period", Order=1, GroupName="Parameters")]
   public int SMAPeriod
   { get; set; }
```

11.5.4.3.3 Compiling

The indicator code is now complete and needs to be compiled. You can compile this indicator from within the NinjaScript Editor right mouse button menu "Compile" menu or simply press the F5 key. It is important to understand that this process makes the indicator ready for real-time use and will run natively within NinjaTrader directly. It does not run interpreted as many other applications do. This provides you with the highest performance possible. If there are any errors reported during compiling, the error messages will be displayed at the bottom of the NinjaScript Editor.

11.5.4.3.4 Using

Your indicator is now ready for use and will be listed in the Indicator Dialog window.



1) The indicator can now be found in the "Available" section of the **Indicators** window
2) Once added to the "Configured" section, our user-defined inputs can be entered along with standard indicator properties.

Once applied to a chart, the indicator should look something like the image below.

11.5.4.4 Intermediate - Your own SMA

Your Own SMA Overview

In this intermediate level tutorial we are going to build a simple moving average indicator. This indicator will show you how to use the "for" loop and a single case "if" statement.

- Set Up
- Entering Calculation Logic
- Compiling
- Using

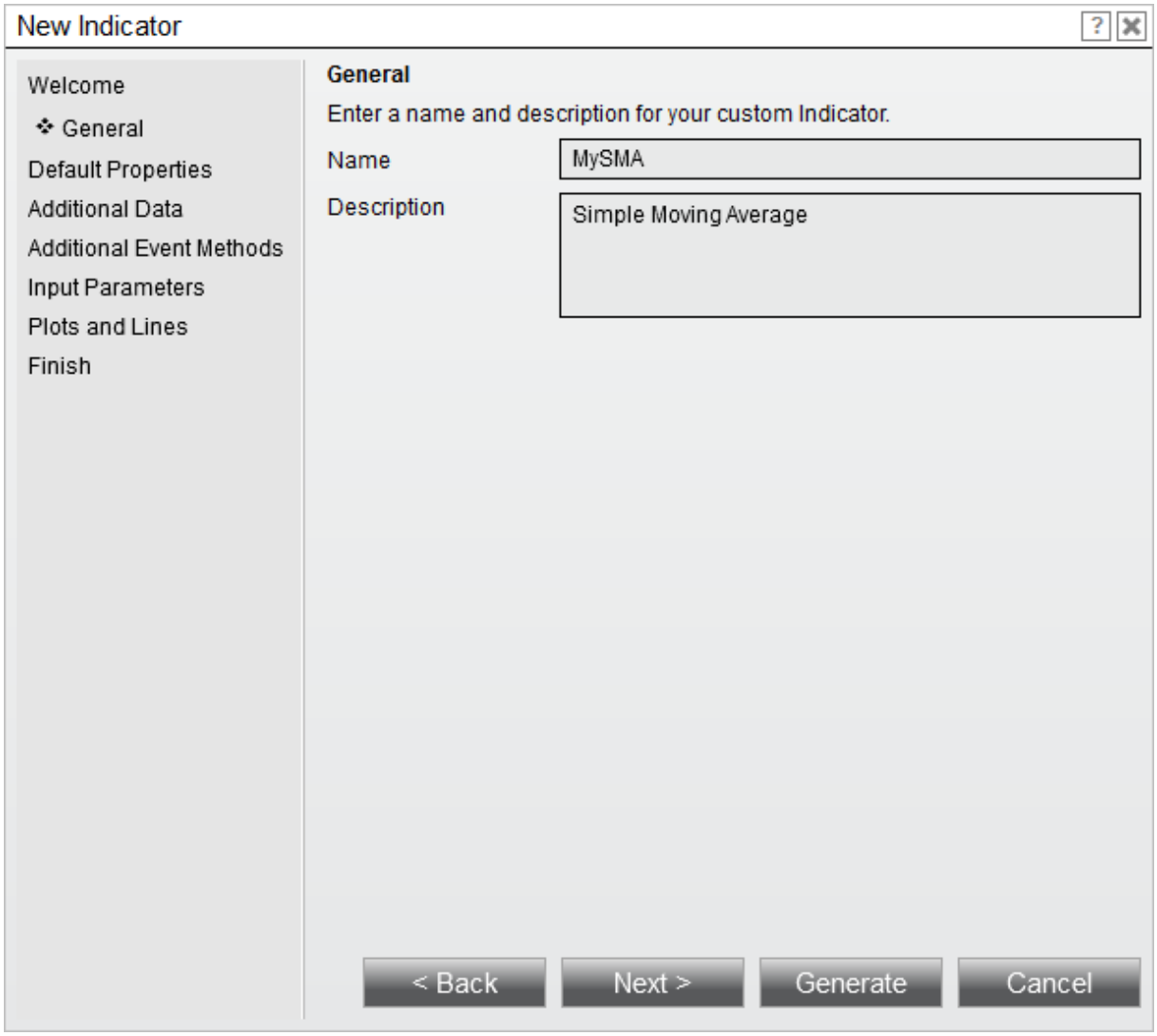
11.5.4.4.1 Set Up

The first step in creating a custom indicator is to use the custom indicator wizard. The wizard will generate the required NinjaScript code that will serve as the foundation for your custom indicator.

1. Within the NinjaTrader Control Center, select the **New** menu, then select the **NinjaScript Editor** menu item.
2. Right mouse click the "Indicators" folder in the **NinjaScript Explorer** section, then select the **New Indicator** menu item to open the **New Indicator Wizard**.

Defining Indicator Properties and Name

First you will define your indicator's name and several indicator properties. Begin by clicking the **Next >** button on the first page of the wizard to view the page shown below.

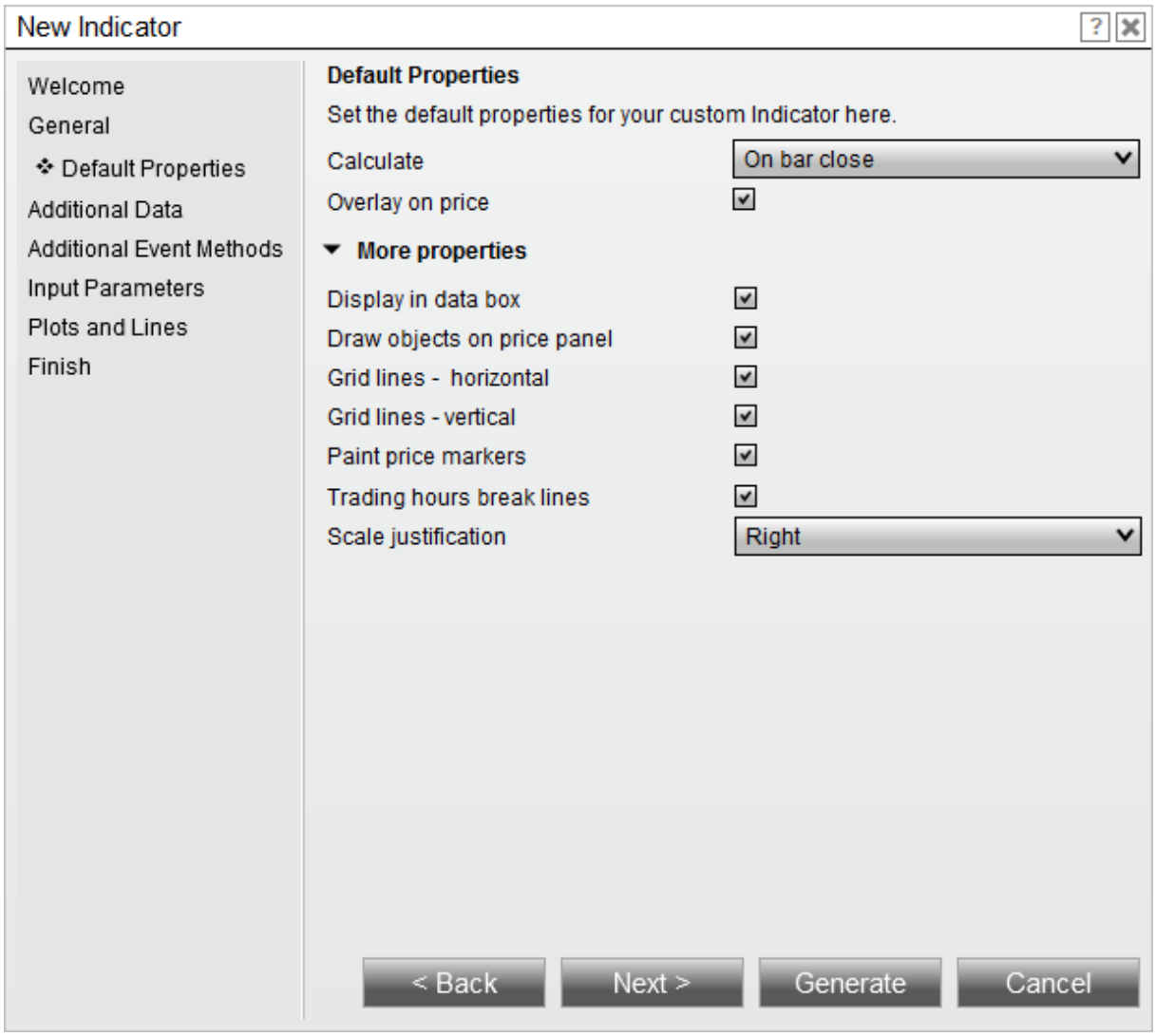


3. Enter the information as shown above
4. Click the **Next >** button

Setting Default Properties

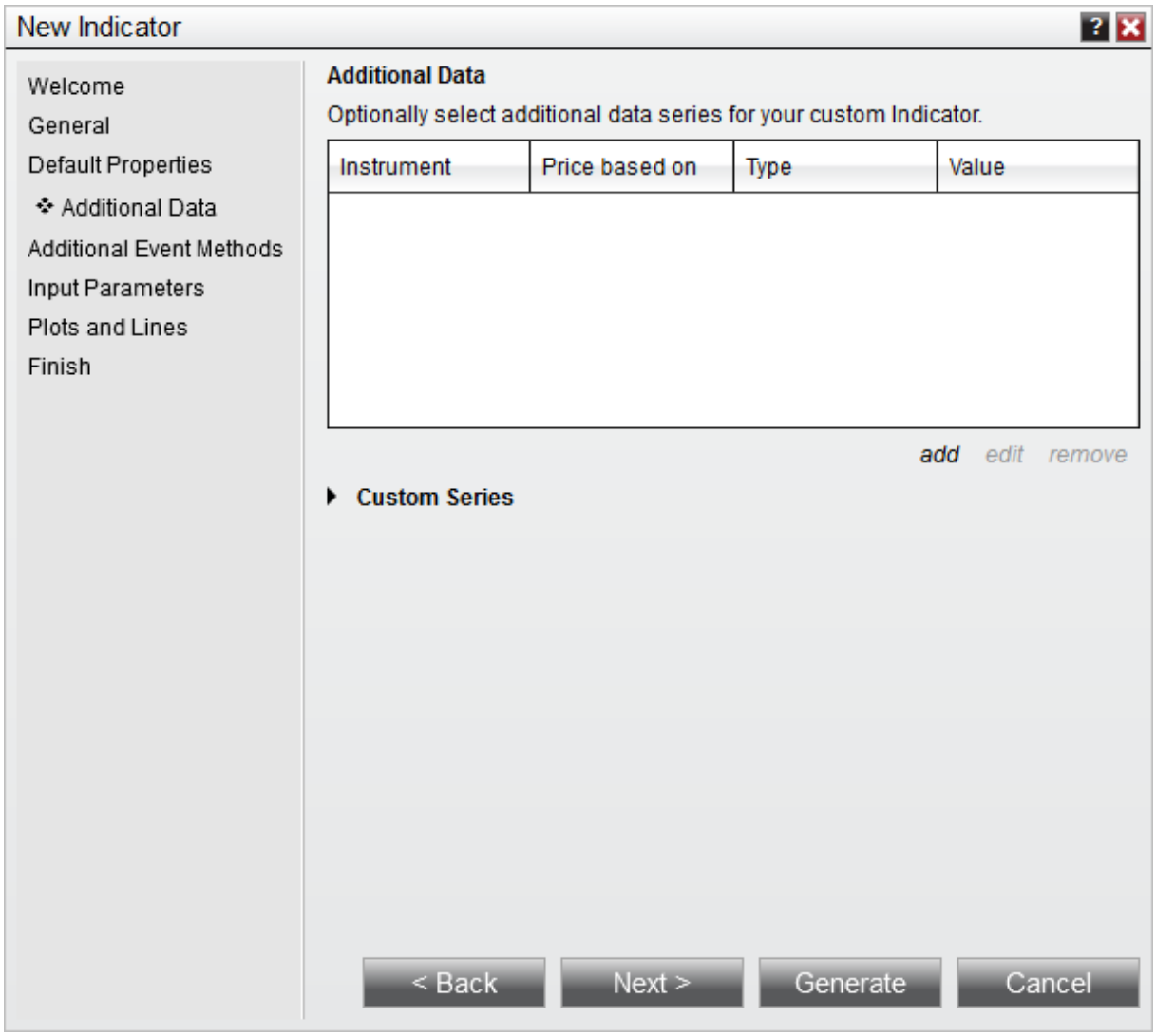
The next page will allow you to set defaults for basic properties related to your indicator, including it's Calculate and Overlay settings. Click the **More Properties** button to expose

additional properties. For this tutorial, we will not change any basic properties' defaults, and instead will leave them all set to the values shown below and move forward by clicking the **Next >** button.:



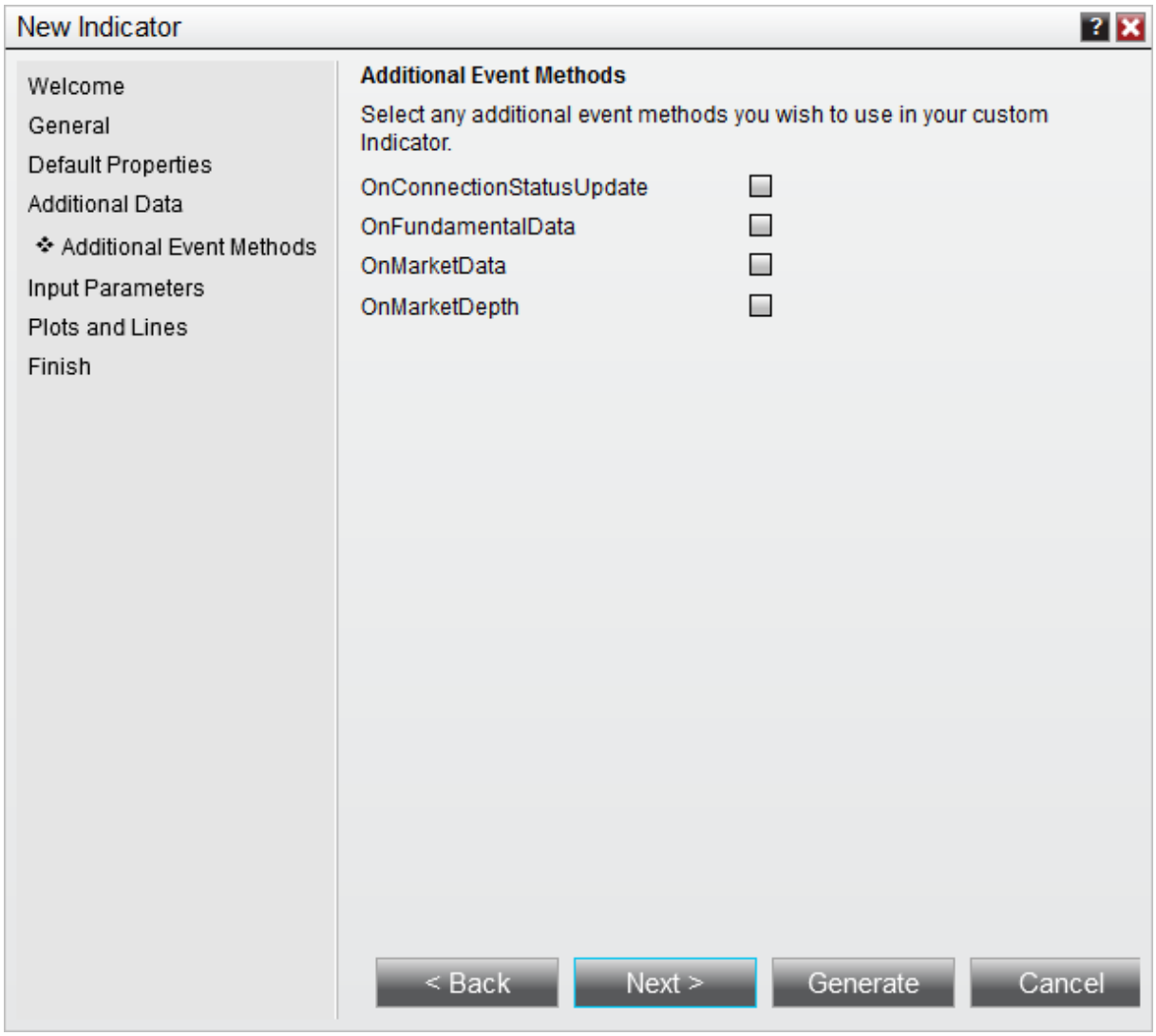
Adding Additional Data

The next page will allow you to configure one or more additional Bars objects for use by the indicator. For our purposes, we will leave this page blank and move forward by clicking the **Next >** button.



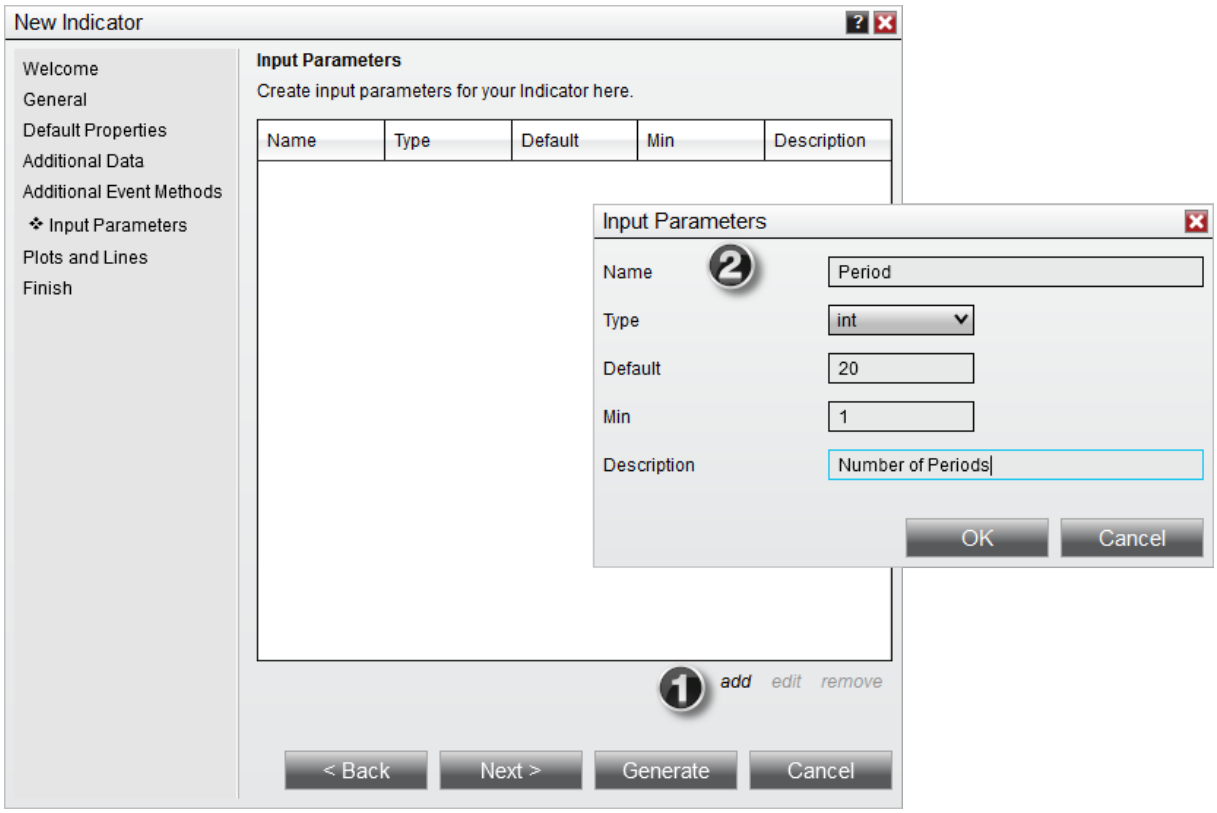
Adding Event Methods

The next page will allow you to pre-populate certain event methods into the NinjaScript code generated by the wizard. For our purposes, we will leave all of the checkboxes corresponding to different event methods unchecked, and will move on by clicking the **Next >** button.



Defining Input Parameters

The next page will allow us to configure user input parameters for the indicator. For our custom indicator, our eventual goal will be to create a simple plot that follows either above or below the bars based upon the Close price of a specified bar compared to the preceding bar. To allow for the variable selection of a number of bars ago, we will create one input parameter and call it "Periods." This variable will then be used to determine the number of bars used in the plot calculation.

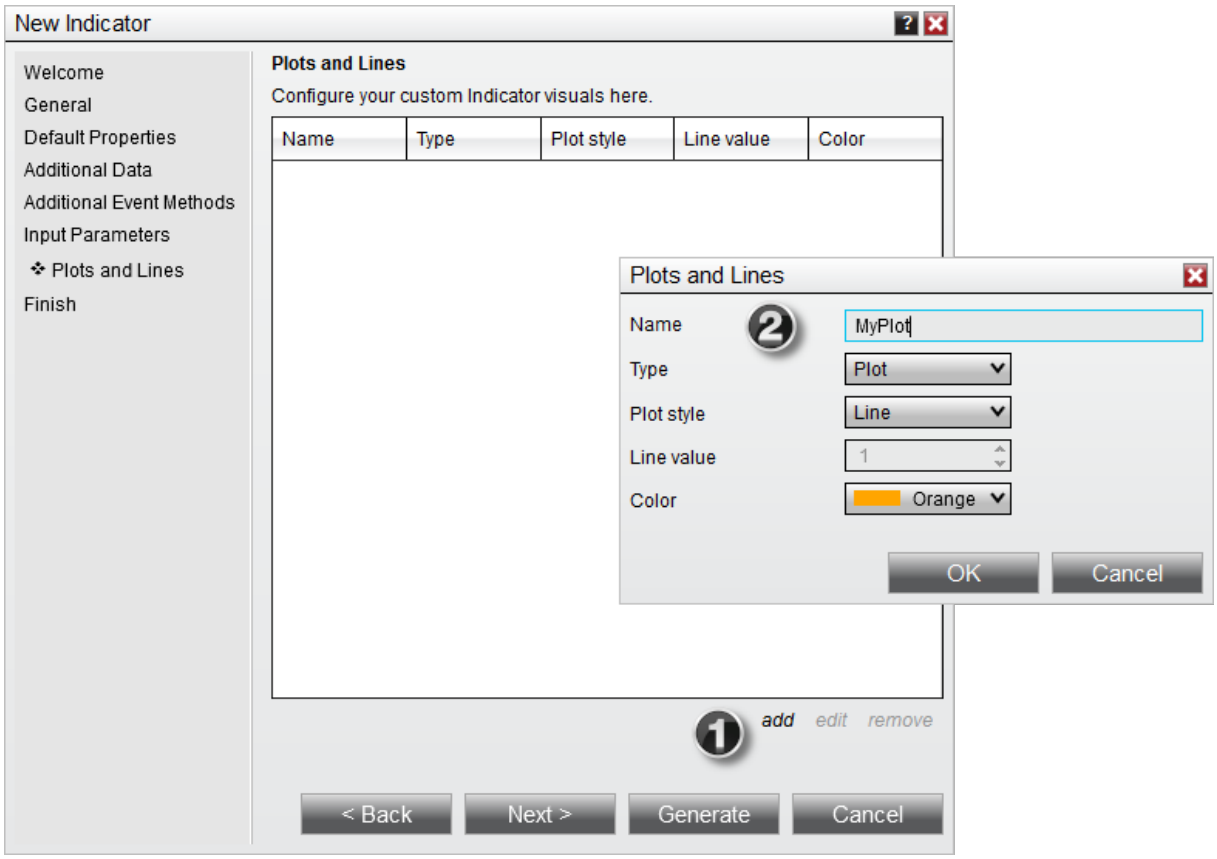


1. Clicking the **add** button on the "Input Parameters" page brings up the **Input Parameters** dialogue
2. The **Input Parameters** dialogue can be used to define user inputs

We specify a default value of 10, which will refer to 10 bars in the calculation. We also specify a minimum value of 1 to ensure that we cannot enter a 0 or negative number for `Periods`.

Defining Plots and lines

The next page will allow us to define plots and static lines for the indicator. For this indicator, we will define a single plot, called "MyPlot."



1. Clicking the **add** button on the "Plots and Lines" page brings up the **Plots and Lines** dialogue
2. The **Plots and Lines** dialogue can be used to define the plot

After this, click the **Finish** button, and the **Indicator Wizard** will generate a basic code structure implementing the parameters that you have set. You are now ready to move on to entering calculation logic in your code.

11.5.4.4.2 Entering Calculation Logic

The OnBarUpdate() method is called for each incoming tick, or on the close of a bar (if enabled) when performing real-time calculations, and is called on each bar of a Bars object when re-calculating the indicator (For example, an indicator would be re-calculated when adding it to an existing chart that has existing price data displayed).. Therefore, this is the main method called for indicator calculation and we will use this method to enter the script that will calculate a simple moving average.

Are there enough bars?

Enter the following code into the OnBarUpdate() method in the NinjaScript Editor:

```
// Do not calculate if we don't have enough bars
if (CurrentBar < Period) return;
```

To calculate a 20 period moving average you will need a minimum of 20 bars of data. The first statement in our OnBarUpdate() method checks to see if there are enough bars of data to perform the moving average calculation. "CurrentBar" returns the index number of the current bar and this is checked against the user defined parameter "Period". If the current bar number is less than the user defined period we "return" which skips calculating the moving average.

Getting a sum of closing prices

Enter the following code into the OnBarUpdate() method and below the code snippet you entered above:

```
// Get a sum of prices over the specified period
double sum = 0;
for (int barsAgo = 0; barsAgo < Period; barsAgo++)
{
    sum = sum + Input[barsAgo];
}
```

First we must declare a variable that will store our sum total.

```
double sum = 0;
```

The variable "sum" whose value is of type "double" will serve as temporary storage.

```
for (int barsAgo = 0; barsAgo < Period; barsAgo++)
{
    sum = sum + Input[barsAgo];
}
```

Next we must calculate the sum. We use a standard "for" loop to skip through prices and add them to the "sum" variable. Although the command that represents the loop may look intimidating, its really quite simple. Let's look at it in English....

What the loop is saying is:

1. the number of bars ago is now zero
2. as long as the number of bars ago is less than the moving average period, then go to line 3 otherwise this loop is finished
3. get the price Input[number of bars ago] and add it to the running sum total
4. add one to the number of bars ago (if number of bars ago was zero it will now be one)
5. go to to line 2

You can find more information on how loops work here. Once the loop has finished, it will have calculated the total sum of closing prices for the period of our moving average.

* We use the value of `Input[barsAgo]` to get a price to use for our calculation. We could have substituted `Close[barsAgo]` to use closing prices or `High[barsAgo]` to use high prices. The reason we use `Input[barsAgo]` is since this allows flexibility for what the indicator is calculated based off of. Remember users have the option to select a price type (High, Open, Close etc...) from the Indicator Dialog window.

The final calculation

Enter the following code into the OnBarUpdate() method and below the code snippet you entered above:

```
// Calculate and set the average value to the 'MyPlot' property
MyPlot[0] = sum / Period;
```

We can now calculate the final moving average value and assign it's value to the property that represents the plot data. We have just finished coding our simple moving average. The class code in your editor should look identical to the image below. You are now ready to compile the indicator and configure it on a chart.

```
public class MySMA : Indicator
{
   protected override void OnStateChange()
   {
      if (State == State.SetDefaults)
      {
         Description                              = @"Simple Moving
Average";
         Name                                     = "MySMA";
         Calculate                                =
Calculate.OnBarClose;
         IsOverlay                                = true;
         DisplayInDataBox                         = true;
         DrawOnPricePanel                         = true;
         DrawHorizontalGridLines                  = true;
         DrawVerticalGridLines                    = true;
         PaintPriceMarkers                        = true;
         ScaleJustification                       =
NinjaTrader.Gui.Chart.ScaleJustification.Right;
         //Disable this property if your indicator requires custom
values that cumulate with each new market data event.
         //See Help Guide for additional information.
         IsSuspendedWhileInactive                 = true;
         Period                                   = 20;
         AddPlot(Brushes.Orange, "MyPlot");
      }
      else if (State == State.Configure)
      {
      }
   }

   protected override void OnBarUpdate()
   {
      // Do not calculate if we don't have enough bars
      if (CurrentBar < Period) return;

      // Get a sum of prices over the specified period
      double sum = 0;
      for (int barsAgo = 0; barsAgo < Period; barsAgo++)
      {
          sum = sum + Input[barsAgo];
      }

      // Calculate and set the average value to the 'MyPlot'
property
      MyPlot[0] = sum / Period;
   }

   #region Properties
   [NinjaScriptProperty]
   [Range(1, int.MaxValue)]
   [Display(Name="Period", Description="Number of Periods",
Order=1, GroupName="Parameters")]
```

Alternate Implementation

In this tutorial we are using a "for" loop to iterate through a collection of prices and accumulate a sum value. We chose this approach to demonstrate the use of a loop. A simple moving average can actually be expressed in a more efficient manner using the built in SUM indicator as show below.

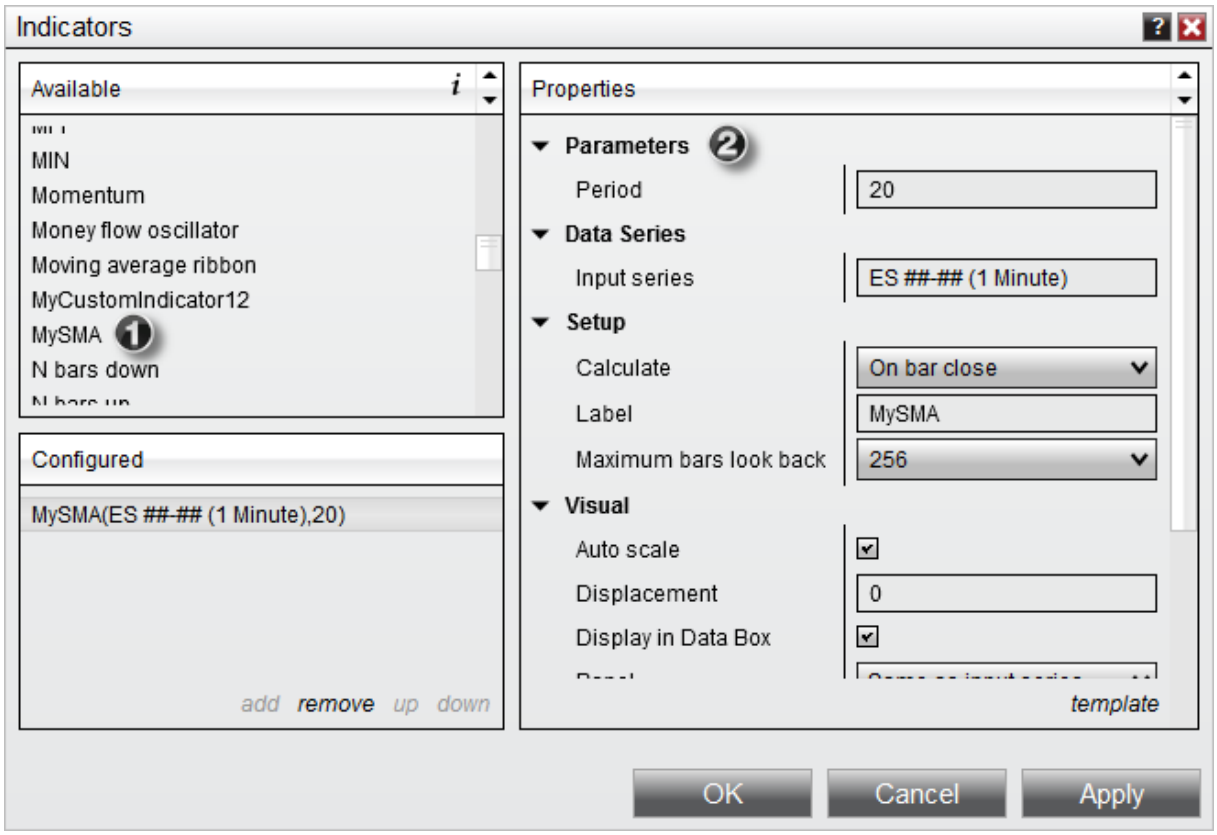
```
// Do not calculate if we don't have enough bars
if (CurrentBar < Period) return;

// Calculate and set the 'average' value to the 'MyPlot' property
MyPlot[0] = SUM(Input, Period)[0] / Period;
```

11.5.4.4.3 Compiling

The indicator code is now complete and needs to be compiled. You can compile this indicator from within the NinjaScript Editor right mouse button menu "Compile" menu or simply press the F5 key. It is important to understand that this process makes the indicator ready for real-time use and will run natively within NinjaTrader directly. It does not run interpreted as many other applications do. This provides you with the highest performance possible. If there are any errors reported during compiling, the error messages will be displayed at the bottom of the NinjaScript Editor.

11.5.4.4.4 Using

Your indicator is now ready for use and will be listed in the Indicator Dialog window.



1) The indicator can now be found in the "Available" section of the **Indicators** window
2) Once added to the "Configured" section, our user-defined inputs can be entered along with standard indicator properties.

Once applied to a chart, the indicator should look something like the image below.

11.5.4.5 Beginner - Indicator on Indicator

Indicator on Indicator Overview

In this beginner level tutorial we are going to build a custom indicator that calculates a moving average of volume. This indicator will show you how to use the built in indicators of Moving Average (SMA) and Volume.

- Set Up
- Entering Calculation Logic
- Compiling
- Using

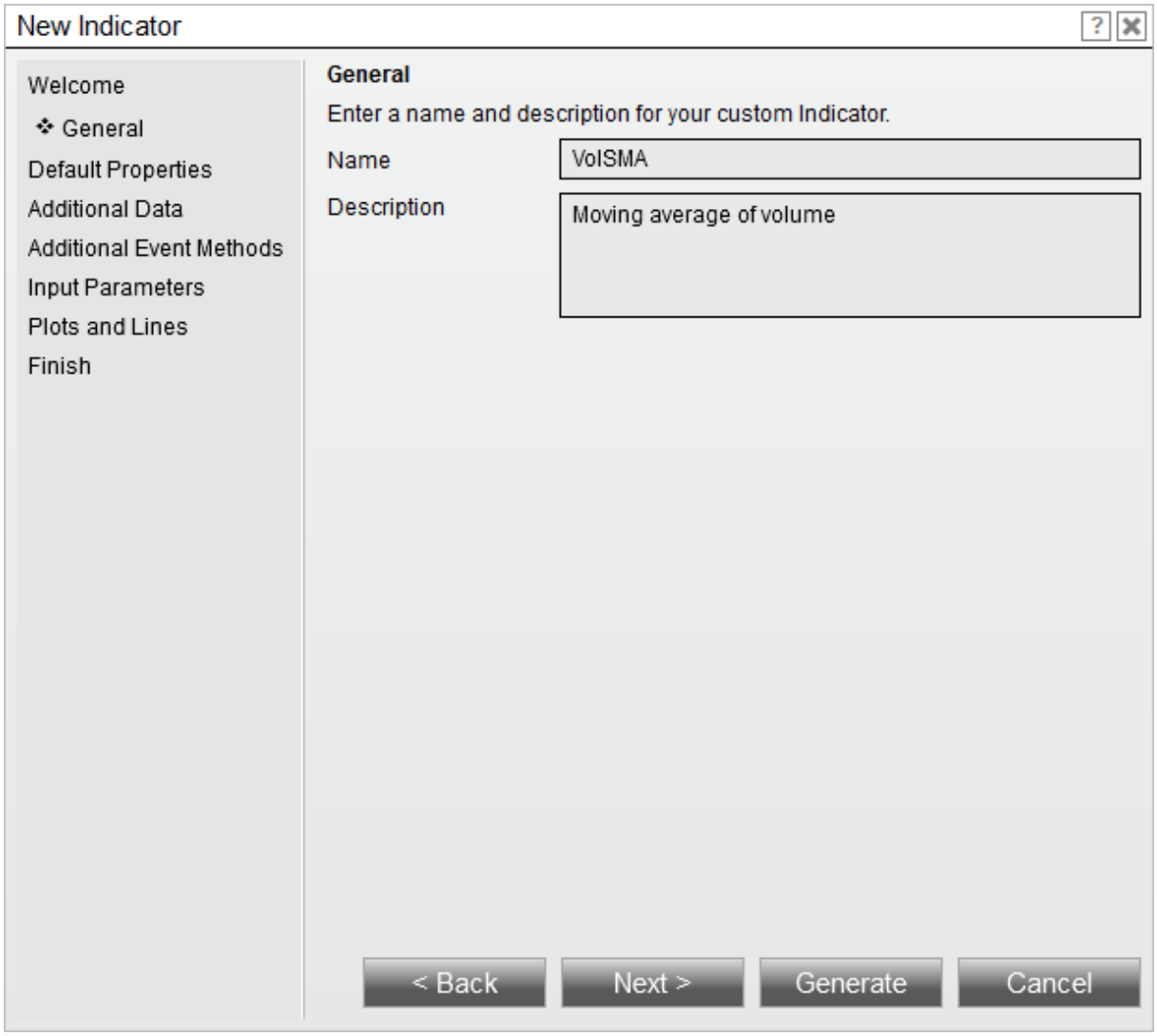
11.5.4.5.1 Set Up

The first step in creating a custom indicator is to use the custom indicator wizard. The wizard will generate the required NinjaScript code that will serve as the foundation for your custom indicator.

1. Within the NinjaTrader Control Center, select the **New** menu, then select the **NinjaScript Editor** menu item.
2. Right mouse click the "Indicators" folder in the **NinjaScript Explorer** section, then select the **New Indicator** menu item to open the **New Indicator Wizard**.

Defining Indicator Properties and Name

First you will define your indicator's name and several indicator properties. Begin by clicking the **Next >** button on the first page of the wizard to view the page shown below.

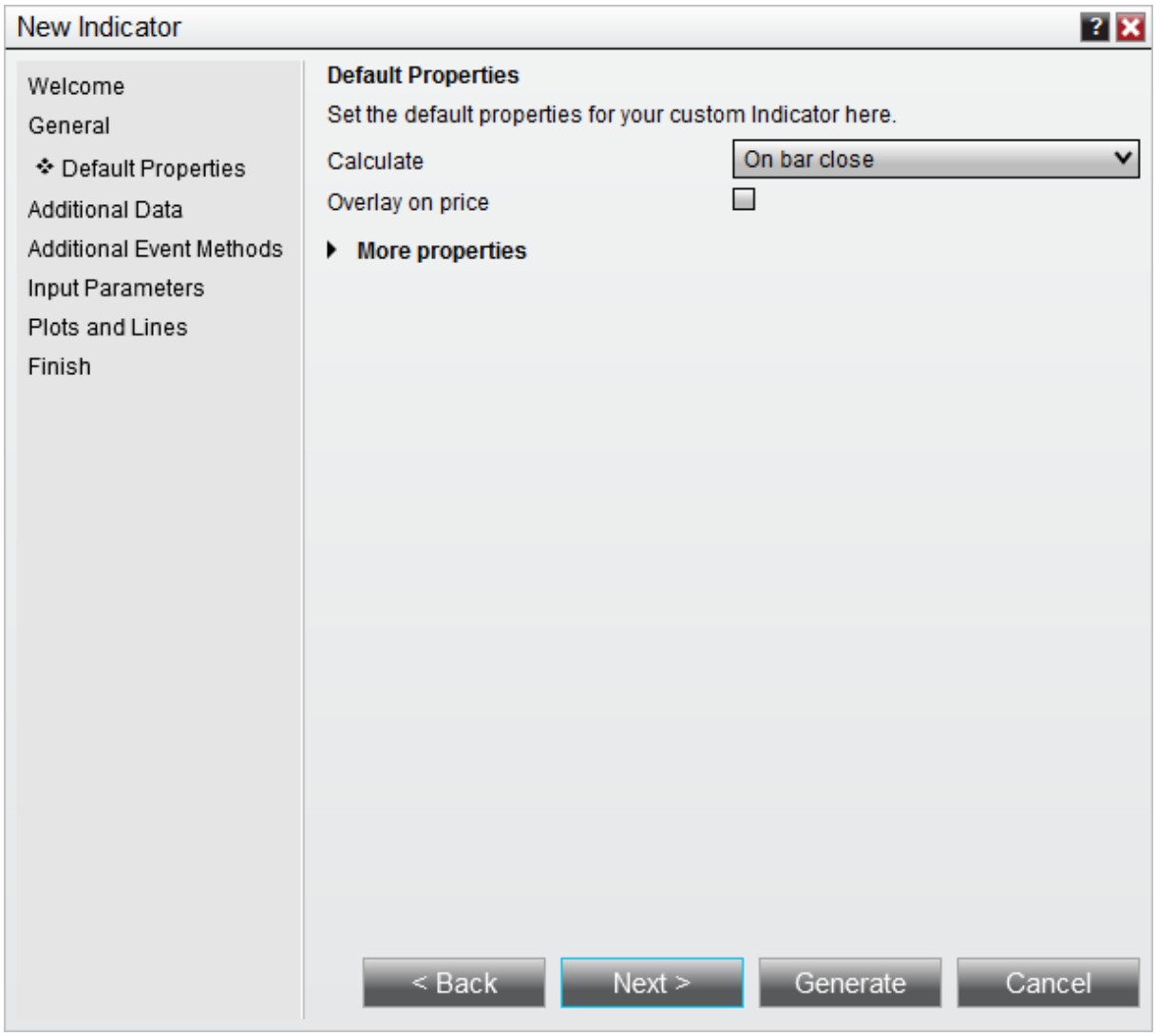


3. Enter the information as shown above
4. Click the **Next >** button

Setting Default Properties

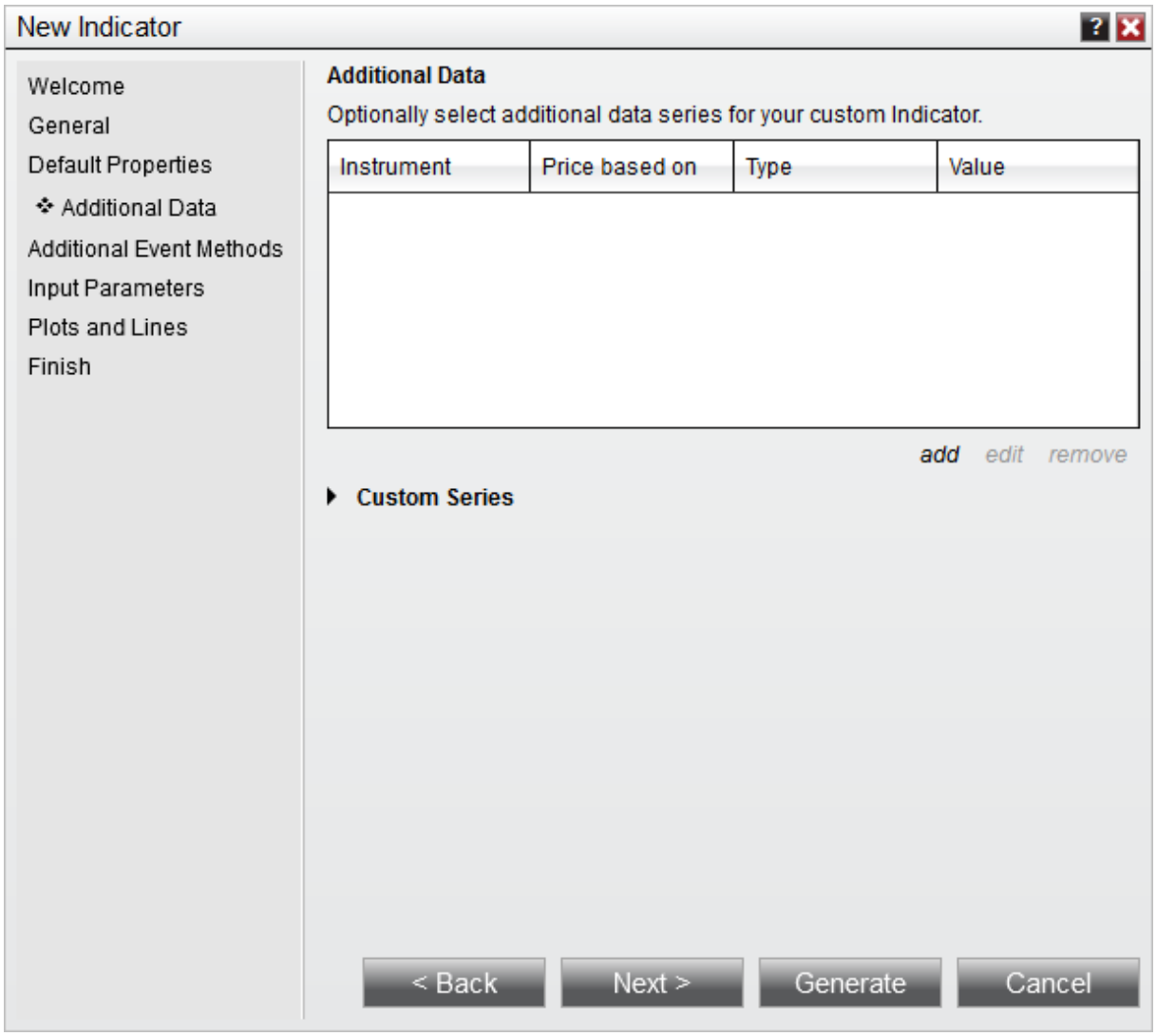
The next page will allow you to set defaults for basic properties related to your indicator, including it's Calculate and Overlay settings. Click the **More Properties** button to expose

additional properties. For this tutorial, we will not change any basic properties' defaults, and instead will leave them all set to the values shown below and move forward by clicking the **Next >** button.:



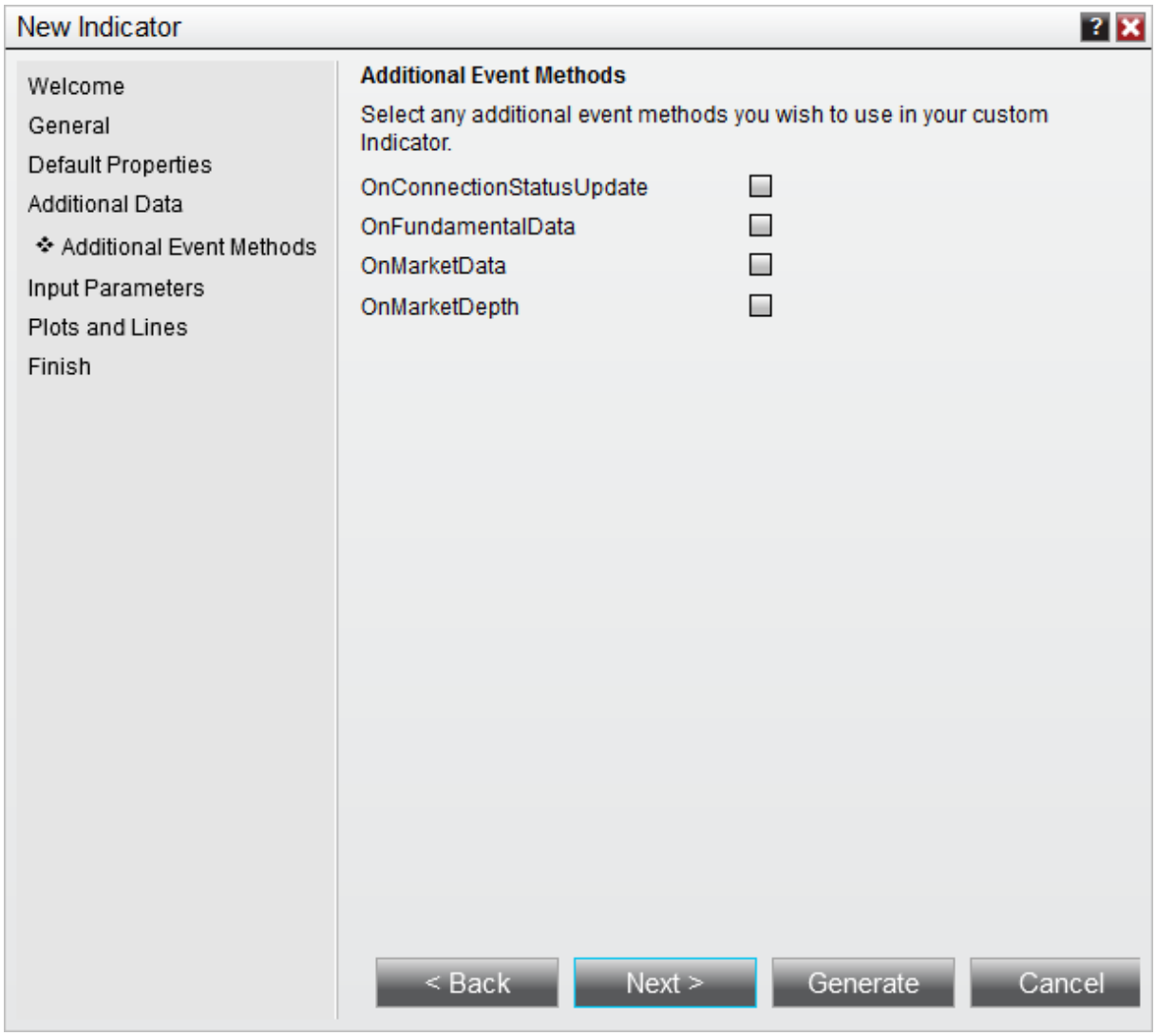
Adding Additional Data

The next page will allow you to configure one or more additional Bars objects for use by the indicator. For our purposes, we will leave this page blank and move forward by clicking the **Next >** button.



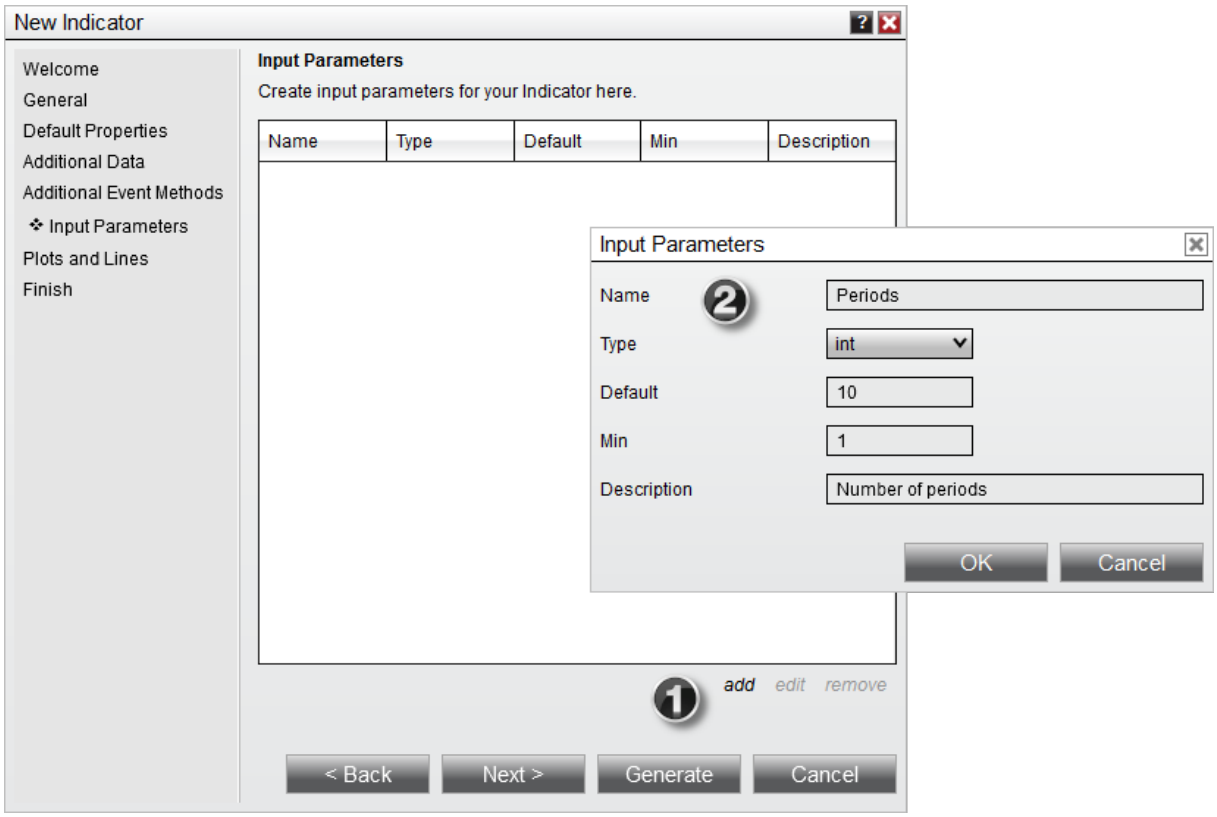
Adding Event Methods

The next page will allow you to pre-populate certain event methods into the NinjaScript code generated by the wizard. For our purposes, we will leave all of the checkboxes corresponding to different event methods unchecked, and will move on by clicking the **Next >** button.



Defining Input Parameters

The next page will allow us to configure user input parameters for the indicator. For our custom indicator, our eventual goal will be to create a simple plot that follows either above or below the bars based upon the Close price of a specified bar compared to the preceding bar. To allow for the variable selection of a number of bars ago, we will create one input parameter and call it "Periods." This variable will then be used to determine the number of bars used in the plot calculation.

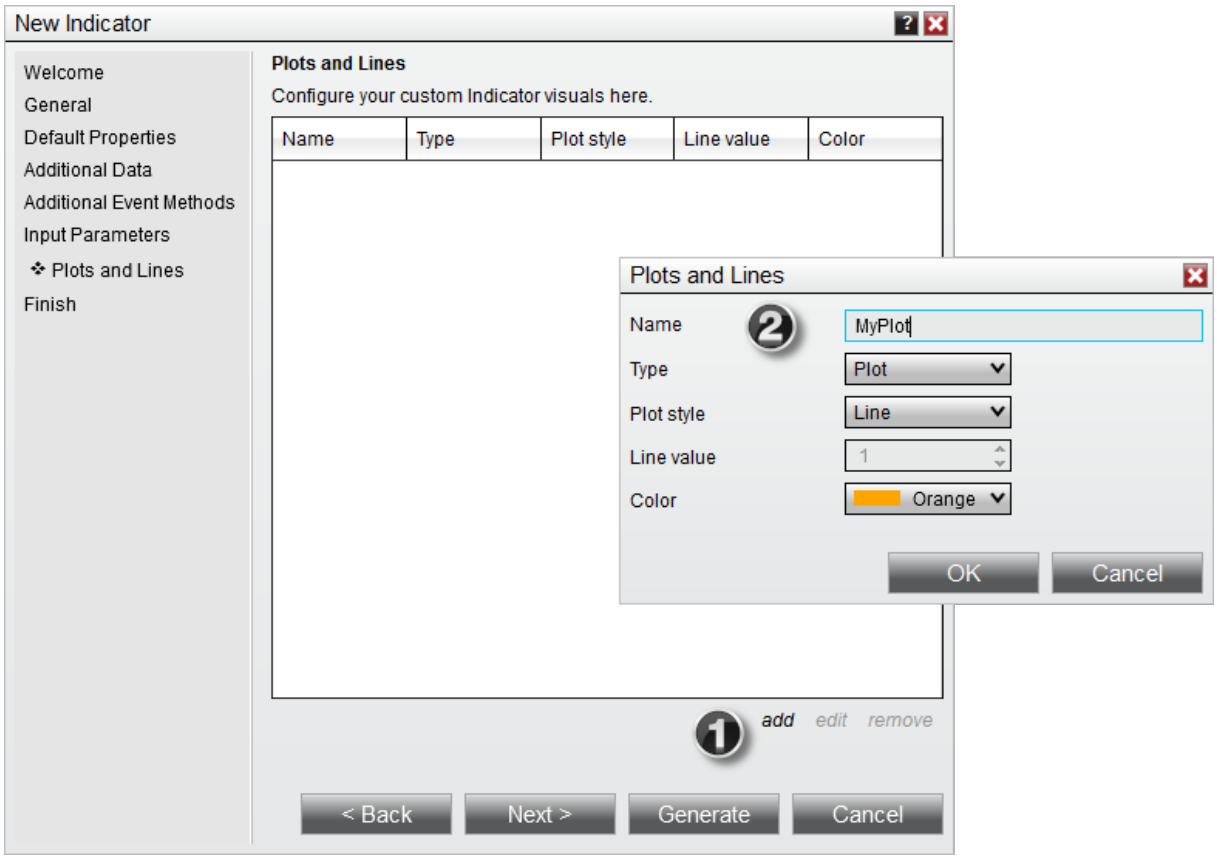


1. Clicking the **add** button on the "Input Parameters" page brings up the **Input Parameters** dialogue
2. The **Input Parameters** dialogue can be used to define user inputs

We specify a default value of 10, which will refer to 10 bars in the calculation. We also specify a minimum value of 1 to ensure that we cannot enter a 0 or negative number for `Periods`.

Defining Plots and lines

The next page will allow us to define plots and static lines for the indicator. For this indicator, we will define a single plot, called "MyPlot."



1. Clicking the **add** button on the "Plots and Lines" page brings up the **Plots and Lines** dialogue
2. The **Plots and Lines** dialogue can be used to define the plot

After this, click the **Finish** button, and the **Indicator Wizard** will generate a basic code structure implementing the parameters that you have set. You are now ready to move on to entering calculation logic in your code.

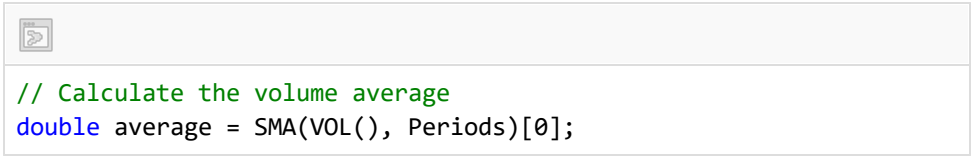
11.5.4.5.2 Entering Calculation Logic

The OnBarUpdate() method is called for each incoming tick, or on the close of a bar (if enabled) when performing real-time calculations, and is called on each bar of a Bars object when re-calculating the indicator (For example, an indicator would be re-calculated when adding it to an existing chart that has existing price data displayed). This is the main method used for indicator calculations, and we will calculate our core indicator logic (calculating an average of volume) within this method.

Calculating the Average

NinjaTrader has built in indicators that you can reference in your calculations. Since we are calculating a simple moving average of volume it would make sense for us to use the built in SMA indicator and Volume indicators.

Enter the following code into the OnBarUpdate() method in the NinjaScript Editor:

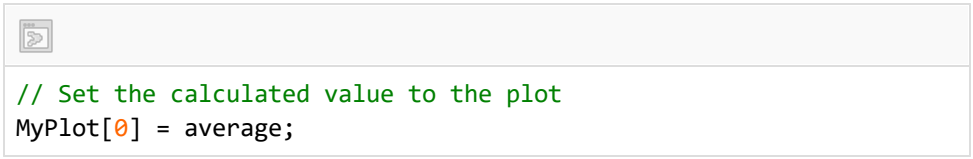


```
// Calculate the volume average
double average = SMA(VOL(), Periods)[0];
```

Here we declared the variable "average" which is of type double. This serves as the temporary storage for the current value of the simple moving average of volume. We then use the simple moving average indicator and pass in the volume indicator as its input, pass in our indicator "Periods" property (a parameter we defined in the wizard) and access the current value "[0]" that we will assign to our variable "average". If we wanted to assign the value one bar ago, we could have used "[1]".

Final Assignment

Enter the following code into the OnBarUpdate() method and below the code snippet you entered above:



```
// Set the calculated value to the plot
MyPlot[0] = average;
```

Here we assign the "average" value to the property that represents the plot data using the '=' assignment operator. We have just finished coding our simple moving average of volume. Your class code should look identical to the code below. You are now ready to compile the indicator and configure it on a chart.

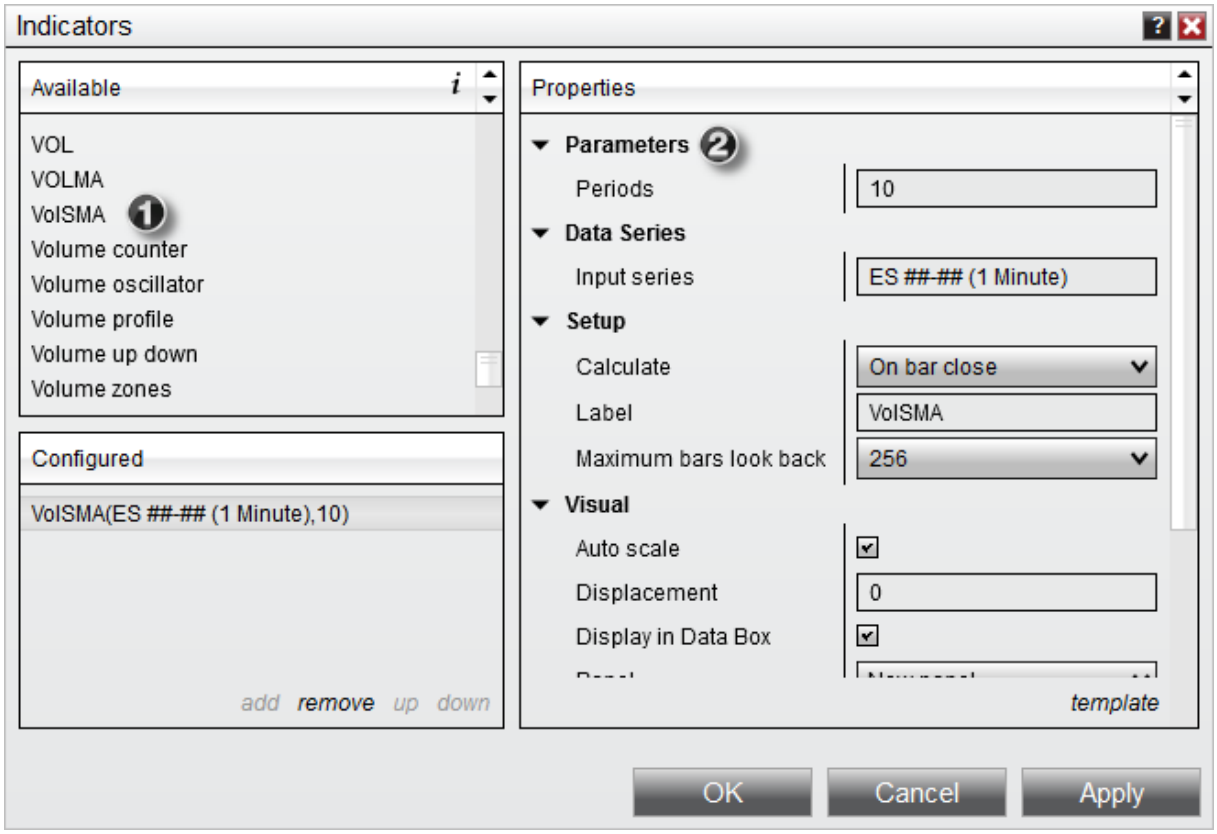
```
public class VolSMA : Indicator
{
   protected override void OnStateChange()
   {
      if (State == State.SetDefaults)
      {
         Description                                 = @"Moving average
of volume";
         Name                                        = "VolSMA";
         Calculate                                   =
Calculate.OnBarClose;
         IsOverlay                                   = false;
         DisplayInDataBox                            = true;
         DrawOnPricePanel                            = true;
         DrawHorizontalGridLines                     = true;
         DrawVerticalGridLines                       = true;
         PaintPriceMarkers                           = true;
         ScaleJustification                          =
NinjaTrader.Gui.Chart.ScaleJustification.Right;
         //Disable this property if your indicator requires custom
values that cumulate with each new market data event.
         //See Help Guide for additional information.
         IsSuspendedWhileInactive                    = true;
         Periods                                     = 10;
         AddPlot(Brushes.Orange, "MyPlot");
      }
      else if (State == State.Configure)
      {
      }
   }

   protected override void OnBarUpdate()
   {
      // Calculate the volume average
      double average = SMA(VOL(), Periods)[0];

      // Set the calculated value to the plot
      MyPlot[0] = average;
   }

   #region Properties
   [NinjaScriptProperty]
   [Range(1, int.MaxValue)]
   [Display(Name="Periods", Description="Number of periods",
Order=1, GroupName="Parameters")]
   public int Periods
   { get; set; }

   [Browsable(false)]
   [XmlIgnore]
   public Series<double> MyPlot
   {
      get { return Values[0]; }
```

11.5.4.5.3 Compiling

The indicator code is now complete and needs to be compiled. You can compile this indicator from within the NinjaScript Editor right mouse button menu "Compile" menu or simply press the F5 key. It is important to understand that this process makes the indicator ready for real-time use and will run natively within NinjaTrader directly. It does not run interpreted as many other applications do. This provides you with the highest performance possible. If there are any errors reported during compiling, the error messages will be displayed at the bottom of the NinjaScript Editor.

11.5.4.5.4 Using

Your indicator is now ready for use and will be listed in the Indicator Dialog window.



1) The indicator can now be found in the "Available" section of the **Indicators** window
2) Once added to the "Configured" section, our user-defined inputs can be entered along with standard indicator properties.

Once applied to a chart, the indicator should look something like the image below.

11.5.4.6 Beginner - Using price variables

Using Price Variables Overview

In this beginner level tutorial we are going to build a custom indicator that searches for a candlestick formation in which the closing price of a specified bar is greater than the closing price of the bar before it. This indicator will show you how to access price variables and use a conditional operator.

- Set Up
- Entering Calculation Logic
- Compiling
- Using

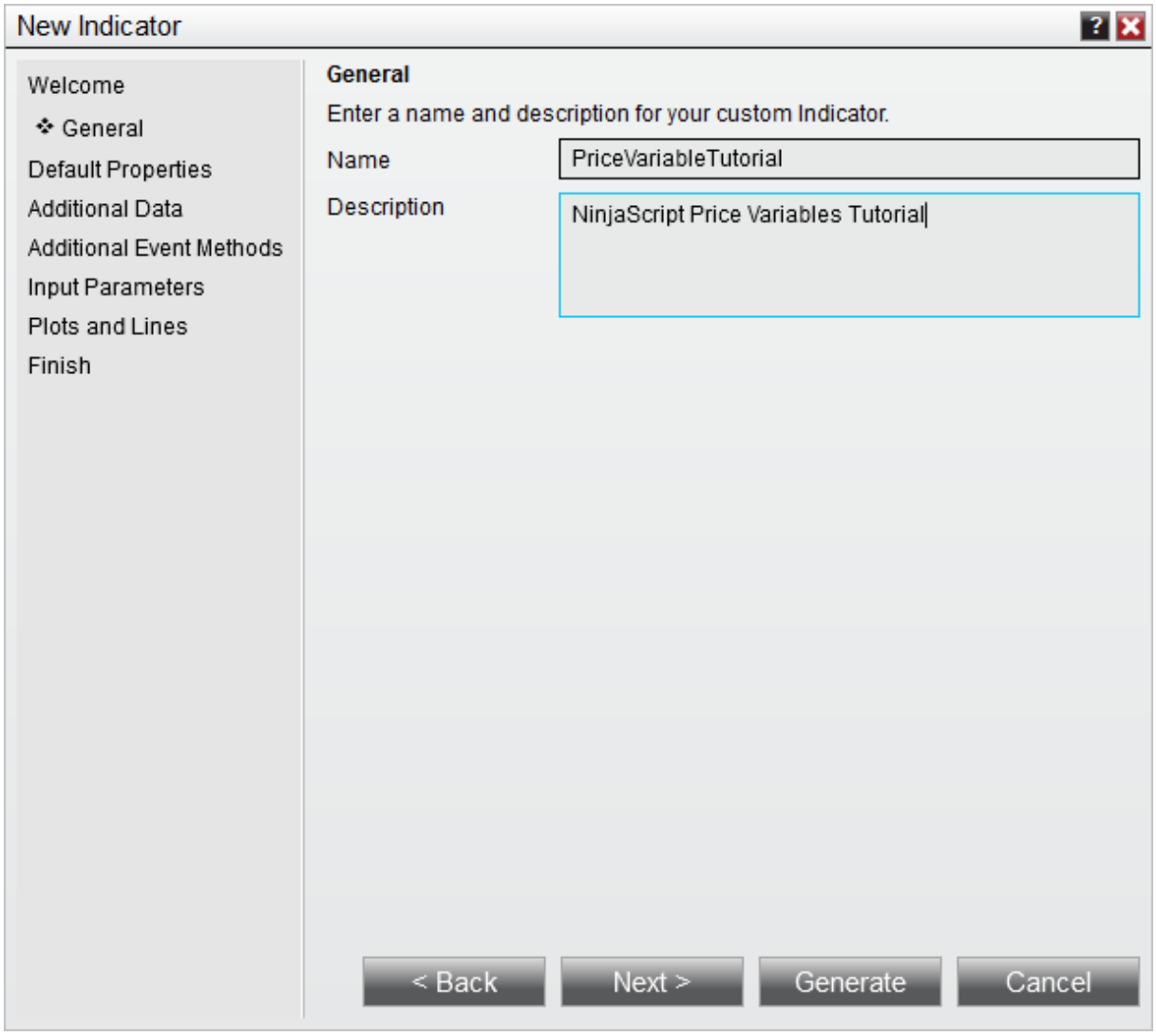
11.5.4.6.1 Set Up

The first step in creating a custom indicator is to use the custom indicator wizard. The wizard will generate the required NinjaScript code that will serve as the foundation for your custom indicator.

1. Within the NinjaTrader Control Center, select the **New** menu, then select the **NinjaScript Editor** menu item.
2. Right mouse click the "Indicators" folder in the **NinjaScript Explorer** section, then select the **New Indicator** menu item to open the **New Indicator Wizard**.

Defining Indicator Properties and Name

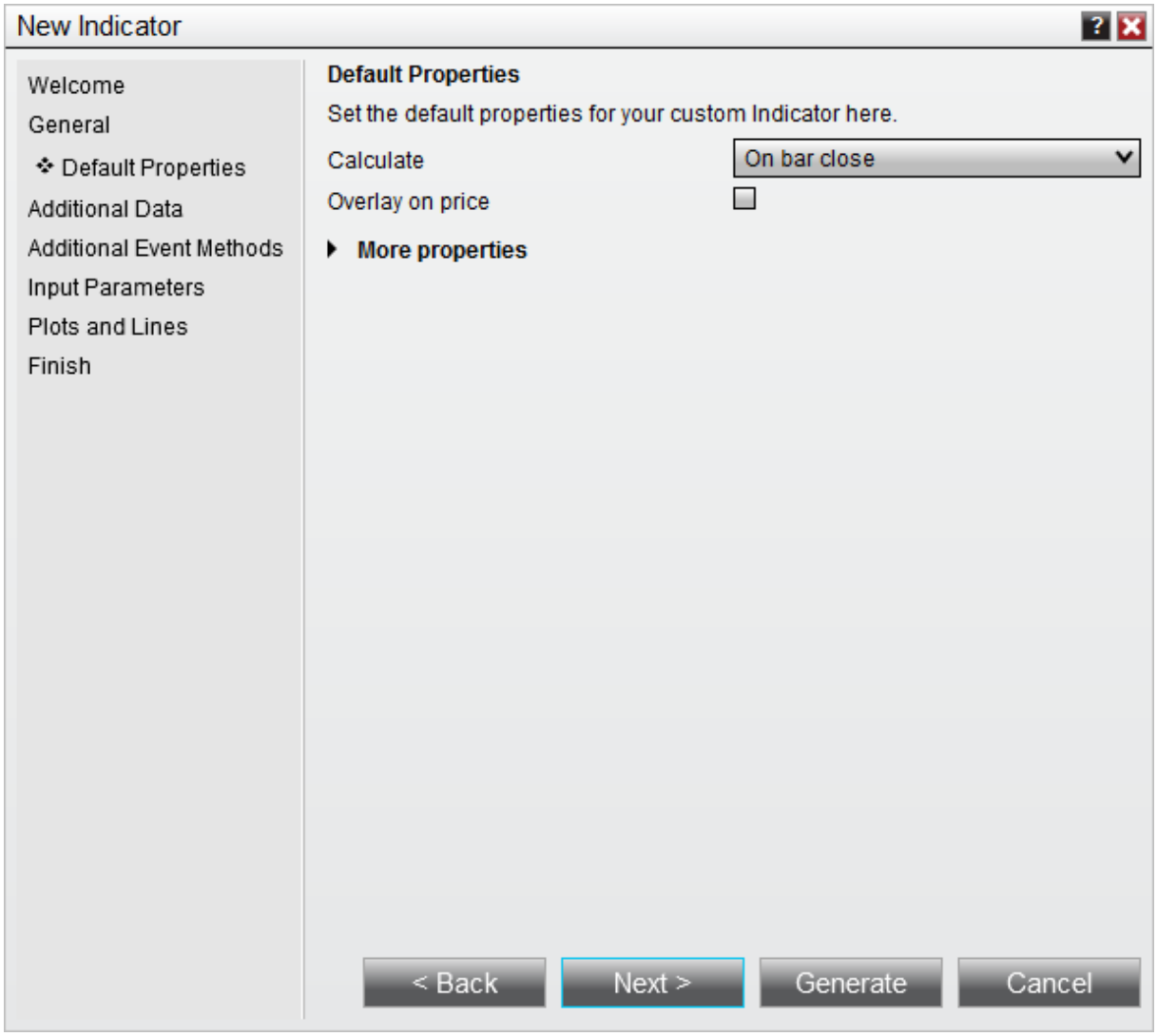
First you will define your indicator's name and several indicator properties. Begin by clicking the **Next >** button on the first page of the wizard to view the page shown below.



3. Enter the information as shown above
4. Click the **Next >** button

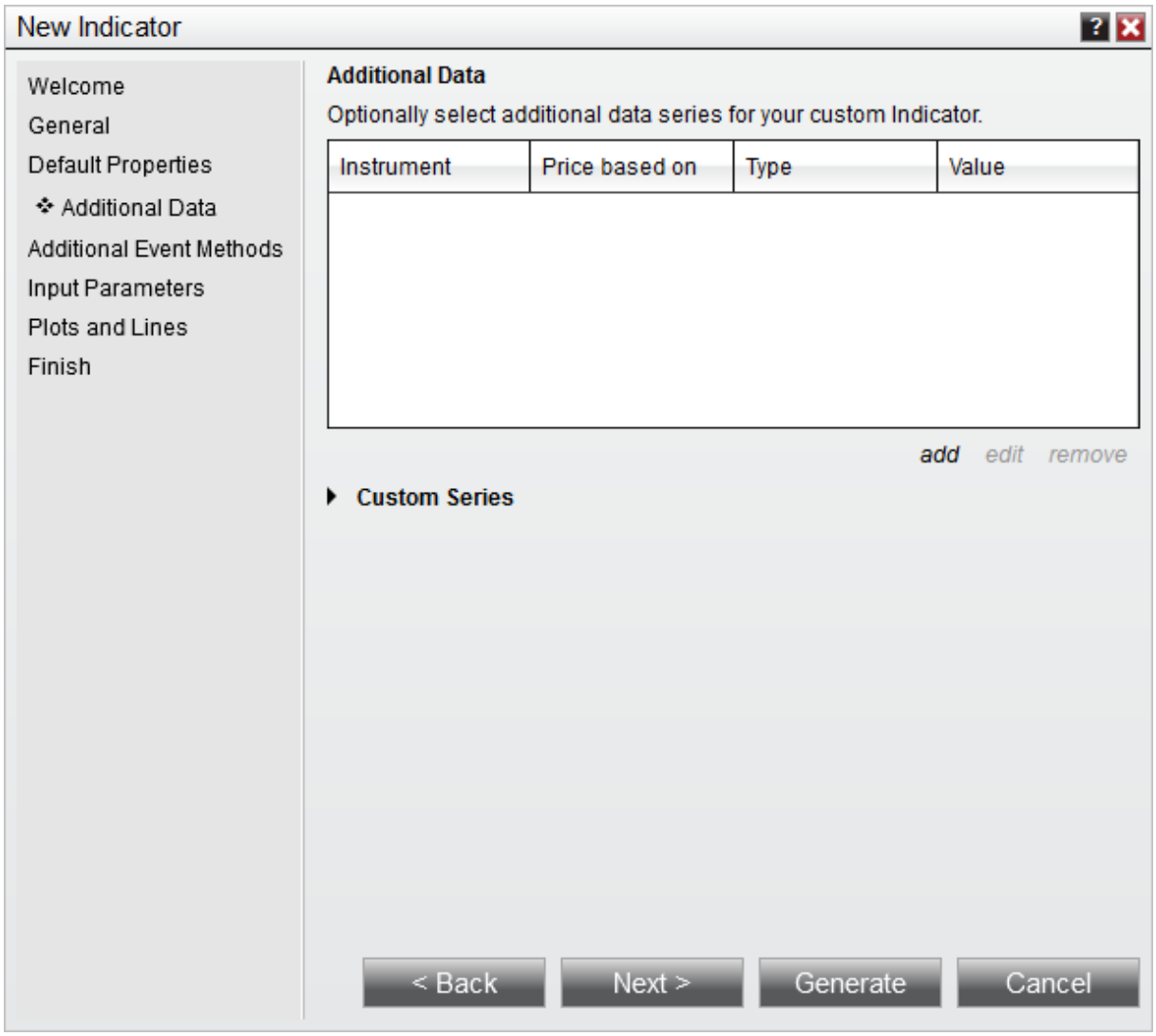
Setting Default Properties

The next page will allow you to set defaults for basic properties related to your indicator, including it's Calculate and Overlay settings. Click the **More Properties** button to expose additional properties. For this tutorial, we will not change any basic properties' defaults, and instead will leave them all set to the values shown below and move forward by clicking the **Next >** button.:



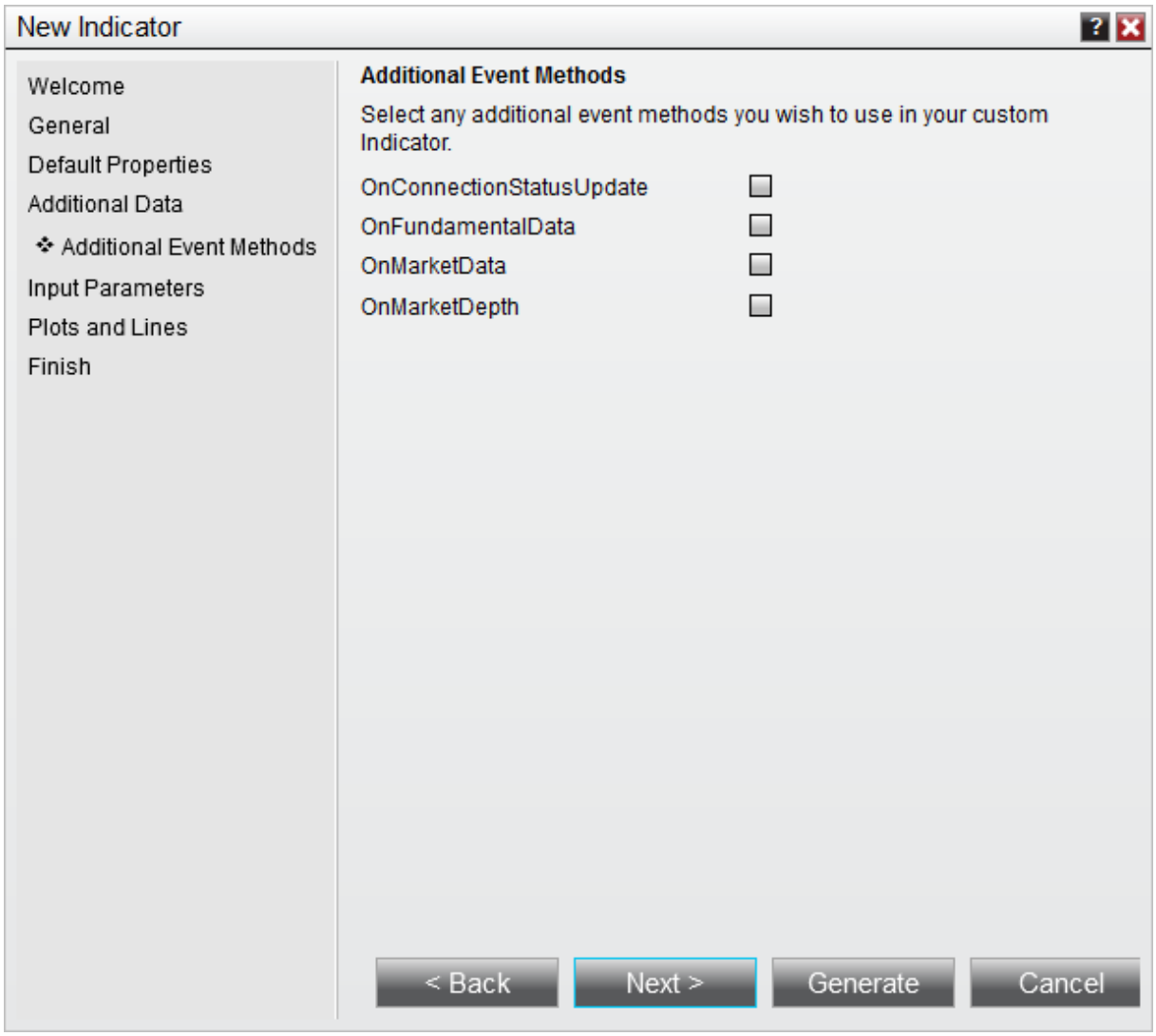
Adding Additional Data

The next page will allow you to configure one or more additional Bars objects for use by the indicator. For our purposes, we will leave this page blank and move forward by clicking the **Next >** button.



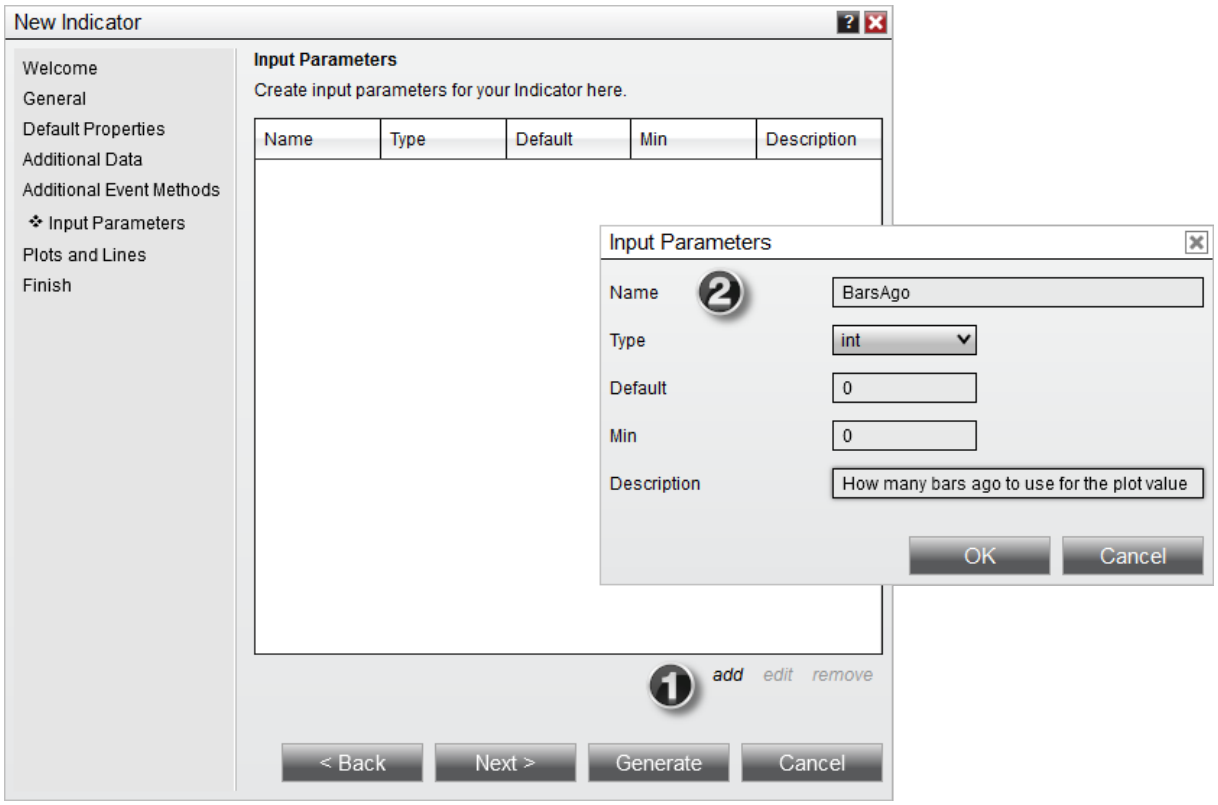
Adding Event Methods

The next page will allow you to pre-populate certain event methods into the NinjaScript code generated by the wizard. For our purposes, we will leave all of the checkboxes corresponding to different event methods unchecked, and will move on by clicking the **Next >** button.



Defining Input Parameters

The next page will allow us to configure user input parameters for the indicator. For our custom indicator, our eventual goal will be to create a simple plot that follows either above or below the bars based upon the Close price of a specified bar compared to the preceding bar. To allow for the variable selection of a number of bars ago, we will create one input parameter and call it "BarsAgo." This variable will then be used in place of a number when specifying which bar's Close price to use for the indicator's condition.

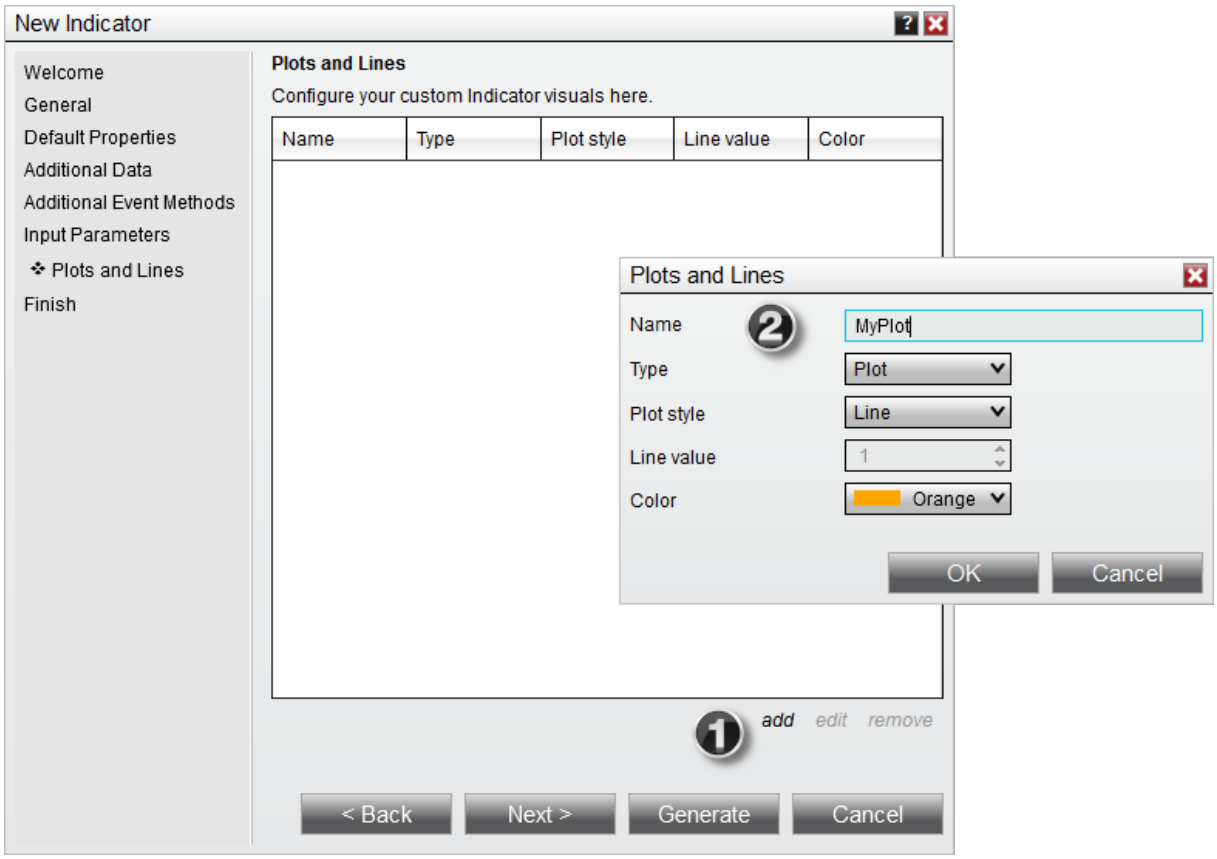


1. Clicking the **add** button on the "Input Parameters" page brings up the **Input Parameters** dialogue
2. The **Input Parameters** dialogue can be used to define user inputs

We specify a default value of 0, which will refer to "zero bars ago," or the current bar. We also specify a minimum value of 0 to ensure that we cannot enter a negative number for `BarsAgo`.

Defining Plots and lines

The next page will allow us to define plots and static lines for the indicator. For this indicator, we will define a single plot, called "MyPlot."



1. Clicking the **add** button on the "Plots and Lines" page brings up the **Plots and Lines** dialogue
2. The **Plots and Lines** dialogue can be used to define the plot

After this, click the **Finish** button, and the **Indicator Wizard** will generate a basic code structure implementing the parameters that you have set. You are now ready to move on to entering calculation logic in your code.

11.5.4.6.2 Entering Calculation Logic

The OnBarUpdate() method is called for each incoming tick, or on the close of a bar (if enabled) when performing real-time calculations, and is called on each bar of a Bars object when re-calculating the indicator (For example, an indicator would be re-calculated when adding it to an existing chart that has existing price data displayed). This is the main method used for indicator calculations, and we will calculate our core indicator logic (testing to see if a Close price on a specified bar was greater than the previous Close price) within this method.

Adding the Condition and Assigning the Plot Value

Enter the following code in the OnBarUpdate() method in the NinjaScript Editor:

```
Values[0][BarsAgo] = (Close[BarsAgo] > Close[(BarsAgo + 1)]) ?
(High[BarsAgo] + (5 * TickSize)) : (Low[BarsAgo] - (5 * TickSize));
```

Although the code above fits on a single line, it is doing several things. Firstly, it is important to understand the structure that we are using in this statement. We are using a Ternary Operator, which provides a way to assign one of two values to a variable based on a condition. We begin by stating that we wish to assign a value to the indicator plot at a bar index corresponding to BarsAgo. We do this by using Values, which is a collection holding values for all plots configured in the indicator:

```
Values[0][BarsAgo] =
```

Next, we add a condition to test. In this case, we are testing to see whether Close at a bar index corresponding to the value of `BarsAgo` was greater than `Close` at a value of `BarsAgo + 1`. If BarsAgo was set to 5, for example, this would compare `Close[5]` to `Close[6]`:

```
Values[0][BarsAgo] = (Close[BarsAgo] > Close[(BarsAgo + 1)]) ?
```

If the condition evaluates to `true`, then the first expression will be run (the expression on the left side of the colon ":"), which will assign the value of the indicator plot to the High price of the specified bar, plus five ticks. We obtain the tick size value for the configured instrument via the TickSize property:

```
Values[0][BarsAgo] = (Close[BarsAgo] > Close[(BarsAgo + 1)]) ?
(High[BarsAgo] + (5 * TickSize)) :
```

if the condition evaluates to `false`, then the second expression will be run (the expression on the right side of the colon ":", which will assign the value of the indicator plot to the Low price of the specified bar, less five ticks:

```
Values[0][BarsAgo] = (Close[BarsAgo] > Close[(BarsAgo + 1)]) ?
(High[BarsAgo] + (5 * TickSize)) : (Low[BarsAgo] - (5 * TickSize));
```

The core indicator logic is now in place, but running this code as it is can result in an "Index out of range" exception. Since we are looking a certain number of bars back in time, we need to make sure that there are always enough bars on the chart for us to look back. For example, if BarsAgo were set to 5, then we would be comparing the value of five bars ago to the value of six bars ago, but on Bars # 1, 2, 3, 4, or 5, at which point we do not have five or six bars to look back, the indicator will cause an error. To resolve this, we will add a condition which will prevent the core calculations from running unless we know there are enough bars on the chart. Add the following line just above the line you have been working on throughout this page:

```
if(CurrentBar < BarsAgo + 1)
    return;
```

This line says, "if there is not a number of bars equal to one number greater than the value of BarsAgo, then exit `OnBarUpdate()`."

Now that everything is in place, your class code should look as below. You are now ready to compile the indicator and configure it on a chart.

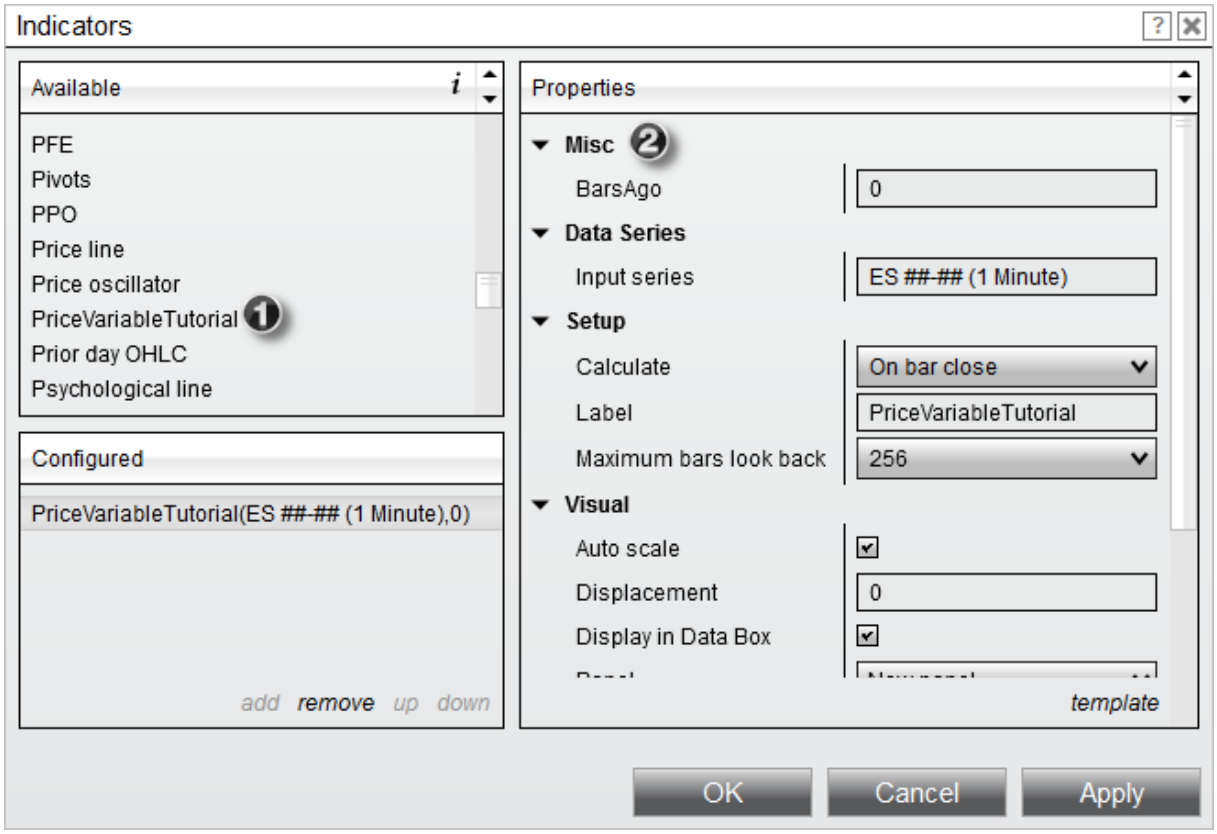
```
public class PriceVariableTutorial : Indicator
{
    protected override void OnStateChange()
    {
        if (State == State.SetDefaults)
        {
            Description                 = @"NinjaScript Price
Variables Tutorial";
            Name                        = "PriceVariableTutorial";
            Calculate                   = Calculate.OnBarClose;
            IsOverlay                   = false;
            DisplayInDataBox            = true;
            DrawOnPricePanel            = true;
            DrawHorizontalGridLines     = true;
            DrawVerticalGridLines       = true;
            PaintPriceMarkers           = true;
            ScaleJustification          =
NinjaTrader.Gui.Chart.ScaleJustification.Right;
            //Disable this property if your indicator requires
custom values that cumulate with each new market data event.
            //See Help Guide for additional information.
            IsSuspendedWhileInactive    = true;
            BarsAgo                 = 0;
            AddPlot(Brushes.Orange, "MyPlot");
        }
        else if (State == State.Configure)
        {
        }
    }

    protected override void OnBarUpdate()
    {
        if(CurrentBar < BarsAgo + 1)
            return;

        Values[0][BarsAgo] = (Close[BarsAgo] > Close[(BarsAgo +
1)]) ? (High[BarsAgo] + (5 * TickSize)) : (Low[BarsAgo] - (5 *
TickSize));
    }

    #region Properties
    [Range(0, int.MaxValue)]
    [NinjaScriptProperty]
    [Display(Name="BarsAgo", Description="How many bars ago to use
for the plot value", Order=1)]
    public int BarsAgo
    { get; set; }

    [Browsable(false)]
    [XmlIgnore]
    public Series<double> MyPlot
    {
        get { return Values[0]; }
```

11.5.4.6.3 Compiling

The indicator code is now complete and needs to be compiled. You can compile this indicator by selecting the **Compile** menu item from within the NinjaScript Editor Right Click, by clicking the Compile icon on the toolbar at the top of the window, or by pressing the F5 key on your keyboard. It is important to understand that this process makes the indicator ready for real-time use natively within NinjaTrader. It does not run as interpreted code, as many other applications do, but rather as a C# assembly. This provides you with the highest performance possible. If there are any errors reported during compiling, the error messages will be displayed at the bottom of the NinjaScript Editor.

11.5.4.6.4 Using

Your indicator is now ready for use and will be listed in the Indicator Dialog window.



1) The indicator can now be found in the "Available" section of the **Indicators** window
2) Once added to the "Configured" section, our user-defined inputs can be entered along with standard indicator properties.

Once applied to a chart, the indicator should look something like the image below.

11.5.4.7 Developing Outside of the NinjaScript Editor

The NinjaScript Editor automatically generates required program code on saving and/or compiling a custom indicator. If you choose to develop custom indicators outside of the NinjaScript Editor environment, please ensure that you use the NinjaScript Editor to compile.

Please see the additional information on this subject.

11.5.5 Developing Strategies

NinjaScript allows you to develop custom strategies in rapid time by using over 100 built-in system indicators, 3rd party indicators or custom indicators. NinjaScript strategies are compiled and run natively within the NinjaScript application providing the highest performance possible.

Please take the time to review this section including the Strategy Development Process.

Prior to running strategies live, please be sure to review the sections about Strategy Position vs. Account Position and Syncing Account Positions.

Tutorial Descriptions

All internal NinjaScript indicators and sample strategies come with full source code and can be viewed within the NinjaScript Editor. Please review the tutorials within this section for detailed walk throughs of custom strategy development.

- Level 1 - Simple MA Cross Over (Demonstrates strategy construction by wizard and scripting)
- Level 2 - RSI with Stop Loss & Profit Target (scripting only)

11.5.5.1 Intermediate - RSI with Stop Loss & Profit Target

RSI with Stop Loss & Profit Target Overview

In this intermediate level tutorial we are going to build a custom automated strategy that goes long when RSI crosses above 20 and exits at a predefined stop loss or profit target, whichever is hit first. This tutorial is demonstrates the use of the NinjaScript Editor.

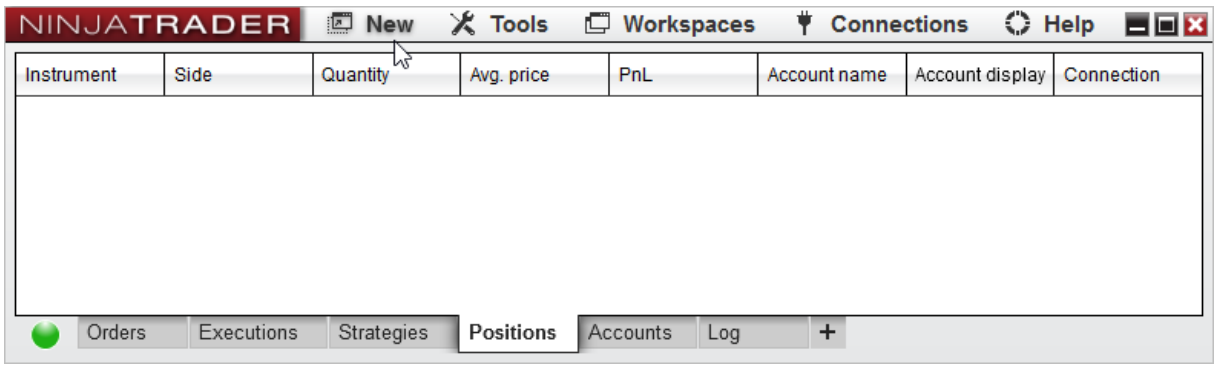
- Set Up
- Entering Strategy Logic
- Compiling

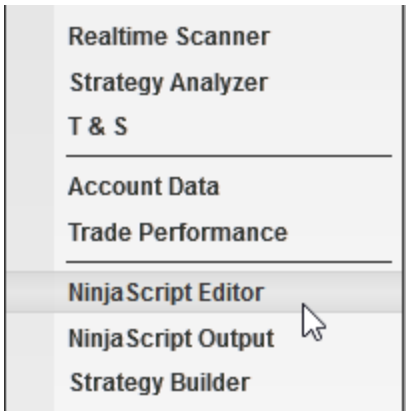
11.5.5.1.1 Set Up

Our first tutorial covered using the Strategy Builder to create simple NinjaScript strategies or to build the framework needed for a more complex strategy.

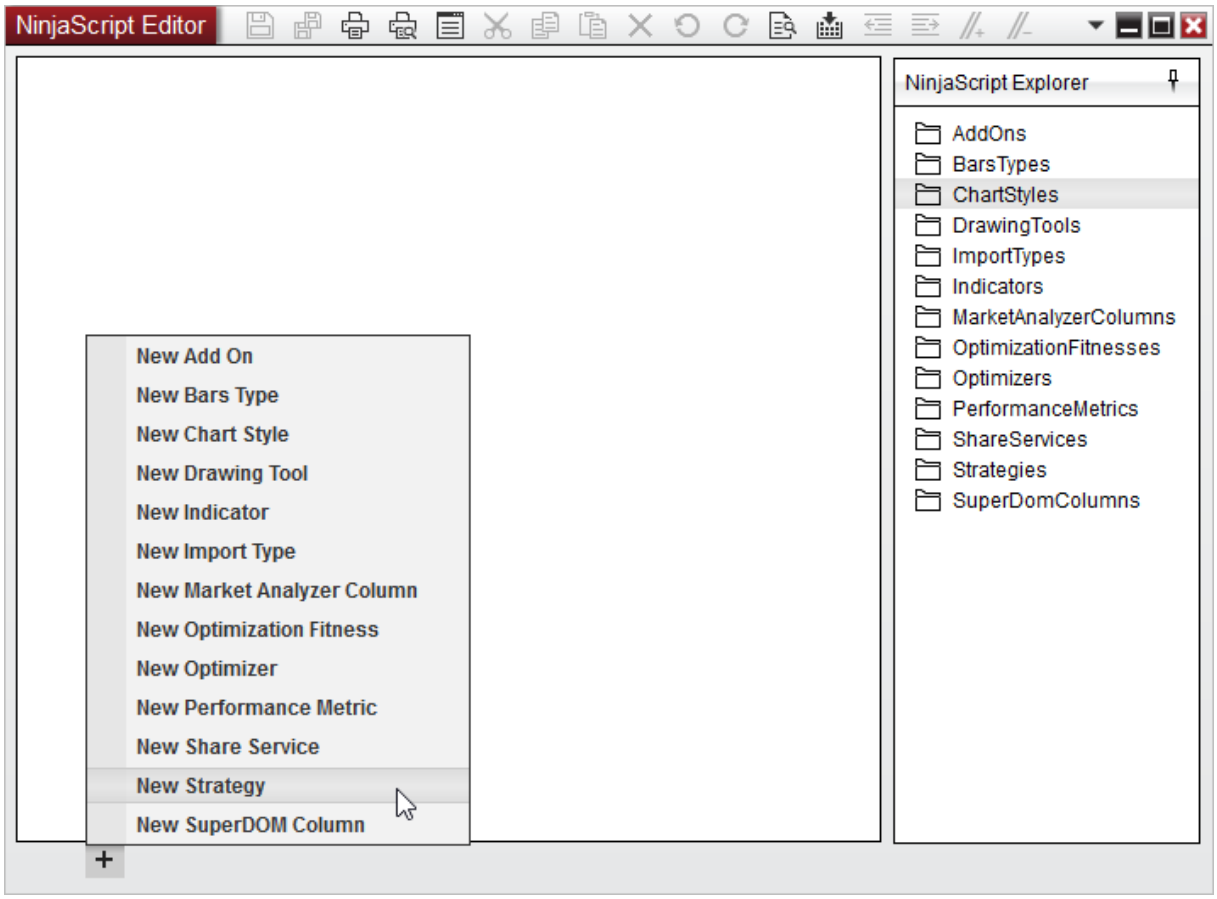
This tutorial will cover another approach, using the NinjaScript Editor and New Strategy Wizard.

1. Within the NinjaTrader Control Center window select the New NinjaScript Editor... menu item

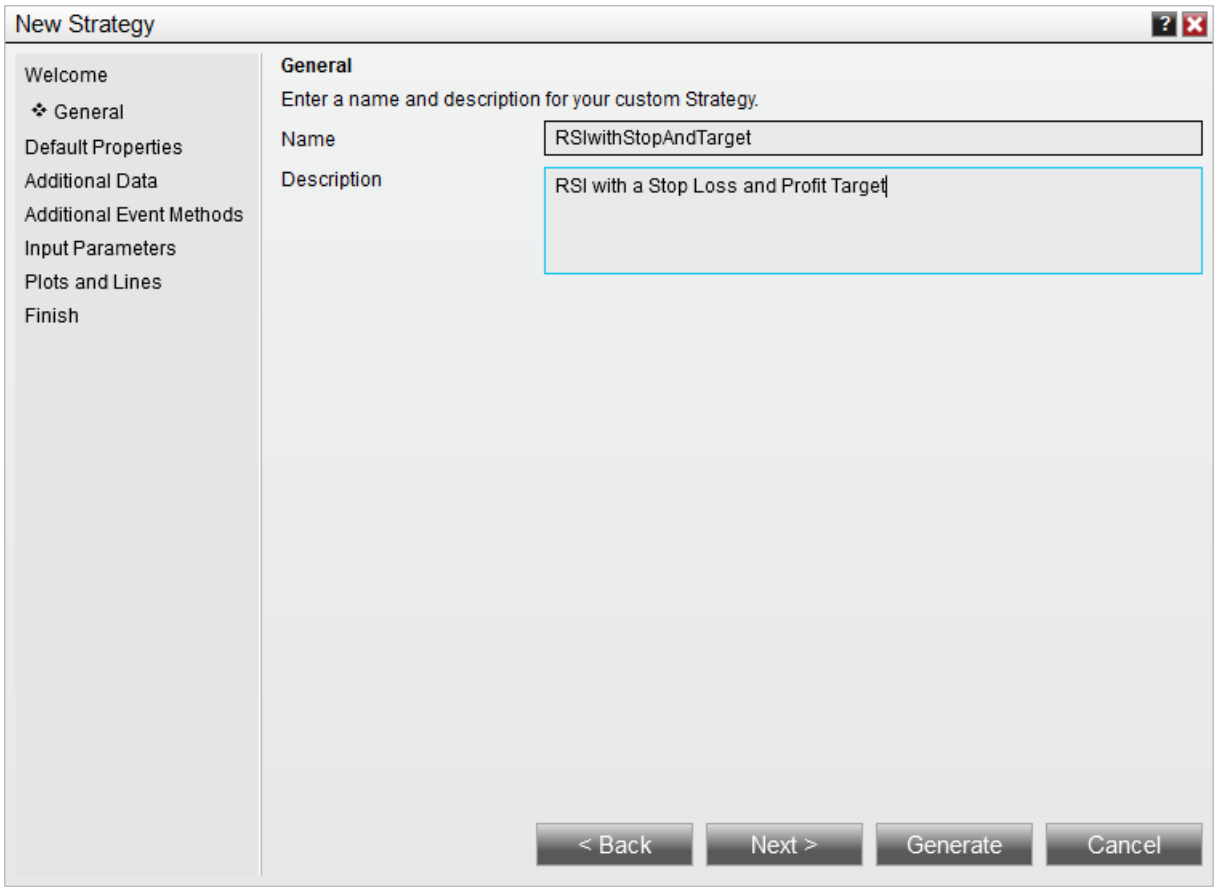




2. Click the **"+"** tab in the lower left, and select **New Strategy** to open a New Strategy Wizard



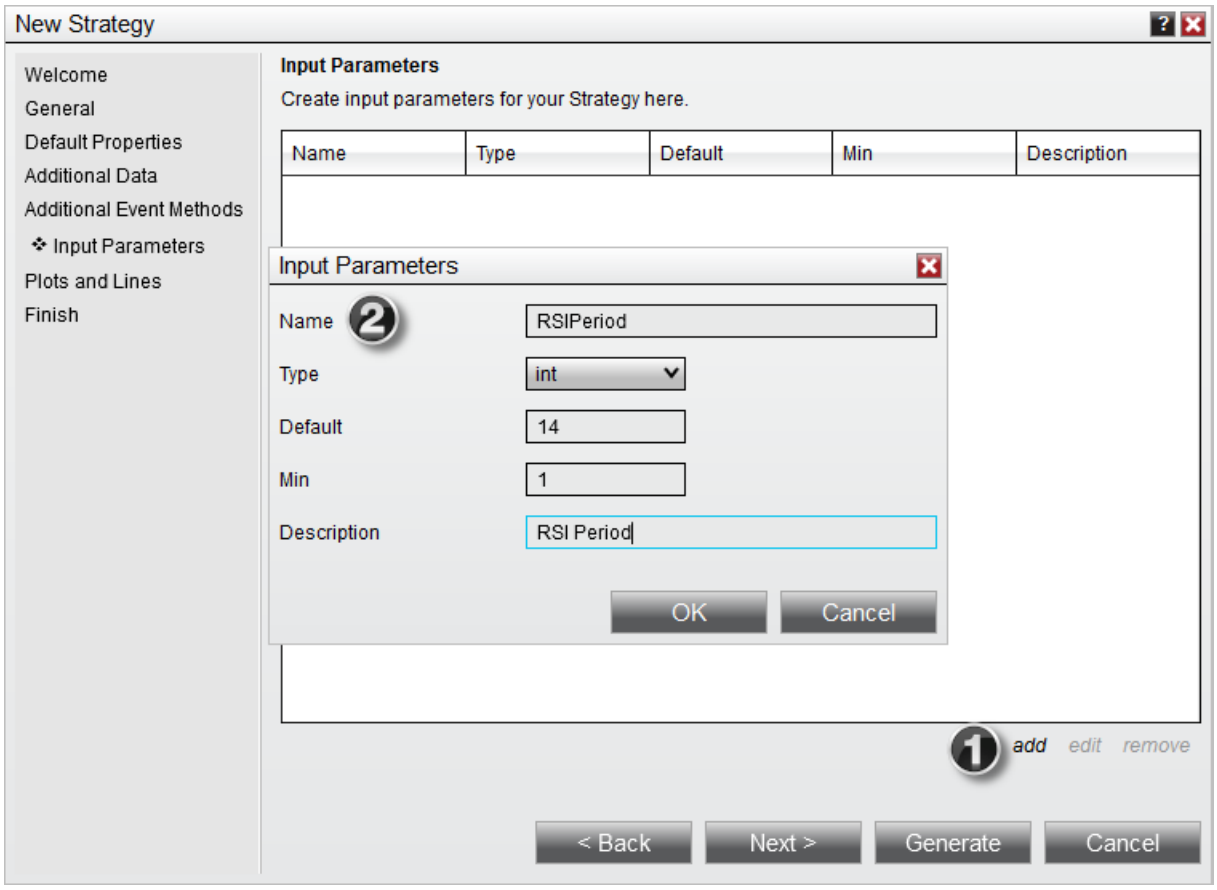
3. Enter the information as shown below
4. Press the "Next >" button until we are at the Inputs and Parameters page



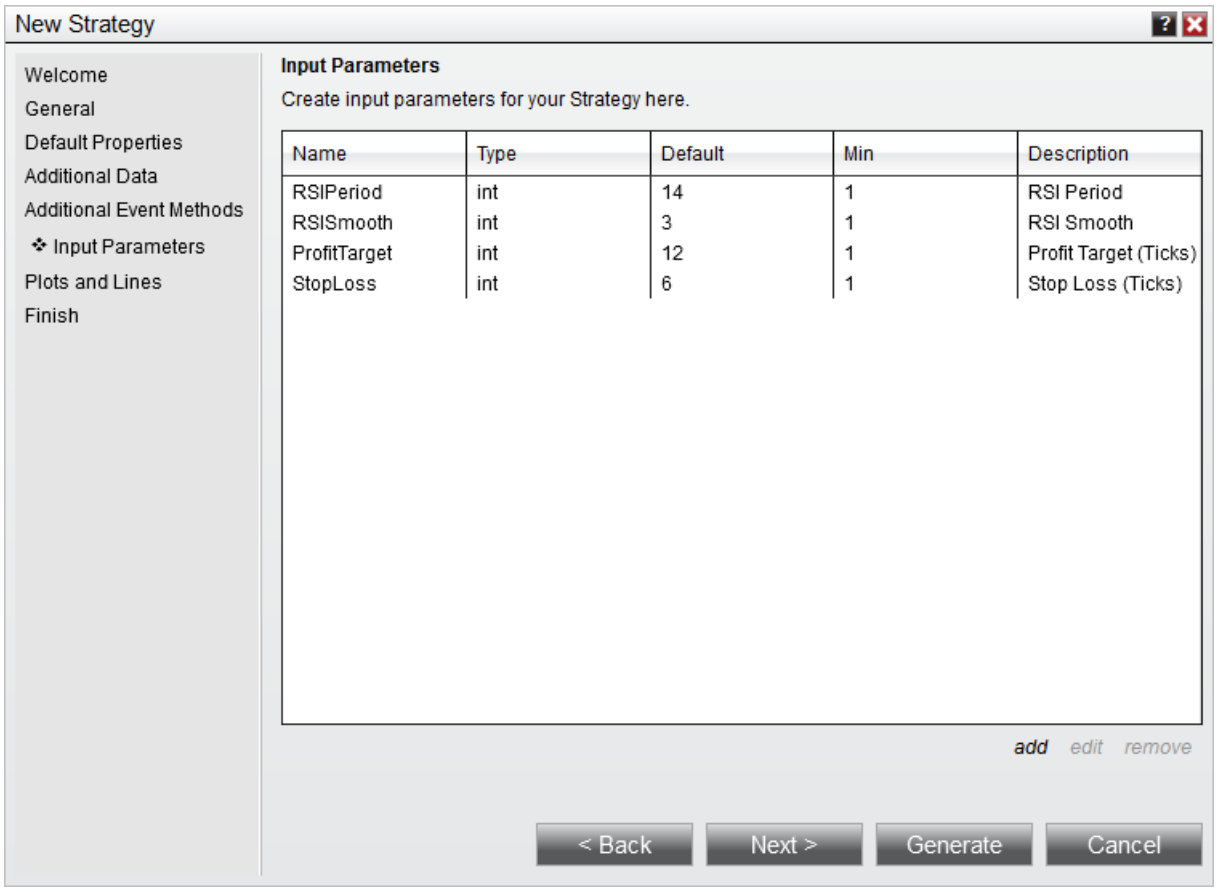
Defining Input Parameters

Below you will define your strategy's input parameters. These are any input parameters that can be changed by the user when running or backtesting a strategy. If your strategy does not require any parameters leave the "Name" fields blank.

5. Click the add button to create a User Input Parameter (See item 1 in the screenshot below)
6. Fill out the Input Parameters window and click OK to create the input parameter (See item 2 in the screenshot below)



7. Add the inputs as per the image below



8. Press the "Generate" button to generate the code in the NinjaScript Editor.

You are now ready to continue to the Entering Strategy Logic page of this tutorial.

11.5.5.1.2 Entering Strategy Logic

Using the OnStateChange() Method to Configure the Strategy

The OnStateChange() method is called once prior to running a strategy and can be used to set properties or call methods in preparation for running a strategy.

Enter the code contained within the OnStateChange() method in the image below into the OnStateChange() method when we are in the `State.DataLoaded` state in the NinjaScript Editor.

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Description                                 = @"RSI with a Stop
Loss and Profit Target";
      Name                                        =
"RSIwithStopAndTarget";
      Calculate                                   =
Calculate.OnBarClose;
      EntriesPerDirection                         = 1;
      EntryHandling                               =
EntryHandling.AllEntries;
      IsExitOnSessionCloseStrategy                = true;
      ExitOnSessionCloseSeconds                   = 30;
      IsFillLimitOnTouch                          = false;
      MaximumBarsLookBack                         =
MaximumBarsLookBack.TwoHundredFiftySix;
      OrderFillResolution                         =
OrderFillResolution.Standard;
      Slippage                                    = 0;
      StartBehavior                               =
StartBehavior.WaitUntilFlat;
      TimeInForce                                 = TimeInForce.Gtc;
      TraceOrders                                 = false;
      RealtimeErrorHandling                       =
RealtimeErrorHandling.StopCancelClose;
      StopTargetHandling                          =
StopTargetHandling.PerEntryExecution;
      BarsRequiredToTrade                         = 20;
      // Disable this property for performance gains in Strategy
Analyzer optimizations
      // See the Help Guide for additional information
      IsInstantiatedOnEachOptimizationIteration = true;
      RSIPeriod                                   = 14;
      RSISmooth                                   = 3;
      ProfitTarget                                = 12;
      StopLoss                                    = 6;
   }
   else if (State == State.DataLoaded)
   {
      AddChartIndicator(RSI(RSIPeriod, RSISmooth));

      SetStopLoss(CalculationMode.Ticks, StopLoss);
      SetProfitTarget(CalculationMode.Ticks, ProfitTarget);
   }
}
```

For more information on the strategy properties added in State.SetDefaults, please see our complete Strategy documentation.
The AddChartIndicator() method is called and the RSI() indicator method is passed in which will automatically plot this indicator on a chart when the strategy runs.

The method signature for the RSI() indicator is:

```
RSI(int period, int smooth);
```

It is valid to have used the Add() method in the following manner:

```
AddChartIndicator(RSI(14, 3));
```

However, instead of hard coding the period value to 14 and the smooth value to 3, we substituted the values for the user defined inputs we defined in the wizard. Calling the Add() method in the following manner:

```
AddChartIndicator(RSI(RSIPeriod, RSISmooth));
```

Allows us to change the period and smooth parameters of the embedded RSI indicator in the strategy at run time. This gives us a higher level of flexibility when working with our strategy.

SetStopLoss() and SetProfitTarget() are called with CalculationMode.Ticks. This means that when a position is opened, the strategy will immediately submit a stop and target order with a price that is calculated based on the StopLoss and ProfitTarget parameters passed in offset from the positions average entry price.

Using the OnBarUpdate() Method for the Core Strategy Logic

The OnBarUpdate() method is called for each incoming tick or on the close of a bar (user defined) when performing real-time calculations. Therefore, this is the main method called for strategy calculation and we will use this method to enter the script that check for entry and exit conditions.

Enter the code contained within the OnBarUpdate() method in the image below into the OnBarUpdate() method in the NinjaScript Editor:

```
protected override void OnBarUpdate()
{
   if (CurrentBar < RSIPeriod)
      return;

   if(CrossAbove(RSI(RSIPeriod, RSISmooth), 20, 1))
      EnterLong();
}
```

Since our strategy exit logic has already been set up in the OnStateChange() method above, we only need to focus on expressing our entry logic. The strategy entry logic is very straight forward and can be translated to English:

if we have not seen the number of bars specified by the user defined input "RSIPeriod" then do not go any further

*if RSI crosses **above** a value of 20 within the last bar, go long*

To accomplish this we used the following methods and properties:

CurrentBar - A value representing the current bar being processed (think of a chart where the left most bar would be equal to one)
CrossAbove() - Checks for a cross above condition and returns true or false
RSI() - Returns the value of the RSI indicator
EnterLong() - Enters a market order long

11.5.5.1.3 Compiling

The strategy code is now complete and needs to be compiled.

- If you completed this tutorial via the **Strategy Wizard**, simply follow the wizard instructions to the end, at which point the strategy will compile.
- If you self coded this tutorial, you can compile this strategy from within the NinjaScript Editor right click menu by selecting the **Compile** menu item, or by pressing the F5 key.

It is important to understand that this process makes the strategy ready for real-time use and will run natively within NinjaTrader directly. It does not run interpreted as many other applications do. This provides you with the highest performance possible. If there are any errors reported during compiling, the error messages will be displayed at the bottom of the NinjaScript Editor.

11.5.5.2 Beginner - Simple MA Cross Over

Simple MA Crossover Overview

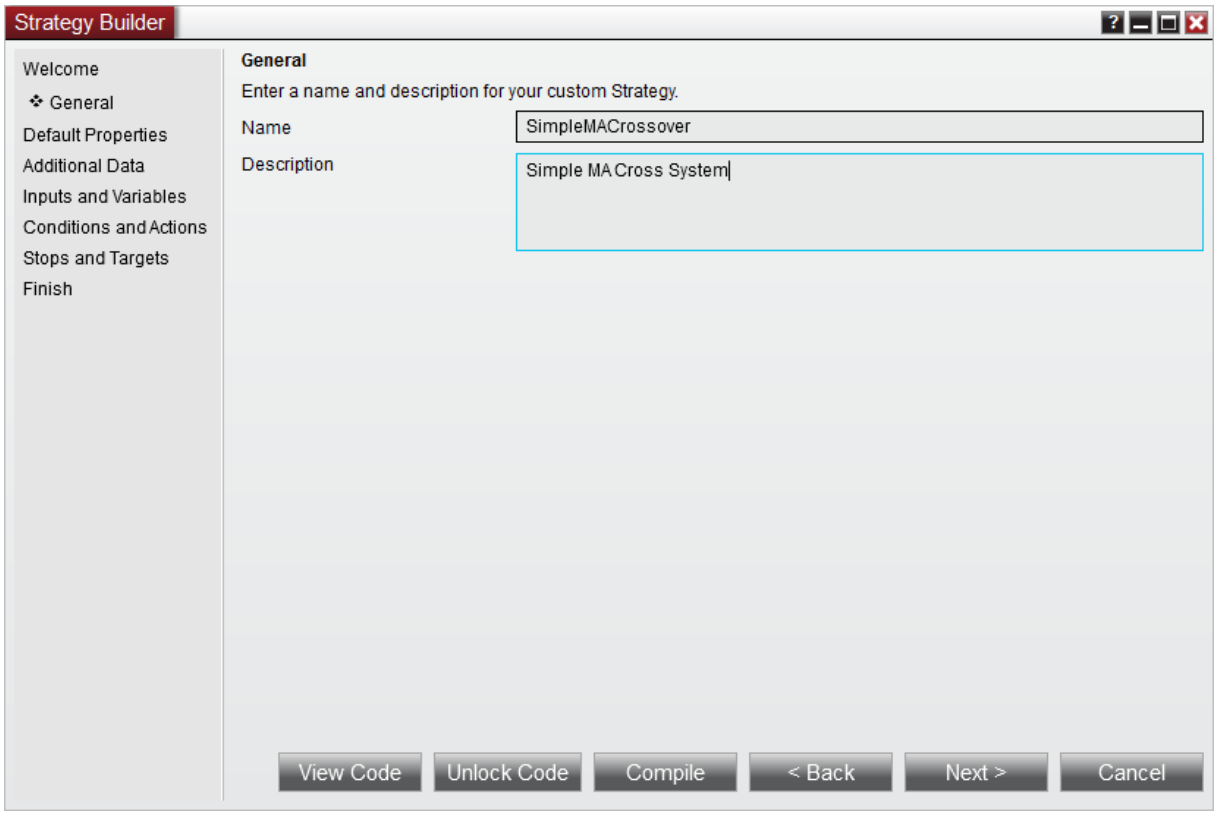
In this beginner level tutorial we are going to build a custom automated strategy that goes long when the fast moving average crosses above the slow moving average and goes short when the fast moving average crosses below the slow moving average.

11.5.5.2.1 Set Up

The first step in creating a custom strategy is to use the custom Strategy Builder. The builder provides two options:

- Allow you to create a functional strategy without any programming
- Generate the required NinjaScript code that will serve as the foundation for your custom strategy for further coding

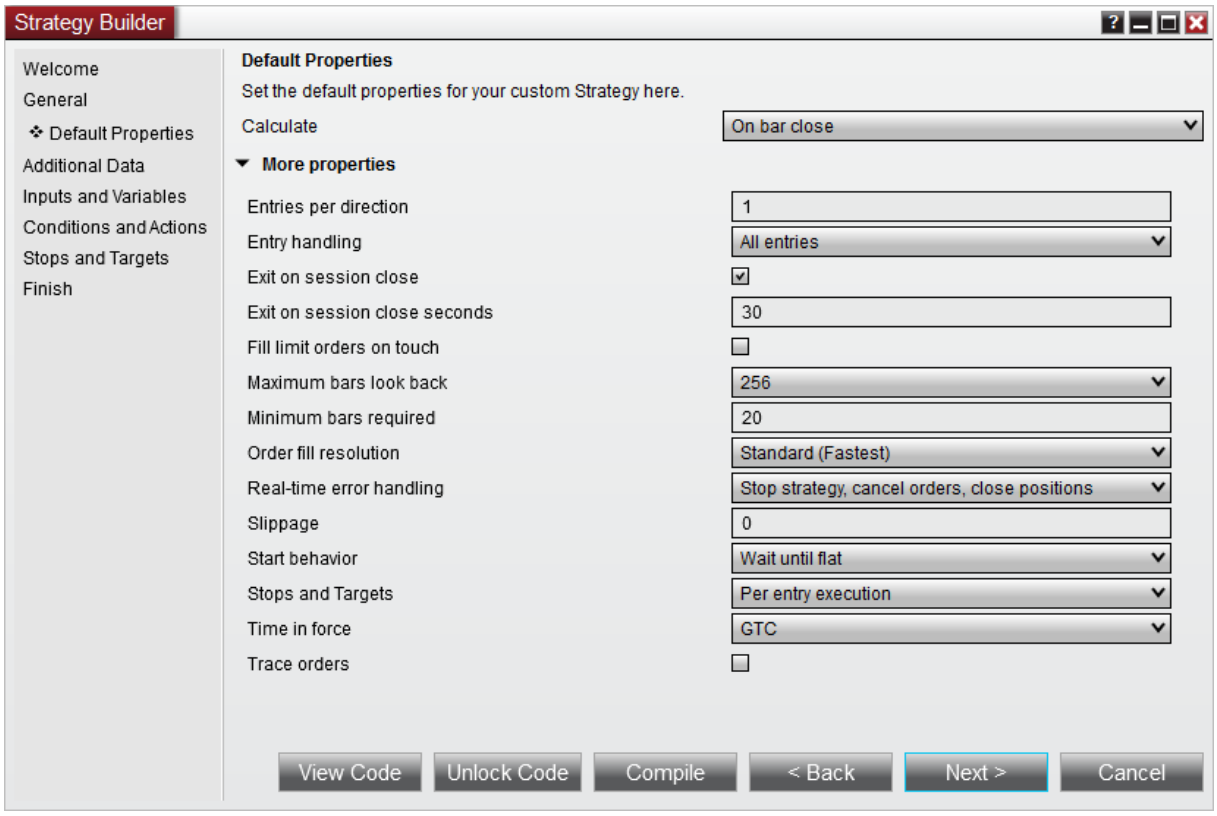
1. Within the NinjaTrader Control Center window select the New **Strategy Builder**... menu
2. Press the "Next >" button



3. Enter the information as shown above
4. Press the "Next >" button

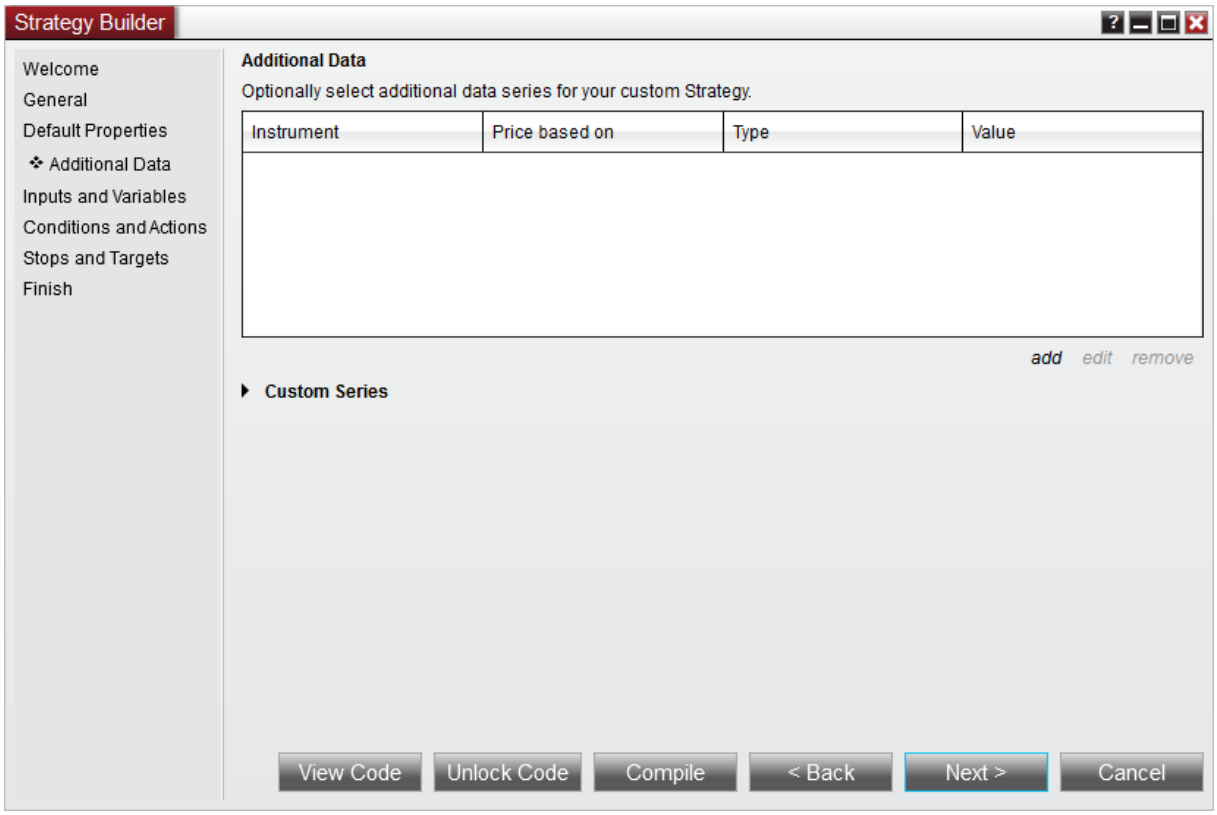
Setting Default Properties

The next page will allow you to set defaults for basic properties related to your strategy, including it's Calculate and EntryHandling settings. Click the **More Properties** button to expose additional properties. For this tutorial, we will not change any basic properties' defaults, and instead will leave them all set to the values shown below:



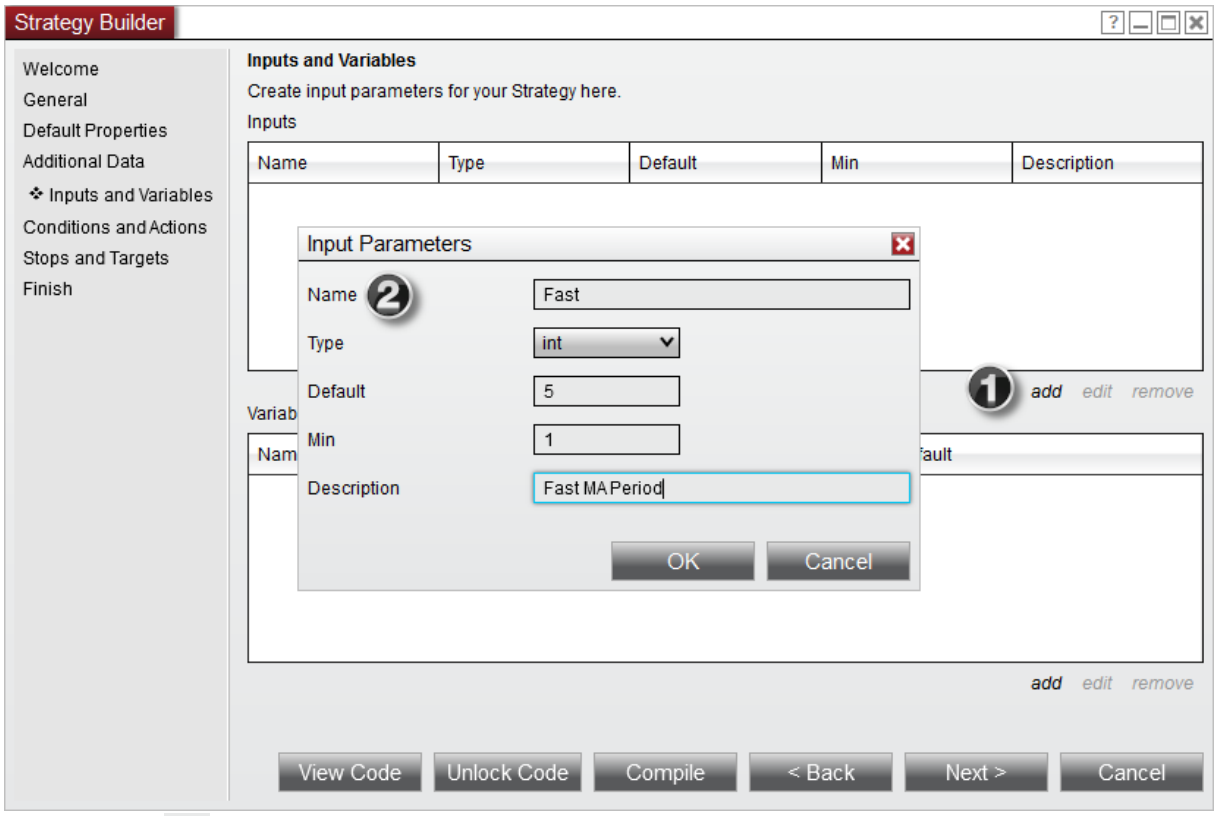
Adding Additional Data

The next page will allow you to configure one or more additional Bars objects for use by the strategy. For our purposes, we will leave this page blank and move forward by clicking the **Next >** button.

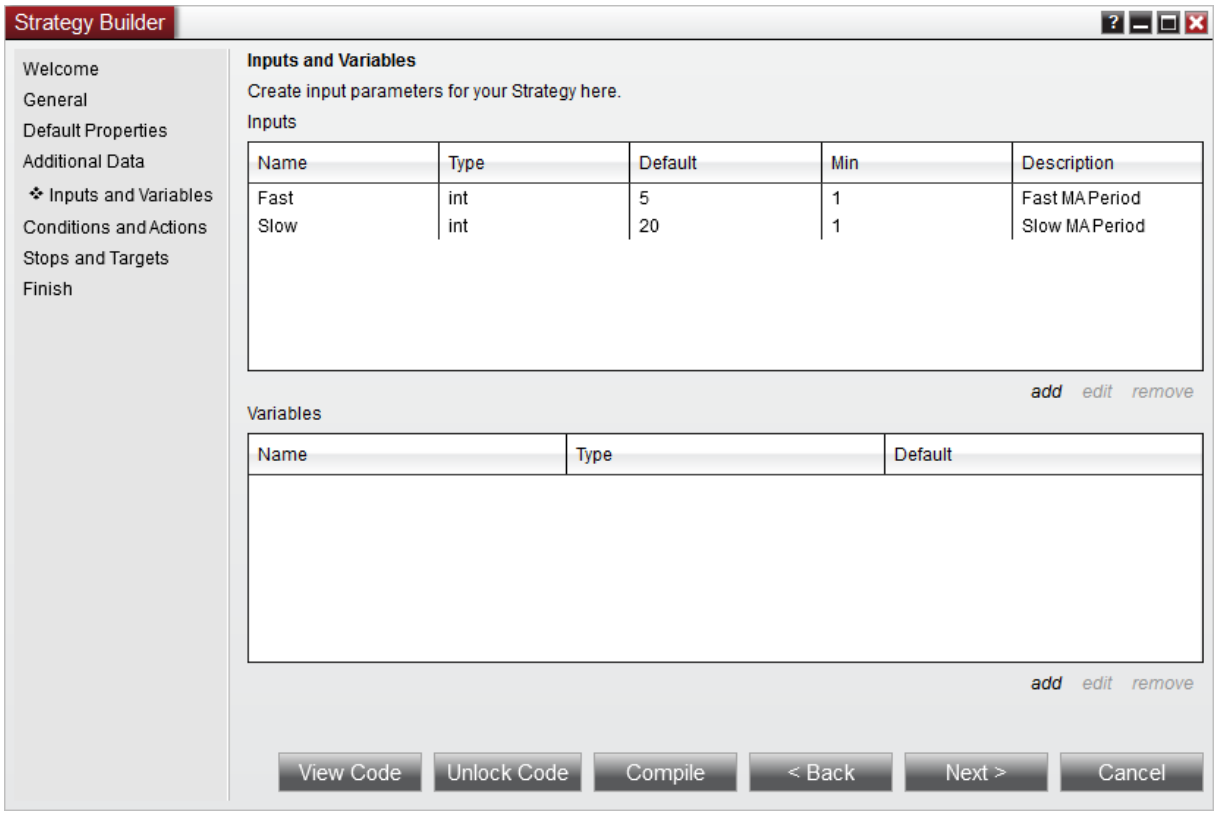


Defining Input Parameters

Below you will define your strategy's input parameters. These are any input parameters that can be changed by the user when running or backtesting a strategy. If your strategy does not require any parameters leave the "Name" fields blank.



1. Click the **add** button to add a property
2. Add input parameters into the newly created **Input Parameters** window and click Ok once the input parameter is set up



5. Add the inputs as per the image above
6. Press the "Next >" button

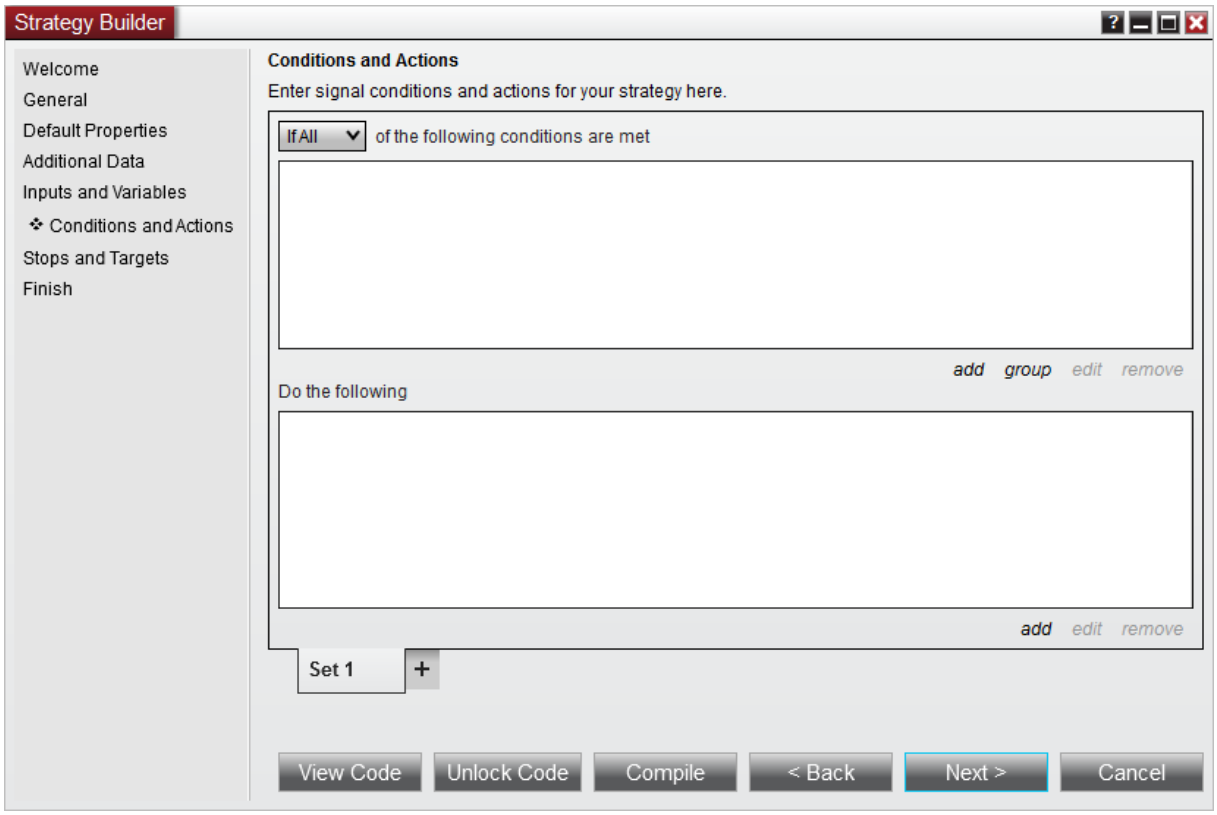
Defining Conditions and Actions

Below you can define conditions that trigger user defined actions such as placing orders, drawing on a chart or creating an alert.

Notice how there are two buttons on the screen below:

View Code... - Pressing this button loads the strategy code in the NinjaScript Editor for viewing purposes only. This is a great approach if you are new to programming or you want to see how the strategy wizard dynamically generates the correct script code on the fly.

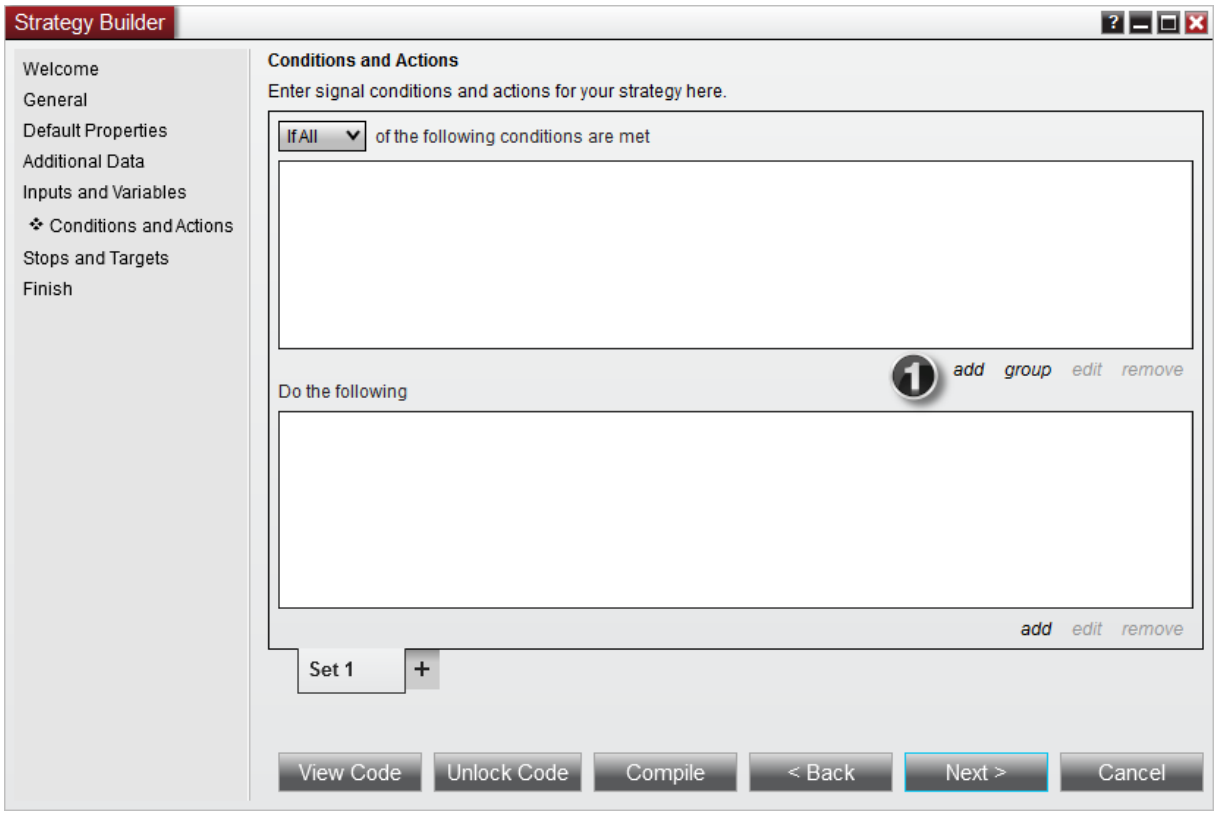
Unlock Code - Pressing this button loads the strategy code in the NinjaScript editor for further manual editing. Once this button is pressed, you can NOT go back to the Wizard for strategy construction and editing.



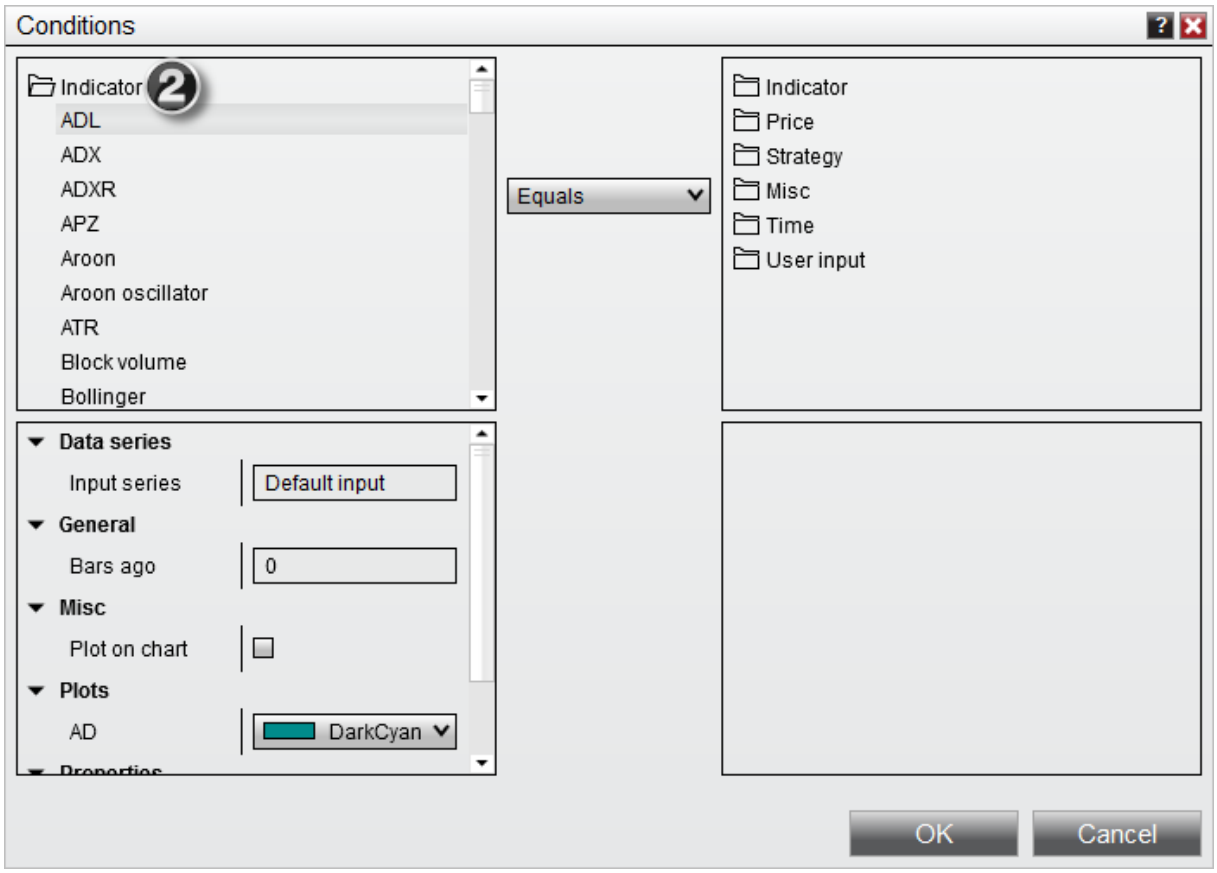
If you want to proceed with this tutorial through self programming continue here after pressing the "Unlock Code" button.
If you want to proceed with this tutorial through using the Strategy Builder please click here.

11.5.5.2.2 Creating the Strategy via the Wizard

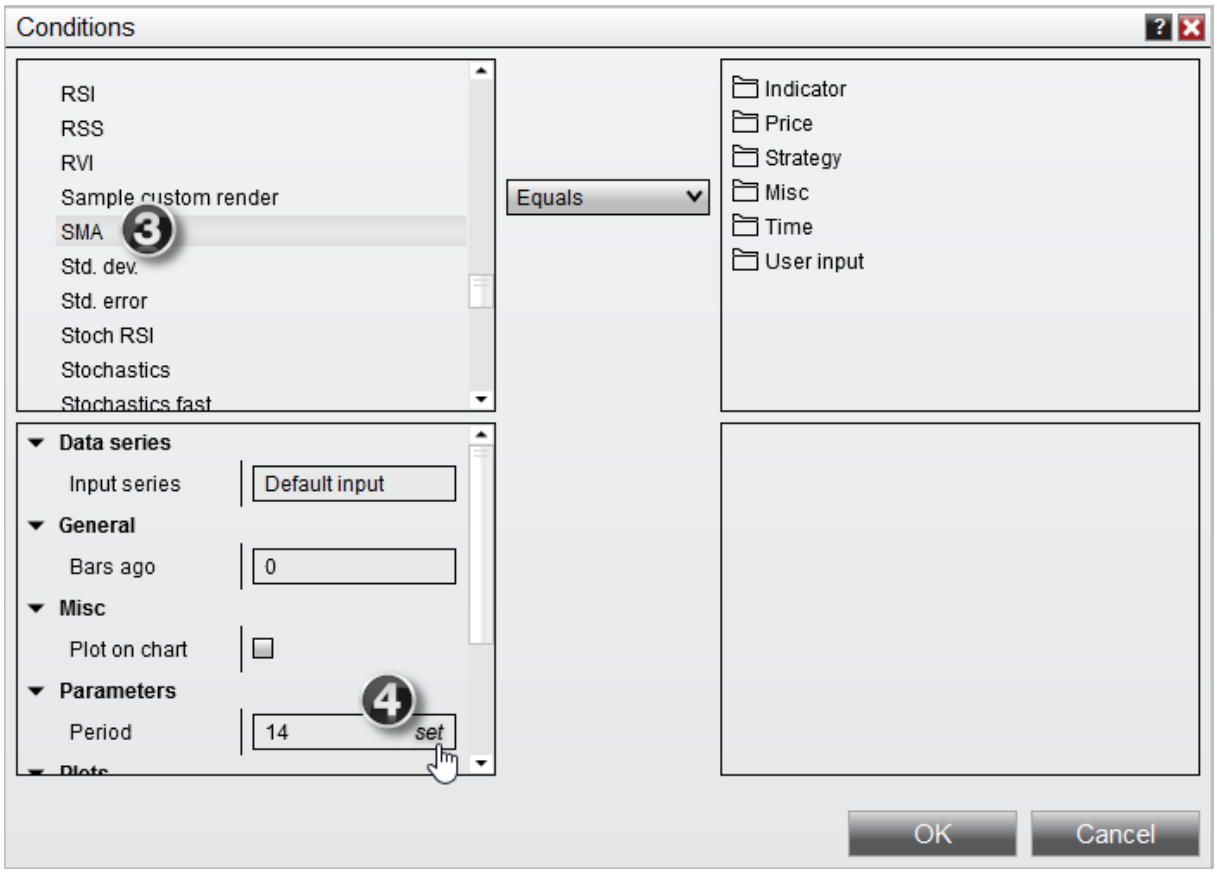
1. Press the "Add" button to display the "Condition Builder" window as per the image below



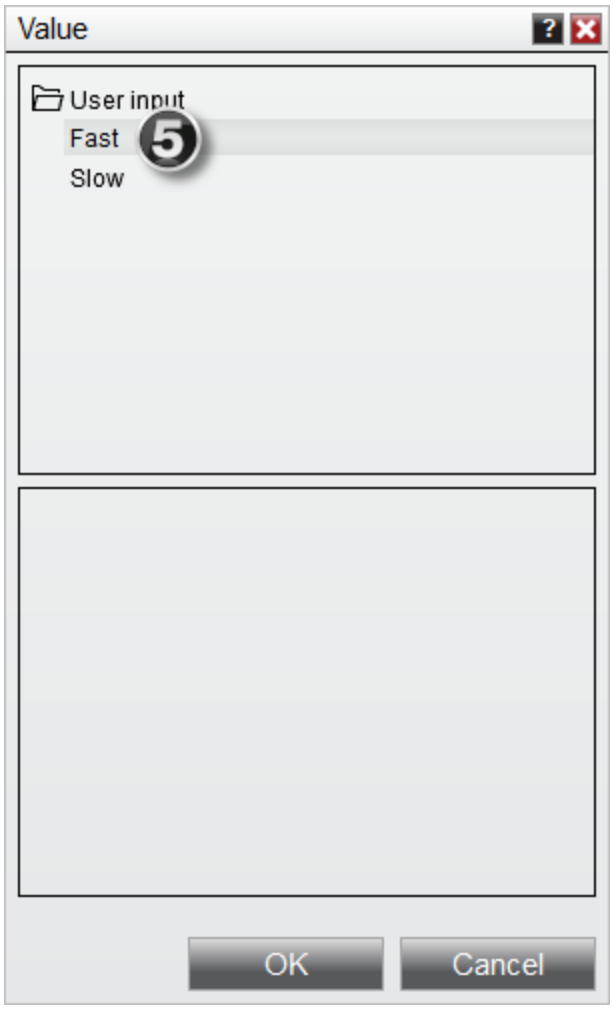
2. Expand the "Indicator" section to be able to select an indicator plot for your condition



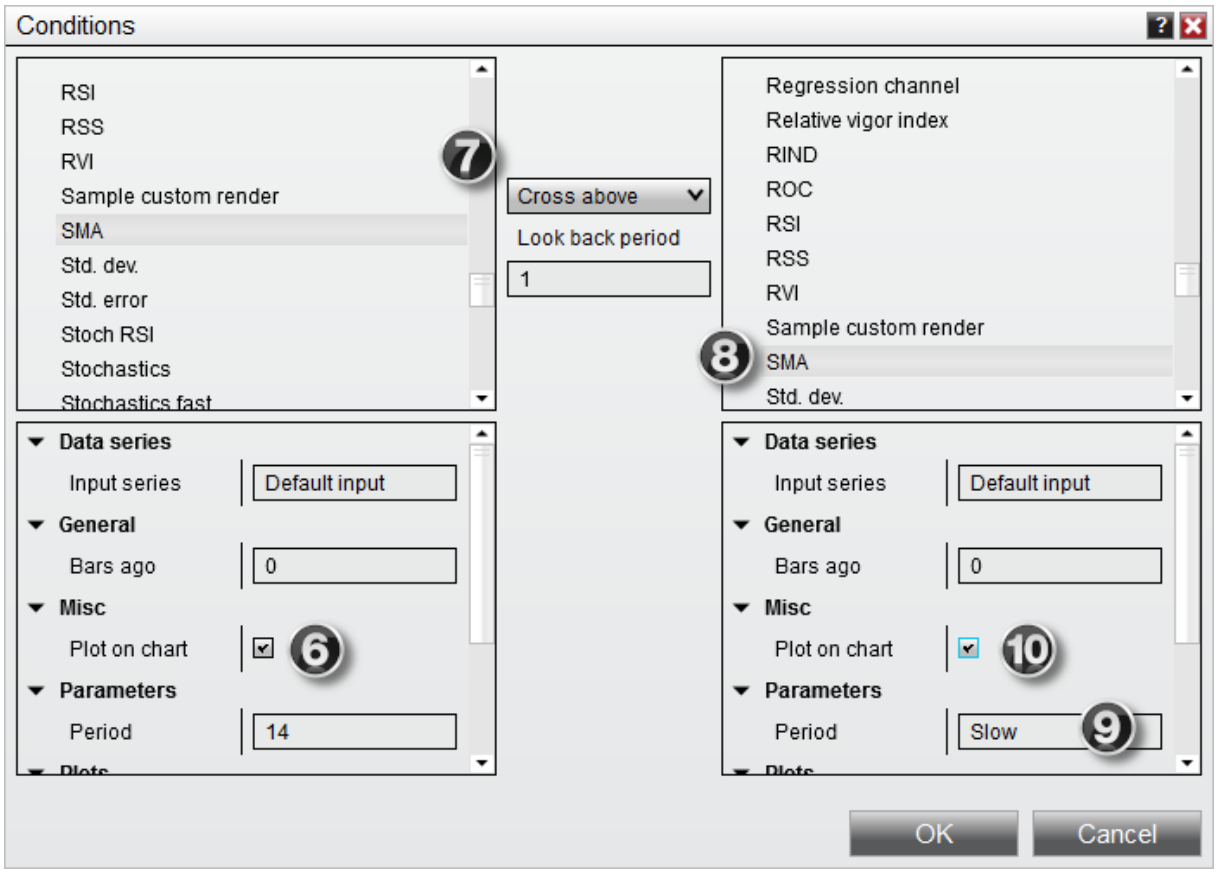
3. Scroll down and select the SMA indicator
4. Click the ***set*** menu item when your mouse is over the Period input field to select our User Defined Input parameter



5. Select User Input > Fast to select the Fast Period user input we created, then click **OK**

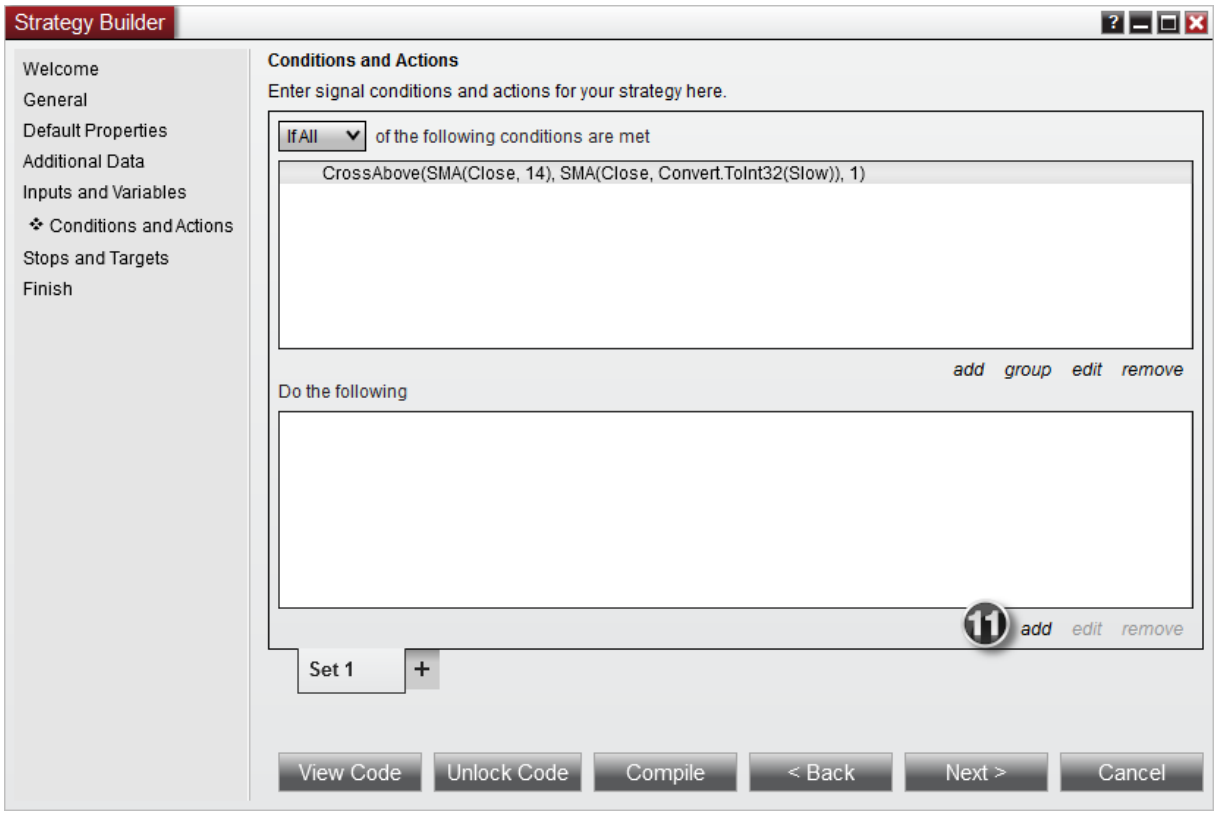


6. Enable this indicator to be plotted on a chart
7. Select "CrossAbove" and set the look back period to a value of "1"
8. Select "SMA" indicator in the right window
9. Set the "Slow" period (just like you did for Fast in step 4)
10. Enable this indicator to be plotted on a chart, then click OK to close the Condition Builder

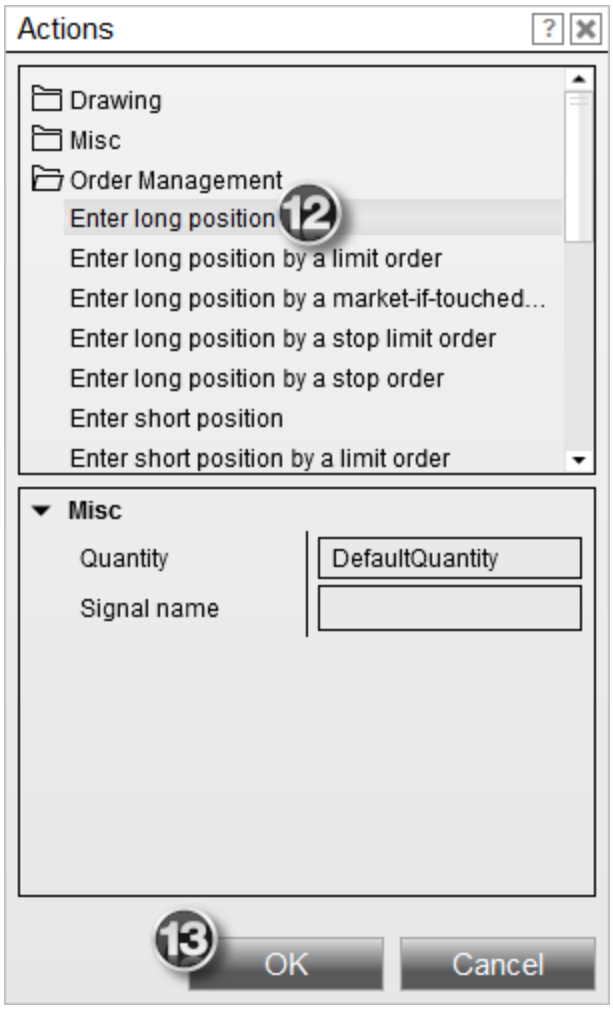


If you look at the image above, you just created an initial condition. The condition is "if the fast simple moving average crosses above the slow simple moving average".

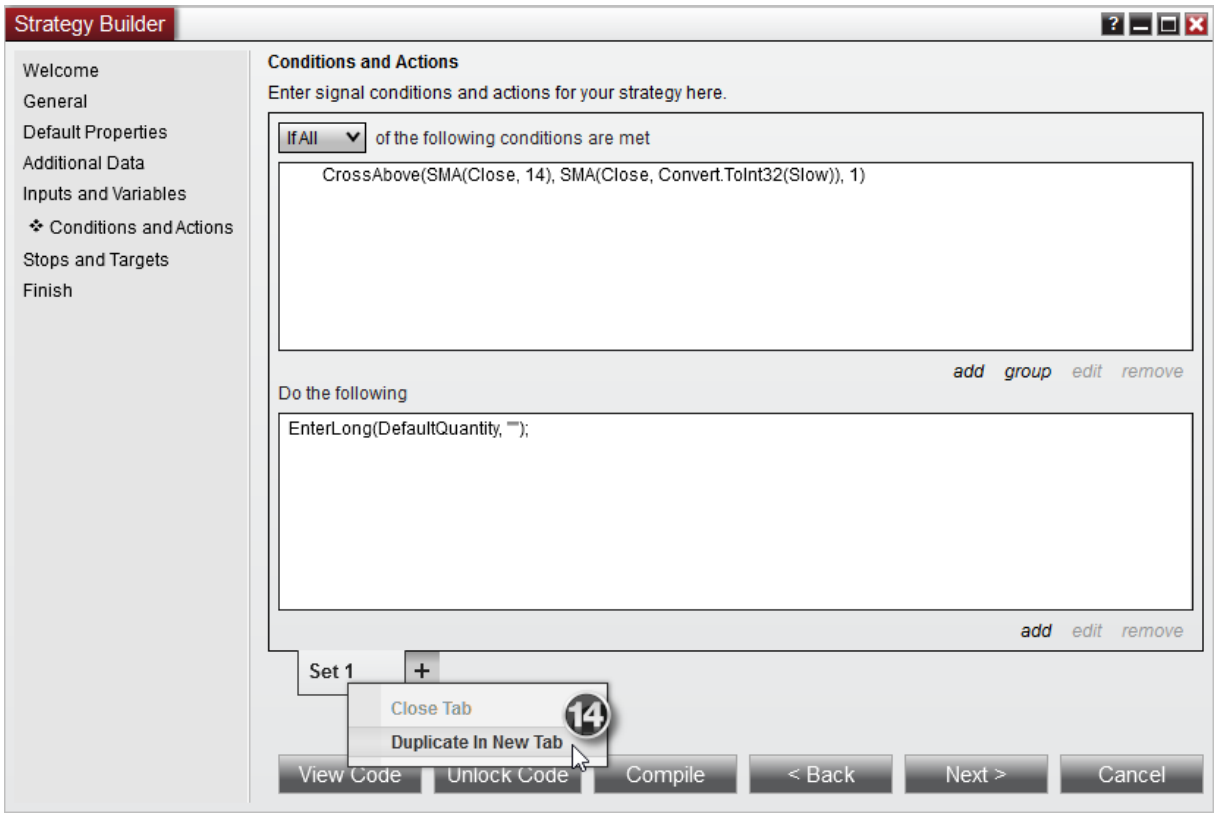
11. Click the **add** button under actions to add an action for this condition



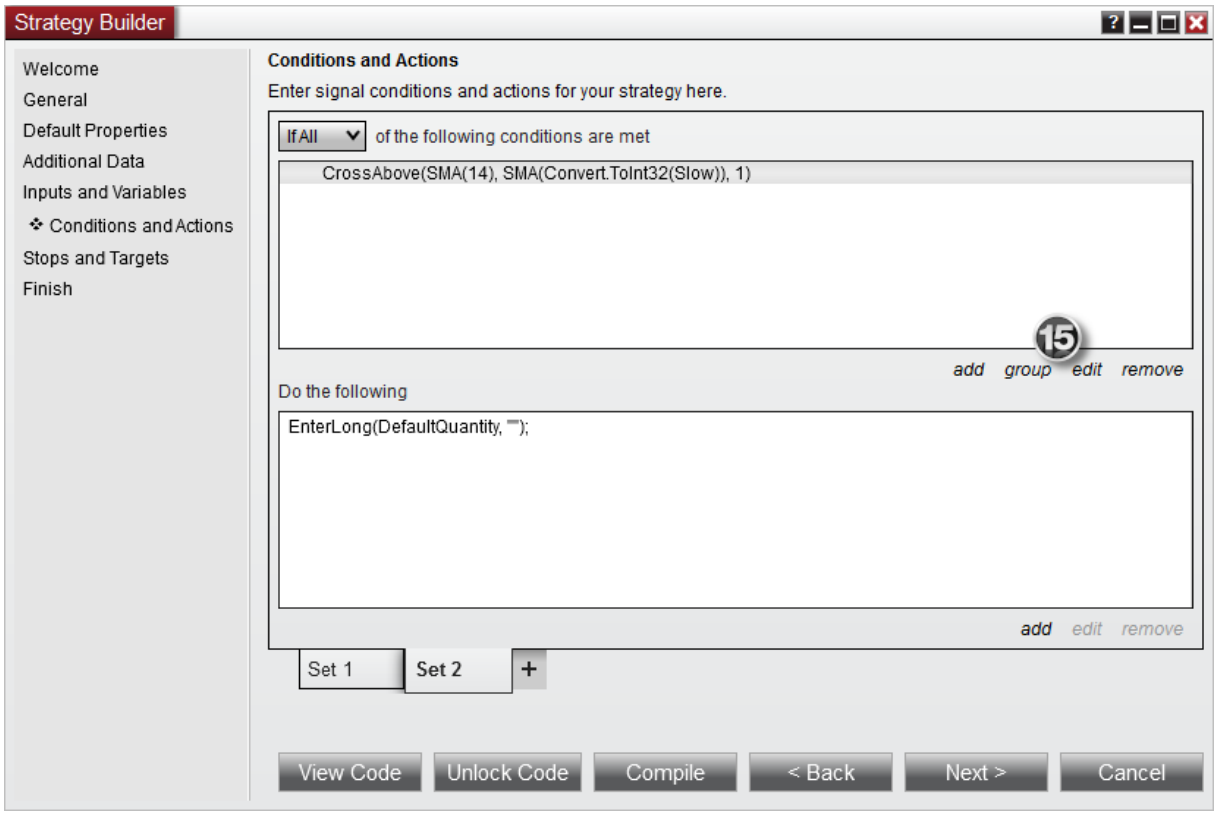
12. Select Order Management > Enter long position to have this condition fire a Buy Market Order
13. Select the OK button to add the action to the condition



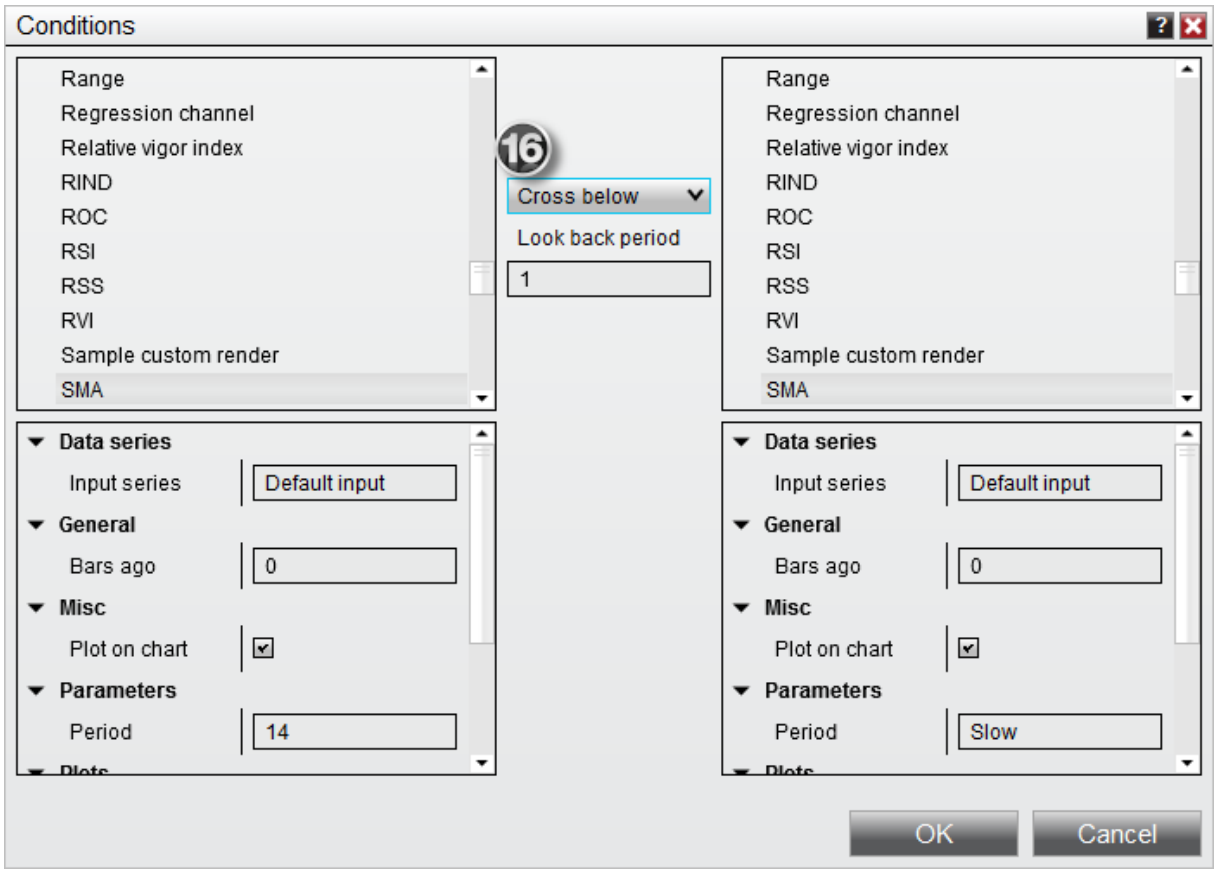
14. Right click on the "Set 1" tab and select **Duplicate in New Tab** to make a copy of this condition and action set



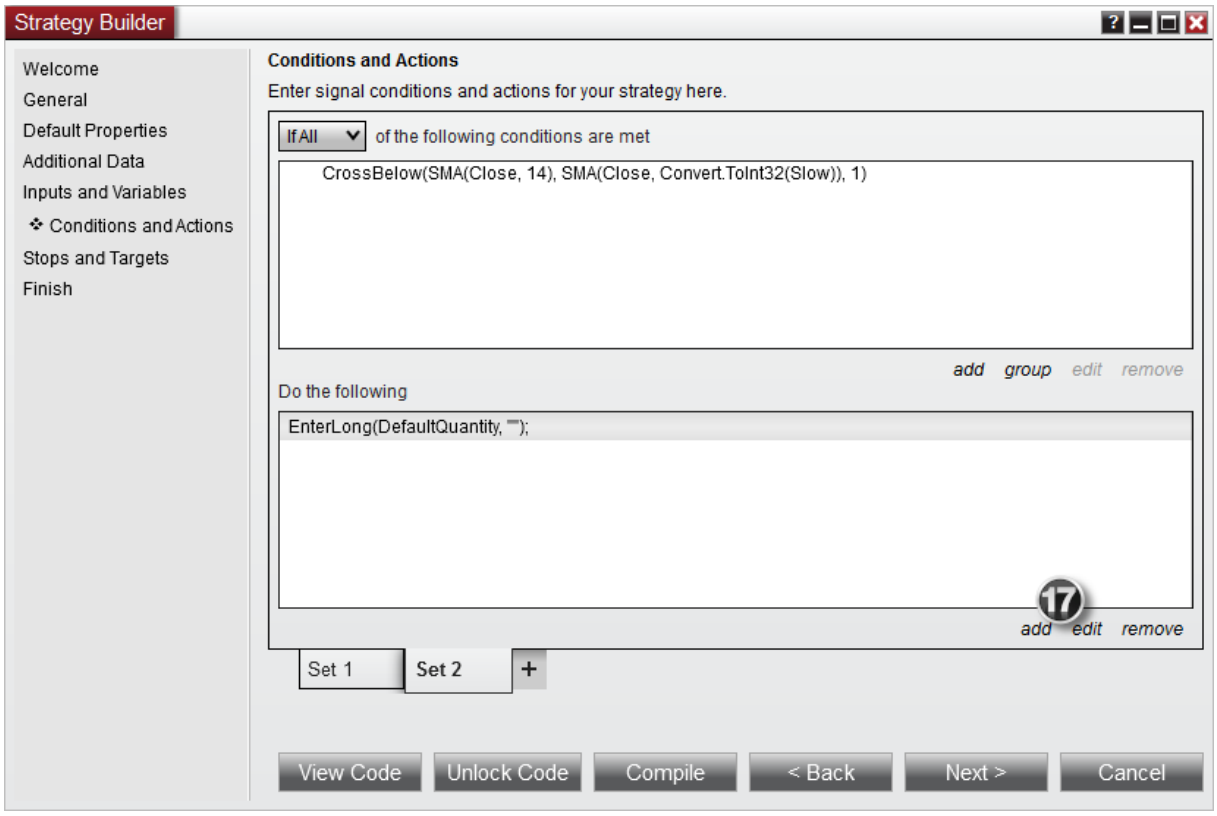
15. We will automatically be moved to the "Set 2" tab. From here, select the condition and click **edit**



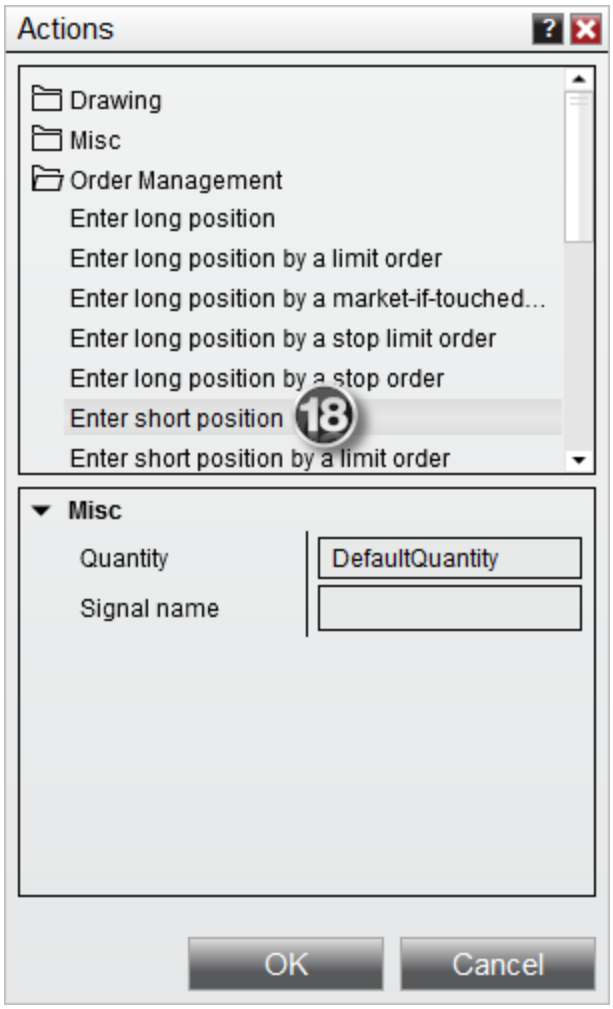
16. Change this Condition to use "Cross Below" so we can create a condition that triggers when the moving averages switch sides, then click **OK**



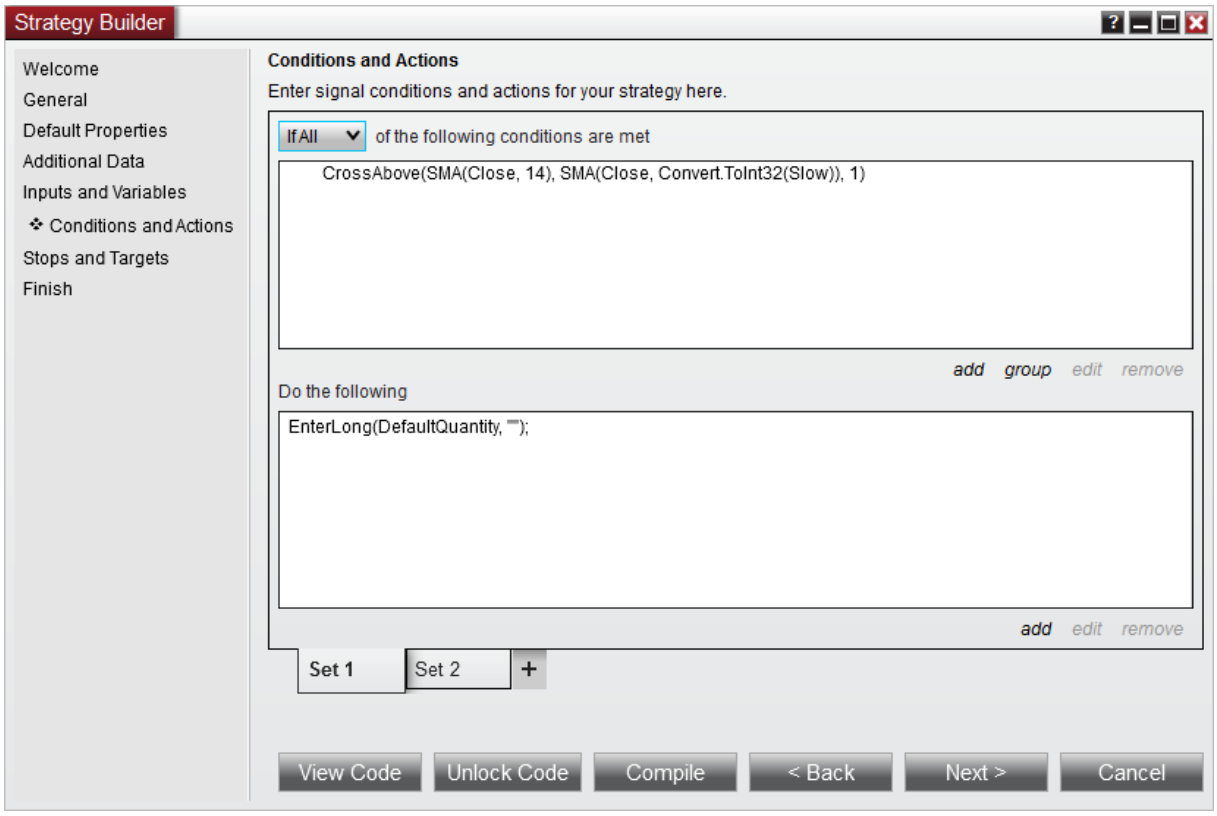
17. Click the action in the Condition Builder, and then select **edit** to edit the action. We will want to change this action so we sell instead of by for the reversed condition

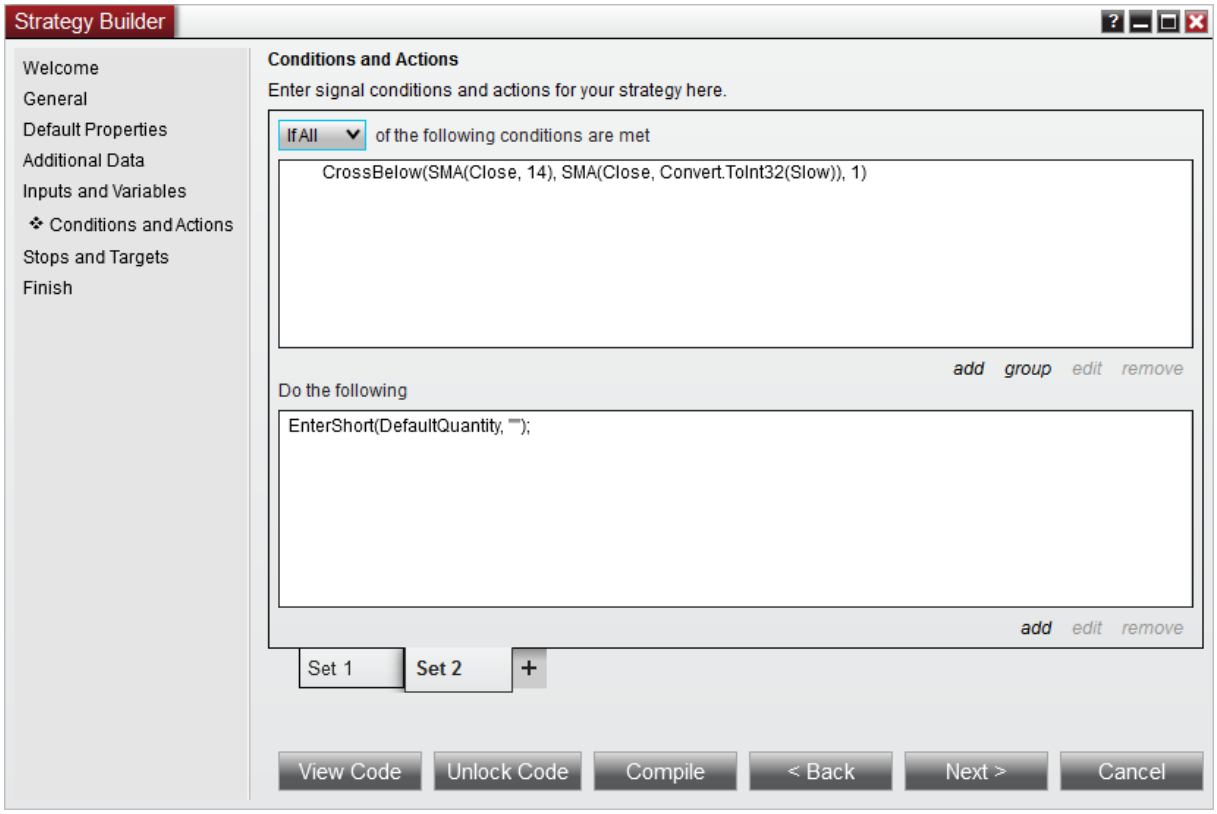


18. Under Order Management, select Enter short position to have the strategy submit a Sell Market order, then click **OK**.



Once finished, we will have condition sets that look like the following:





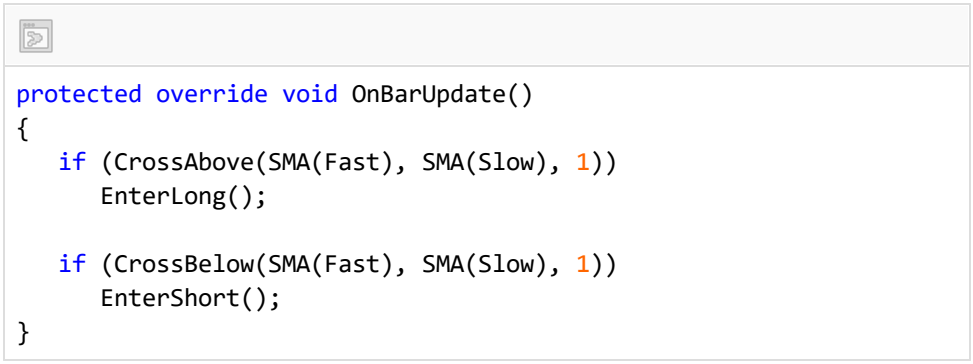
11.5.5.2.3 Creating the Strategy via Self Programming

If you have not done so already, press the "Unlock Code" button within the wizard to launch the NinjaScript Editor.

The OnBarUpdate() method is called for each incoming tick or on the close of a bar (user defined) when performing real-time calculations. Therefore, this is the main method called for strategy calculation and we will use this method to enter the script that check for entry and exit conditions.

The Entry and Exit Condition

Enter the code contained within the OnBarUpdate() method in the image below into the OnBarUpdate() method in the NinjaScript Editor:



```
protected override void OnBarUpdate()
{
   if (CrossAbove(SMA(Fast), SMA(Slow), 1))
      EnterLong();

   if (CrossBelow(SMA(Fast), SMA(Slow), 1))
      EnterShort();
}
```

Translated into English, the code contained within the OnBarUpdate() method above reads:

*if the fast simple moving average crosses **above** the slow simple moving average within the last bar, go long*

*if the fast simple moving average crosses **below** the slow simple moving average within the last bar, go short*

To accomplish this we used the following methods:

CrossAbove() - Checks for a cross above condition and returns true or false
CrossBelow() - Checks for a cross below condition and returns true or false
SMA() - Returns the value of a simple moving average
EnterLong() - Enters a market order long
EnterShort() - Enters a market order short

11.5.5.2.4 Compiling

The strategy code is now complete and needs to be compiled.

- If you completed this tutorial via the Strategy Wizard, simply follow the wizard instructions to the end at which time the strategy will compile.

- If you self coded this tutorial you can compile this strategy from within the NinjaScript Editor right mouse button menu "Compile" menu or simply press the F5 key.

It is important to understand that this process makes the strategy ready for real-time use and will run natively within NinjaTrader directly. It does not run interpreted as many other applications do. This provides you with the highest performance possible. If there are any errors reported during compiling, the error messages will be displayed at the bottom of the NinjaScript Editor.

11.5.5.3 The Strategy Development Process

Describe your Strategy

Describing your strategy means creating a set of objective rules that define the conditions used to enter and exit a market. Describing your strategy always starts with the wizard and then provides the following choices:

- Strategy Wizard with Condition Builder - This is a point and click approach for strategy description which is ideal for everyone from the non-programmer, novice programmer and advanced programmer.
- NinjaScript Editor - This is a modern scripting editor with full inline syntax checking and Intelliprompt. This is a great approach for those who want to manually code their strategy logic. If you are going to self code your strategy, please be familiar with the OnStateChange() and OnBarUpdate() methods.

Backtest and Optimize your Strategy

Once you have completed describing your strategy you can then test it against historical data to objectively determine how the strategy performed on a specific market(s) in the past.

- Strategy Analyzer - You can backtest, optimize, and analyze your historical results

At this point in the process you will likely go through an iterative cycle by where you change your strategy description, backtest, change description and backtest until you have a strategy that meets your requirements.

Real-Time Test your Strategy

It is critical that before you deploy your strategy against your live trading account, that you test it in real-time operation to ensure that the mechanics (operation) of your strategy behaves as you would expect it to. In addition, you can also forward test your strategy using real-time market data against the NinjaTrader trade simulation engine. NinjaTrader provides several options for real-time testing:

- Simulated Data Feed Connection - This is an random internally generated market with user controlled trend and is great for force testing operation of a strategy
- Playback Connection - Record, replay at user defined speeds multiple markets simultaneously and run your strategies

- Real-time Simulation - Connect to your broker or market data vendor in real-time and run your strategies through our state of the art simulation engine

You can run your strategy from either a chart or the Strategies tab of the Control Center window. You can generate real-time strategy performance data from the Strategies tab.

Running on your Live Trading Account

Now that you have described, backtested and real-time tested your strategy, you are ready to automate your strategy against your live trading account. A few tips you should know:

- Please make sure you fully understand the live run-time options
- Live strategy performance will vary from your backtested results
- Please make sure you fully understand Strategy Position vs Account Position... your strategy position is not a one-to-one relationship with your brokerage account position... you may need to synchronize if they are not synchronized.
- Strategies are automatically terminated (stop running) on NinjaTrader shut down
- Automated trading does not mean go fishing while your computer trades for you. We highly recommend that you are within close proximity to your computer while it is running an automated trading strategy; you never know what can go wrong
- You can run multiple trading strategies at the same time in the same market

11.5.5.4 Working with Accounts

There are a couple of fundamental concepts that should be understood in terms of how NinjaScript strategies behave in a live-trading environment. More information can be found on the Strategy Position vs Account Position, and Syncing Account Positions pages.

11.5.6 Historical Order Backfill Logic

Understanding How Orders are backfilled for NinjaScript strategies

NinjaScript strategies use an algorithm to process order fills on historical data in two scenarios: when processing fills in the Strategy Analyzer, or when processing historical orders for a live running strategy. The algorithm fills historical orders using the same set of logic in both scenarios. Below is an outline of the logic used to determine the appropriate fill price for each historical order. When more than one order needs to be filled at once, the logic below will be ran for each individual order in succession.

General Outline

The steps involved in determining the appropriate fill price for an order are documented in their own sections below. The general, top-level outline of the logic can be broken into three steps:

1. Prepare to calculate fill prices
2. Take three passes to calculate the appropriate fill price for each order which needs filled
3. Fill the orders using the calculated fill price

Step 1 - Prepare to Calculate Fill Prices

1. Determine all orders that need filled
2. Determine the current bar being formed at the time
3. Determine whether the current bar's first move was upward or downward
4. Determine the strategy being run
5. Determine the Bars In Progress the strategy is currently processing

Step 2 - Take Three Passes To Determine Fill Price

The bulk of the backfill logic takes place in this step. Here orders are tested for their order types and prices, and are compared against current bar data to determine the appropriate fill prices per order type in different scenarios.

> **Note**: Throughout these three passes, prices are temporarily stored in two variables: a "next high price" and a "next low price." These are used to approximate the price that would be hit on the next tick, for the purpose of setting the fill price.

1. First Pass
    a. If the current bar moved up first, save the current bar high price as the "next high price," then save the current bar Open price as the "next low price."
        i. If it moved down first, save the current bar Open price as the "next high price," then save the current bar Low price as the "next low price."

    b. if it's a Market Buy order, set the fill price to the lesser of the "next high price" or the bar Open
        i. If it's a Market Sell order, set the fill price to the greater of the "next high price" or the bar Open

    c. Ensure the strategy is currently processing the bar series on which the order resides, then:
        i. if the current order is Long, set the fill price to the lesser of the "next high price" or the current bar Open, taking slippage into account
            1. if it is Short, set the fill price to the greater of the saved "next low price" or the current bar Open, taking slippage into account

    d. Handle the special case of Limit orders with "Fill Limit Orders on Touch" enabled
        i. If the limit price has been touched, set the fill price to current bar Open

    e. Ensure the order would be filled without errors by comparing its stop and/or limit prices against each other and the current bar, then:
        i. For Limit orders, set the fill price to the current order's Limit price (however this is 'clamped' to happen inside the bar though)
        ii. For Stop Limit orders:
            1. if the order is Long, set the fill price to the greater of the existing fill price value or the current order's Limit price

2. if it is Short, set the fill price to the lesser of the existing fill price value or the current order's Limit price

2. Second Pass
   a. If the current bar moved up first, save the current bar High price as the "next high price," then save the current bar Low price as the "next low price."
      i. If it moved down first, save the current bar Low price as the "next high price," then save the current bar Low price as the "next low price."

   b. if it's a Market Buy order and the bar moved up first, set the fill price to the lesser of the "next high price" or the bar High
      i. If the bar moved down first, set the fill price to the lesser of the "next high price" or the bar Low
   c. If it's a Market Sell order and the bar moved up first, set the fill price to the greater of the "next high price" or the bar High
      i. If the bar moved down first, set the fill price to the greater of the "next high price" or the bar Low

   d. Ensure the strategy is currently processing the bar series on which the order resides, then:
      i. if the current order is Long, set fill price to the lesser of the "next high price" or the current bar Open, taking slippage into account
         1. if it is Short, set the fill price to the greater of the "next low price" or the current bar Open, taking slippage into account

   e. Handle the special case of Limit orders with "Fill Limit on Touch" enabled
      i. If the limit price has been touched, set the fill price to current bar Open

   f. Ensure the order would be filled without errors by comparing its stop and/or limit prices against each other and the current bar, then:
      i. For Limit orders, set the fill price to the current order's Limit price (however this is 'clamped' to happen inside the bar though)
      ii. For Stop Limit orders:
         1. if the order is Long, set the fill price to the greater of the existing fill price value or the current order's Limit price
         2. if it is Short, set the fill price to the lesser of the existing fill price or the current order's Limit price

3. Third Pass
   a. If the current bar moved up first, save the current bar Close price as the "next high price," then save the current bar Low price as the "next low price."
      i. If it moved down first, save the current bar High price as the "next high price," then save the current bar Close price as the "next low price."

b. If it's a Market Buy order and the bar moved up first, set the fill price to the lesser of the "next high price" or the bar Low
   i. If the bar moved down first, set the fill price to the lesser of the "next high price" or the bar High
c. If it's a Market Sell order and the bar moved up first, set the fill price to the greater of the "next high price" or the bar Low
   i. If the bar moved down first, set the fill price to the greater of the "next high price" or the bar High

d. Ensure the strategy is currently processing the bar series on which the order resides, then:
   i. if the current order is Long, set the fill price to the lesser of the "next high price" or the current bar Open, taking slippage into account
      1. if it is Short, set the fill price to the greater of the "next low price" or the current bar Open, taking slippage into account

e. Handle the special case of Limit orders with "Fill Limit on Touch" enabled
   i. If the limit price has been touched, set the fill price to current bar Open

f. Ensure the order would be filled without errors by comparing its stop and/or limit prices against each other and the current bar, then:
   i. For Limit orders, set the fill price to the current order's Limit price (however this is 'clamped' to happen inside the bar though)
   ii. For Stop Limit orders:
      1. if the order is Long, set the fill price to the greater of the existing fill price or the current order's Limit price
      2. if the order is Short, set the fill price to the lesser of the existing fill price or the current order's Limit price

Step 3 - Fill the Order

Each order is filled using the final fill price calculated for that particular order. If an order cannot be filled at this step, no further attempts will be made. Possible scenarios which would cause an order not to be filled at this stage include switching from State.Historical to State.Realtime when the strategy is currently waiting for a flat position before submitting orders, or a connectivity issue.

1. If the order is an entry, first temporarily clear all Entry Signals and pending orders from internally held collections of pending Entry Signals and orders

2. If its an exit, first determine the quantity that needs to be filled
   a. If the position being closed has not been partially closed already, use the full order quantity
   b. If the position has already been partially closed by other orders, set the order quantity to the remaining position quantity

3. Determine whether the strategy needs to wait until flat before filling the order
   a. This would apply if an exit order is being processed in real time, attempting to exit a position that was simulated on historical data

4. Create and parameterize a new Execution object (set Account, Commission, Instrument, Name, etc.)

5. Set properties of the Order object being analyzed
   a. AvgFillPrice, Filled (quantity), OrderState (set to OrderState.Filled)

6. Add the new Execution to the Executions collection

7. Add the order to the Orders collection

8. Fill the order

11.5.7 Multi-Threading Consideration for NinjaScript

Multi-Threading Overview

With the introduction of multi-threading in NinjaTrader special considerations should be made when programming your NinjaScript objects. Multi-threading basically allows NinjaTrader to take advantage of multi-core CPUs commonplace in modern computing to do multiple tasks at the same time. While this has many advantages for multi-tasking, it can cause new types of issues you may have not needed to consider before. This page was designed to serve as a high-level overview of some of the most common scenarios that can arise due to multi-threading, but should not be considered an exhaustive list.

Using A Dispatcher

Depending on your CPU configuration, the NinjaTrader application will usually consist of multiple main UI threads, where various features like Charts or NinjaScript objects run, along with a number of background worker threads where events such as market data updates will be distributed throughout the product. In principle, an object can only access information related to objects that exist on the same thread. It is possible (and quite likely), that the thread which a NinjaScript object is running will not be the same thread as the event which is calling the object. In cases where you need to access objects on the UI from a NinjaScript objects calling event thread, a dispatcher can be used.

> **Note**: As a best practice, you should always make sure to use Dispatcher.InvokeAsync() to ensure your action is done asynchronously to any internal NinjaTrader actions. Calling the synchronous **Dispatcher.Invoke()** method can potentially result in a deadlock scenarios as your script is loaded.

```
if (State == State.Historical)
{
   if (ChartControl != null)
   {
      // add some text to the UserControlCollection through the
ChartControls dispatcher
      ChartControl.Dispatcher.InvokeAsync(new Action(() =>  {
         UserControlCollection.Add(new
System.Windows.Controls.TextBlock {
            Text = "\nAdded by the ChartControl Dispatcher."
         });
      }));
   }
}
```

Thread Access

Since market data is distributed across the entire application by a randomly assigned UI thread, there is no guarantee that your object will be running on the same event thread that is calling the object. Therefore it is recommend that you call Dispatcher.CheckAccess() in order to test if you truly need to dispatch the requested action.

```
// check if the current object is already on the calling thread
if (Dispatcher.CheckAccess())
{
   // execute action directly
   action(args);
}
// otherwise run the action from the thread that created the object
else
{
   // dispatch action to calling thread
   Dispatcher.InvokeAsync(action, args);
}
```

Cross Thread Exceptions

When accessing objects included on the UI, you may receive the following error if you attempt to access a certain property/method from the wrong thread:

"Error on calling 'OnBarUpdate' method on bar 0: You are accessing an object which resides on another thread. I.E. creating your own Brush without calling .Freeze(), or trying to access a UI control from the wrong thread without using a Dispatcher"

This error can be avoided by invoking the **Dispatcher** used on the appropriate UI thread.

Access Violation Exception

Should you be using custom resources like text files, static members, etc. it is important to protect your resources from concurrent access. If NinjaTrader tried to use the resource at the same time you would run into errors similar to this one:

8/20/2010 12:14:29 PM|3|128|Error on calling 'OnBarUpdate' method for strategy 'SampleStrategy/1740b50bfe5d4bd896b0533725622400': The process cannot access the file 'c:\sample.txt' because it is being used by another process.

```
private object lockObj = new object();

private void WriteFile()
{
    // lock a generic object to ensure only one thread is accessing
the following code block at a time
    lock (lockObj)
    {
        string filePath = @"C:\sample.txt";
        using (System.IO.FileStream file = new
System.IO.FileStream(filePath, FileMode.Append, FileAccess.Write,
FileShare.None))
        {
          // write something to the file...

          // be sure to flush the buffer so everything is written
to the file.
          file.Flush();

          // The "using" block implicitly closes the FileStream
object,
          // giving other threads access to the file
        }
    }
}
```

Multi-threaded consideration for Order, Execution and Position objects

These considerations apply to the OnOrderUpdate(), OnExecutionUpdate() and OnPositionUpdate() handlers, where both the actual 'core' objects are passed by reference and updating method value parameters are provided. Examplary the OnOrderUpdate() is discussed in below.

- OnOrderUpdate() method guarantees that you will see each order state change in sequence
- The "order" method parameter represents the core order object updated by NinjaTrader
- The supplementary method parameters provide an updating value representing each order change in sequence. Think of this as the relevant information on the order at the time the state changed.
- Since the "order" method parameter represents the current order object state, it is possible for the updating values of that object to be out of sync with the correspond method parameters during a particular order update event.

As an example, the NinjaTrader core may have received "Working" and then "PartFilled" order state change events back from the broker API on thread "B". At some point in time (milliseconds later) the NinjaTrader core will take these events and trigger the OnOrderUpdate() method in the strategy on thread "A". Thus, when the strategy receives the first "Working" state for an order, the **orderState** method parameter will reflect the "Working" state although the actual **order.OrderState** is really in a state of "Part Filled". You would see that current value truly reflected in the core Order object method parameter or any order objects returned in any of the order methods such as EnterLong(). Of course, the OnOrderUpdate() method parameters will eventually receive the event for "PartFilled" state in the sequence the events were received.

Considering the concept above, if you are unsure if you should be using the core order object value vs the updating method parameter value value, ask your self if you are truly looking for the *most current order state*, or the *sequence of order states*:

- For the most current order state, use the core "order" object property (e.g., order.OrderState, order.LimitPrice, order.StopPrice, etc)
- For the sequence of order states, use the updating method parameter value (e.g., orderState, limitPrice, stopPrice, etc)

11.5.8 Multi-Time Frame & Instruments

Multi-Series Scripting Overview

NinjaScript supports multiple time frames and instruments in a single script. This is possible because you can add additional Bars objects to indicators or strategies, in addition to the primary Bars object to which they are applied. A Bars object represents all of the bars of data on a chart. For example, if you had a MSFT 1 minute chart with 200 bars on it, the 200 bars represent one Bars object. In addition to adding Bars objects for reference or for use with indicator methods, you can execute trades across all the different instruments in a script. There is extreme flexibility in the NinjaScript model that NinjaTrader uses for multiple-bars

scripts, so it is very important that you understand how it all works before you incorporate additional Bars objects in a script. An important fact to understand is that NinjaScript is truly event driven; every Bars object in a script will call the OnBarUpdate() method. The significance of this will become evident throughout this page.

Note: If using OnMarketData(), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same). The market data subscription behavior occurs both in real-time and during TickReplay historical

It is also important that you understand the following method and properties:

- AddDataSeries()
- BarsArray
- BarsInProgress
- CurrentBars

Note: As we move through this section, the term "Primary Bars" will be used and for the purpose of clarification, this will always refer to the first Bars object loaded into a script. For example, if you apply a script on MSFT 1 minute chart, the primary Bars would be MSFT 1 minute data set.

This section is written in sequential fashion. Example code is re-used and built upon from sub section to sub section.

Working With Multi-Time Frame Objects

Data processing sequence

Understanding the sequence in which bars series process and the granularity provided by market data vendors is essential for efficient multi-series development. Let's assume we have two series (primary and secondary) in our script, which is representing the same instrument, yet different intervals. During historical data processing, NinjaTrader updates the two series *strictly* according to their timestamps, calling the primary bar series of the corresponding timestamps first, and then calling the secondary series.

Note: Historical bars are processed according to their timestamps with the primary bars first, followed by the secondary, which is **NOT** guaranteed to be the same sequence that these events occurred in real-time. If your development requires

ticks to process in the same sequence historically as well as in real-time, you will need to enable Tick Replay (utilizes more PC resources).

Shared Timestamps

In circumstances where multiple bars share the same exact timestamps, your primary bars series will *always* be processed first, followed by the secondary bars series (regardless of the period value used). Consequently, if you were looking to obtain a value from the secondary bars series, it would **ONLY** be available *after* the primary series has been processed for the same timestamps. For example, consider a news event or a fast moving market with an influx of ticks (session begin/session end). This activity will yield a wider range of bars than usual and the probability of those bars sharing the same timestamps increases. If such a succession of bars with the same timestamps is processed, the primary bars would be processed first and then the secondary bars during this period.

Tip: While the following behavior applies to all period types, the effects are amplified on smaller time frames. If you plan on using a high-resolution (e.g., 1-second, 10-tick, etc), please make sure to thoroughly read and understand the material below when working with these additional series. It is also important to keep in mind that the granularity of the timestamps will dictate how accurately NinjaTrader can synchronize the bars in historical processing. The available level of granularity will be dependent upon which data provider you use with NinjaTrader.

Let's look at an illustration of how the multi-time frame bar processing sequence can be understood. Assume our primary series is a 5-tick bar series, and our secondary series is a 1-tick bar series. The time of day is near the session close, so a rapid sequence of bars is generated.

In the figure below, the 1st group of bars (colored orange), and the 4th group of bars (colored purple) process in an exact logical sequence (i.e., a single primary bar update, followed by five secondary series updates). This is because each bar in these groups have *unique timestamps* and NinjaTrader can synchronize those bars logically in the exact time sequence each series updated. However, all of the bars marked with red text share the *same exact timestamps* down to the millisecond (14:59:00:480). Since there were six ticks in sequence with the shared timestamps, this range of ticks expands two of the primary bars (colored green and blue). As a result, the primary bar #3 appears to update earlier when compared to the secondary series. In reality, both

bars series are incrementing in their exact sequence according to the timestamps of each series.

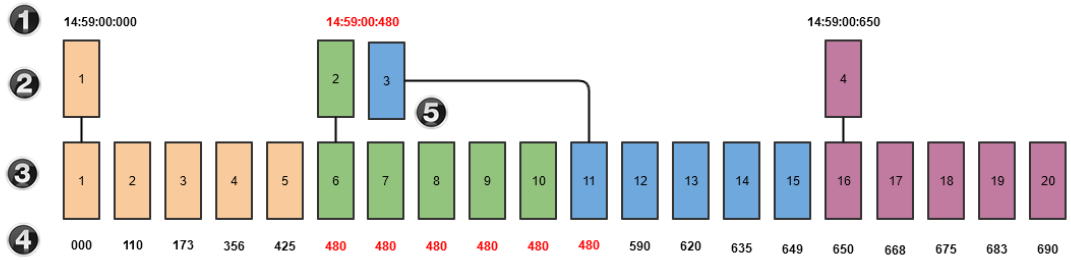


Figure 1. Bar processing with shared timestamps

1. Timestamps of primary series (hour, minute, second, millisecond)
2. Current bars numbered in series representing 5-tick primary series
3. Current bars numbered in series representing 1-tick secondary series
4. Millisecond time stamps of secondary series
5. A sequence of bars sharing the same time stamps

Adding Additional Bars Objects to NinjaScript

Additional Bars are added to a script via the AddDataSeries() method in the OnStateChange() method when the State has reached **State.Configure**. When a Bars object is added to a script, it is also added to the BarsArray array. **BarsArray** functions like a container in the script that holds all Bars objects added to the script. As a Bars object is added to the script, it's added to **BarsArray** and given an index number so we can retrieve this Bars object later.

Warning:

- This method should **ONLY** be called from the OnStateChange() method during **State.Configure**
- Arguments supplied to **AddDataSeries()** should be hardcoded and **NOT** dependent on run-time variables which cannot be reliably obtained during State.Configure (e.g., Instrument, Bars, or user input). Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided. Trying to load bars dynamically may result in an error similar to: **Unable to load bars series. Your NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.**

- When instantiating indicators in a Multi-Series script in OnStateChange, the input any hosted indicator is running on should be explicitly stated (since a specific BarsInProgress is not guaranteed)

For the purpose of demonstration, let's assume that a MSFT 1 minute bar is our primary Bars object (set when the script is applied to a 1 minute MSFT chart) and that the OnStateChange() method is adding a 3 minute Bars object of MSFT, then adding a 1 minute Bars object of AAPL, for a total of 3 unique Bars objects.

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
         Name    = "Multi-Time Frame & Instruments Example";
   }
   else if (State == State.Configure)
   {
      AddDataSeries(BarsPeriodType.Minute, 3);
      AddDataSeries("AAPL", BarsPeriodType.Minute, 1);
   }
}
```

Note: To maximize data loading performance, any NinjaScript object (indicator or strategy as host) which references a multi-series indicator which calls AddDataSeries must include it's own calls to AddDataSeries(). For example, if the code above was included in an indicator, and that indicator was referenced in a NinjaScript strategy, then the hosting strategy will need to include the same calls to AddDataSeries(). When the strategy adds the additional Bars objects, the calls to AddDataSeries() within the indicator will be ignored. If the Bars objects are not added by the strategy in such cases, and error will be thrown in the Log tab of the Control Center that would read - "A hosted indicator tried to load additional data. All data must first be loaded by the hosting NinjaScript in its configure state."

Creating Series<T> Objects

Series<T> Objects

Series<T> is the base class for PriceSeries, TimeSeries, and VolumeSeries. Rather than using one of these pre-defined derived classes, you can create your own Series<T> collection to hold any Type that you choose. The advantage that Series<T> has over other collections is that it can be quickly initialized to contain a number of index slots equal to the number of bars in one of the Bars objects on the chart, with each index slot corresponding to a specific bar.

Initializing a Series<T> with BarsArray

A Series<T> can be constructed by passing in a specific index of BarsArray. Initializing a Series<T> this way produces an empty Series<T> container holding the same number of index slots as the BarsArray that was passed in as an argument. For example, assuming that BarsArray[1] holds 500 bars, the code below will create an empty Series<T> with 500 index slots:

Initializing Series<T> with BarsArray

```
private Series<double> myEmptyIndexedSeries; // Define a
Series<T> objectvariable.

// Initialize the Series object to have the same number of
index slots as BarsArray[1]
protected override void OnStateChange()
{
    if (State == State.DataLoaded)
    {
        // Passing in BarsArray[1] as an argument results
in an empty Series with an identical number of index slots
        myEmptyIndexedSeries = new
Series<double>(BarsArray[1]);
    }
}
```

This method of initializing a Series<T> can be especially useful when you wish to store user-defined information related to each bar in a Bars object on the chart. This process ensures that index slots are available for every bar on the chart right away.

Initializing a Series<T> with an Indicator Method

Passing in an indicator method as an argument when instantiating a Series<T> object provides an alternative to the process outlined above. Because indicator methods already contain Series objects synced to the bars on a chart, they can be used to inform the constructor of Series<T> of how many index slots to create.

Initializing Series<T> with an Indicator Method

```
// Declare two Series objects
private Series<double> primarySeries;
private Series<double> secondarySeries;

protected override void OnStateChange()
{
     if (State == State.Configure)
    {
        // Adds a secondary bar object to the strategy.
        AddDataSeries(BarsPeriodType.Minute, 5);
    }
    else if (State == State.DataLoaded)
    {
        // Syncs a Series object to the primary bar object
        primarySeries = new Series<double>(this);

        /* Syncs another Series object to the secondary bar
object.
        We use an arbitrary indicator overloaded with an
ISeries<double> input to achieve the sync.
        The indicator can be any indicator. The
Series<double> will be synced to whatever the
        BarsArray[] is provided.*/
        secondarySeries = new
Series<double>(SMA(BarsArray[1], 50));

        // Stop-loss orders are placed 5 ticks below
average entry price
        SetStopLoss(CalculationMode.Ticks, 5);

        // Profit target orders are placed 10 ticks above
average entry price
        SetProfitTarget(CalculationMode.Ticks, 10);
    }
}
```

How Bars Data is Referenced

Understanding how multi-time frame bars are processed and which OHLCV data is referenced is critical.

Figure 1 below demonstrates the concept of bar processing on historical data or in real-time when the Calculate property is set to **Calculate.OnBarClose**. The 1 minute bars in yellow will only know the OHLCV of the 3 minute bar in yellow. The 1 minute bars in cyan will only know the OHLCV data of the 3 minute bar in cyan. Take a look at "Bar #5," which is the fifth one minute bar. If you wanted to know the current high value for the 3-minute time frame, you would get the value of the first 3 minute bar since this is the last "closed" bar. The second 3 minute bar (cyan) would not be known at that time.

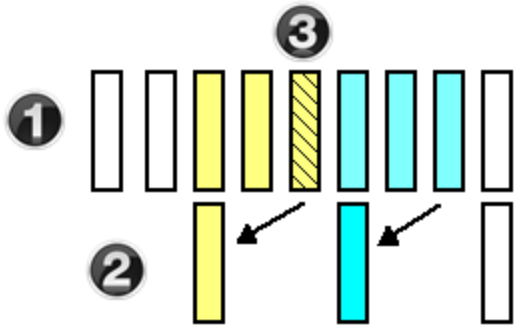


Figure 1. Bar processing on historical data using Calculate.OnBarClose
1. Primary 1-minute bar series
2. Secondary 3-minute bar series
3. Bar #5

Contrast the above image and concept with the image below, which demonstrates bar processing in real-time when the **Calculate** property is set to **Calculate.OnEachTick** (tick by tick processing) or **Calculate.OnPriceChange (**processing by change in price). The 1 minute bars in yellow will know the current OHLCV of the 3 minute bar in yellow (second 3 minute bar) which is still in formation and has not yet closed.

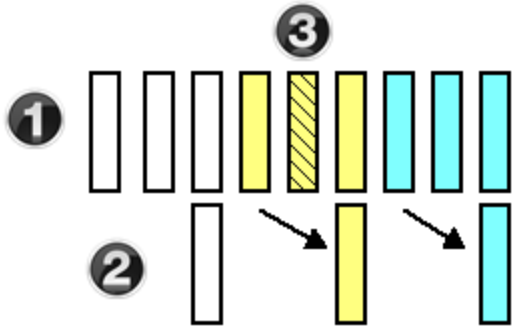


Figure 2. Bar processing in real-time using Calculate.OnEachTick or Calculate.OnPriceChange
1. Primary 1-minute bar series
2. Secondary 3-minute bar series

3. Bar #5

If you have a multi-time frame script in real-time, and it is processing tick by tick instead of on the close of each bar, understand that the OHLCV data you access in real-time is different than on historical data.

Below is another example to illustrate this point:

Your script has complex logic that changes the bar color on the chart. You are running tick by tick, as per the above "Figure 2" image, the 5th 1 minute bar is looking at OHLCV data from the second 3 minute bar. Your script changes the fifth 1 minute bar color to green. In the future, you reload your script into the chart and the fifth 1 minute bar is now a historical bar. As per Figure 1 above, the fifth 1 minute bar now references the OHLCV data of the first 3 minute bar (instead of the 2nd 3 minute bar as per Figure 2) and as a result, your script logic condition for coloring the bar green is no longer valid. The result is that now your chart looks different.

Special considerations for session boundaries :

Bars are not considered closed until the first tick of the following bar comes in (see also "True Event Driven OnBarUpdate" below). As a consequence, if the above series 2 cyan bar represents the final bar of a session, and this bar is referenced from the matching series 1 cyan bar, or anywhere after that, the data from the close of the bar (the beginning of the next session) will be referenced. If you plan on using multiple session templates, you will need to handle the final bar of a trading day case explicitly (for example, using a Session Iterator and the PriorDayOHLC) if you would like to reference data from the end of the previous trading day instead of the beginning of the current trading day.

Using Bars Objects as Input to Indicator Methods

In the sub-section above, the concept of index values was introduced. This is a critical concept to understand since it is used consistently when working with multi-Bars script.

Let's demonstrate this concept:

Carrying on from the example above, our primary Bars is set from a MSFT 1 minute chart

MSFT 1 minute Bars is given an index value of 0 in BarsArray

In the OnStateChange() method we added a MSFT 3 minute Bars object and an AAPL 1 minute Bars object to the script

MSFT 3 minute Bars is given an index value of 1 in BarsArray
AAPL 1 minute Bars is given an index value of 2 in BarsArray

Incremental index values are given to Bars objects as they are added to a script. If there are 10 Bars objects in a script, then you will have index values ranging from 0 to 9.

Our script now has 3 Bars objects in the container **BarsArray**. From this point forward, we can ask this container to give us the Bars object we want to work with by providing the index value. The syntax for this is:

```
BarsArray[index]
```

This allows us to get the correct Bars object and use it as input for an indicator method. For example:

```
ADX(14)[0] > 30 && ADX(BarsArray[2], 14)[0] > 30
```

The above expression in English would translate to:

If the 14 period ADX of MSFT 1 minute is greater than 30 and the 14 period ADX of AAPL 1 minute is greater than 30

Before we can apply this concept, we need to ensure that our Bars objects actually contain bars that we can use to run calculations. This can be done by checking the CurrentBars array, which returns the number of the current bar in each Bars object. Using this in conjunction with BarsRequiredToPlot will ensure each Bars object has sufficient data before we begin processing.

> **Note**: By default, the **CurrentBars** starting value will be -1 until all series have processed the first bar.

```
protected override void OnBarUpdate()
{
     // Checks to ensure all Bars objects contain enough
bars before beginning.
     // If this is a strategy, use BarsRequiredToTrade
instead of BarsRequiredToPlot
     if (CurrentBars[0] <= BarsRequiredToPlot ||
CurrentBars[1] <= BarsRequiredToPlot || CurrentBars[2] <=
BarsRequiredToPlot)
          return;
}
```

Putting it all together now, the following example checks if the current CCI value for all Bars objects is above 200. You will notice that BarsInProgress is used. This is to check which Bars object is calling the OnBarUpdate() method. More on this later in this section.

```
protected override void OnBarUpdate()
{
     // Checks to ensure all Bars objects contain enough
bars before beginning
     // If this is a strategy, use BarsRequiredToTrade
instead of BarsRequiredToPlot
     if (CurrentBars[0] <= BarsRequiredToPlot ||
CurrentBars[1] <= BarsRequiredToPlot || CurrentBars[2] <=
BarsRequiredToPlot)
          return;

     if (BarsInProgress == 0)
     {
          if (CCI(20)[0] > 200 && CCI(BarsArray[1], 20)[0]
> 200
           && CCI(BarsArray[2], 20)[0] > 200)
          {
               // Do something
          }
     }
}
```

True Event Driven OnBarUpdate() Method

Since a NinjaScript script is truly event driven, the OnBarUpdate() method is called for every bar update event for each Bars object added to the script. This model maximizes flexibility. For example, you could have multiple trading systems combined into one strategy, each dependent on one another. For example, you could have a 1 minute MSFT Bars object and a 1 minute AAPL Bars object, process different trading rules on each Bars object and check to see if MSFT is long when AAPL trading logic is being processed.

The BarsInProgress property is used to identify which Bars object is calling the OnBarUpdate() method. This allows you to filter out the events that you are looking for.

Continuing our example above, let's take a closer look at what is happening. Remember, we have three Bars objects working in our script, a primary Bars MSFT 1 minute, an MSFT 3 minute, and an AAPL 1 minute.

```
protected override void OnBarUpdate()
{
     // Checks to ensure all Bars objects contain enough
bars before beginning
     // If this is a strategy, use BarsRequiredToTrade
instead of BarsRequiredToPlot
     if (CurrentBars[0] <= BarsRequiredToPlot ||
CurrentBars[1] <= BarsRequiredToPlot || CurrentBars[2] <=
BarsRequiredToPlot)
          return;

     // Checks if OnBarUpdate() is called from an update on
the primary Bars
     if (BarsInProgress == 0)
     {
          if (Close[0] > Open[0])
               // Do something
     }

     // Checks if OnBarUpdate() is called from an update on
MSFT 3 minute Bars
     if (BarsInProgress == 1)
     {
          if (Close[0] > Open[0])
               // Do something
     }

     // Checks if OnBarUpdate() is called from an update on
AAPL 1 minute Bars
     if (BarsInProgress == 2)
     {
          if (Close[0] > Open[0])
               // Do something
     }
}
```

What is important to understand in the above sample code is that we have "if" branches that check to see what Bars object is calling the OnBarUpdate() method in order to process relevant trading logic. If we only wanted to process the events from the primary Bars we could add the following condition at the top of the OnBarUpdate() method:

```
if (BarsInProgress != 0)
    return;
```

What is also important to understand is the concept of context. When the OnBarUpdate() method is called, it will be called within the context of the calling Bars object. This means that if the primary Bars triggers the OnBarUpdate() method, all indicator methods and price data will point to that Bars object's data. Notice how the statement "if (Close[0] > Open[0]" exists under each "if" branch in the code sample above. The values returned by Close[0] and Open[0] will be the close and open price values for the calling Bars object. So when the BarsInProgress == 0 (primary Bars) the close value returned is the close price of the MSFT 1 minute bar. When the BarsInProgress == 1 the close value returned is the close price of the MSFT 3 minute Bars object.

Notes:

- A multi-series script only processes bar update events from the primary Bars (the series the script is applied to) and any additional Bars objects the script adds itself. Additional Bars objects from a multi-series chart or from other multi-series scripts that may be running concurrently will not be processed by this multi-series script.

- If a multi-series script adds an additional Bars object that already exists on the chart, the script will use the preexisting series instead of creating a new one to conserve memory. This includes that series' session template as applied from the chart. If the Bars object does not exist on the chart, the session template of the added Bars object will be the session template of the primary Bars object. If the primary Bars object is using the "<Use instrument settings>" session template, then the additional Bars objects will use the default session templates as defined for their particular instruments in the Instruments window.

- In a multi-series script, **CurrentBars** starting value will be -1 until all series have processed the first bar. To ensure you have satisfied this requirement on all your Bars objects, it is recommend you start your OnBarUpdate() method with CurrentBars checks, as seen in the code sample above.

- A multi-series indicator will hold the same number of data points for plots as the primary series. Setting values to plots should be done in the primary series in OnBarUpdate(). If you are using calculations based off of a larger secondary series, it may plot like a step ladder because there are more data points available than there are actual meaningful data values.

- The default CloseStrategy() handling will only be applied to the primary series of a MultiSeries NinjaScript strategy.
- An indicator / strategy with multiple DataSeries of the same instrument will only process realtime OnBarUpdate() calls when a tick occurs in session of the trading hour templates of all added series. Any ticks not processed will be queued and processed as a tick comes in for all subsequent DataSeries.

Accessing the Price Data in a Multi-Bars NinjaScript

As you probably know already, you can access the current bar's closing price with the following statement:

```
Close[0];
```

You can also access price data such as the close price of other Bars objects at any time. This is accomplished by accessing the Opens, Highs, Lows, Closes, Volumes, Medians, Typicals and Times series by index value. These properties hold collections (containers) that hold their named values for all Bars objects in a script.

Continuing with our example code above, if you wanted to access the high price of the MSFT 3 minute Bars object at index 1 you would write:

```
Highs[1][0];
```

This is just saying "give me the series of high prices for the Bars object at index 1 'Highs[1]' and return to me the current high value '[0]'". If the BarsInProgress index was equal to 1, the current context is of the MSFT 3 min Bars object so you could just write:

```
High[0];
```

The following example demonstrates various ways to access price data.

```
protected override void OnBarUpdate()
{
     // Checks to ensure all Bars objects contain enough
bars before beginning
     // If this is a strategy, use BarsRequiredToTrade
instead of BarsRequiredToPlot
     if (CurrentBars[0] <= BarsRequiredToPlot ||
CurrentBars[1] <= BarsRequiredToPlot || CurrentBars[2] <=
BarsRequiredToPlot)
          return;

     // Checks if OnBarUpdate() is called from an update on
the primary Bars
     if (BarsInProgress == 0)
     {
          double primaryClose  = Close[0];
          double msft3minClose = Closes[1][0];
          double aapl1minClose = Closes[2][0];

          // primaryClose could also be expressed as
          // primaryClose = Closes[0][0];
     }

     // Checks if OnBarUpdate() is called from an update on
MSFT 3 minute Bars object
     if (BarsInProgress == 1)
     {
          double primaryClose  = Closes[0][0];
          double msft3minClose = Close[0];
          double aapl1minClose = Closes[2][0];
     }
}
```

Entering, Exiting and Retrieving Position Information

This section is relevant for NinjaScript strategies only. Entry and Exit methods are executed within the BarsInProgress context. Let's demonstrate with an example:

```
protected override void OnBarUpdate()
{
     // Checks to ensure all Bars objects contain enough
bars before beginning
     // If this is an indicator, use BarsRequiredToPlot
instead of BarsRequiredToTrade
     if (CurrentBars[0] <= BarsRequiredToPlot ||
CurrentBars[1] <= BarsRequiredToPlot || CurrentBars[2] <=
BarsRequiredToPlot)
          return;

     // Checks if OnBarUpdate() is called from an update on
the primary Bars
     if (BarsInProgress == 0)
     {
          // Submits a buy market order for MSFT
          EnterLong();
     }

     // Checks if OnBarUpdate() is called from an update on
AAPL 1 minute Bars object
     if (BarsInProgress == 2)
     {
          // Submits a buy market order for AAPL
          EnterLong();

          // Submits a buy market for MSFT when
OnBarUpdate() is called for AAPL
          EnterLong(0, 100, "BUY MSFT");
     }
}
```

As you can see above, orders are submitted for MSFT when BarsInProgress is equal to 0 and for AAPL when BarsInProgress is equal to 2. The orders submitted are within the context of the Bars object calling the OnBarUpdate() method and the instrument associated to the calling Bars object. There is one exception, which is the order placed for MSFT within the context of the OnBarUpdate() call for AAPL. Each order method has a variation that allows you to specify the BarsInProgress index value which enables submission of orders for any instrument within the context of another instrument.

Notes:

1. Should you have multiple Bars objects of the same instrument while using Set() methods in your strategy, you should only submit orders for this instrument to the first Bars context of that instrument. This is to ensure your order logic is processed correctly and any necessary order amendments are done properly.

2. Should you have multiple Bars objects of the same instrument while using options to terminate orders/positions at the end of the session (TIF=Day or IsExitOnSessionCloseStrategy=true), you should not submit orders to Bars objects other than the first Bars context for that instrument when on the last bar of the session. This is necessary because some of the end of session handling is applied only to the first Bars context of an instrument, and submitting orders to other Bars objects for that instrument can bypass the end-of-session handling.

3. For advanced order methods, if you **DO NOT** specify a BarsInProgress , the order will be submitted to the current bars in progress updating. If the current BarsInProgress is a higher time frame, this could delay the time that the order is filled during historical backtesting. As a result, you should always submit historical orders to the most granular of time frames.

4. When backtesting and submitting orders 'On bar close' and utilizing OnExecutionUpdate or OnOrderUpdate to submit orders, these orders will be processed immediately and filled by the fill engine depending on if the order satisfies its fill condition. This evaluation is done by looking ahead to the next bar of the current series. This is done prior to any secondary higher granularity series having a chance to run its 'OnBarUpdate' logic. If you planned on running order logic in your highest granularity added series then please insure that you submit orders in all cases to the highest granularity series.

The Position property always references the position of the instrument of the current context. If the BarsInProgress is equal to 2 (AAPL 1 minute Bars), Position would refer to the position being held for AAPL. The Positions property holds a collection of Position objects for each instrument in a strategy. Note that there is a critical difference here. Throughout this entire section we have been dealing with Bars objects. Although in our sample we have three Bars objects (MSFT 1 and 3 min and AAPL 1 min) we only have two instruments in the strategy.

MSFT position is given an index value of 0
AAPL position is given an index value of 1

In the example below, when the OnBarUpdate() method is called for the primary Bars we also check if the position held for AAPL is NOT flat and then enter a long position in MSFT. The net result of this strategy is that a long position is entered for AAPL, and then once AAPL is long, we go long MSFT.

```
protected override void OnBarUpdate()
{
     // Checks to ensure all Bars objects contain enough
bars before beginning
     // If this is an indicator, use BarsRequiredToPlot
instead of BarsRequiredToTrade
     if (CurrentBars[0] <= BarsRequiredToPlot ||
CurrentBars[1] <= BarsRequiredToPlot || CurrentBars[2] <=
BarsRequiredToPlot)
          return;

     // Checks if OnBarUpdate() is called from an update on
the primary Bars
     if (BarsInProgress == 0 && Positions[1].MarketPosition
 != MarketPosition.Flat)
     {
          // Submits a buy market order for MSFT
          EnterLong();
     }

     // Checks if OnBarUpdate() is called from an update on
AAPL 1 minute Bars
     if (BarsInProgress == 2)
     {
          // Submits a buy market order for AAPL
          EnterLong();
     }
}
```

11.5.9 NinjaScript Lifecycle

NinjaTrader uses a State change system to represent various life cycles of your NinjaScript object. For more basic indicators and strategies, simply understanding each **State** described on the OnStateChange() page is sufficient. However, for more advanced development projects, it is critical to understand how NinjaTrader calls these states for various instances throughout the lifetime of the entire application.

When NinjaTrader instantiates a NinjaScript object

There are two categories of instances instantiated by NinjaTrader:

- "UI" instances representing its default properties on various user interfaces
- The "configured" instance executing your custom instructions

In both categories, OnStateChange() is called at least twice: once to **State.SetDefaults** acquiring various default property values, and then again to **State.Terminated** handling internal references cleanup.

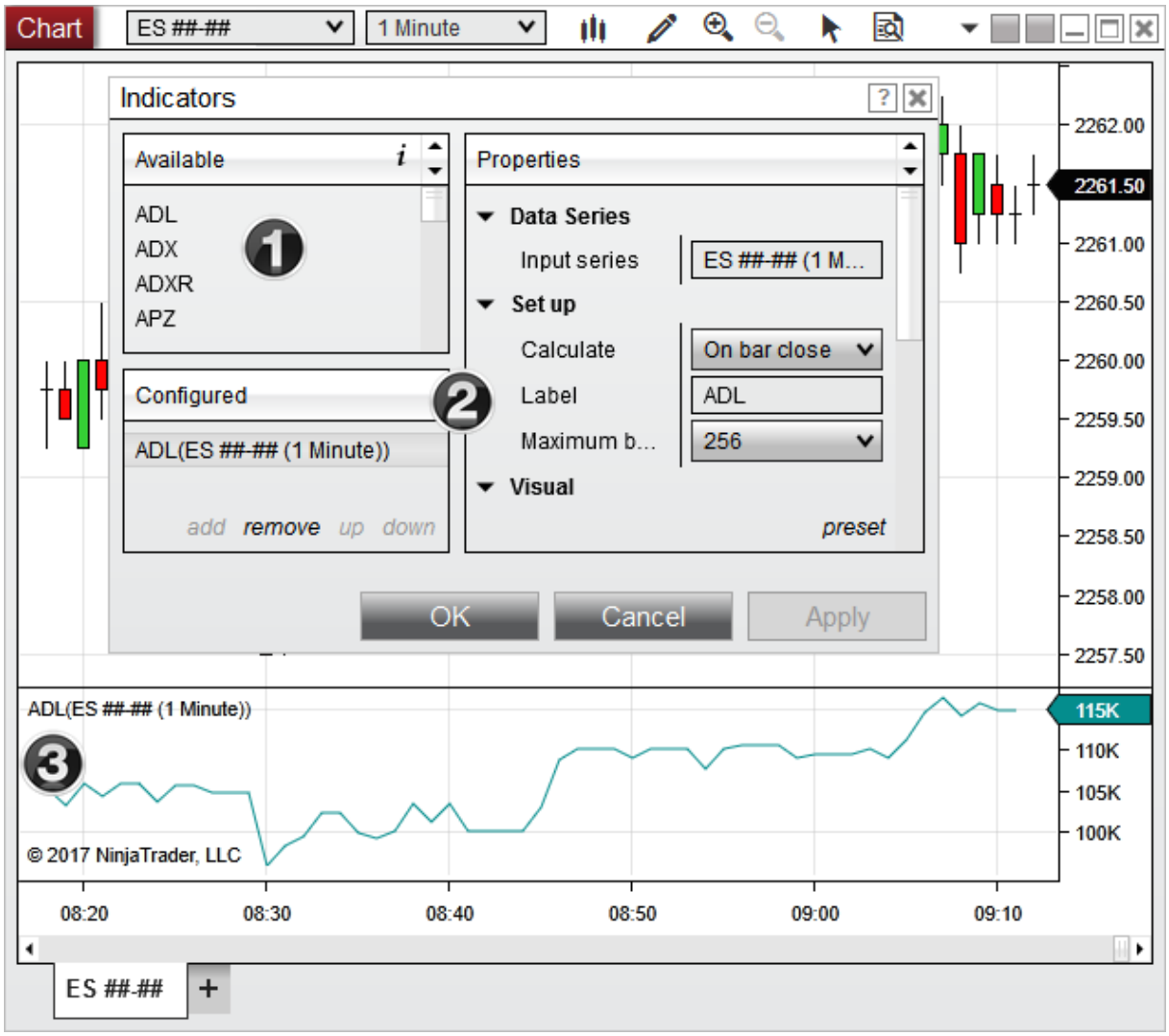
> **Note**: It is important to understand that previous major versions of NinjaTrader were not so diligent in running termination logic for UI instances and the current major NinjaTrader 8 version has been changed to help properly address related issues.

To elaborate on that process, imagine the sequence of user events required to start an indicator on a chart:

1. User right clicks on a Chart and select "**Indicator**"
2. User adds an Indicator from the **Available** list
3. User configures desired **Properties** and presses "**Apply**" or "**OK**"

During this sequence, there are actually 3 instances of the same indicator created by NinjaTrader:

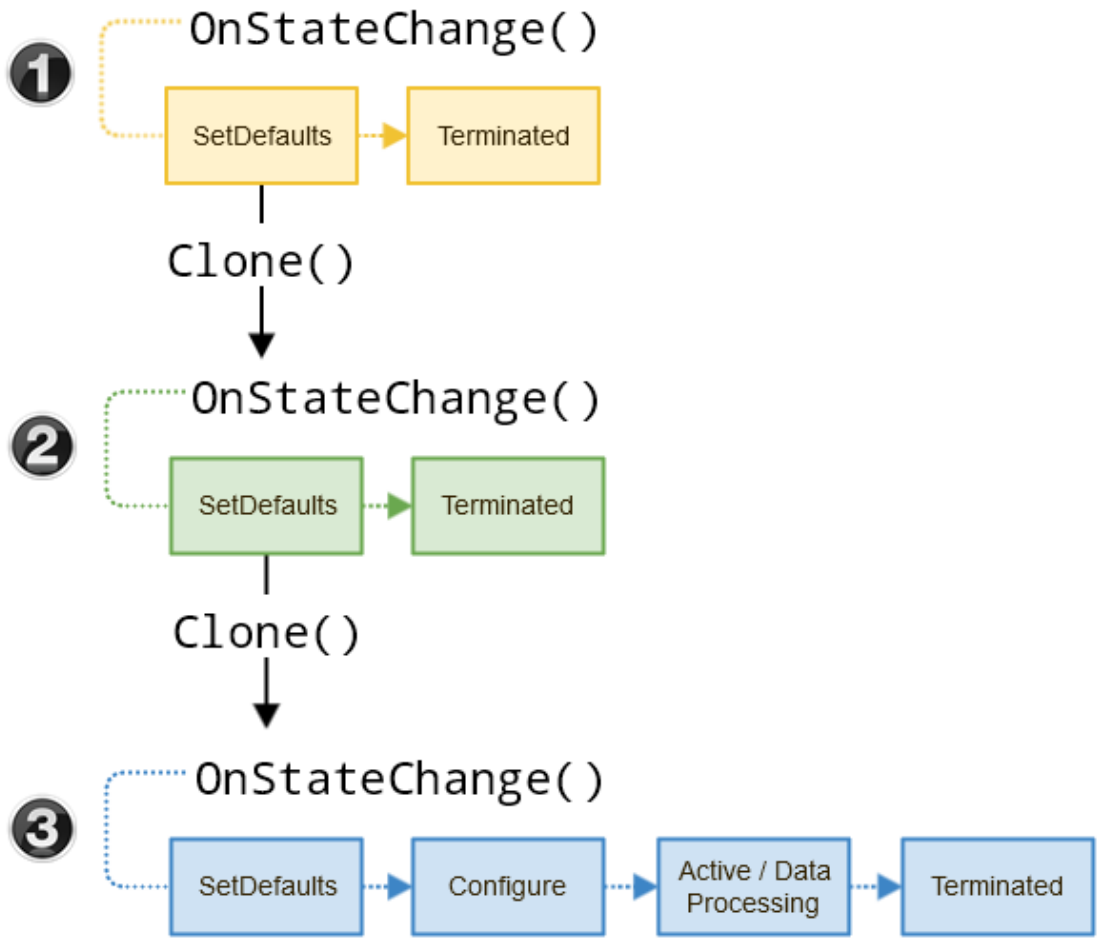
1. The instance displaying the **Name** property to the list of "**Available**" indicators (**Note**: this process involves creating an instance of *all* indicators in order to build the complete list)
2. The instance displaying the individual **Name** and its default **Properties**
3. The instance configured and executing on the chart



To visualize how each instance goes through its **States**, please consider the logic and flow chart below:

1. In order to display the indicator name in the list of **"Available"** indicators, the NinjaTrader core must find the **Name** of each installed indicator defined in their **SetDefaults**. This occurs simultaneously for *every indicator installed on the system* in order to build the full list of available indicators.
2. The selected indicator is then cloned and **SetDefaults** is called again in order to display the default properties to the "**Properties**" grid. This only occurs for the individual indicator.
3. After the user has set their desired property settings and press **OK** or **Apply**, the indicator is once again cloned and runs through its full state management. This only occurs for the indicator configured to execute on the chart.

Warning: Since NinjaTrader is multi-threaded, it is possible the **OnStateChange()** logic will be operating on a different thread than your indicator instances. Due to this fact, if logic in your **OnStateChange()** method is thread sensitivity (e.g., dependent on UI threads vs Instrument threads) please make sure to read the section on multi-threading considerations and check for thread access in your **OnStateChange()** logic



It is the 3rd "configured" instance you are concerned with developing, but you should also be aware of the "UI" instances which are triggered at various stages of NinjaTrader.

Notes:
1. The example above is written for an indicator, but the same concept of state management applies to every NinjaScript object type

2. The UI instances do not reach **State.Terminated** until the user closes out of the UI feature displaying the object
3. Since AddOns run in the background and are not dependent on UI elements, they will run through their **SetDefaults/Terminated** states after each NinjaScript compile and startup/shutdown of NinjaTrader.
4. The configured instance will also be cloned back to UI instances during various user actions (e.g, re-opening an indicator dialog to reconfigure settings, or user copying & pasting the indicator to a new panel or chart). Therefore you should not assume that objects (such as ChartControl) will not be accessible in the UI instances.
5. In some extreme scenarios, you may need to execute custom logic before or after an object is cloned. Overriding the default behavior can be done via the virtual Clone() method

What does this mean for me?

Since **OnStateChange()** can be called at various times throughout NinjaTrader, you should be diligent in handling each state and managing resources only when it is appropriate that your NinjaScript object was actually configured:

- **State.SetDefaults** should be kept as lean as possible to prevent logic from processing superfluously and causing problems unrelated to the configured instance. Only properties which need to be displayed on the UI should be set in this state.
- Resources should only be set up once an object has reached **State.Configure** or **State.DataLoaded** (see best practices for more information)
- **State.Terminated** logic should be specific in when it resets a value or destroys a resource. Since the running instance can be cloned back to a UI instance, checking that a mutable property exists before accessing sometimes is not enough. You may need to consider adding a flag to help decide when a resource needs to be reset or destroyed.

Example

Let's say your object was an indicator looking to add a custom toolbar element to the chart, and when the indicator is removed from the chart, you would want to make sure your toolbar elements are also properly removed. In the OnStateChange() handler you change could add the custom toolbar once the **State** has reached **State.Historical**, and remove the toolbar once the State has reached **State.Terminated**.

To ensure that the remove logic only runs in instances that were actually configured, you can see we in the example below we also track that the toolbar needs a reset in **State.Terminated** state via a custom bool variable. In other words, the proper reset request comes from our configured instance and would be ignored if the **State.Terminated** is called from outside our object (i.e., a UI instance). This will prepare our object to properly handle both situations in which **State.Terminated** could be called in the NinjaTrader state management system.

```
// custom flag to help time termination logic
private bool toolBarNeedsReset = false;

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "State lifetime indicator";
   }
   else if (State == State.Historical)
   {
      // before indicator starts historical processing
      // add a custom tool bar using a custom method
      AddToolBarButton(); // this is a pseudo-method for example
purposes
      toolBarNeedsReset = true; // use a flag to track this logic
was executed
   }

   else if (State == State.Terminated)
   {
      // here we intend to remove the custom tool bar when the
indicator shuts down
      if (toolBarNeedsReset) // flag is only true after actually
added
         RemoveToolBarButton();
   }
}
```

Cloning NinjaScript

Clone is the operation of iterating over all public browsable properties on a NinjaScript object and duplicating the values over to a freshly generated instance. For the majority of NinjaScript with standard properties the clone process is transparent to you and you do not need to be concerned the the clone process. For those of you that want more control or will be utilizing complex properties then knowledge about clone is essential. Cloning is performed in 2 primary use cases:

1. Configuring an instance in an object dialog and then cloning the configured data to an actual NinjaScript instance applied for example to a Chart. (Configuration then Run)
2. When triggering 'Reload NinjaScript' or "Reload All Historical Data'

NinjaScript objects have a base clone method implemented which will iterating over all browsable properties and copy by value to the next instance. The rules follow the 'clone' rules described in the clone documentation located here and described above. The default behavior will work in almost all cases except for when you have some complex custom property which

needs specific clone behavior. In which case we allow the ability to override Clone() and specify your own behavior.

> **Note**: If you plan to utilize complex class properties on NinjaScript, you can specify your own clone method. However when NinjaScript is compiled in NinjaTrader a new DLL holding the compiled IL code is 'hot-loaded' into NinjaTrader. As a user or developer would try to reload NinjaScript or configure an existing NinjaScript object, any complex class will not resolve since the class will be residing in two different assemblies. This problem cannot be solved with custom clone method and workarounds for this are setting Browsable(false) attribute on that property so it is not cloned or putting the property it its own dedicated assembly.

Saving NinjaScript Properties to the Workspace via XML Serialization

XML Serialization comes into play when you have a set of properties and want those properties to persist the user saved workspace (or any templates that are user created).

By default basic types such as int, string, bool will all serialize without issue, if you have a complex property you want its setting maintained on restore then you need to create a string serialized representation of that class. The technique is shown in this example post here where we show how to serialize a color brush.

11.5.10 Using 3rd Party Indicators

3rd Party Indicators Overview

You can use 3rd party indicators within your strategies or custom indicators. A 3rd party indicator is an indicator that was not developed by NinjaTrader.

> **Note**: It is important to understand the functionality provided or **NOT** provided in a 3rd party proprietary indicator. Just because they provide an indicator that displays a bullish or bearish trend on a chart does **NOT** mean that you can access this trend state from their indicator. It is up to the developer of the indicator to determine what information is accessible.

3rd party indicators can be provided to you in one of the following ways:

- NinjaScript archive file that can be directly imported into NinjaTrader
- A custom installer
- A set of files and instructions for saving them in the correct folders

If you were provided with a NinjaScript archive file that you have successfully imported via the Control Center window "File > Utilities > Import NinjaScript" menu, you can skip over the information below since NinjaTrader automatically configures the indicators ready for use.

If you were provided with a custom installer or a compiled assembly (.DLL) file that you had to manually save in the folder My Documents\NinjaTrader Folder>\bin\Custom then you must follow the instructions below.

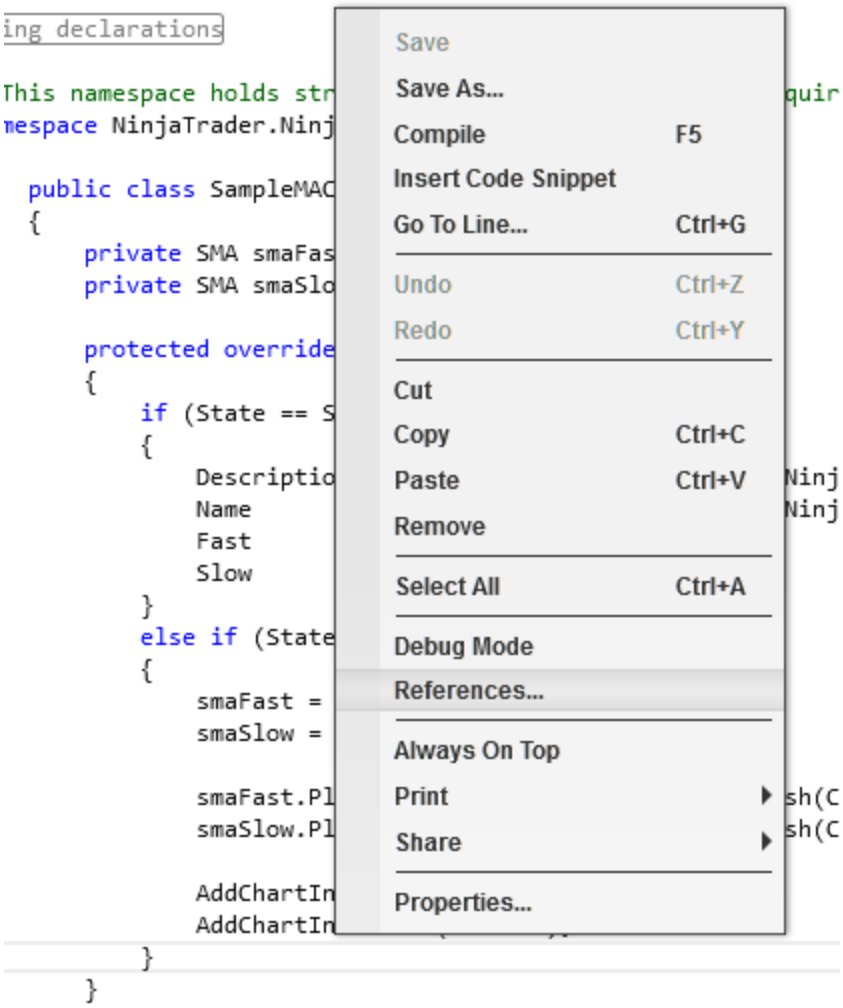
Vendor File

The 3rd party developer should have either installed a "Vendor" file or provided you with one. Its likely in the format "NinjaTrader.VendorName.cs" where VendorName is the name of the 3rd party vendor. This file allows you to conveniently access their indicators.

- If you were provided an installer, you can check with the vendor if this file was included or;
- If they provided you this file, save it to "My Documents\<NinjaTrader Folder>\bin\Custom" and restart NinjaTrader

Adding a Reference

1. From within the NinjaScript Editor, right click on your mouse to bring up the context menu and select the sub-menu **References...** as per the image to the right.



2. A **References** window will appear

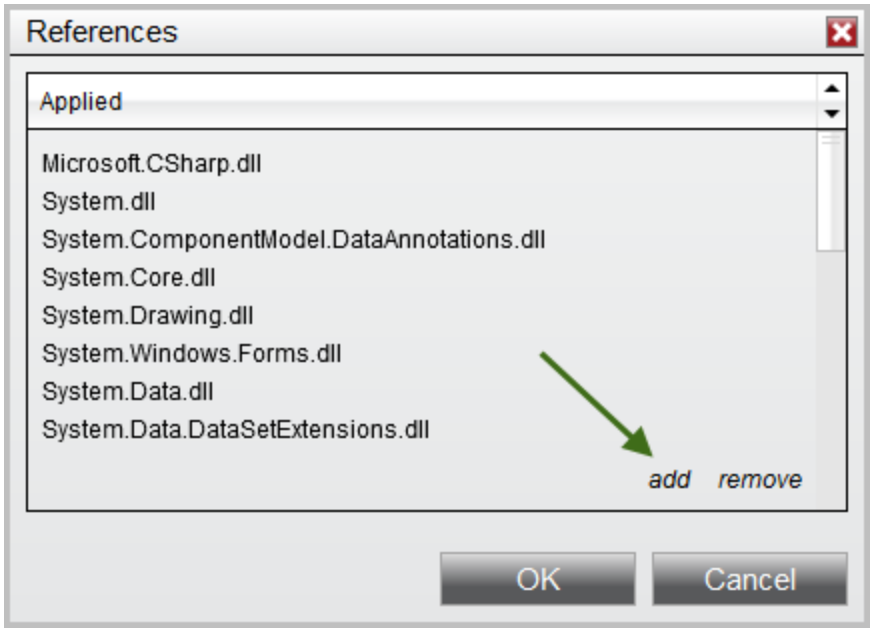
3. Press the ***"add"*** and select the 3rd party vendor DLL file

> **Warning**: Please make sure in this step to select only the 'true' DLL file needed for reference, which would not contain any X86 or X64 suffixes in its file-name, otherwise you could run into compile issues later.

4. You will see a reference to the 3rd party vendor DLL in the **References** window

5. Press the **OK** button

You will now be able to access the indicator methods provided by the 3rd party vendor



11.5.11 Using ATM Strategies

You can create an automated strategy that generates a trade signal that executes a NinjaTrader ATM Strategy.

- **ATM Strategies** operate in real-time only and will not execute on historical data thus they can't be backtested
- Executions resulting from an **ATM Strategy** that is created from within a NinjaScript automated strategy will not plot on a chart during real-time operation
- Strategy set up parameters such as EntriesPerDirection, EntryHandling, IsExitOnSessionCloseStrategy do not apply when calling the AtmStrategyCreate() method

- Executions from **ATM Strategies** will not have an impact on the hosting NinjaScript strategy position and PnL - the NinjaScript strategy hands off the execution aspects to the ATM, thus no monitoring via the regular NinjaScript strategy methods will take place (also applies to strategy performance tracking)
- **ATM Strategy** stop orders can either be StopMarket or StopLimit orders, depending on which type is defined in the ATM Strategy Template (Advanced Options) you call in the AtmStrategyCreate() method in your NinjaScript strategy. To make the distinction clear which is used, following a naming convention for the template name is highly suggested (i.e. AtmStrategyTemplate_STPLMT)
- A general sample for calling ATM's is preinstalled with NinjaTrader under the 'SampleATMStrategy' script - for a script showing how to implement reversal type setups, please see this link to our online resources.

There is a Clear Line...

There is a clear line between a NinjaScript Strategy and an **ATM Strategy**. The use model for creating an **ATM Strategy** within a NinjaScript Strategy is when you want to programmatically monitor and generate an entry signal and then manualy manage the resulting open position via an ATM Strategy in one of NinjaTrader's order entry windows.

!!! IMPORTANT: Manually Closing an ATM Strategy from an Order Entry Window such as the SuperDOM

It is crucial that when running **ATM Strategies** created by a NinjaScript strategy that you understand how to properly manually close the **ATM Strategy** from any of the order entry windows.

- If the order entry window ATM Strategy Selection Mode is NOT in "**DisplaySelectedATMStrategyOnly**" click on the "**CLOSE**" button via your middle mouse button (scroll wheel)
- If the order entry window **ATM Strategy Selection Mode** is in "**DisplaySelectedATMStrategyOnly**" you can click on the "**CLOSE**" button with your left mouse button to close the selected active **ATM strategy**

Following the approaches above will internally close the **ATM Strategy**. Not following the approach will close the account/instrument position, terminate all strategies and cancel all orders. The result is that your NinjaScript strategy will be terminated.

11.5.12 Using BitmapImage Objects with Buttons

Images as Buttons Overview

BitmapImage objects can be used to apply an image as a background to a Button object added to a NinjaTrader window.

> **Note**: The following topic covers methods and properties outside of the NinjaScript

libraries. Most of the items covered in the example below belong to .NET's System.Windows.Media.Imaging and System.Windows.Controls namespaces. More information on these namespaces can be found at the links below:

- System.Windows.Controls
- System.Windows.Media.Imaging

Using an image as the background for a button can be achieved through a fairly straightforward process using some of the .NET framework's Controls and Imaging methods

There are a few best practices to keep in mind when working with Buttons:

- Dispose of any leftover objects in State.Terminated for efficient memory use
- Use your object's main Dispatcher when adding or removing Buttons to or from your chart, to ensure that the correct thread is used
- Be aware of the proper States in which to initialize objects related to the Button (State.Configure), apply the Button (State.Historical), and dispose of unneeded objects (State.Terminated)

Adding a Button to a Chart Toolbar Using an Image as the Background

The example below walks through the process of adding a Button to a chart toolbar specifically, and applying a .jpg image as the Button's background. This example also displays several best practices when working with Buttons, such as proper object disposal and ensuring that the Button is not populated when the indicator is applied in an inactive chart tab.

```
//Add the following Using statements
using System.Windows.Media.Imaging;
using System.Windows.Controls;


public class addButton : Indicator
{
    // Define a Chart object to refer to the chart on which the
indicator resides
    private Chart chartWindow;

    // Define a Button
    private System.Windows.Controls.Button myButton = null;

    // Instantiate a BitmapImage to hold an image
    BitmapImage myBitmapImage = new BitmapImage();

    // Instantiate an ImageBrush to apply to the Button
    ImageBrush backgroundImage = new ImageBrush();

    private bool IsToolBarButtonAdded;

    protected override void OnStateChange()
    {
        if (State == State.Configure)
        {
            // Assign an image on the filesystem to the
BitmapImage.
            // This example assumes that a jpg image named
"ButtonBackground" resides in the install directory
            myBitmapImage.BeginInit();
            myBitmapImage.UriSource = new
Uri(NinjaTrader.Core.Globals.InstallDir + "ButtonBackground.jpg");
            myBitmapImage.EndInit();

            // Assign the BitmapImage as the ImageSource of the
ImageBrush
            backgroundImage.ImageSource = myBitmapImage;
        }
        else if (State == State.Historical)
        {
            //Call the custom addButtonToToolbar method in
State.Historical to ensure it is only done when applied to a chart
            // -- not when loaded in the Indicators window
            if (!IsToolBarButtonAdded) AddButtonToToolbar();
        }
        else if (State == State.Terminated)
        {
            //Call a custom method to dispose of any leftover
objects in State.Terminated
            DisposeCleanUp();
        }
    }
```

11.5.13 Using Historical Bid/Ask Series

Historical Bid/Ask Series Overview

NinjaTrader has the ability to use historical bid and ask price series in your NinjaScript instead of only being able to use a last price series. The following outlines the intricacies of this capability:

Notes:
- You can have multiple bid/ask/last series in your NinjaScript indicator/strategy. Please use the AddDataSeries() method to add these series to your script.
- The historical bid/ask series holds *all* bid/ask events sent out by the exchange. This would *not* be equivalent to the bid/ask at a specific time a trade went off.
- When processing your NinjaScript, the historical bid/ask series would have the historical portion triggered in the OnBarUpdate() method only. OnMarketData() method events for the historical bid/ask series would only be triggered in real-time.

Tips:
- For using **OnMarketData()** events historically, please see the educational topic on Developing for Tick Replay
- Changing the price type used for the primary Bars object to which a script is applied can be done in the Data Series window from any open chart.

Accessing Bid/Ask Series

When calling AddDataSeries() to add an additional Bars object to your script, a constructor overload will be available which takes a MarketDataType enumeration as an argument. This will allow you to specify the price series which will be used in that particular object. If you were to pass in MarketDataType.Ask or MarketDataType.Bid, as in the example below, that particular data series will use that price type for all of its PriceSeries collections, such as Close, Open, High, and Low.

Warning: A **Tick Replay** indicator or strategy **CANNOT** use a **MarketDataType.Ask** or **MarketDataType.Bid** series. Please see Developing for Tick Replay for more information.

Example

```
protected override void OnStateChange()
{
    if (State == State.Configure)
    {
        // Add an AAPL data series using the Ask series
        AddDataSeries("AAPL", BarsPeriodType.Minute, 30,
MarketDataType.Ask);

        //Add another AAPL data series using the Bid series, with
other settings identical
        AddDataSeries("AAPL", BarsPeriodType.Minute, 30,
MarketDataType.Bid);
    }
}
```

11.5.14 Using Images and Geometry with Custom Icons

Custom Icon Overview

When overriding the Icon method in a Share Service, Drawing Object, or Chart Style, you can use a variety of inputs to specify what will be displayed on the icon, including UniCode characters (if they exist in the icon pack for the font family used in NinjaTrader), custom Geometry Paths from the System.Windows.Shapes namespace, or image files. Using an image file for a custom icon can allow the flexibility of creating your icon's visuals outside of your code via image editing software. For more information about adding custom Icons, see the "Icon" page under the topics for each of the NinjaScript object types listed above.

▼ Using an Image as an Icon

Using an Image as an Icon

The process for using an image as an icon is fairly straightforward using WPF objects, and is the same for different NinjaScript objects.

1. Instantiate a new BitmapImage object
2. Assign a Uri to the BitmapImage, pointing to an image file
3. Instantiate a Grid of the same dimensions as the icon
4. Instantiate an Image object
5. Assign the BitmapImage as the Image's Source
6. Add the Image to the Grid
7. Return the Grid by overriding the Icon property

Note: Be careful to instantiate the Grid to be same size as the needed icon.

Some icon sizes differ from others. For example, the icon for Share Services is substantially larger than the icon for a Chart Style in the Chart Toolbar.

```
// Add the following Using statements
using System.Windows.Controls;
using System.Windows.Media;
using System.Windows.Media.Imaging;

BitmapImage iconBitmapImage = new BitmapImage();

protected override void OnStateChange()
{
   if (State == State.Configure)
   {
      // Set the BitmapImage's UriSource to the location of
an image file
        iconBitmapImage.BeginInit();
        iconBitmapImage.UriSource = new
Uri(NinjaTrader.Core.Globals.InstallDir + "icon.jpg");
        iconBitmapImage.EndInit();
   }
}

// Override Icon (read-only) to return the custom Grid and
Image
public override object Icon
{
    get
    {
       // Instantiate a Grid on which to place the image
        Grid myCanvas = new Grid { Height = 16, Width =
16 };

        // Instantiate an Image to place on the Grid
        Image image = new Image
        {
            Height = 16,
            Width = 16,
            Source = iconBitmapImage
        };

        // Add the image to the Grid
        myCanvas.Children.Add(image);

        return myCanvas;
    }
}
```

Using Geometry on an Icon

Using Geometry on an Icon

Custom geometry Paths can be used to draw and fill custom shapes, which can then be applied directly to a Canvas returned for use in an Icon. The process for using a Path is similar to that for using an Image:

1. Instantiate a new Path object
2. Instantiate a Grid of the same dimensions as the icon
3. Define the visual properties of the Path
4. Add the Path to the Grid
5. Return the Grid by overriding the Icon property

```
// Add the following namespace to use Path objects
using System.Windows.Shapes;
using System.Windows.Controls;

public override object Icon
{
    get
    {
        // Instantiate a Grid on which to place the Path
        Grid myCanvas = new Grid { Height = 16, Width =
16 };

        // Instantiate a Path object on which to draw
geometry
        System.Windows.Shapes.Path myPath = new
System.Windows.Shapes.Path();

        // Define the Path's visual properties
        myPath.Fill = Brushes.Red;
        myPath.Data =
System.Windows.Media.Geometry.Parse("M 0 0 L 5 0 L 5 5 L 10
5 L 10 0 L 15 0 L 15 5 L 10 5 L 10 10 L 5 10 L 5 5 L 0 5
Z");

        // Add the Path to the Canvas, then return the
Canvas
        myCanvas.Children.Add(myPath);
        return myCanvas;
    }
}
```

11.5.15 Using SharpDX for Custom Chart Rendering

Understanding the SharpDX .NET Library

NinjaTrader Chart objects (such as Indicators, Strategies, DrawingTools, ChartStyles) implement an OnRender() method aimed to render custom lines, shapes, and text to the chart. To achieve the level of performance required to keep up with market data events, NinjaTrader uses a 3rd-party open-source .NET library named SharpDX. This 3rd party library provides a C# wrapper for the powerful Microsoft DirectX API used for graphics processing and known for its hardware-accelerated performance, including 2D vector and text layout graphics used for **NinjaTrader Chart Rendering**. The SharpDX/DirectX library is extensive, although NinjaTrader only uses a handful of namespaces and classes, which are documented as a guide in this reference. In addition to this educational resource, we have also compiled a more focused collection of SharpDX SDK Reference resources to help you learn the **SharpDX** concepts used in **NinjaTrader Chart Rendering**.

Tips:

1. There are several pre-installed examples of **OnRender()** and **SharpDX** objects used in the **NinjaTrader.Custom** project. For starters, please look at the **SampleCustomRender** indicator file
2. Although not entirely identical, the **SharpDX** wrapper is designed to resemble **System.Drawing** namespace; experienced GDI developers will be familiar with concepts discussed in this section.
3. Microsoft provides various DirectX Programming Guides aimed to educate users with the underlying **C++ DirectX API**. While **SharpDX (C#)** syntax is different, you may find these guides helpful for understanding **SharpDX** concepts not offered by this guide.

There are three main **SharpDX** namespaces you need to be familiar with:

SharpDX	Contains basic objects used by SharpDX.
SharpDX.Direct2D1	Contains objects used for rendering for 2D geometry, bitmaps, and text.
SharpDX.DirectWrite	Contains objects used for text rendering

The rest of this page will help you navigate the fundamental concepts needed to achieve custom rendering to your charts.

SharpDX Vectors and Charting Coordinates

Understanding the SharpDX.Vector2

SharpDX Draw methods use a SharpDX.Vector2 object which describes where to render a command relative to the chart panel. These **Vector2** objects can be thought as a two-dimensional point in the chart panels X and Y axis. Since the chart canvas used to draw on consists of the full panel of the chart, a vector using a value of 0 for both the X and Y coordinates would be located in the top left corner of the chart:

```
// creates a vector located at the top left corner of the
chart
float x = 0;
float y = 0;
SharpDX.Vector2 myVector2 = new Vector2(x, y);
```

Tip: You can learn about Understanding Chart Canvas Coordinates on another topic

Vector2 objects contain **X** and **Y** properties helpful to recalculate new properties based on the initial vector:

```
float width = endPoint.X - startPoint.X;
float height = endPoint.Y - startPoint.Y;
```

Additionally, you can recalculate a new vector from existing vector objects:

```
SharpDX.Vector2 center = (startPoint + endPoint) / 2;
```

It is also helpful to know that **Vector2** objects are similar to the Windows Point structure and these two types can be used interchangeably. Depending on the mechanism used to obtain user input or other application values, you may receive the coordinates in a **Point**. For convenience, NinjaTrader provides a DXExtension.ToVector2() method used for converting between these two objects if needed:

```
SharpDX.Vector2 dxVector2 = wpfPoint.ToVector2();
```

Calculating Chart Coordinates

If you simply used a vector with static values, your **Vector2** objects would never change, and your drawing would remain fixed on a particular area of the chart (which may be desired). However, since NinjaTrader charts are dynamic and responded to various market data updates, scroll, resize, and scale operations - you also need a way to recalculate **vectors** to display information dynamically. To assist in this process, NinjaTrader provides some GUI related utilities to help navigate the chart and calculate values for your custom rendering.

```
// creates a vector located at the top left corner of the
chart panel
startPoint = new SharpDX.Vector2(ChartPanel.X,
ChartPanel.Y);

// creates a vector located at the bottom right corner of
the chart panel
endPoint = new SharpDX.Vector2(ChartPanel.X + ChartPanel.W,
ChartPanel.Y + ChartPanel.H);
```

Common utilities fall under 4 key components, and you can learn more about their specific functions from the help guide topics linked in the table below:

ChartControl	The entire hosting grid of the Chart
ChartBars	The primary bars series configured on the Chart
ChartPanel	The panel on which the calling script resides
ChartScale	The Y-Axis values of the configured ChartPanel

Note: For full absolute device coordinates always use **ChartPanel** X, Y, W, H values. **ChartScale** and **ChartControl** properties return WPF units, so they can be drastically different depending on DPI of the user's display. You can learn about Working with Pixel Coordinates on another topic.

SharpDX Brush Resources

Understanding SharpDX Brush Resources

To color or "paint" an area of the chart, you must define custom resources which describe how you wish the custom render to appear. **SharpDX** contains special resources modeled after the familiar WPF Brushes. However, the two objects are different in the way they are constructed and also in how they are managed after they are used.

There are many types of **SharpDX Brush Resources** which all derive from the same base Direct2D1.Brush class. This base object is not enough to describe how your object should be presented, so in order to use a brush for rendering purposes, you will need to determine exactly what type of brush you wish to use:

Direct2D1.SolidColorBrush	Paints an area with a solid color.
Direct2D1.RadialGradientBrush	Paints an area with a radial gradient.
Direct2D1.LinearGradientBrush	Paints an area with a linear gradient.

Describing SolidColorBrush Colors

The most common and simple brush to use is a Direct2D1.SolidColorBrush which allows you to paint using a solid color (or with transparency). In the most basic form, **SolidColorBrush** can be constructed using a predefined SharpDX.Color

```
SharpDX.Direct2D1.SolidColorBrush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue);
```

You can also use a SharpDX.Color3 or SharpDX.Color4 structure as a way to get more customizable colors in your rendering:

```
// create a 3 component color using rgb values in float notation
SharpDX.Color3 dxColor3 = new SharpDX.Color3(1.0f, 0.0f, 0.0f);

// create a 4 component color using rgb + alpha (transparency) in float notation
SharpDX.Color4 dxColor4 = new SharpDX.Color4(dxColor3, 0.5f);

// solid color brush uses a Color4 during construction
SharpDX.Direct2D1.SolidColorBrush argbColorBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, dxColor4);
```

Alternatively, you can set the "transparency" of an existing brush by accessing its Opacity property:

```
customDXBrush.Opacity = .25f;
```

Note: Unlike their WPF counterparts, **SharpDX** brushes are thread-safe and do **NOT** need to be frozen.

Converting SharpDX Brushes

SharpDX Brushes are **device-dependent resources**, which means they can only be used with the device (i.e., RenderTarget) which created them. In practice, this mean you should **ONLY** create your **SharpDX** brushes during the chart object's OnRender() or OnRenderTargetChanged() methods.

Warning: Failure to create device-dependent resources during the **OnRender()** or **OnRenderTargetChanged()** can lead to a host of issues including memory and application corruption which can negatively impact the stability of NinjaTrader. Please be careful your **SharpDX** device-dependent resources are only created and updated during either of these two run-time

methods. Please see the Best Practices for SharpDX Resources section on this page for more information.

Because of this detail, a common problem you may run into is the requirement to share a **SharpDX** device brush resource with a **WPF** application brush. For example, you may have **WPF** brushes defined in the UI during OnStateChange() or recalculated conditionally during OnBarUpdate(), but ultimately wish to use also in custom rendering routines. For convenience, NinjaTrader provide a DXExtension.ToDxBrush() method used for converting these objects if necessary:

```
areaBrushDx = areaBrush.ToDxBrush(RenderTarget);
smallAreaBrushDx = smallAreaBrush.ToDxBrush(RenderTarget);
textBrushDx = textBrush.ToDxBrush(RenderTarget);
```

Note: If you are using a large number of brushes, and are not tied to WPF resources, you should favor creating the **SharpDX Brush** directly since the ToDxBrush() method can lead to performance issues if called too frequently during a single render pass. Please see the Best Practices for SharpDX Resources section on this page for more information.

SharpDX RenderTarget

Understanding the RenderTarget

A SharpDX Render Target is a general purpose object resource used for receiving and executing drawing commands. When using a NinjaTrader chart object, a pre-constructed Chart RenderTarget object is available for you to use and ready to receive commands. You can think of the **RenderTarget** as the device context you are using to render to (i.e. the Chart Panel). While there is nothing special you need to do to setup this resource, it is important to understand some details regarding the **RenderTarget** to learn how it can be used.

The **RenderTarget** is primarily used for executing commands such as drawing shapes or text:

```
RenderTarget.DrawLine(startPoint, endPoint, areaBrushDx)
```

It is commonly used for creating various resources such as **Brushes** and other **SharpDX** objects:

```
SharpDX.Direct2D1.SolidColorBrush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue);
```

It can also be used to set various properties to describe how the **RenderTarget** should render:

```
RenderTarget.AntialiasMode    =
SharpDX.Direct2D1.AntialiasMode.PerPrimitive;
```

Sequencing RenderTarget commands

If the sequence in which objects render is essential to your custom rendering, you will need to be mindful of the order in which you call various **RenderTarget** members. For example, we can draw a second line which uses a different AntialiasMode and the renders each line in the order the render target received its commands:

```
RenderTarget.AntialiasMode =
SharpDX.Direct2D1.AntialiasMode.Aliased;
RenderTarget.DrawLine(startPoint, endPoint, areaBrushDx,
8);

RenderTarget.AntialiasMode =
SharpDX.Direct2D1.AntialiasMode.PerPrimitive;
RenderTarget.DrawLine(startPoint, endPoint, customDXBrush,
2);
```

In the above example, this order of operations would result in the second RenderTarget.DrawLine() to be rendered "on top" of the first **RenderTarget.DrawLine().** If you instead called these two methods in reverse order, you would not see the thinner line since it would be covered up by the thicker line.

Note: It is important to realize that **RenderTarget sequencing** and the Chart Object ZOrder are two different concepts. The **ZOrder** property controls the overall layer your entire chart object appears relative to other chart objects existing on the same chart. **RenderTarget sequencing** only affects the order objects are rendered relative itself. Therefore, it is not possible to sequence your chart object's **RenderTarget** to draw on two different **ZOrders** (e.g., one line above chart bars and another line below).

Using the RenderTarget with Device Resources

Throughout the lifetime of a chart, the render target is created and destroyed several times to satisfy various user commands. As a result, any resources that are created need to be recreated and destroyed as that render target is updated. The NinjaTrader OnRenderTargetChanged() method was designed to help with this process and will be called anytime the **RenderTarget** has changed. You should use this method if you have objects which are passed around from various other resources.

Warning: Failure to create device-dependent resources during the **OnRender()** or **OnRenderTargetChanged()** can lead to a host of issues including memory and application corruption which can negatively impact the stability of NinjaTrader. Please be careful your **SharpDX** device-dependent resources are only created and updated during either of these two run-time methods. Please see the Best Practices for SharpDX Resources section on this page for more information.

SharpDX Lines and Shapes

RenderTarget Draw Methods

All drawings consistent of a few basic shapes which can be called through a handful of **RenderTarget** commands. "Draw..." methods create just the outline of the shape, and "Fill..." will paint the interior of the shape.

RenderTarget.DrawEllipse()	Draws the outline of the specified ellipse using the specified stroke style.

RenderTarget.DrawGeometry()	Draws the outline of the specified geometry using the specified stroke style.
RenderTarget.DrawLine()	Draws a line between the specified points.
RenderTarget.DrawRectangle()	Draws the outline of a rectangle that has the specified dimensions and stroke style.
RenderTarget.FillEllipse()	Paints the interior of the specified ellipse.
RenderTarget.FillGeometry()	Paints the interior of the specified geometry.
RenderTarget.FillRectangle()	Paints the interior of the specified rectangle.

Note: AntialiasMode.PerPrimitive allows for graphics to render more sharply, but comes at a performance cost. It is recommended to set the RenderTarget.AntialiasMode back to the default **AntialiasMode.Aliased** after you finish your **RenderTarget** Draw command. Please see the Best Practices for SharpDX Resources section on this page for more information.

Line

The simplest shape is a Line, executed by the RenderTarget.DrawLine() command which just takes two Vector2 objects which describe where to draw the line, and (optionally) the width of the line to draw:

```
// create two vectors for the line to draw
SharpDX.Vector2 startPoint =  new
SharpDX.Vector2(ChartPanel.X, ChartPanel.Y);
SharpDX.Vector2 endPoint = new SharpDX.Vector2(ChartPanel.X
 + ChartPanel.W, ChartPanel.Y + ChartPanel.H);

// define the brush used in the line
SharpDX.Direct2D1.SolidColorBrush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue);

// execute the render target draw line with desired values
RenderTarget.DrawLine(startPoint, endPoint, customDXBrush,
2);

// always dispose of a brush when finished
customDXBrush.Dispose();
```

Rectangle

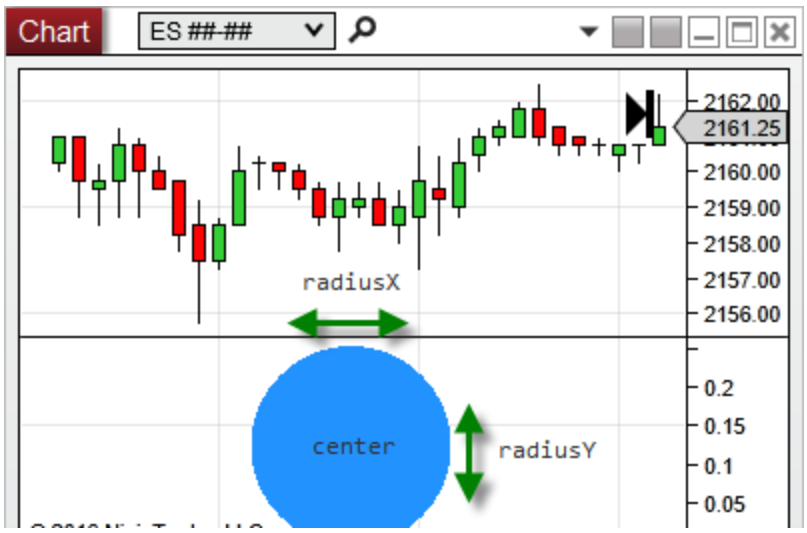
Using either the RenderTarget.FillRectangle() or RenderTarget.DrawRectangle() requires a SharpDX.RectangleF structure, constructed using four values to represent the location (x, y) and size (width, height) of the rectangle to draw.

```
// create two vectors to position the rectangle
SharpDX.Vector2 startPoint =  new
SharpDX.Vector2(ChartPanel.X, ChartPanel.Y);
SharpDX.Vector2 endPoint = new SharpDX.Vector2(ChartPanel.X
 + ChartPanel.W, ChartPanel.Y + ChartPanel.H);

// calculate the desired width and heigh of the rectangle
float width = endPoint.X - startPoint.X;
float height = endPoint.Y - startPoint.Y;

// define the brush used in the rectangle
SharpDX.Direct2D1.SolidColorBrush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue);

// construct the rectangleF struct to describe the with
position and size the drawing
SharpDX.RectangleF rect = new
SharpDX.RectangleF(startPoint.X, startPoint.Y, width,
height);

// execute the render target fill rectangle with desired
values
RenderTarget.FillRectangle(rect, customDXBrush);

// always dispose of a brush when finished
customDXBrush.Dispose();
```

Ellipse

Similar to the **Rectangle**, you can draw an **Ellipse** (or circle) using either the RenderTarget.FillEllipse() or RenderTarget.DrawEllipse() methods using a SharpDX Direct2D1 Ellipse struct. For this structure, you will need to use a Vector2 object to determine the **Center** position of the ellipse, a **RadiusX,** and a **RadiusY** which determines the size of the ellipse:

```
// create two vectors to position the ellipse
SharpDX.Vector2 startPoint =  new
SharpDX.Vector2(ChartPanel.X, ChartPanel.Y);
SharpDX.Vector2 endPoint = new SharpDX.Vector2(ChartPanel.X
 + ChartPanel.W, ChartPanel.Y + ChartPanel.H);

// calculate the center point of the ellipse from start/end
points
SharpDX.Vector2 centerPoint = (startPoint + endPoint) / 2;

// set the radius of the ellipse
float radiusX = 50;
float radiusY = 50;

// construct the rectangleF struct to describe the position
and size the drawing
SharpDX.Direct2D1.Ellipse ellipse = new
SharpDX.Direct2D1.Ellipse(centerPoint, radiusX, radiusY);

// define the brush used in the rectangle
SharpDX.Direct2D1.SolidColorBrush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue);

// execute the render target fill ellipse with desired
values
RenderTarget.FillEllipse(ellipse, customDXBrush);

// always dispose of a brush when finished
customDXBrush.Dispose();
```



Geometry

For more complicated shapes, you can use the RenderTarget.FillGeometry() or RenderTarget.DrawGeometry() methods using a Direct2D1.PathGeometry object, which is ultimately defined by a Direct2D1.GeometrySink interface.

Warning: Any **SharpDX PathGeometry** object used in your development must be disposed of after they have been used. NinjaTrader is **NOT** guaranteed to dispose of these resources for you! Please see the Best Practices for SharpDX Resources section on this page for more information.

To describe a **PathGeometry** object's path, use the object's PathGeometry.Open() method to retrieve an **GeometrySink**. Then, use the **GeometrySink** to populate the geometry with figures and segments. To create a figure, call the GeometrySink.BeginFigure() method, specify the figure's start point, and then use its Add methods (such as GeometrySink.AddLine()) to add segments. When you are finished adding segments, call the GeometrySink.EndFigure() method. You can repeat this sequence to create additional figures. When you are finished creating figures, call the GeometrySink.Close() method.

```
// create three vectors to position the geometry
SharpDX.Vector2 startPoint =  new
SharpDX.Vector2(ChartPanel.X, ChartPanel.Y);
SharpDX.Vector2 endPoint = new SharpDX.Vector2(ChartPanel.X
 + ChartPanel.W, ChartPanel.Y + ChartPanel.H);
SharpDX.Vector2 centerPoint = (startPoint + endPoint) / 2;

// create the PathGeometry used by the RenderTarget
Fill/Draw method
SharpDX.Direct2D1.PathGeometry trianglePathGeometry   = new
 SharpDX.Direct2D1.PathGeometry(Core.Globals.D2DFactory);

// retrieve the GeometrySink used to describe the
PathGeometry
SharpDX.Direct2D1.GeometrySink geometrySink    =
trianglePathGeometry.Open();

// create the points used to define the GeometrySink
SharpDX.Vector2 beginPoint = new
SharpDX.Vector2(centerPoint.X, startPoint.Y);

// Create a figure using the beginPoint
geometrySink.BeginFigure(beginPoint,
SharpDX.Direct2D1.FigureBegin.Filled);

// add lines to the figure
SharpDX.Vector2 line1 = new SharpDX.Vector2(endPoint.X,
centerPoint.Y);
geometrySink.AddLine(line1);
SharpDX.Vector2 line2 = new SharpDX.Vector2(centerPoint.X,
endPoint.Y);
geometrySink.AddLine(line2);

// end and close figure when finished
geometrySink.EndFigure(SharpDX.Direct2D1.FigureEnd.Closed);
geometrySink.Close();

// define the brush used in the geometry
SharpDX.Direct2D1.SolidColorBrush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue);

// execute the render target fill geometry with desired
values
RenderTarget.FillGeometry(trianglePathGeometry,
customDXBrush);
```

Tip: For more examples of using **Shapes** for custom rendering, many of the DrawingTools included in the **NinjaTrader.Custom** project use these types of **SharpDX** objects and methods extensively.

SharpDX Text Rendering

Using SharpDX for rendering Text

Up until this point, we have been using the SharpDX.Direct2D1 namespace to render shapes. When dealing with text, there is a separate SharpDX.DirectWrite namespace which works along with the **Direct2D1** objects.

There are two principle objects used for text rendering: A **TextFormat** object which sets the style of the text, and a **TextLayout** object used to construct complex texts with various settings and provides metrics for measuring the shape the formatted text.

Each one of these objects has their own **RenderTarget** methods: RenderTarget.DrawText() for simple **TextFormat** objects and RenderTarget.DrawTextLayout() for more advanced layouts. Both methods accept a **TextFormat** object; **DrawTextLayout** is more complicated but has better performance since it reuses the same text layout which does not need to be recalculated.

Tip: Both the **TextFormat** and **TextLayout** objects require a **DirectWrite** factory during construction. For convenience, you can simply use the pre-built NinjaTrader.Core.Globals.DirectWriteFactory property.

Formatting Text

The **TextFormat** object determines the font size, style and family, among other properties.

Warning: Any **SharpDX TextFormat** object used in your development must be disposed of after they have been used. NinjaTrader is **NOT** guaranteed to dispose of these resources for you! Please see the Best Practices for SharpDX Resources section on this page for more information.

```
SharpDX.DirectWrite.TextFormat textFormat = new
SharpDX.DirectWrite.TextFormat(Core.Globals.DirectWriteFact
ory, "Arial", 12);
```

Once the text formatting has been described, you can use this object to immediately start rendering text in the DrawText() method. This approach also requires a SharpDX.RectangleF to help determine the size and position the text renders on the chart.

```
// define the point for the text to render
SharpDX.Vector2 startPoint =  new
SharpDX.Vector2(ChartPanel.X, ChartPanel.Y);

// construct the text format with desired font family and
size
SharpDX.DirectWrite.TextFormat textFormat = new
SharpDX.DirectWrite.TextFormat(Core.Globals.DirectWriteFact
ory, "Arial", 36);

// construct the rectangleF struct to describe the position
and size the text
SharpDX.RectangleF rectangleF = new
SharpDX.RectangleF(startPoint.X, startPoint.Y,
ChartPanel.W, ChartPanel.H);

// define the brush used for the text
SharpDX.Direct2D1.SolidColorBrush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue);

// execute the render target text command with desired
values
RenderTarget.DrawText("I am some text", textFormat,
rectangleF, customDXBrush);

// always dispose of textFormat when finished
textFormat.Dispose();
// always dipose of brush when finished
customDXBrush.Dispose();
```

Converting Text

One common approach to text formatting is to use the same formats as existing chart objects. This provides familiar text format matching other objects which exist on the chart. To accomplish this, you can simply use the **ChartControl** NinjaTrader.Gui.SimpleFont object and convert to **SharpDX** using the ToDirectWriteTextFormat() method.

```
SharpDX.DirectWrite.TextFormat textFormat =
ChartControl.Properties.LabelFont.ToDirectWriteTextFormat()
;
```

Text Layouts

The **TextLayout** object works in combination with the **TextFormat** object by extending its functionality and providing an interface more powerful than a simple Rectangle, enabling you to position, measure, or clip the text to a surrounding shape.

When constructing the **TextLayout** object, you will pass in the exact text as a string you wish to render, along with the desired **TextFormat**. This gives you the ability to measure the text string after it has been formatted. During construction, you also have an opportunity to specify the maximum height and width of the **TextLayout**. For example, we can set the text layout to bound to height and width chart panel:

```
SharpDX.DirectWrite.TextLayout textLayout = new
SharpDX.DirectWrite.TextLayout(Core.Globals.DirectWriteFact
ory, "I am also some text", textFormat, ChartPanel.W,
ChartPanel.H);
```

After the text has its format and layout, you can use the RenderTarget.DrawTextLayout() method to specify the exact location as a Vector2, as well as the Brush used to draw the text.

```
RenderTarget.DrawTextLayout(startPoint, textLayout,
customDXBrush);
```

Measuring Text Layouts

Working with an existing **TextLayout** object, you can use its TextLayout.Metrics object to retrieve metadata related to the size of the formatted text. This is helpful if you are unsure of the size of the text before it is rendered. For example, you may wish to draw a rectangle around the formatted text calculated width and height. Using the approach below, the rectangle will dynamically resize to fit the text values used:

```
// define the point for the text to render
SharpDX.Vector2 startPoint =  new
SharpDX.Vector2(ChartPanel.X + 20, ChartPanel.Y + 20);

// construct the text format with desired font family and
size
SharpDX.DirectWrite.TextFormat textFormat = new
SharpDX.DirectWrite.TextFormat(Core.Globals.DirectWriteFact
ory, "Arial", 36);

// construct the text layout with desired text, text
format, max width and height
SharpDX.DirectWrite.TextLayout textLayout = new
SharpDX.DirectWrite.TextLayout(Core.Globals.DirectWriteFact
ory, "I am also some text", textFormat, ChartPanel.W,
ChartPanel.H);

// create a rectangle which will automatically resize to
the width/height of the textLayout
SharpDX.RectangleF rectangleF = new
SharpDX.RectangleF(startPoint.X, startPoint.Y,
textLayout.Metrics.Width, textLayout.Metrics.Height);

// define the brush used for the text and rectangle
SharpDX.Direct2D1.SolidColorBrush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue);

// execute the render target draw rectangle with desired
values
RenderTarget.DrawRectangle(rectangleF, customDXBrush);

// execute the render target text layout command with
desired values
RenderTarget.DrawTextLayout(startPoint, textLayout,
customDXBrush);

// always dispose of textLayout, textFormat, or brush when
finished
textLayout.Dispose();
textFormat.Dispose();
customDXBrush.Dispose();
```

Note: The **TextLayout.Metrics** height and width properties return the text pixel height, including the line spacing of the font. Due to the nature of most font families, there will be an amount of line spacing above and below the text. You can use the TextLayout.GetLineMetrics() method to help calculate the distance from the top of the text line to its baseline.

SharpDX Stroke Style

Using the StrokeStyle Object

When rendering **SharpDX** Lines and Shapes, you can optionally configure a SharpDX.Direct2D1.StrokeStyle allowing you to utilize several pre-made dash styles, or even create a custom dash pattern.

Note: Unlike other **SharpDX** objects such as **brushes**, the **StrokeStyle** is a device-independent resource. This means you only need to create the object once throughout the lifetime of the script. However, the **StrokeStyle** needs to be disposed of when the script is terminated. The **Creating a Custom DashStyle** example below shows how to use a stroke style from the beginning to end of the lifetime of your script. Please see the Best Practices for SharpDX Resources section on this page for more information.

For convenience, **SharpDX** provides the StrokeStyleProperties struct for creating new a **StrokeStyle:**

```
// create a stroke style property using a pre-configured
"DashDot" dash style
SharpDX.Direct2D1.StrokeStyleProperties
dxStrokeStyleProperties = new
SharpDX.Direct2D1.StrokeStyleProperties
{
   DashStyle = SharpDX.Direct2D1.DashStyle.DashDot,
};
```

Once you have your desired stroke style properties, you can create a new stroke style object.

Warning: Any **SharpDX StrokeStyle** object used in your development must be disposed of after they have been used. NinjaTrader is **NOT** guaranteed to dispose of these resources for you! Please see the Best Practices for SharpDX Resources section on this page for more information.

```
SharpDX.Direct2D1.StrokeStyle dxStrokeStyle = new
SharpDX.Direct2D1.StrokeStyle(NinjaTrader.Core.Globals.D2DF
actory, dxStrokeStyleProperties);
```

Tip: The **SharpDX.Direct2D1.StrokeStyle** require a **Direct2D1** factory during construction. For convenience, you can simply use the pre-built NinjaTrader.Core.Globals.D2DFactory property. The DirectD2D factory should only be instantiated and access from OnRender() or OnRenderTargetChanged(), as access outside those methods could cause performance issues.

And then use that object with the RenderTarget.DrawLine() method:

```
RenderTarget.DrawLine(startPoint, endPoint, dxBrush, width,
dxStrokeStyle);
```

Creating a Custom DashStyle

By setting the StrokeStyle.DashStyle property to "**Custom**", you can further refine the appearance of a **SharpDX** rendered line or shape by describing the length and space between the lines. Creating a custom **DashStyle** is not only useful for using **RenderTarget methods**, but also can be used for customizing the appearance of standard NinjaScript Plots.

The code example creates a single **StrokeStyle** object using custom dash style properties. The example then uses those the custom stroke style object with user defined dashes for overriding the default NinjaTrader plot appearances, and using the same stroke style in a RenderTarget.DrawLine() command.

```
// a SharpDX.Direct2D1.StrokeStyle is device independent
// it only needs to be setup once throughout the lifetime
of your script
private SharpDX.Direct2D1.StrokeStyle dxStrokeStyle;

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Custom StrokeStyle";

      AddPlot(Brushes.Blue, "Custom StrokeStyle");
   }
   else if (State == State.Configure)
   {
      // create a custom stroke style when configured
      SharpDX.Direct2D1.StrokeStyleProperties
dxStrokeStyleProperties = new
SharpDX.Direct2D1.StrokeStyleProperties
      {
         // set the dash style to "Custom" define the dash
pattern
         DashStyle = SharpDX.Direct2D1.DashStyle.Custom,

         // set further custom/optional StrokeStyle
appearances
         DashCap = CapStyle.Round,
         EndCap   = CapStyle.Flat,
         StartCap = CapStyle.Square,
         LineJoin = LineJoin.Miter,

         // offset in the dash sequence
         DashOffset = 10.0f,
      };

      // define the an array of floating-point values
      float[] dashes = { 1.0f, 2.0f, 2.0f, 3.0f, 2.0f,
2.0f };

      // create the stroke style using the custom
properties and dash array
      dxStrokeStyle = new
SharpDX.Direct2D1.StrokeStyle(NinjaTrader.Core.Globals.D2DF
actory,
                dxStrokeStyleProperties, dashes);
   }
   else if (State == State.Terminated)
   {
      // make sure to dispose of stroke style when finished
      if (dxStrokeStyle != null)
      {
         if (!dxStrokeStyle.IsDisposed)
```

Best Practices for SharpDX Resources

Understanding Device-dependent vs Device-independent resources

Direct2D has several types of resources which may be mapped to the different hardware devices:

- **Device-independent** resources are on the CPU
- **Device-dependent** resources are on the GPU

When **device-dependent** resources are created, system resources are dedicated to that object. Resources which are **device-dependent** are associated with a particular **RenderTarget** device and are only available on that device. Therefore, objects which were created using a **RenderTarget** can only be used by that device. As the **RenderTarget** updates, objects which were previously created will no longer be compatible and can lead to errors. You can use the NinjaTrader OnRenderTargetChanged() method to detect when the render target has updated and gives you an opportunity to recreate resources.

Device-dependent resources

The following objects are associated with a specific **RenderTarget**. They must be created and dispose of any time the **RenderTarget** is updated:

- Brush
- GeometrySink
- GradientStopCollection
- LinearGradientBrush
- RadialGradientBrush
- SolidColorBrush

Device-independent resources

The following objects are **NOT** associated with a specific device. They can be created once and last for the lifetime of your script, or until they need to be modified:

- PathGeometry
- StrokeStyle
- TextFormat
- TextLayout

Note: For more technical information on device resources, please see the MSDN Direct2D Resources Overview

SharpDX DisposeBase

Although most C# objects stored in memory are handled by the operating system, there are a few **SharpDX** resources which are not managed. It is important to take care of these resources during the lifetime of your script as there is no guarantee that NinjaTrader will be able to dispose of these unmanaged references for you.

The following commonly used objects implement from the SharpDX.DisposeBase and should be disposed any time they are created:

- Brush
- GeometrySink
- GradientStopCollection
- LinearGradientBrush
- PathGeometry
- RadialGradientBrush
- SolidColorBrush
- StrokeStyle
- TextFormat
- TextLayout

Warning: The list above is **NOT** exhaustive and there are other less common **SharpDX** objects that could implement **DisposeBase**. Failure to clean up these resources **WILL** result in NinjaTrader using more memory than necessary and may expose potential "memory leaks" coming from your script. If you experience unusual amounts of memory being utilized over time, an unmanaged **SharpDX** resource is often times the culprit.

Since there is no guarantee that NinjaTrader will release objects from memory when your script is terminated, it is best to protect these resources from issues and call Dispose() as soon as possible. This commonly involves calling **Dispose()** at the end of OnRender(),or during OnRenderTargetChanged() when dealing with **device- dependent** resources such as brush. **Device-independent** resources can be created once and then retained for the life of your application.

```
protected override void OnRender(ChartControl chartControl,
 ChartScale chartScale)
{
   // 1 - setup your resource
   SharpDX.Direct2D1.SolidColorBrush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue
   // 2 - use your resource
   RenderTarget.DrawLine(startPoint, endPoint,
customDXBrush);
   // 3- dispose of your resource
   customDXBrush.Dispose()
}
```

Note: If your resource is setup (i.e., uses the "new" keyword) during **OnRender()** or **OnRenderTargetChange()**, calling **.Dispose()** during State.Terminated will **ONLY** dispose of the *very last reference in memory* and is **NOT** sufficient to completely manage all instances created during the lifetime of your script. You should be diligent in calling **Dispose()** throughout the lifetime of the script.

You can also consider implementing the using Statement (C# Reference) which will implicitly call **Dispose() for** you when you are done:

```
// customDXBrush implicitly calls Dispose() after this
block executes
using (SharpDX.Direct2D1.SolidColorBrush customDXBrush =
new SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue))
{
   RenderTarget.DrawLine(startPoint, endPoint,
customDXBrush);
}
```

Critical: Attempting to use an object which has already been disposed can lead to memory corruption that NinjaTrader may not be able to recover. Attempts to use an object in this manner can result in an error similar to: Error

on calling 'OnRender' method on bar 0: Attempted to read or write protected memory. This is often an indication that other memory is corrupt.

You can check to see if can object has been disposed of by using the DisposeBase.IsDiposed property:

```
SharpDX.Direct2D1.Brush customDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.DodgerBlue);

// checks the object is not disposed of before using
if(!customDXBrush.IsDisposed)
{
   RenderTarget.DrawLine(startPoint, endPoint,
customDXBrush);
   customDXBrush.Dispose();
}
```

You should also favor managing these resources yourself, which means methods which accept a **SharpDX DisposeBase** object as an argument should be created before they are passed into the method and disposed of after they are used. For example, the code below should be avoided:

Practice to avoid

```
// do NOT convert an object as it is passed to an argument.

// You may have no chance to Dispose of the object!
// Finalizer is not guaranteed to release of these
resources
RenderTarget.DrawLine(startPoint, endPoint,
Brushes.AliceBlue.ToDxBrush(RenderTarget));

MyCustomMethod(Brushes.AliceBlue.ToDxBrush(RenderTarget));
```

Instead, you should manage these objects yourself:

Best practice

```
// Do create and store this reference yourself so you can
control when it is released (Y)
SharpDX.Direct2D1.Brush customDXBrush  =
WPFBrush.ToDxBrush(RenderTarget);

RenderTarget.DrawLine(startPoint, endPoint,
customDXBrush));

MyCustomMethod(customDXBrush);

customDXBrush.Dipose()
```

Other Best Practices

If possible, you should avoid using the ToDxBrush() method if it is not necessary. It is relatively harmless to use this approach for a few brushes, but can introduce performance issues if used too liberally.

Practice to avoid

```
// do NOT convert from WPF brushes unnecessarily
SharpDX.Direct2D1.Brush dxBrush1 =
System.Windows.Media.Brushes.Blue.ToDxBrush(RenderTarget);
SharpDX.Direct2D1.Brush dxBrush2 =
System.Windows.Media.Brushes.Red.ToDxBrush(RenderTarget);
SharpDX.Direct2D1.Brush dxBrush3 =
System.Windows.Media.Brushes.Green.ToDxBrush(RenderTarget);
SharpDX.Direct2D1.Brush dxBrush4 =
System.Windows.Media.Brushes.Purple.ToDxBrush(RenderTarget)
;
SharpDX.Direct2D1.Brush dxBrush5 =
System.Windows.Media.Brushes.Orange.ToDxBrush(RenderTarget)
;
SharpDX.Direct2D1.Brush dxBrush6 =
System.Windows.Media.Brushes.Yellow.ToDxBrush(RenderTarget)
;
```

Instead, you should construct a SharpDX Brush directly if a WPF brush is not ever needed:

Best practice

```
// Do create SharpDX Brushes directly if you have a large
amount of brushes
SharpDX.Direct2D1.Brush dxBrush1 = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.Blue);
SharpDX.Direct2D1.Brush dxBrush2 = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.Red);
SharpDX.Direct2D1.Brush dxBrush3 = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.Green);
SharpDX.Direct2D1.Brush dxBrush4 = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.Purple);
SharpDX.Direct2D1.Brush dxBrush5 = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.Orange);
SharpDX.Direct2D1.Brush dxBrush6 = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.Yellow);
```

Rendering with anti-aliasing disabled can be used to render a higher qualify shapes but comes as a performance impact. You should make sure to set this render target property back to its default when you are finished with a render routine.

Best practice

```
// AntialiasMode.PerPrimitive is more resource intensive
// store the old reference before setting the desired value
SharpDX.Direct2D1.AntialiasMode oldAntialiasMode =
RenderTarget.AntialiasMode;
RenderTarget.AntialiasMode =
SharpDX.Direct2D1.AntialiasMode.PerPrimitive;

// execute your render routines

// and then set back to initial AntialiasMode when finished
RenderTarget.AntialiasMode = oldAntialiasMode;
```

11.5.16 Working with Brushes

In order to achieve custom rendering for various chart related objects, a Brush is used to "paint" an area or another chart object. There are a number of different brushes which are

available through the .NET Framework, where the most common type of brush is a SolidColorBrush which is used to paint an area with a single solid color.

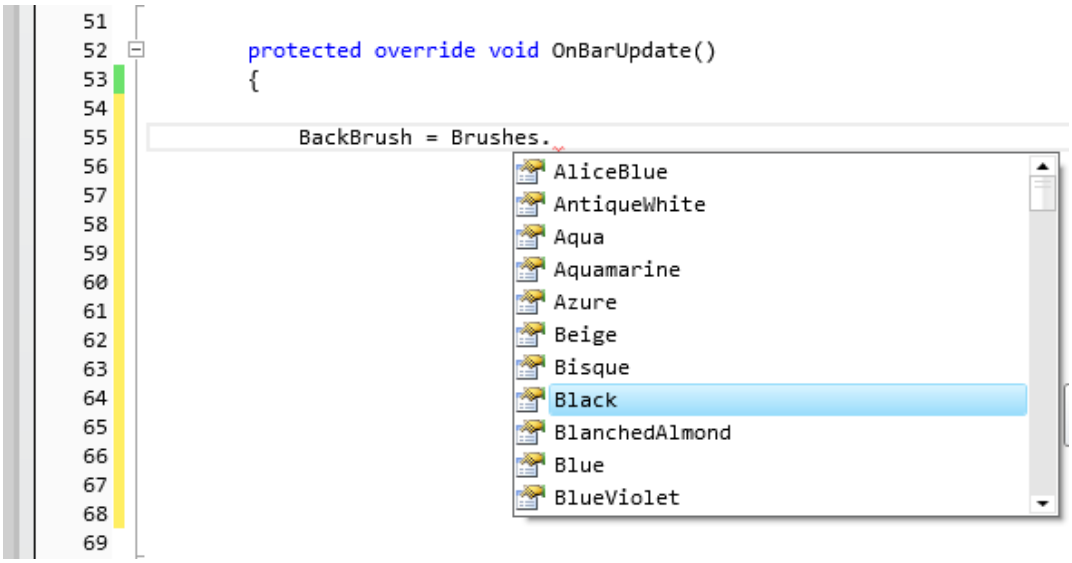
Notes: The following document is written in sequential fashion, starting with the most simple concepts, to the more advance topics. The majority of the brushes discussed in this document will be referred to as "**WPF" brushes** which exist in the System.Windows.Media namespace, however there are also **"SharpDX" brushes** which exist in the 3rd party SharpDX.Direct2D1 namespace used for advanced chart rendering. Advanced brush types should **ONLY** be used by experienced programmers familiar with .NET graphics functionality.

Understanding predefined brushes

Using Predefined Brushes

For convenience, the .NET Framework supplies a collection of static predefined Brushes, such as Red or Green. The advantage to using these brushes is that they are readily available, properly named to quickly find a simple color value, and can be reused on-the-fly without having to recreate an instance of the brush at run time, and do not need to be otherwise managed. There are 256 predefined named brushes which are available in the Brushes class. You can browse this list in the NinjaScript editor just by typing `Brushes.` and using Intelliprompt to find the desired named brush of your choice.

Note: Since predefined brushes are static, properties of the brush object (such as Color, Opacity, etc.) **CANNOT** be modified. However, this also means predefined brushes are thread-safe and do **NOT** need to be frozen. For customizing and freezing a brush, please see the section below on *Creating a Custom Solid Color Brush*.



Tip: You can also find a list of these predefined brushes as well as their hexadecimal value on the MSDN article for the Brushes Class

```
// set the chart's background color to a predefined "Blue" brush
BackBrush = Brushes.Blue;

//draw a line using a predefined "LimeGreen" brush.
Draw.Line(this, "tag1", false, 10, 1000, 0, 1001,
Brushes.LimeGreen, DashStyleHelper.Dot, 2);
```

Understanding custom brushes

Creating a Custom Solid Color Brush

In cases where you would like more specific color than one of the predefined brushes, you can optionally create your own **Brush** object to be used for custom rendering. In order to achieve this, you will need to initiate your own custom brush object, where you can then specify your color using RGB (red, green, blue) values Color.FromRgb().

Notes:
- Anytime you create a custom brush that will be used by NinjaTrader rendering it must be frozen using the .Freeze() method due to the multi-threaded nature of NinjaTrader.
- You may have up to 65535 unique Brush instances, therefore, using static predefined brushes (as in the section above) should be favored. Alternatively, in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created.

```
// initiate new solid color brush with custom blue color
Brush myBrush = new SolidColorBrush(Color.FromRgb(56, 120,
153));
myBrush.Freeze();

Draw.Line(this, "tag1", true, 10, 1000, 0, 1001, myBrush,
DashStyleHelper.Dot, 2);
```

Warning: If you do not call .Freeze() on a custom defined brush **WILL** eventually result in threading errors should you try to modify or access that brush after it is defined.

Creating a Transparent Solid Color Brush

You can create a transparent brush using the Color.FromArgb() where the A parameter defines alpha transparency.

Note: Anytime you create a custom brush that will be used by NinjaTrader rendering it must be frozen using the .Freeze() method due to the multi-threaded nature of NinjaTrader.

```
// initiate new solid color brush which has an alpha
(transparency) value of 100
myBrush = new SolidColorBrush(Color.FromArgb(100, 56, 120,
153));
myBrush.Freeze();

Draw.Line(this, "tag1", true, 10, 1000, 0, 1001, myBrush,
DashStyleHelper.Dot, 2);
```

Warning: If you do not call .Freeze() on a custom defined brush **WILL** eventually result in threading errors should you try to modify or access that brush after it is defined.

Using brushes defined on the user interface

Saving a Brush as a user defined property (Serialization)

If you would like a brush to become a public UI property, meaning the brush can be set up and defined by a user during configuration, it is important to be able to save the user's brush selection in order to restore that brush either from a workspace or from a template file at a later time. Saving a custom defined user input is done through a concept of Serialization which writes the object and its value to a .xml file. This process normally works fine for a simple user defined value type (such as a `double` or an `int`) but for more complex types such as Brushes, the object itself cannot be serialized directly to the .xml file and will result in errors upon saving the indicator or strategy to a workspace or template file. The example below will demonstrate and explain how to properly store a user define brush input which will be correctly serialized.

In order to achieve the desired behavior of saving the user defined brush input, we will add the XmlIgnore property attribute to the public brush resource, which essentially tells the serialization routine to ignore this property.

```
[XmlIgnore]
public Brush MyBrush { get; set; }
```

In its place, we create a new public string called "MyBrushSerialize" which will convert the public "MyBrush" to a string type which can then be processed by the serialization routines. We also add the Browsable(false) attribute to this public string to prevent this property from showing up on the UI, which is of no value to the end user.

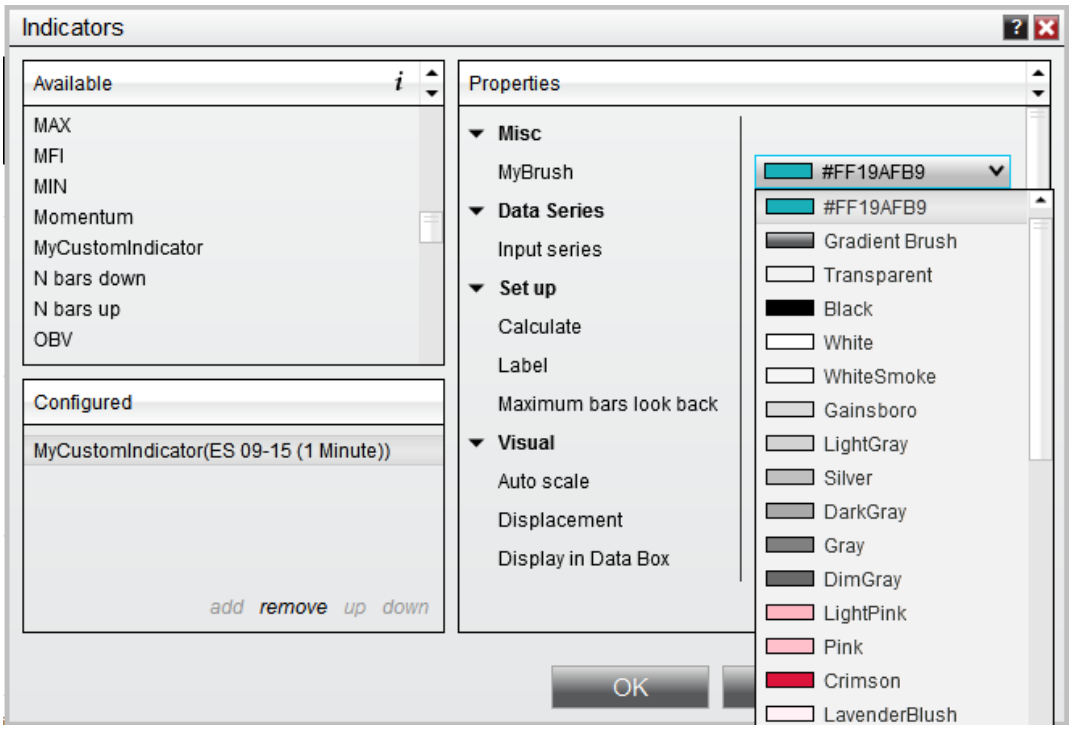
```
[Browsable(false)]
public string MyBrushSerialize
{
   get { return Serialize.BrushToString(MyBrush); }
   set { MyBrush = Serialize.StringToBrush(value); }
}
```

Tip: For a complete example of **User Definable Color Inputs**, please see the reference sample here.

Adding a User Defined Brush to the Color Picker

You can optionally define a custom brush to be added to the standard color picker by using a [CustomBrush] attribute to a public brush. The CustomBrush attribute will then add it to the color picker menu for that indicator when you look through the plots, lines, or other brushes from the indicators configured menu and will be listed toward the top of the list (as pictured below)

```
[CustomBrush]
public Brush MyBrush
{
   get { return new SolidColorBrush(Color.FromRgb(25, 175,
185)); }
   set { }
}
```



Using advanced brush types (SharpDX)

Understanding SharpDX Brushes

While the majority of the NinjaTrader platform's UI is **WPF**, under the hood, chart's use a **DirectX API** for faster performance. To render custom objects to a chart during OnRender(), a particular **SharpDX Brush** object must be implemented which reside in the **SharpDX.Direct2D1** namespace. These brushes can then be passed as arguments to the **SharpDX** RenderTarget methods such FillRectangle(), DrawLine(), etc. While **SharpDX Brushes** behave much the same as previously discussed **WPF Brushes**, there are a few special considerations you must take as detailed in the following sections.

> **Note**: The **SharpDX Brushes** used in RenderTarget methods should **NOT** be confused with the **WPF Brushes** used with DrawingTool Draw methods.

Creating a SharpDX Brush

A SharpDX Brush must be created either in **OnRender()** or **RenderTargetChanged()**. If you have custom brushes which may be changed

on various conditions such as in OnBarUpdate() or by a user during OnStateChange(), or you are pre-computing a custom brush for performance optimization, you will need to ensure the actual SharpDX instance is updated in OnRender() or RenderTargetChange().

Warning: Each DirectX render target requires its own brushes. You **MUST** create brushes directly in **OnRender()** or using **OnRenderTargetChanged()**. If you do not you will receive an error at runtime similar to:

"A direct X error has occured while rendering the chart: HRESULT: [0x88990015], Module: [SharpDX.Direct2D1], ApiCode: [D2DERR_WRONG_RESOURCE_DOMAIN/WrongResourceDomain], Message: The resource was realized on the wrong render target. : Each DirectX render target requires its own brushes. You must create brushes directly in OnRender() or using OnRenderTargetChanged().

Please see OnRenderTargetChanged() for examples of a brush that needs to be recalculated, or OnRender() for an example of recreating a static brush.

```
// use predefined "Blue" SharpDX Color
SharpDX.Direct2D1.SolidColorBrush solidBlueDXBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.Blue);

// create custom Brush using a "Red" SharpDX Color with
"Alpha" (0.100f) transparency/opacity
SharpDX.Direct2D1.SolidColorBrush transparentRedDXBrush =
new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, new
SharpDX.Color4(new SharpDX.Color3(220f, 0f, 0f), 0.100f));
```

Converting to SharpDX Brush

For convenience, you can convert a computed WPF Brush to a SharpDX Brush using the ToDxBrush() extension method.

Warning: Converting **ToDxBrush()** can result in performance issues depending on the number of brushes being used. If you experience performance issues with your custom **SharpDX** rendering, you should favor

using **SharpDX** brushes directly instead of converting the brush using **ToDxBrush().**

```
// convert predefined WPF "Blue" to SharpDX Brush
SharpDX.Direct2D1.Brush blueDXBrush =
Brushes.Blue.ToDxBrush(RenderTarget);

// convert the computed WPF Brush to SharpDX Brush
SharpDX.Direct2D1.Brush customDXBrush =
customWPFBrush.ToDxBrush(RenderTarget);
```

Disposing DXBrush

Since **SharpDX Brushes** reference unmanaged resources, these brushes should always be disposed of after they have been used.

Warning: Failing to dispose of a SharpDX Brush and other unmanaged resources can cause the platform to utilize more memory than necessary.

```
customDXBrush.Dipose();
```

Using Complex Brushes

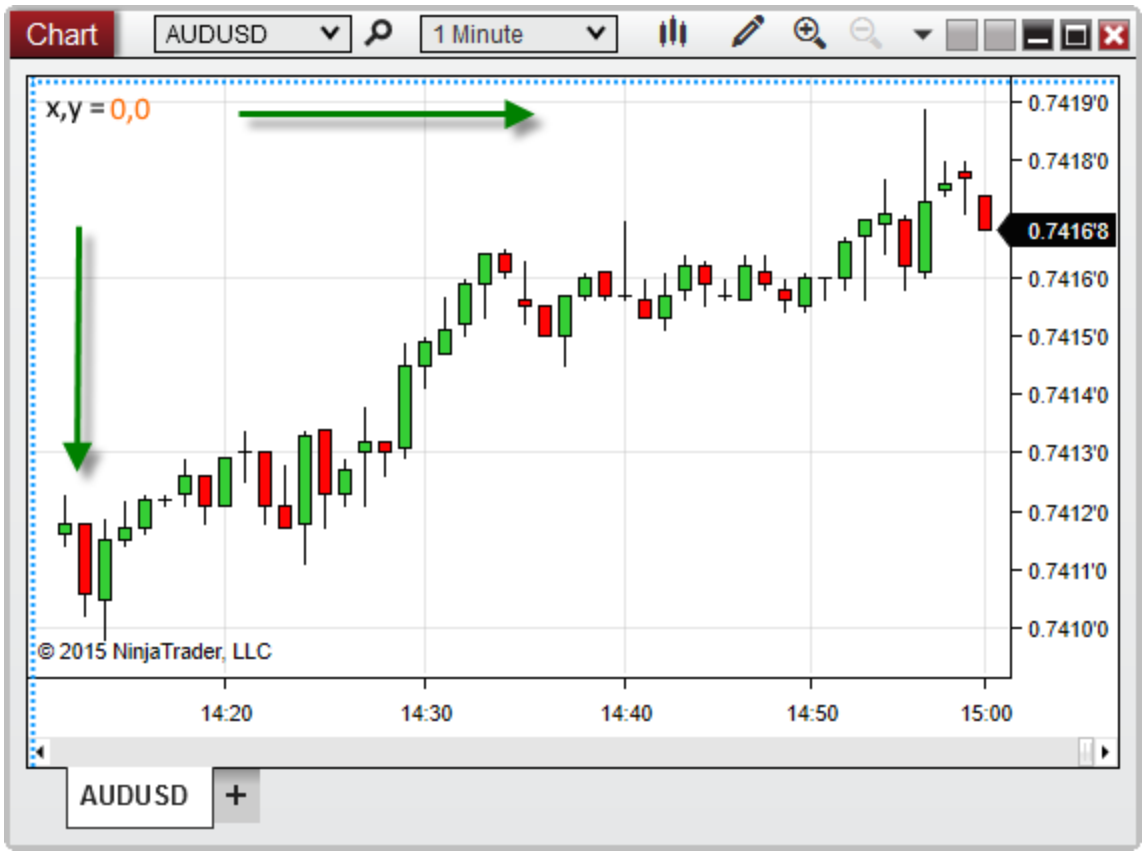
In addition to the SolidColorBrush object demonstrated on this page, the .NET Framework provides more complex brushes which have more attributes than just filling an area with a solid color. Information on these special types of brushes can be found on the MSDN website: LinearGradientBrush, RadialGradientBrush, ImageBrush.

These complex types also have an equivalent found in the **SharpDX SDK Reference**: SharpDX.Direct2D1.LinearGradientBrush, SharpDX.Direct2D1.RadialGradientBrush

11.5.17 Working with Chart Object Coordinates

Understanding Chart Canvas Coordinates

The chart canvas represents the portion of a chart window on which objects can be painted (the area outlined in blue in the image below). The canvas area is measured by an x-axis and y-axis independent of the price and time-axis of the chart itself. When working with coordinates on a chart canvas, it is important to note that the origin point (coordinates 0,0) is in the top-left corner of the canvas, **NOT** the bottom-left. Moving down the canvas increases the y-coordinate, and moving the the right on the canvas increases the x-coordinate.



Understanding Chart Areas

When using `ChartControl` properties and methods, it is important to understand the layout of a chart window, and which specific area of the window is being measured by a specific property. The image below shows the three primary areas of a chart window.

The three regions shaded in the image above are labeled as follows:

1. The **chart canvas** covers the area in which bars, drawing objects, and indicator plots can be painted. It is bounded on the bottom by the x-axis, and on the right, left, or both by the y-axis. This is measured by properties such as CanvasLeft and CanvasRight.
2. The **y-axis** extends vertically from the chart's horizontal scroll bar to the top of the chart canvas, and can be displayed to the right or left (or both) of the canvas area, depending the "Scale Justification" properties of the Bars object or indicators painted on the chart. This is measured by properties such as AxisYLeftWidth and AxisYRightWidth.
3. The **x-axis** sits beneath the chart canvas, and extends horizontally from the left edge of the chart canvas (or the left edge of the y-axis if it is visible on the left) to the right edge of the y-axis applied to the right of the canvas (or the right edge of the canvas itself if the y-axis is not visible on the right). This is measured by properties such as AxisXHeight.

11.5.18 Working with Pixel Coordinates

Understanding Device Pixels vs. Application Pixels (WPF)

When working with pixel coordinates (for example, when using SharpDX drawing methods for custom drawing), it is important to note if the coordinates specified in method arguments refer to application pixels (i.e., WPF coordinates), or the larger concept of Device Independent Pixels (DIP).

The physical size of an application-specific pixel can vary based on PC hardware and operating-system settings, which introduces a challenge for developers using pixel coordinates for processes such as custom drawing on a chart canvas. By specifying the number of pixels when defining a coordinate, the object placed at that coordinate could render in a very different position depending on the users display settings. **Device Independent Pixels** provide a way to measure or quantify pixel coordinates without being impacted by different sizes of application pixels. Specifying **Device Independent Pixels** can ensure that objects render in the intended location or position, regardless of these unpredictable factors.

Converting to Device Pixels

NinjaScript provides helper methods to convert from application pixels to device pixels (or vice versa) within the ChartingExtensions class. Since some NinjaScript methods and properties return application pixels where device pixels are needed, using these helper methods can provide great flexibility by allowing you to define physical application pixels, then converting them to device independent pixels before passing them to a method. Using this process, the application pixel values used will result in objects being rendered exactly where intended.

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // get the point the user clicked, (which returns application
pixel)
   Point clickPoint = chartControl.MouseDownPoint;

   // Convert the clickPoint X and Y coordinates to device
independent pixels (DIP)
   // This will ensure that the MouseDownPoint will work across all
screen displays
   clickPoint.X =
ChartingExtensions.ConvertToHorizontalPixels(clickPoint.X,
chartControl.PresentationSource);
   clickPoint.Y =
ChartingExtensions.ConvertToVerticalPixels(clickPoint.Y,
chartControl.PresentationSource);

   // set the location (vector) from the user clickPoint
   SharpDX.Vector2 vectorForEllipse = clickPoint.ToVector2();

   // create the shape (ellipse), and color (brush) for our object
to render
   SharpDX.Direct2D1.Ellipse ellipse = new
SharpDX.Direct2D1.Ellipse(vectorForEllipse, 10f, 10f);
   SharpDX.Direct2D1.Brush ellipseBrushDX =
Brushes.Blue.ToDxBrush(RenderTarget);

   // finally, render a ellipse at the exact point the user clicked
   RenderTarget.FillEllipse(ellipse, ellipseBrushDX);
}
```

11.5.19 Working with Price Series

Price Data Overview

The core objective of developing custom Indicators and Strategies with NinjaScript is to evaluate price data. NinjaScript allows you to reference current and historical price data. There are several categories of price data which include ISeries<T>, Indicator and Custom Historical Series.

Definitions

ISeries<T>	Standard bar based price types such as closing, opening, high, low prices and volume

Indicator	Calculated values based on price type values such as a simple moving average
Custom Historical Series<T>	Custom calculated values that you wish to store and associate to each historical bar

Referencing Series

ISeries<T>	Syntax	Editor Shortcut	Definition
Close	`Close[int barsAgo]`	"c" + Tab Key	Last traded price of a bar
Open	`Open[int barsAgo]`	"o" + Tab Key	Opening price of a bar
High	`High[int barsAgo]`	"h" + Tab Key	Highest traded price of a bar
Low	`Low[int barsAgo]`	"l" + Tab Key	Lowest traded price of a bar
Volume	`Volume[int barsAgo]`	"v" + Tab Key	Number of shares/contracts traded of a bar
Input	`Input[int barsAgo]`	"i" + Tab Key	Default price type of a bar

You will notice that to reference any price data you need to include a value for `[int barsAgo]`. This is a very simple concept; barsAgo represents the number of bars ago to reference and int indicates that barsAgo is an integer value. As an example, we could write a statement to check if the the high price of 1 bar ago is less than the high price of the current bar like this:

```
High[1] < High[0];
```

You could write a statement to calculate the average closing price of the last three bars like this:

```
( Close[2] + Close[1] + Close[0] ) / 3;
```

As you may have already figured out, referencing the current bar data is accomplished by passing in a value of 0 (zero) to the barsAgo parameter. Basically, we are saying show me the price data of zero bars ago, which means the current bar.

Note: In most cases, you will access the historical price series using a core event handler such as OnBarUpdate. For more advance developers, you may find situations where you wish to access historical price series outside of the core event methods, such as your own custom mouse click. In these advanced scenarios, you may run into situations where the barsAgo pointer is not in sync with the current bar, and may result in errors when trying to obtain this information. In those cases, please use the Bars.Get...() methods with the absolute bar index (e.g., Bars.GetClose(), Bars.GetTime(), etc.)

Referencing Indicator Data

NinjaScript includes a library of built in indicators that you can access. Please see the Indicator Methods reference section for clear definitions for how to access each indicator.

All indicator values can be accessed in the following way:

```
indicator(parameters)[int barsAgo]
```

where indicator is the name of the indicator you want to access, parameters is any associated parameters the indicator requires and barsAgo is the number of bars we wish to offset from the current bar.

As an example, we could write a statement to check if the current closing price is greater than the 20 period simple moving average like this:

```
Close[0] > SMA(20)[0];
```

If you wanted to perform the same check but only check against a 20 period simple moving average of high prices you would write it like this:

```
Close[0] > SMA(High, 20)[0];
```

You could write a statement to see if a 14 period CCI indicator is rising like this:

```
CCI(14)[0] > CCI(14)[1];
```

Value of a 10 period CCI 1 bar ago = CCI(10)[1]

Please review the Indicator Methods section for proper syntax for accessing different indicator values.

11.5.20 Reference Samples

Reference Samples Overview

- Indicator
- Strategy

11.5.20.1 Indicator

Indicator Overview

- Calculating the highest high or lowest low for a specified time range
- Changing fonts for draw objects
- Coloring a region
- Creating a user-defined parameter type (enum)
- Creating your own Level II data book (Accessing market depth)
- Draw Objects
- Ensuring indicator plots are valid before programmatically accessing them
- Exposing indicator values that are not plots
- Getting indicator values from a specified time
- Manipulating DateTime objects
- Manipulating string objects
- Multi-Colored Plots
- Removing and Custom Formatting an Indicator's Chart Label
- Using a secondary series as an input series for an indicator
- Using a Series or DataSeries object to store calculations
- Using a TypeConverter to Customize Property Grid Behavior
- Using custom events to output the current Level II data book
- Using StreamReader to read from a text file
- Using StreamWriter to write to a text file
- Using System.IO File properties to write to and read from a text file
- Using Try-Catch Blocks
- Creating Chart WPF (UI) Modifications from an Indicator

11.5.20.1.1 Calculating the highest high or lowest low for a specified time range

Determining a high or low value for given time range can be useful.

Key concepts in this example

- Converting time to bars ago values

- Getting the highest high and lowest low values

Important related documentation

- GetBar()
- MAX()
- MIN()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleGetHighLowByTimeRange_NT8.zip

11.5.20.1.2 Changing fonts for draw objects

Drawing text on a chart can be useful for outputting information, but when all information is displayed with the same font and size it could be difficult to quickly see the key information. Since NinjaScript is based on C#, it is possible to use Font objects to create more styles for your text

Key concepts in this example

- Drawing text on a chart
- Changing the font size on a chart

Important related documentation

- Text()
- SimpleFont()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleChangeFont_NT8.zip

11.5.20.1.3 Coloring a region

Filling in a region between two DataSeries objects on your indicators can be beneficial for creating visual indicators. The colored regions allow for immediate recognition of various zones that can help a discretionary trader quickly identify what's important and trade accordingly.

This reference sample demonstrates the following concept

- Coloring a region between two DataSeries objects
- Coloring a region between a DataSeries object and a double value

Important methods and properties used include

- Bollinger()
- Draw.Region()

Other methods and properties of interest include:

- Draw.Diamond()
- Draw.Rectangle()
- DrawOnPricePanel

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleDrawRegion_NT8.zip

11.5.20.1.4 Creating a user-defined parameter type (enum)

Creating user-defined parameters allows you to present the user with hard coded options they can choose. These options provide flexibility in your indicators and can be of value to the user if they like to switch settings often.

Key concepts in this example

- Hard code various Moving Average types the user can select
- Use a switch to determine which code logic is executed based on the Moving Average type selected

Important related documentation

- enum
- branching statements

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleUniversalMovingAverage_NT8.zip

11.5.20.1.5 Creating your own Level II data book (Accessing market depth)

Level II data is important for the momentum trader. It allows them to determine which way the market makers are trading and can be useful in helping the trader decide which way the momentum is going.

Key concepts in this example

- Storing Level II data in a custom object list

- Printing Level II books for discretionary trading

Important related documentation

- List<>
- MarketDepthEventArgs
- Operation
- Position
- Price
- Volume
- Time

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleLevel2Book_NT8.zip

11.5.20.1.6 Draw Objects

Being able to mark visually conditions of interest on the chart is useful for the discretionary trader. With NinjaScript you can draw various objects onto your chart to alert you of these points of interest.

Key concepts in this example

- Drawing unique diamonds to mark the beginning and end of uptrends
- Drawing and updating a single rectangle that marks the current uptrend

Important related documentation

- Drawing
- Draw.Diamond()
- Draw.Rectangle()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleDrawObject_NT8.zip

11.5.20.1.7 Ensuring indicator plots are valid before programmatically accessing them

When accessing NinjaScript indicators in other scripts it is important to check if the hosted indicator's plot values are already set prior to use in the hosting script. This check ensures that proper values are always used and that irrelevant values do not throw off the script logic.

This reference sample demonstrates how to run these checks in a hosting indicator by checking another hosted indicator for set plot values.

Another example for when you would want to use this is if you were trying to access the Pivots indicator, but did not have enough days loaded to properly calculate those values yet. Basing logic on the Pivots in such a scenario would yield values that are not useful and can be detrimental if not handled correctly in your code.

Key concepts in this example

- Checking indicator plots for valid values
- Handling logic for when the indicator plots are not valid

Attached archive contains two indicator files

- SampleEveryNBarTest is the hosting indicator
- SampleEveryNBar is the hosted indicator

> **Note**: When hosting an indicator in an Indicator column in the Market Analyzer you will need to manually ensure enough bars back are loaded for the indicator to calculate correctly.

Important related documentation

- IsValidDataPoint()
- Series

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleEveryNBarTest_NT8.zip

11.5.20.1.8 Exposing indicator values that are not plots

There may be cases where you want to have your indicator calculate non-plotted values that you will want to access when using this indicator inside of another indicator or strategy.

Key concepts in this example

- Creating exposed BoolSeries objects
- Storing and retrieving values from BoolSeries objects

Important related documentation

- Series<T>

We suggest using an available class that implements the Series interface.

- Price Series
- Time Series
- Volume Series

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleBoolSeries_NT8.zip

11.5.20.1.9 Getting indicator values from a specified time

Sometimes, you may want to access a value from a historical point in time, but have not kept track of the value to make this readily available. With NinjaScript, it is possible to pick a bar based on time to access that value. GetBar() returns the number of bars ago that holds the same timestamp of the time you request. This sample demonstrates how to get an indicator value from 9:30AM of the previous trading day.

Key concepts in this example

- Obtaining a Simple Moving Average value from a specific time by referencing the bar number for that time.

Important related documentation

- GetBar()
- Draw.Line()
- Time
- Sessions
- DateTime

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleGetBar_NT8.zip

11.5.20.1.10 Manipulating DateTime objects

An essential element of any trader's strategies or indicators is time. You may find yourself wanting a high and low marker for a certain timeframe or you might want something drawn on your charts during those choppy lunch hours. DateTime objects are included in the .NET framework, and they can be used to do any time related action, like limiting trading hours or finding the highest high between 9:30AM and 10:30AM.

Key concepts in this example

- Common manipulation of DateTime objects

Important related documentation

- DateTime
- DateTime.Add()
- DateTime.Compare()
- DateTime.Now
- DateTime.TryParse()
- TimeSpan
- DateTime.ToString(string)
- string.Format()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleDateTimeFunctions_NT8.zip

11.5.20.1.11 Manipulating string objects

Dealing with strings and other related concepts are essential to many computer programs. This sample is a collection of some of the most common string and text related functions including splitting a string, replacing a string with another string, and a few other string functions.

Key concepts in this example

- Simple text/string manipulation ideas

Important related documentation

C#

- string.IndexOf()
- string.Replace()
- string.Split()
- Escape characters
- Foreach iterator
- String literals

NinjaTrader

- ClearOutputWindow()

> **Note**: A related sample demonstrating how to format numbers can be found here.

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleStringFunctions_NT8.zip

11.5.20.1.12 Multi-Colored Plots

With multi-colored plots it becomes easy to pick out changes in value of your indicator from a quick glance.

Key concepts in this example

- Adding plots for each color used
- Plotting a SMA line with three different colors depending on the rising, falling, or neutral nature of the line

Important related documentation

- AddPlot()
- IsFalling()
- IsRising()
- PlotBrushes

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleMultiColoredPlot_NT8.zip

11.5.20.1.13 Removing and Custom Formatting an Indicator's Chart Label

If you create a NinjaScript indicator or strategy with many customizable parameters, you will have a long label when you load the NinjaScript onto your chart. This may be visually cumbersome so you may want to trim the displayed label to a more manageable size that only contains the most important parameters.

Key concepts in this example:

- Creating a custom string for the label of the NinjaScript item.

Important related documentation

- Draw.TextFixed()
- Draw.Text()
- Override DisplayName()

Tip: When adding an indicator onto a chart you can also completely remove any labeling on the chart of the indicator name. You can do this by clearing the "Label" field under the "General" category when you add the indicator onto the chart.

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleDisplayName_NT8.zip

11.5.20.1.14 Using a secondary series as an input series for an indicator

Adding additional series to a script can be useful. You may also want to use this added data for an indicator's Input Series.

Key concepts in this example

- Adding series
- Supplying a series object to an indicator as the input series parameter
- Plotting using data from two different series

Important related documentation

- AddDataSeries()
- AddPlot()
- IsValidDataPoint()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleSecondarySeriesAsInputSeries_NT8.zip

11.5.20.1.15 Using a Series or DataSeries object to store calculations

When creating Indicators or Strategies you may find that you need to store values in a way that is similar to the way price data is stored in NinjaTrader.

Series<T> objects are useful for storing various types of values.

Since they are linked to your historical bars object, you can store and link a value to each bar. This allows you the flexibility of accessing the values at any point in the future for further calculations or plotting.

Key concepts in this example

- Creating objects that store data
- Storing and retrieving values from these objects

Important related documentation

- Series<T>

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleCustomSeries_NT8.zip

11.5.20.1.16 Using a TypeConverter to Customize Property Grid Behavior

The default behavior of the NinjaTrader property grid is designed to handle the most basic display of your custom Indicator and Strategy properties. However, there are special cases where you may want to control how unique properties display to other users. Since using a TypeConverter is more of a general C# concept used to convert values between data types (a string to an enum for example), so the possibilities of what you can do are within the bounds of the .NET Framework. This NinjaScript sample was produced to help NinjaScript developers understand how to leverage the IndicatorBaseConverter and StrategyBaseConverter helper classes to customize property grid behavior without affecting general NinjaTrader property behavior.

Note

- NinjaTrader expects custom properties to be of value type or type which implements ICloneable interface
- This reference sample assumes you are familiar with basic C# TypeConverter practices and is intended as a starting point. There are extensive 3rd party guides available online which can help you implement a particular type converter goal not covered in this sample

Key concepts in this example

There are 5 use cases demonstrated in this sample which fall into two different categories:

1. Dynamically manipulate what is displayed on the UI Property Grid
a. Show / hide properties based on secondary input
b. Disable / enable properties based on secondary input

2. Customizing how a property is displayed
a. Display "Friendly" enum values
b. Re-implement a "bool" CheckBox as "Friendly" a ComboBox
c. Display a custom collection / list with user defined values at run time

Important related documentation

C#

- ICloneable Interface
- PropertyDescriptor Class
- RefreshPropertiesAttribute
- TypeConverter Class
- DisplayAttribute Class

NinjaTrader

- IndicatorBaseConverter Class
- StrategyBaseConverter Class
- TypeConverterAttribute

Note: The reference sample files on this page are written for an indicator using the **IndicatorBaseConverter** class, but the same key concepts are available to strategies by replacing the **StrategyBaseConverter** where noted in the sample code.

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

Additional resources

MSDN - How to: Implement a Type Converter

SampleIndicatorTypeConverter_NT8.zip

11.5.20.1.17 Using custom events to output the current Level II data book

Custom events allow you the flexibility to access indicator/price information whenever you deem it necessary. You do not need to wait for the next incoming tick or next bar update before you can process some code logic.

Key concepts in this example

- Creating a custom event from a Timer object
- Printing Level II books whenever you receive a Timer event

Important related documentation

- List
- Timer
- TriggerCustomEvent()
- State.Terminated

- MarketDepthEventArgs

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleCustomEvents_NT8.zip

11.5.20.1.18 Using StreamReader to read from a text file

Sometimes you may have data stored outside of NinjaTrader that you want to bring in and use for calculations. Using StreamReader will allow us to bring in the data stored in text files and allow us to do manipulations with them.

Key concepts in this example

- Opening a text file with StreamReader
- Parsing Open-High-Low-Close data with date stamps from a file
- Determining current day Open-High-Low-Close after reading data from the text file with StreamReader

Important related documentation

- StreamReader
- ReadLine
- StringReader
- TextReader

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleStreamReader_NT8.zip

11.5.20.1.19 Using StreamWriter to write to a text file

Sometimes you may want to store information related to certain market conditions or trades outside of NinjaTrader. The formatting of the text file can also make for easy importing into Microsoft Excel in the future.

Key concepts in this example

- Creating a text file with StreamWriter
- Writing Open-High-Low-Close data with date stamps to the file

Important related documentation

- StreamWriter
- WriteLine
- StringWriter
- TextWriter

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleStreamWriter_NT8.zip

11.5.20.1.20 Using System.IO File properties to write to and read from a text file

Using Stream objects can be cumbersome when you only want to write/read a small amount of data. Fortunately, you can handle this operation with another method. Keep in mind that in exchange for the convenience you will lose some performance.

Key concepts in this example

- Creating and appending a text file with Open-High-Low-Close data and date stamps
- Determining current day Open-High-Low-Close after reading data from the text file

Important related documentation

- File.AppendAllText()
- File.ReadAllText()

Other methods and properties of interest include:

- File.ReadAllLines()
- File.WriteAllLines()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleFileReadWrite_NT8.zip

11.5.20.1.21 Using Try-Catch Blocks

Debugging your NinjaScript code can be time consuming and frustrating. When NinjaTrader encounters a run-time exception in your programming logic it will terminate the execution of the script and log the exception to the Control Center Log tab. This in itself can be of value however, there may be times where your script is too large and the exception error message provided is not granular enough. This is where standard C# exception handling using the keywords "try" and "catch" (try-catch block) can be very useful. A try-catch block allows you to encapsulate a section of your code to trap exceptions and write out meaningful information that can help you resolve your run-time errors.

Key concepts in this example
- try-catch blocks

Important related documentation
- Log()
- Print()
- PrintTo()

Import instructions
1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleTryCatch_NT8.zip

11.5.20.1.22 Creating Chart WPF (UI) Modifications from an Indicator

NinjaTrader's extensive C# backend allows for powerful expandability that is unmatched in other trading platforms. Within the context of C# and NinjaScript it is possible to manipulate the window in which the NinjaScript is added. This example demonstrates how chart window modifications can be performed to add your own WPF controls to your chart for custom functionality. These window modifications could be, but are not limited to: adding custom buttons, menus or toolbars.

Key concepts in this example
- Adding your own toolbar with WPF Controls to the left/right side of a chart
- Adding your own toolbar with WPF Controls to the top of a chart
- Adding WPF Controls to the MainMenu title bar of a chart window
- Adding custom WPF Controls to Chart Trader
- Modifying existing Chart Trader buttons

Important related documentation
C#
- Button
- Grid
- GridSplitter
- Menu
- MenuItem
- StackPanel

NinjaTrader
- NTMenuItem
- TabControlManager

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleWPFModifications.zip

11.5.20.2 Strategy

Strategy Overview

11.5.20.2.1 Backtesting NinjaScript Strategies with an intrabar granularity

You can submit orders to different Bars objects. This allows you the flexibility of submitting orders to different timeframes. Like in live trading, taking entry conditions from a 5min chart means executing your order as soon as possible instead of waiting until the next 5min bar starts building. You can achieve this by submitting your orders to a more granular secondary bar series to achieve an "intrabar" fill.

Key concepts in this example

- Finding entry conditions on the primary bar object

- Submitting orders to the secondary bar object for an intrabar fill

Important related documentation

- AddDataSeries()
- BarsInProgress
- EnterLong()
- BarsArray
- EnterLongLimit()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleIntrabarBacktest_NT8.zip

11.5.20.2.2 Entering on one time frame and exiting on another

You can submit orders to different bars objects. This allows you the flexibility of submitting orders to different timeframes. You can watch for trade conditions across different time frames and place orders on whichever one you want. This is useful for strategies that require more finesse in the exit than the entry. You can now enter trades on longer time frames and then monitor and exit your trade on a more granular time frame.

Key concepts in this example

- Comparing values across multiple time frames
- Submitting orders to a non-primary bar object

Important related documentation

- BarsArray
- BarsInProgress
- AddDataSeries()
- BarsSinceExitExecution()
- BarsRequiredToTrade()
- EnterLongLimit()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleMultiTimeFrameOrders_NT8.zip

11.5.20.2.3 Getting PnL from an ATM strategy

When integrating Advanced Trade Management templates into NinjaScript Strategies, the procedure for obtaining trade performance and order status is a little different than for strategies without ATM templates. The difference is that ATM templates basically take over and manage the trade separately from the NinjaScript portion of the strategy. Until the ATM trade terminates, the NinjaScript portion just observes and waits for another trading opportunity. Accessing all the trade data is still possible, but the set of commands to use differ from the traditional NinjaScript commands. This sample borrows code from the built-in strategy SampleAtmStrategy and doesn't include all the references specific to that strategy. In addition, this sample draws some profit/loss information right on the chart.

Note: This is a real-time strategy only. You will also need to first setup an ATM template titled "AtmStrategyTemplate" for this sample to work. You can set this up in the SuperDOM or Chart Trader windows.

Key concepts in this example

- Obtaining unrealized and realized profit/loss from ATM templates initiated by a NinjaScript strategy
- Keeping a running total of all the realized profits/losses

Important related documentation

- GetAtmStrategyRealizedProfitLoss()
- GetAtmStrategyUnrealizedProfitLoss()
- Draw.TextFixed()
- RoundToTickSize()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleATMPnL_NT8.zip

11.5.20.2.4 Halting a Strategy Once User Defined Conditions Are Met

For error-handling, money-management or any other reason you may want to halt your strategy from processing its' core program logic. Before you halt your strategy, it is best to close all positions and cancel all active orders to prevent the risk of having an unmanaged position in the market. We have provided two reference samples for these topics.

Key concepts in the SampleHaltBasicStrategy example*

- Using PnL statistics to determine when to halt processing of the strategy

- Cancelling active orders
- Closing active positions

Key concepts in the SampleHaltAdvancedStrategy example**

- Using a custom method to halt processing on all event-driven methods
- Advanced order handling in error situations with the OnOrderUpdate() method

* This is intended for strategies driven exclusively by the OnBarUpdate() method.
** This sample's intended audience is for advanced programmers who have programmed strategies that take advantage of event-driven methods such as, but not limited to, OnMarketData() or OnOrderUpdate() in addition to the OnBarUpdate() method.

Important related documentation

- CancelOrder()
- Order
- SystemPerformance
- AllTrades*
- TradesPerformance
- OnMarketData()
- OnOrderUpdate()

* This reference sample uses the .AllTrades property. This property will include all historical virtual trades as well as real-time trades. If you wish to only make calculations based on real-time trades you can use the .RealtimeTrades property.

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleHaltAdvancedStrategy_NT8.zip
SampleHaltBasicStrategy_NT8.zip

11.5.20.2.5 Keeping orders alive

The default behavior for NinjaTrader is to cancel limit orders if the trigger conditions are no longer true. It is possible to submit orders that stay active until cancelled by setting liveUntilCancelled to true. This sample demonstrates and explains the difference between submitting an order with isLiveUntilCancelled true and false. The comments contain a longer, more detailed explanation.

Key concepts in this example:

- How to submit an order that stays active until it is explicitly canceled*

*Another sample demonstrating how to explicitly cancel orders can be found here: Using CancelOrder() method to cancel orders

Important related documentation

- EnterLongLimit()
- isliveUntilCancelled
- CrossAbove()
- CrossBelow()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleIsLiveUntilCanceled_NT8.zip

11.5.20.2.6 Modifying the price of stop loss and profit target orders

One of the benefits of NinjaScript is the ability to automatically submit stop loss and profit target orders in real-time triggered when your entry order is filled.

Key concepts in this example

- Submitting a stop loss and profit target order using default values offset from your entry order average fill price
- Modification of the stop loss order to a break even price once a desired level of profit has been reached

Important related documentation

- SetStopLoss()
- SetProfitTarget()
- SetTrailStop()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SamplePriceModification_NT8.zip

11.5.20.2.7 Monitoring for and trading a breakout

A common concept many traders use is the idea of a breakout. Points of interest are when the price breaks out from a consolidation range or from previous highs and lows.

Key concepts in this example

- Determining and storing the first 30 bar high

- Submitting a long stop order to be filled when price breaks out from the 30 bar high
- Closing positions after a certain amount of bars have passed
- Resetting the 30 bar high at the start of every new trading session

Important related documentation

- IsFirstBarOfSession
- BarsSinceNewTradingDay
- BarsSinceEntryExecution()
- BarsSinceExitExecution()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleBreakoutStrategy_NT8.zip

11.5.20.2.8 Monitoring Stop-Loss and Profit Target Orders

At times you may have a need to monitor the stop-loss and profit target orders generated by the SetStopLoss(), SetTrailStop() or SetProfitTarget() methods. You can accomplish this by capturing their unique Order object as the the OnOrderUpdate() method is called and process them into a collection for future referencing.

Key concepts in this example

- Monitoring stop-loss order states
- Monitoring profit target order states

Important related documentation

- Order
- OrderState
- OnOrderUpdate()
- System.Collections (for List<>)

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleMonitorStopProfit_NT8.zip

11.5.20.2.9 Plotting from within a NinjaScript Strategy

When running a strategy on a chart you may find the need to plot values onto a chart. If these values are internal strategy calculations that are difficult to migrate to an indicator, you can use the following technique to achieve a plot.

With NinjaTrader 8 we introduced strategy plots which provide the ability for a strategy to render its own plots. These plots must be specific to a single panel just like indicators. If you need to have strategy plots on more than a single panel then please use the technique seen in the attached sample. You can find documentation on the standard methods for plotting in the Indicator help guide section, although the documents are for indicators the plotting items are shared between indicators and strategies.

Important related documentation:

- Plotting from a strategy with Indicator plot methods

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleStrategyPlot_NT8.zip

11.5.20.2.10 Removing draw objects from the chart

Drawing objects can be used for a number of different purposes, like keeping track of where a strategy has its entry point, profit target, and stop loss. If a strategy draws an object(s) for every trade it takes, the chart could quickly become cluttered. This sample will show how to remove the objects that aren't necessary anymore.

> **Note**: This is a real-time only strategy. Please view this strategy on a real-time data connection or the Simulated Data Feed.

Key concepts in this example

- Drawing lines at the price where the orders are that extend for the duration of the trade
- Removing those lines when the trade is over

Important related documentation

- Draw
- Line()
- RemoveDrawObject()
- RemoveDrawObjects()
- CrossAbove()

- CrossBelow()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleRemoveDrawObjects_NT8.zip

11.5.20.2.11 Resetting values at the beginning of new trading sessions

Normally calculated values are carried over between trading sessions, but sometimes you may want to reset these values to begin a trading session fresh. The technique demonstrated in this reference sample can be useful to do things like resetting counters you may be running or clearing bool flags you may have set.

Key concepts in this example

- Resetting a variable at the beginning of a new trading session
- Limiting the number of trades a strategy can make per trading session

Important related documentation

- IsFirstBarOfSession
- IsFirstTickOfBar
- EnterLong()
- ExitLong()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

https://ninjatrader.com/support/helpGuides/nt8/samples/SampleTradeLimiter_NT8.zip

11.5.20.2.12 Rounding values to the nearest tick size

When NinjaTrader receives a request to submit an order, it automatically rounds any limit price or stop price to the nearest tick for that specific instrument.

When debugging and/or printing out order information, this may not be apparent. NinjaTrader includes a Method named RoundToTickSize to apply the same internal rounding to any value you wish, which can help make comparisons easier.

Key concepts in this example

- Rounding a value to the nearest tick

Important related documentation

- RoundToTickSize()
- EnterLongLimit()
- ExitLong()
- CrossAbove()
- CrossBelow()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleRoundToTickSize_NT8.zip

11.5.20.2.13 Scaling out of a position

A common technique used by discretionary traders is scaling in and scaling out of a position. To scale out of a position refers to closing a portion of your position when you hit a profit target and then raising your stop to close your remaining portion later.

Key concepts in this example

- Submitting Profit Target orders
- Submitting Trailing Stop orders
- Closing half of your position at a time

Important related documentation

- MarketPosition
- SetProfitTarget()
- SetTrailStop()
- EntriesPerDirection*
- EntryHandling*
- SetStopLoss()

* Entry handling properties can be either programmatically set or set through the Strategy dialog window

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleScaleOut_NT8.zip

11.5.20.2.14 Separating logic to either calculate once on bar close or on every tick

Depending on your trade ideas, the timing of entries and exits could be crucial. Sometimes waiting 30 seconds for a bar to close is too long when you are trying to exit a position. To address this you could select your strategy to calculate on every single tick, but this may impact your entry timings. For example, crossover entries could flip back and forth making it difficult to place entry orders. If you are facing this issue, it is possible to separate out parts of your strategy logic to calculate on every single tick and other parts to calculate once at the end of each bar.

Key concepts in this example

- Running some logic once per bar
- Running other logic on every single tick

Important related documentation

- Calculate
- IsFirstTickOfBar
- CrossBelow()
- EnterLong()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleEnterOnceExitEveryTick_NT8.zip

11.5.20.2.15 Stopping a strategy after consecutive losers

Trending days or ranging days can make or break a strategy. If you have a system that does extremely well on trending days, you may look for a way to turn that system off during range-bound days. A simple filter you may use could be something like, "If the last three trades were consecutive losers, stop trading for the rest of the session

Key concepts in this example

- Obtaining previous trade information to decide whether or not to keep trading for the day

Important related documentation

- SystemPerformance
- TradeCollection
- AllTrades*
- EnterLong()
- ExitLong()
- IsFirstBarOfSession
- IsFirstTickOfBar

* This reference sample uses the .AllTrades property. This property will include all historical virtual trades as well as real-time trades. If you wish to only make calculations based on real-time trades you can use the .RealtimeTrades property.

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleTradeObjects_Nt8.zip

11.5.20.2.16 Trading crossovers

Similar in concept to a breakout, many traders like to trade crossovers. This can be a crossover of price from a certain threshold or even an indicator crossing over another indicator.

Key concepts in this example

- Determining and storing the first 15 bar high and low values for the current session
- Submitting long or short entry orders depending on which threshold is crossed
- Using a trail stop to exit positions

Tip: This reference sample sets Calculat to OnEachTick. The reason we are doing this is so we can submit orders as soon as a crossover occurs instead of waiting for the bar to close before submitting the order.

Important related documentation

- Calculate
- CrossAbove()
- CrossBelow()
- SetTrailStop()
- SetStopLoss()
- SetProfitTarget()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleHighLowCross_NT8.zip

11.5.20.2.17 Using a time filter to limit trading hours

A common observation many traders have made is that certain hours of the day are more volatile than others.
Depending on the trader's style they may want to trade only during the volatile hours of the day or the less volatile days of the week.

Key concepts in this example

- Comparing days of the week
- Comparing the time of day

Important related documentation

- DayOfWeek
- Time
- ToTime()
- ToDay()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleTimeFilter_NT8.zip

11.5.20.2.18 Using CancelOrder() method to cancel orders

When using NinjaTrader's Enter() and Exit() methods, the default behavior is to automatically expire them at the end of a bar unless they are resubmitted to keep them alive. Sometimes you may want more flexibility in this behavior and wish to submit orders as live-until-cancelled. When orders are submitted as live-until-cancelled, the way to cancel them is by using the CancelOrder() method.

Key concepts in this example

- Submitting live-until-cancelled entry orders
- Manually cancelling orders

Important related documentation

- CancelOrder()
- Order
- OnOrderUpdate()
- OnExecutionUpdate()
- EnterLongLimit()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop

2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleCancelOrder_NT8.zip

11.5.20.2.19 Using multiple entry/exit signals simultaneously

Sometimes you want to trade an instrument with several different possible entry conditions. To keep track of which trade used which conditions can become cumbersome if done on paper.

The attached reference sample demonstrates the following key concepts:

- Adding user definable indicators to the strategy for display on the chart
- Setting the manner in which NinjaTrader handles entry orders
- Using unique identifiers for entry and exit orders

Important methods and properties used include:

- AddChartIndicator()
- EntriesPerDirection*
- EntryHandling*

* Entry handling properties can be either programmatically set or set through the Strategy dialog window

Other methods and properties of interest include:

- EnterLongLimit()
- EnterLongStopMarket()
- EnterLongStopLimit()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleMultipleEntryExitSignals_NT8.zip

11.5.20.2.20 Using OnOrderUpdate() and OnExecution() methods to submit protective orders

The OnOrderUpdate() and OnExecutionUpdate() methods are reserved for experienced programmers.

Instead of using Set() methods to submit stop-loss and profit target orders, you can submit and update them manually through the use of Order and Execution objects in the OnOrderUpdate() and OnExecutionUpdate() methods.

The OnOrderUpdate() method is updated whenever the state of an order changes which allows you to submit and control your stop-loss and profit target orders the instant your entry order is filled.

The OnExecutionUpdate() method is updated whenever you receive an execution or a fill on your orders. This method provides you the fastest possible submission of protective orders. Utilizing the increased granularity provided in these advanced methods can be advantageous to you by providing you with maximum control of how your stop-loss and profit target orders behave.

Key concepts in this example

- Submitting live-until-cancelled entry orders
- Modifying stop-loss order to breakeven after a certain amount in profit

Important related documentation

- Order
- Execution
- OnOrderUpdate()
- OnExecutionUpdate()
- SetStopLoss()
- SetProfitTarget()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleOnOrderUpdate_NT8.zip

11.5.20.2.21 Using IsRising and IsFalling conditions in the Strategy Builder

NinjaTrader's Strategy Builder includes access to many methods and properties, including the IsRising() and IsFalling() methods. It is possible to check if an indicator (or any Series<t>) is rising or falling using these methods. You can also use High values, Low values, or any other Series<t> with IsRising() or IsFalling() to qualitatively determine the direction of the DataSeries. This simple sample demonstrates IsRising() and IsFalling() in the Strategy Builder.

> **Note**: This is a Strategy Builder sample.

Key concepts in this example

- Using IsRising() and IsFalling() in the Strategy Builder

Important methods and properties used include

- IsRising()
- IsFalling()

Other methods and properties of interest include:

- EnterLong()
- ExitLong()

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SampleIsFallingIsRising.zip

11.5.20.2.22 Using trade performance statistics for money management

For money management reasons you may want to limit your strategy from aggressive daytrading or you may want to cut your losses short on volatile sessions that are not playing out in your favor. This can be done through the utilization of the Performance object.

Key concepts in this example

- Locking in realized profits after a certain amount of gains have been achieved during the trading session
- Cutting realized losses short after a certain amount of losses have been accrued over the trading session
- Preventing aggressive amounts of trading

Important related documentation

- SystemPerformance
- TradeCollection
- AllTrades*

* This reference sample uses the .AllTrades property. This property will include all historical virtual trades as well as real-time trades. If you wish to only make calculations based on real-time trades you can use the .RealtimeTrades property.

Import instructions

1. Download the file contained in this Help Guide topic to your PC desktop
2. From the Control Center window, select the menu Tools > Import > NinjaScript
3. Select the downloaded file

SamplePnL_NT8.zip

11.5.21 Tips

Tips Overview

11.5.21.1 Adding Indicators to Strategies

When backtesting strategies it can be useful to add the indicators you use for calculations onto the chart to make it easier to check your strategy for accuracy. Instead of doing this step manually every time you run the strategy you can program it to automatically load the indicators for you.

For example:

To add a volume indicator to your charts you need to add this code snippet into the OnStateChange section of your code for the State: State.DataLoaded

```
protected override void OnStateChange()
{
     if (State == State.DataLoaded)
     {
          AddChartIndicator(VOL());
     }
}
```

To choose which panel you want your indicator plotted on you can use this code snippet into the State.DataLoaded state:

```
VOL().Panel = 2;
AddChartIndicator(VOL());
```

To customize plot colors:

```
VOL().Plots[0].Brush = Brushes.Red;     // Plots the VOL with a red plot
```

To customize plot width:

```
VOL().Plots[0].Width = 4;    // Plots the VOL bars with a width of 4
```

To customize the plot dash style:

```
VOL().Plots[0].DashStyleHelper = DashStyleHelper.Dash;
```

To customize the plot style:

```
VOL().Plots[0].PlotStyle = PlotStyle.Bar;
VOL().Plots[0].AutoWidth = true;
```

To customize lines you can do it the same way as above.

```
RSI(14, 3).Lines[0].Value = 20;
RSI(14, 3).Lines[0].Brush = Brushes.Green;
```

Remember, you need to use the AddChartIndicator() method to add your indicator if you wish to use any of the plot / line indicator customization examples.

11.5.21.2 Checking for Null References

A common object-oriented programming error is not checking for null references on your object variables This will cause an "Object reference not set to an instance of an object" error.

For example:
You create a variable that holds an Order object

```
private Order entryOrder = null;
```

But in the OnBarUpdate() method you do not check if this variable as been assigned an Order object, thus when trying to access object properties it fails and yields the "Object reference not set" error since the variable is null.

```
protected override void OnBarUpdate()
{
     if (entryOrder.Filled > 0)
          // Do something
}
```

This will generate an error because you cannot access the object or any of its properties yet. You must always check if an object variable is null before attempting to access the object.

```
protected override void OnBarUpdate()
{
     if (entryOrder == null)
     {
          entryOrder = EnterLong();
     }
     else if (entryOrder != null)
     {
          if (entryOrder.Filled > 0)
               // Do something
     }
}
```

11.5.21.3 Creating User Defined Input Parameters

You can create user defined input parameters for both NinjaScript Indicators and Strategies. Although user defined input parameters can be specified as part of the initial set up of NinjaScript Indicator or Strategies using the Wizard you may have a requirement to add new parameters at a later point in your development process. To create these parameters you will need to edit your NinjaScript code and follow these steps.

1. Open your NinjaScript file
2. Inside of the if (State == State.SetDefaults) statement, assign a value to the variable for your parameter

```
Period = 5;
```

Note: This is also where you set the default value for your parameter.

3. Scroll down to the bottom of the editor and expand the minimized "Properties" section by clicking on the + sign on the left.
4. Use the following template code for each parameter you wish to create. Please note that the type (int, double, etc) will differ depending on what type of variable you wish to create

```
[Range(1, int.MaxValue)]
[NinjaScriptProperty]
[Display(Name="Period", Description="Numbers of bars used for
calculations", Order=1, GroupName="Parameters")]
public int Period
{ get; set; }
```

5. To specify lower and upper bounds, you would modify [Range(1, int.MaxValue)]. For example:

```
// No upper bound, lower bound of 1
[Range(1, int.MaxValue)]
// No lower bound, upper bound of 100
[Range(int.MinValue, 100)]
// No lower or upper bound
[Range(int.MinValue, int.MaxValue)]
```

6. Use the "Description" field to provide a brief description of what the parameter does.
7. Pay attention to this line as the object type will vary depending on the type of parameter you wish to make:

```
public int Period
```

8. Now, wherever in your code you want to call the user-definable parameter, just use "Period".

```
if (SMA(Period)[0] > SMA(Period)[1])
    // Do something
```

11.5.21.4 Debugging your NinjaScript Code

Debugging can be a frustrating and time-consuming task. In order to make the most of your time, it is best to proceed in a methodical manner. The first step you should do is to strip your code down into simple code segments. You want to start your debugging at a point where you know the code works as expected. From there you can then add more layers of complexity. With each additional layer, you want to ensure it works as expected before adding more layers.

To begin the process of stripping down your code you can either make a new temporary NinjaScript and copy over only the key relevant code segments or you can comment out segments that are not vital to the test.

To comment out code segments you can either press the "Comment selection" button on the top toolbar in the NinjaScript editor or type "//" in front of the line. To mass comment code segments, you can use your mouse cursor and select multiple lines and press the "Comment selection" button as well. To uncomment code, remove the "//" or select the line and press the "Uncomment selection" button.

```
double limitPrice = 0; //You can comment after working code
// double stopPrice = 0;
// you can also comment out whole lines to prevent it from being
used by using //
```

Commenting code segments is also useful if you wanted to just temporarily check if your code compiles. When your NinjaScript editor pops up with errors you can click on the error message and it will bring you to the erroneous line. After you comment out the erroneous lines you should be able to compile.

The most common method you can use to ensure your code works as expected is through the use of the Print() command. The Print command will print the supplied value to the New -> NinjaScript Output window.

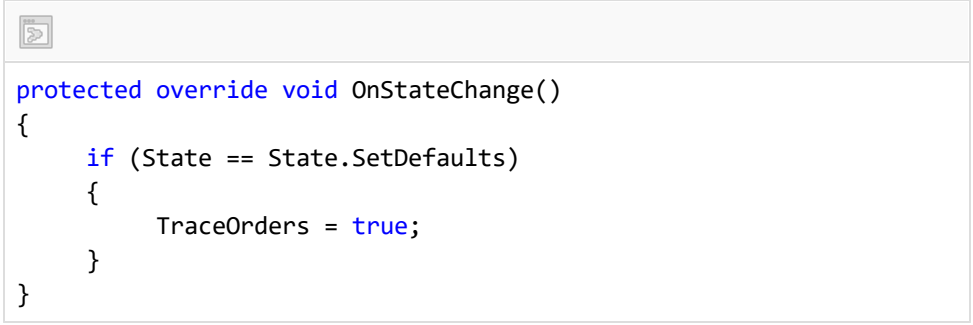
Example:

```
if (Close[0] > Open[0])
{
     Print("Code has entered If statement. Close: " + Close[0] + "
Open: " + Open[0]);
     // Do something
}
```

Placing Print() commands throughout your code logic allows you to trace where you are exactly. You can see where your code goes and where it does not go by viewing the output

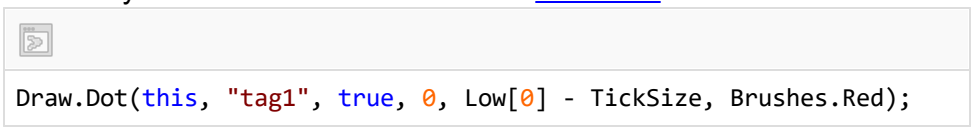
from the Output Window. Coupled with reading the error log from the Log tab in the Control Center, you can pinpoint where your code is crashing or locking up and make changes accordingly.

Debugging orders can be a bit harder though because you cannot discern the behavior state of your orders through the Print() command easily. In addition to Print(), you can use TraceOrders to help you decipher what is happening under the hood for orders. TraceOrders will print information into the Output Window that will contain details about your orders.



```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          TraceOrders = true;
     }
}
```

When debugging stop or limit orders it can also be extremely useful to draw dots along your chart of the actual stop/limit prices. This way you can visually see where your orders are at and if they should have been filled or not. Draw.Dot



```
Draw.Dot(this, "tag1", true, 0, Low[0] - TickSize, Brushes.Red);
```

Sometimes your code will compile and run fine, but it will produce inaccurate results. To check for accuracy, you can use Print() along with the Data Box to compare values. If you are doing a complex mathematical calculation, you can print the value at every step to ensure the calculations are as expected.

External references can also be very useful when debugging. They are great for gaining insight into syntax and usage of various methods. Searching Google or MSDN can provide useful examples and code snippets you can adapt to use with your own code.

Some useful resources:
NinjaScript Debugging
CSharp Essentials
C# Station
Microsoft Developer Network
The Code Project

11.5.21.5 Floating-Point Arithmetic

Some common problems that you may encounter when comparing different double values are the caveats involved with floating-point arithmetic. Because of the way computers store floating-point numbers, under certain conditions your value will be an approximate of the

actual decimal number you wanted. If this situation arises in your code, your comparison logic may not execute as you had intended even if your logic was mathematically sound on paper. To address this issue you will need to use a range comparison that takes into account the slight differences in the least significant digits of the floats.

For example, under normal mathematics we would assume double x is equivalent to double y.

```
double x = 90.10;
double y = 100 * 0.9010;
Print("double x: " + x);
Print("double y: " + y);
```

Even the output of this code segment suggests they are the same:

```
double x = 90.1
double y = 90.1
```

Unfortunately, as demonstrated by this code segment, they are not.

```
bool c = (x == y);
Print("x equals y: " + c);
```

This segment outputs the following:

```
x equals y: False
```

This means when we try to check for equality it would never evaluate to true even if it does mathematically.

```
if (x == y)
    // Do something. This will never be true.
```

Instead of comparing double x to y for an exact equality we will need to check a range.

```
if (Math.Abs(x - y) < 0.0001)
    // Do something
```

The arbitrary constant you choose to compare the range with should match the precision and accuracy of the floating-point numbers you are comparing.

Alternatively, you can check the difference between the two variables against the double.Epsilon field. double.Epsilon field represents the smallest possible double value.

```
if (x - y < double.Epsilon)
     // Do something
```

You can also use a Compare() method to accurately compare floating-point numbers. Take note that this method should only be used to compare price values since its precision is based on the instrument's tick size and may be unsuited for use in other floating-point situations.

```
double newPriceRange = Close[0] - Open[0];
double oldPriceRange = Close[1] - Open[1];
if (Instrument.MasterInstrument.Compare(newPriceRange,
oldPriceRange) == 1)
{
 // Do something
}
```

The Compare() method returns a value of "1" if the first parameter is greater than the second, "-1" if the first parameter is less than the second, and "0" if the first parameter is equal to the second.

For a more formal analysis of floating-point arithmetic, there are many resources online:
http://docs.sun.com/source/806-3568/ncg_goldberg.html
http://www.codeproject.com/dotnet/ExtremeFloatingPoint1.asp#terms

11.5.21.6 Formatting numbers

String formatting on numbers is very useful for creating readable output. This can be done through the use of the number object's ToString() method.

A common practice is printing out mathematical operations with the use of the ToString() method on the double object. What usually happens is the printing of a long string containing all the decimal places existing in the double. This sometimes makes output cluttered and hard to read. Luckily, C# has a robust set of string formatting options available to make the string more comprehendible.

Here is a list of common formatting options available in the ToString() method:

```
double c = 10.25693;
Print("No formatting: " + c.ToString());
Print("Currency formatting: " + c.ToString("C"));
Print("Exponential formatting: " + c.ToString("E"));
Print("Fixed-point formatting: " + c.ToString("F2"));
Print("General formatting: " + c.ToString("G"));
Print("Percent formatting: " + c.ToString("P0"));
Print("Formatted to 2 decimal places: " + c.ToString("N2"));
Print("Formatted to 3 decimal places: " + c.ToString("N3"));
Print("Formatted to 4 decimal places: " + c.ToString("N4"));
```

The corresponding output is as follows:

```
No formatting: 10.25693
Currency formatting: $10.26
Exponential formatting: 1.025693E+001
Fixed-point formatting: 10.26
General formatting: 10.25693
Percent formatting: 1,026 %
Formatted to 2 decimal places: 10.26
Formatted to 3 decimal places: 10.257
Formatted to 4 decimal places: 10.2569
```

For custom formatting you can use the following:

```
double phoneNumber = 9165551022;
Print("Phone number: " + phoneNumber.ToString("(###) ### - ####"));
```

Corresponding output:

```
Phone number: (916) 555 - 1022
```

For more information on general string formatting the Microsoft documentation may be of use. Many other resources can be found online through a Google search as well.
https://learn.microsoft.com/en-us/dotnet/standard/base-types/standard-numeric-format-strings
https://learn.microsoft.com/en-us/dotnet/standard/base-types/custom-numeric-format-strings

11.5.21.7 How do I resolve NinjaScript Programming Errors?

You may come across various situations where your NinjaScript files will not compile. This can include situations such as:

- You are trying to import a NinjaScript Archive File and you receive an error such as "You have custom NinjaScript files on your PC that have programming errors..."
- You are new to NinjaScript development and somehow your files will no longer compile

Because importing NinjaScript files requires compilation of your entire NinjaScript library you will first need to resolve the errors to allow for a successful compilation.

1st Step in Resolving Errors

1. Backup NinjaScript files (Tools > Export > Backup File, ensure that "NinjaScript Files" is checked and then press the "Export" button) .
2. Open the NinjaScript Editor via the menu New > NinjaScript Editor
3. Press the "F5" key on your keyboard to compile your NinjaScript library or right click in the window and click "Compile". A list of errors will appear at the bottom of the NinjaScript Editor along with the file name where each error is located.
4. Determine if you no longer need the NinjaScript file with the errors. If you no longer need it, skip to step 9.
5. Double click on the error message to open the NinjaScript with the errors. Try to resolve the error and recompile by pressing the "F5" key.
6. If the error still exists in this file, "comment out" some or all of the content in the OnBarUpdate() method and recompile by pressing the "F5" key
7. If errors still exist in this file, "comment out" some or all of the content in the OnStateChange() method and recompile by pressing the "F5" key
8. If errors still exist in this file, "comment out" any properties that in the "Properties" region that may be causing the problems and recompile by pressing the "F5" key
9. If errors still exist in this file, try to remove the file from compilation or delete the file (See "To remove or delete the erroneous file" instructions below)
10. If another NinjaScript file references a file you wish to delete, open the file that references the file you wish to delete and "comment out" or delete the reference
11. Repeat steps 2 through 10 for every NinjaScript that still has errors

To remove or delete the erroneous file

With NinjaTrader 8, we now have the option to remove a file from compilation but not delete it. This means all your code is intact but is not compiled so it will not produce errors. This also means the item is not available for use until you add it back into compilation.

1. Open the NinjaScript Editor via the menu New > NinjaScript Editor
2. Highlight the NinjaScript file you wish to prevent from being compiled, Right click on it and click "Exclude from Compilation".
3. If another NinjaScript file references the file you wish to delete, you must first remove the reference to the file you wish to delete, see step 10 above
4. You also have the option to completely delete the file as well, this is the same process as above except you would select "Remove" instead.

2nd Step in Resolving Errors

If the above procedure does not resolve all errors, you may need to reinstall NinjaTrader.

1. Backup NinjaScript files (Tools > Export > Backup File, ensure that "NinjaScript Files" is checked and then press the "Export" button) .
2. Shut down NinjaTrader
3. Uninstall NinjaTrader from the windows Control Panel Add/Remove Programs
4. Manually delete or move the folder My Documents\NinjaTrader 8
5. Reinstall the latest version of NinjaTrader from our website

11.5.21.8 Make sure you have enough bars in the data series you are accessing

A common programming error is not checking to ensure there are enough bars contained in the data series you are accessing. This will explain some of the concepts to check for this situation.

For example:

```
protected override void OnBarUpdate()
{
    if (Close[0] > Close[1])
        // Do something
}
```

In the code snippet above, the OnBarUpdate() method is called for each bar contained in your data series.

On the very first bar (think of the 1st bar on the chart from left to right) the value of "close of 1 bar ago" (Close[1]) does not yet exist and your indicator/strategy will not work and throw an exception to the Control Center Log tab "Index was out of range...".

Following are two ways to ways to resolve this:

```
protected override void OnBarUpdate()
{
    if (CurrentBar < 1)
        return;
    if (Close[0] > Close[1])
        // Do something
}
```

The resolution above is to check how many bars we have seen (CurrentBar) and to exit the OnBarUpdate() method if an insufficient number of bars has been seen.

```
protected override void OnBarUpdate()
{
    if (Close[0] > Close[Math.Min(CurrentBar, 1)])
        // Do something
}
```

The resolution above substitutes the minimum value between the current bar being processed and the desired number of bars ago value, in this case 1.

Multi Time Frame and Instrument Scripts

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
         Name    = "Multi-Time Frame & Instruments Example";
    }
    else if (State == State.Configure)
    {
        // Adds a secondary bar object to the script.
        AddDataSeries(BarsPeriodType.Minute, 5);

        // Adds an additional bar object to the script.
        AddDataSeries(BarsPeriodType.Minute, 5);
    }
}

protected override void OnBarUpdate()
{
     // Checks to ensure all Bars objects contain enough bars
before beginning
     // If this is a strategy, use BarsRequiredToTrade instead of
BarsRequiredToPlot
     if (CurrentBars[0] <= BarsRequiredToPlot || CurrentBars[1] <=
BarsRequiredToPlot || CurrentBars[2] <= BarsRequiredToPlot)
          return;
}
```

The resolution above would be used in a Multi Time Frame script. Since OnBarUpdate() processes multiple data series, we need to make sure each Data Series we reference has processed enough bars.

11.5.21.9 Order Types

Understanding the different types of entry and exit orders you can place through NinjaScript is important. As a trader, it is critical you place the right type of order depending on the current market conditions and your trading style.

Order Methods

```
EnterLong();
EnterShort();
ExitLong();
ExitShort();
```

These place market orders to either buy or sell. Market orders offer the fastest execution speed and under most conditions, guarantee that your order is filled. Be wary about using them on low volatility securities with large spreads though. You might get filled at a much higher/lower price than you expected.

```
EnterLongStopMarket();
EnterShortStopMarket();
ExitLongStopMarket();
ExitShortStopMarket();
```

These orders wait for the price of the instrument to pass your stop price. Once it passes the stop price the order becomes a market order for execution. Stop orders increase your chances of getting filled at a particular price, but are not guaranteed because they are still ultimately market orders.

```
EnterLongLimit();
EnterShortLimit();
ExitLongLimit();
ExitShortLimit();
```

Limit orders allow you to specify the price you want to be filled at. These orders are useful on low volatility instruments because they ensure you get filled at the price you specified or better. Take note that limit orders are not guaranteed to execute and may cause only partial fills.

```
EnterLongStopLimit();
EnterShortStopLimit();
ExitLongStopLimit();
ExitShortStopLimit();
```

The stop-limit order offers the trader complete control over the order. Like a stop order, the stop-limit order waits until the specified stop price has been reached. Unlike the stop order though, the stop-limit order becomes a limit order instead of a market order when the stop price is reached. The drawback for a stop-limit order is the same as all limit orders; the trader might not be filled if the limit price is never reached.

```
EnterLongMIT();
EnterShortMIT();
ExitLongMIT();
ExitShortMIT();
```

The MIT (Market If Touched) order allows the order to be submitted at Market once the price is touched. This order is similar to a stop order except the buy and sell actions are reversed. For example, a buy MIT order is submitted below market where a buy Stop would be submitted above market.

11.5.21.1 Parameter sequencing

Indicator and strategy parameters (user defined inputs) will always be displayed in an order that the user specifies in the NinjaScript file.

In the NinjaScript Editor, expand the "Properties" region of your code where all of your parameters are defined. In this example, this will be our Properties section:

```
[Range(1, int.MaxValue), NinjaScriptProperty]
[Display(ResourceType = typeof(Custom.Resource), Name = "Fast",
GroupName = "NinjaScriptStrategyParameters", Order = 0)]
public int Fast
{ get; set; }
[Range(1, int.MaxValue), NinjaScriptProperty]
[Display(ResourceType = typeof(Custom.Resource), Name = "Slow",
GroupName = "NinjaScriptStrategyParameters", Order = 1)]
public int Slow
{ get; set; }
```

In this case, the Fast parameter will show up as the first parameter with the Slow parameter showing as the second.

To switch the order around, we could modify Order. If we change Slow's Order to 0 and Fast's Order to 1 as shown below ...

```
[Range(1, int.MaxValue), NinjaScriptProperty]
[Display(ResourceType = typeof(Custom.Resource), Name = "Fast",
GroupName = "NinjaScriptStrategyParameters", Order = 1)]
public int Fast
{ get; set; }
[Range(1, int.MaxValue), NinjaScriptProperty]
[Display(ResourceType = typeof(Custom.Resource), Name = "Slow",
GroupName = "NinjaScriptStrategyParameters", Order = 0)]
public int Slow
{ get; set; }
```

... the Slow property will show first and the Fast property second.

11.5.21.1 Referencing the correct bar

When coding an indicator or strategy it is important to be able to access the intended bars for correct calculations. In NinjaScript we are able to access the bars we want through proper use of the bar's indexing.

The bar's indexing is setup in a reverse chronological order. This means "0" refers to the most recent bar, "1" refers to the previous bar, "2" refers to the bar before that one, etc.

For example, if we wanted to subtract the high and low of 10 bars ago from each other we would do this:

```
double value = High[10] - Low[10];
```

Now that we know how the indexing works there are several properties and methods at our disposal that can help us access important keystone bars. The more important ones are CurrentBar and BarsSinceNewTradingDay.

CurrentBar

CurrentBar returns an int representing the number of bars existing on the chart. This property is most useful when you want to run calculations from the very beginning of the chart.

For example, if you wanted to find the average high value of the first 10 bars on the chart you could do this:

```
double highValue = 0;
int x = CurrentBar;
while (x > CurrentBar - 10)
{
   highValue += High[x];
   x--;
}
Print("The average high value: " + highValue/10);
```

Note: A common mistake in using CurrentBar is using it in the index to access the most recent bar. In this situation, instead of doing something like Close[CurrentBar] you will want to do Close[0].

BarsSinceNewTradingDay

BarsSinceNewTradingDay is another property that can help you find the first bar of the current trading day. The difference between BarsSinceNewTradingDay and CurrentBar is that BarsSinceNewTradingDay resets its count whenever a new session begins. This means if you use it in an index it will only get you to the beginning of the current session and not any previous sessions.

For example, if you wanted to find the open of the current session you could do this:

```
double openValue = Open[Bars.BarsSinceNewTradingDay];
```

The example used in the discussion about CurrentBar can also be done with Bars.BarsSinceNewTradingDay if you wanted to calculate values based on the current session instead of the start of the chart too.

Note: If you wish to access values older than 256 bars ago you will need to ensure the MaximumBarsLookBack is set to .Infinite.

Other Properties and Methods

There are also a number of other properties and methods that can be useful in helping you locate the correct bars index to reference. Please take a look at these in the help guide:
BarsSinceEntryExecution()
BarsSinceExitExecution()
GetBar()
GetDayBar()
HighestBar()

LowestBar()
LRO()
MRO()

11.5.21.1 Strategy Position vs. Account Position

An important concept to understand prior to using NinjaScript strategies in a real-time trading environment (live brokerage account for example) is the difference between a Strategy Positionand an Account Position. A Strategy Position is a virtual position that is created by the entry and exit executions generated by a strategy and is independent from any other running strategy's position or an account position. An Account Position is the position you actually hold in a real-time trading account, whether it is a NinjaTrader internal simulation account (Sim101) or your live real-money brokerage account. In most cases, a trader would want their Strategy Position's size and market direction to be equal (in sync) to their Account Position but there are situations when this may not be the case.

For example:

- You want to run multiple strategies in the same market simultaneously where strategy A holds a LONG 1 position, strategy B holds a LONG 2 position resulting in an account that should hold a LONG 3 position in order to be in sync with both strategies
- You want to run a strategy and at the same time trade the same market the strategy is running on using discretionary tactics through one of NinjaTrader's advanced order entry window such as the SuperDOM or Chart Trader

An extremely common scenario...
An extremely common scenario is starting a NinjaScript strategy in the middle of a trading session, such as one hour after the session has begun. The NinjaScript strategy is run on each historical bar for the 1st hour of the session (it will actually run on all historical data loaded in a chart) to determine the current position state it would be in if it had been running live since the start of the session. This position state then becomes the Strategy Position for your strategy. Let us assume that during the historical hour your strategy would have entered a LONG 1 position and the position is still open. This would mean the Strategy Position is LONG 1 and since this trade was not actually executed on an account your Account Position is FLAT.

What can you do in this case?

If you want your Account Position to match your Strategy Position, you will need to place a manual order into the account the strategy is running on. Continuing from the above example, you would need to place a 1-lot market order for the market being traded into the account the strategy is running on. Alternatively, there is the ability to have your account automatically synced to your strategy position on strategy startup. To use this feature, please set "Sync account position" to true in the Strategy dialogue window. For more information on this feature please see the article here about syncing Account Positions to Strategy Positions.

What if I do not submit a manual order to sync my account?

The resulting behavior when the Strategy Position and Account Position are out of sync is when your strategy (continuing with the example above) closes the long position with a sell order it would bring the Strategy Position to flat and your Account Position to SHORT 1.

> **Critical**: *** TD AMERITRADE Users *** When starting a NinjaScript strategy, please be absolutely sure your strategy position is in sync with your account position. It is imperative that they are in sync for your NinjaScript strategy to run properly.

11.5.21.1 TraceOrders

TraceOrders is a useful property when debugging the behavior of your orders. With the use of this property, you can track orders placed, amended, and canceled. The traces displayed in the NinjaScript Output window or if used, in the OnOrderTrace Override in the script where this was set. This will provide meaningful information for diagnosis when NinjaTrader ignores, changes or cancels orders when various strategy order methods are called.

To enable TraceOrders, add this line into the OnStateChange() method in the state SetDefaults of your NinjaScript strategy:

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          TraceOrders = true;
     }
}
```

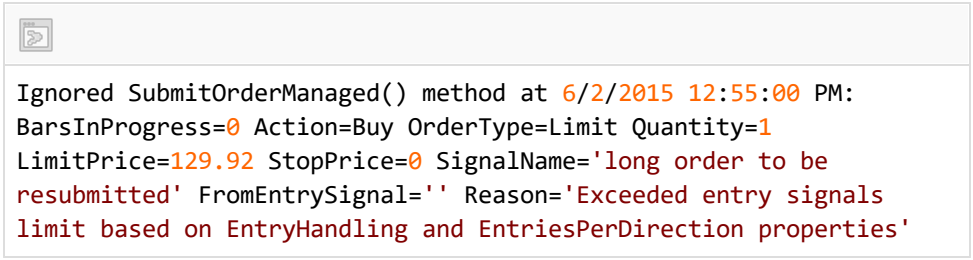
Trace output examples:

```
Entered internal SubmitOrderManaged() method at 6/2/2015 8:42:00
AM: BarsInProgress=0 Action=Buy OrderType=Market Quantity=1
LimitPrice=0 StopPrice=0 SignalName='market order'
FromEntrySignal=''
```

This trace is outputted when we place an entry order. It tells us all the pertaining properties of our order as well as the time it was submitted.

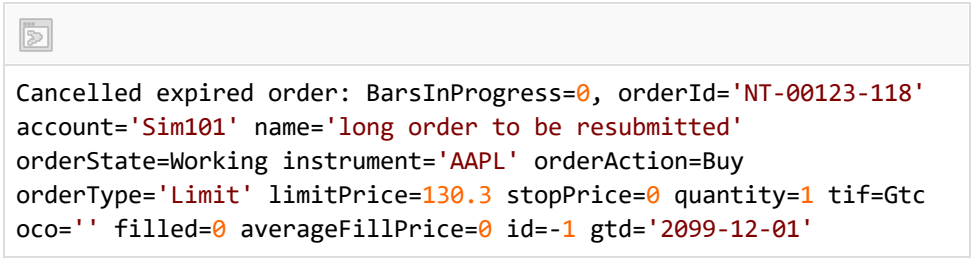
```
Amended open order at 6/2/2015 11:39:00 AM: BarsInProgress=0
Action=Buy OrderType=Limit Quantity=1 LimitPrice=130.41 StopPrice=0
 SignalName='long order to be resubmitted' FromEntrySignal=''
```

This trace tells us that a previously submitted order was modified instead of submitting a completely new order.



```
Ignored SubmitOrderManaged() method at 6/2/2015 12:55:00 PM:
BarsInProgress=0 Action=Buy OrderType=Limit Quantity=1
LimitPrice=129.92 StopPrice=0 SignalName='long order to be
resubmitted' FromEntrySignal='' Reason='Exceeded entry signals
limit based on EntryHandling and EntriesPerDirection properties'
```

This trace provides the reason why our Limit order was ignored.

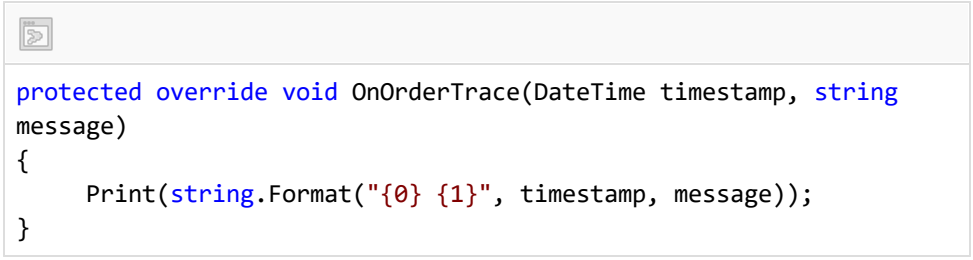


```
Cancelled expired order: BarsInProgress=0, orderId='NT-00123-118'
account='Sim101' name='long order to be resubmitted'
orderState=Working instrument='AAPL' orderAction=Buy
orderType='Limit' limitPrice=130.3 stopPrice=0 quantity=1 tif=Gtc
oco='' filled=0 averageFillPrice=0 id=-1 gtd='2099-12-01'
```

This trace tells us that our Limit order was canceled because it had expired.

A new concept in NinjaTrader 8 is the OnOrderTrace override method.

This method prevents TraceOrders from printing the traces directly to the output window but instead sends this information to the OnOrderTrace override where you can do logic or format the trace how you would like and then print only what you need to see.



```
protected override void OnOrderTrace(DateTime timestamp, string
message)
{
     Print(string.Format("{0} {1}", timestamp, message));
}
```

These examples illustrate the most common traces you will run across. They are mostly useful in determining the reason your orders are not submitted or cancelled. TraceOrders will only show you what is happening under the hood when you submit orders, but it will not tell you what happens after the order is submitted. To determine the behavior of your orders after submission you will need to look into your NinjaTrader trace logs. You can view those either through the "Log" tab on the "Control Center" or from the trace folder in My Documents\NinjaTrader 8\trace\.

For more information on how to debug your NinjaScript please review the Debugging tip.

11.5.21.14 User Definable Color Inputs

User definable inputs do not need to be limited to numeric values. You can have brushes as an input as well. To do this you will need to make a public property for a Brush object instead of an int or double.

In the "Properties" region of your code, there is only a slight change in the code snippet you would normally use to create user definable inputs.

```
[XmlIgnore()]
[Display(Name = "BorderBrush", GroupName = "NinjaScriptParameters",
 Order = 0)]
```

The second difference is in the next line of code:

```
public Brush BorderBrush
{ get; set; }
```

This creates a brush input for use in the dialog window when we try to add the NinjaScript to a chart.

Some additional extra code that is required for creating a color input is to serialize the brush. Serialization is necessary for NinjaTrader to use the brush input throughout the program. Please note that serialization is a general concept not exclusive to brush inputs. There may be other struct/classes (for a TimeSpan example, please cross reference this page) that you could use in your code that would also need to have their "value" properties serialized.

```
[Browsable(false)]
public string BorderBrushSerialize
{
 get { return Serialize.BrushToString(BorderBrush); }
    set { BorderBrush = Serialize.StringToBrush(value); }
}
```

Attached is a NinjaScript indicator sample that uses two user definable brush inputs to determine the color of a drawn rectangle.
SampleBrushInput.zip

11.5.21.15 Using [] brackets

In C#, square brackets represent a way to access values stored within an collection. NinjaScript comes with quite a few collections that we call ISeries objects which could be accessed with square brackets. ISeries objects are linked to the underlying bars series in that

they hold the same number of values as the number of bars on a chart. For example, to get the close price one bar ago, you would use Close[1] since the value of 1 within the square brackets represents the number of bars ago whose value you wish to reference. As another example, to get the high three bars ago, you would use High[3].

```
double close1 = Close[1]; // gets the close price one bar ago
double high3 = High[3]; // gets the high of three bars ago
double low = Low; // results in compile error. Low is an array, and
can't be accessed directly. It should be Low[n Bars ago].
```

Many of NinjaTrader's indicators store their values in Series as well, generally in a Plot. Plots are essentially a Series<double> object and to retrieve values from them you need to specify which value you want to access. In most cases, you'd like the current value, so you could use SMA(14)[0], not just SMA(14). SMA(14) is the Indicator its self or Series, and you can't access its values by calling it directly. Using SMA(14)[0] retrieves the part of the Series you're interested in--the most current value.

```
double SMA_current = SMA(14)[0]; // gets the current value of the
SMA
double SMA_1 = SMA(14)[1]; // gets the SMA value one bar ago
double SMA_value = SMA(14); // results in compile error. SMA(14)
is a Series and the variable SMA_value of type double can't hold a
Series.
```

Most of the time, you need an index value (number in the square brackets), but there are also cases when you need to use the ISeries instead. CrossAbove() and CrossBelow() are two key examples. If you look at the reference page for CrossAbove(), the two method signatures (overloads) look like this:

```
CrossAbove(ISeries<double> series1, ISeries<double> series2, int
lookBackPeriod)
CrossAbove(ISeries<double> series1, double value, int lookBackPeriod)
```

This means the first variable must always be a ISeries<double> object, and the second variable can be either another ISeries<double> or a double value (100, 70.25, etc). To specify a ISeries<double> object, you can just leave off the square brackets. For example if(CrossAbove(SMA(14), SMA(28), 1)) checks if the 14 period SMA has crossed above the 28 period SMA within the last bar. if(CrossAbove(SMA(14)[0], SMA(28)[0], 1)) would give you a

compile error because it expects a ISeries<double> as input, not a double value (which is returned when an index is present).

```
if (CrossAbove(SMA(14), SMA(28), 1)) // works fine
if (CrossAbove(SMA(14), 1000, 1)) // works fine, this uses a double
for the second argument. See the above overload.
if (CrossAbove(SMA(14)[0], SMA(28)[0], 1)) // compile error: SMA(14)
[0] is a double, not a ISeries<double>
if (CrossAbove(SMA(14), SMA(28)[0], 1)) // would work fine with a
ISeries<double> as first argument and a double as the second
argument
```

11.6 Language Reference

NinjaScript Language Reference

- Add On
- Bars Type
- Chart
- Chart Style
- Common
- Drawing
- Drawing Tool
- Import Type
- Indicator
- Indicator Methods
- ISeries<T>
- Market Analyzer Column
- Instrument
- Optimization Fitness
- Optimizer
- Performance Metrics
- Share Service
- Strategy
- SuperDOM Column

11.6.1 Alphabetical Reference

11.6.2 Common

The following section documents methods and properties available to every NinjaScript type that access various forms of data including bar data, price data, and statistical forms of data. The Common section is broken into several categories pertaining to distinct NinjaScript objects or concepts. An index of topics under the Common section can be found below:

Attributes	Documents both .NET native and NinjaScript custom attributes which are commonly used to define the behavior of a NinjaScript property or object
Alert, Debug, Share	Documents methods for triggering alerts, printing debug messages, and using Share Services
Analytical	Documents methods and properties useful for analyzing and identifying specific conditions within Series<T> collections
Bars	Represents the data returned from the historical data repository
Charts	Covers information related to accessing chart related data
Drawing	Documents the drawing of custom shapes, lines, text and colors on your price and indicator panels from both Indicators and Strategies
Instrument	Represents an instance of a Master Instrument
ISeries<T>	Documents the interface that is implemented by all NinjaScript classes that manage historical data as an `ISeries<double>` used for indicator input, and other object data
OnBarUpdate()	An event driven method which is called whenever a bar is updated
OnFundamentalData()	An event driven method which is called for every change in fundamental data

OnMarketDepth()	An event driven method which is called and guaranteed to be in the correct sequence for every change in level two market data
OnStateChange()	An event driven method which is called whenever the script enters a new State
SessionIterator	An interface which allows you to traverse through various trading hours data elements which apply to a segment of bars
System Indicator Methods	Documents syntax and return values for system indicator methods
TradingHours	Represents the Trading Hours information returned from the current bars series
Name	Determines the listed name of the NinjaScript object
IsVisible	Determines if the current NinjaScript object should be visible on the chart
DisplayName	Determines the text display on the chart panel
Description	Text which is used on the UI's information box to be display to a user when configuration a NinjaScript object
Clone()	Used to override the default NinjaScript Clone() method which is called any time an instance of a NinjaScript object is created
TriggerCustomEvent()	Provides a way to use your own custom events (such as a Timer object) so that internal NinjaScript indexes and pointers are correctly set prior to processing user code triggered by your custom event

11.6.2.1 AddDataSeries()

Definition

Adds a Bars object for developing a multi-series (multi-time frame or multi-instrument) NinjaScript.

Related Methods and Properties

AddHeikenAshi()	This method adds a Heiken Ashi Bars object for multi-series NinjaScript.
AddKagi()	This method adds a Kagi Bars object for multi-series NinjaScript.
AddLineBreak()	This method adds a Line Break Bars object for multi-series NinjaScript.
AddPointAndFigure()	This method adds a Point-and-Figure Bars object for multi-series NinjaScript.
AddRenko()	This method adds a Renko Bars object for multi-series NinjaScript.
AddVolumetric()	This method adds a Order Flow Volumetric Bars object for multi-series NinjaScript.
BarsArray	An array holding Bars objects that are added via the AddDataSeries() method.
BarsInProgress	An index value of the current Bars object that has called the OnBarUpdate() method.
BarsPeriods	Holds an array of BarsPeriod objects synchronized to the number of unique Bars objects held within the parent NinjaScript object.
CurrentBars	Holds an array of int values representing the number of the current bar in a Bars object.

Syntax

The following syntax will add another Bars object for the primary instrument of the script.

```
AddDataSeries(BarsPeriod barsPeriod)
AddDataSeries(BarsPeriodType periodType, int period)
```

The following syntax allows you to add another Bars object for a different instrument to the script:

```
AddDataSeries(string instrumentName, BarsPeriodType periodType, int period)
AddDataSeries(string instrumentName, BarsPeriodType periodType, int period,
MarketDataType marketDataType)
```

```
AddDataSeries(string instrumentName, BarsPeriod barsPeriod)
AddDataSeries(string instrumentName, BarsPeriod barsPeriod, string tradingHoursName)
AddDataSeries(string instrumentName, BarsPeriod barsPeriod, string tradingHoursName,
bool? isResetOnNewTradingDay)
AddDataSeries(string instrumentName, BarsPeriod barsPeriod, int barsToLoad, string
tradingHoursName, bool? isResetOnNewTradingDay)
AddDataSeries(string instrumentName) //only for R15 and higher
```

Warning:

- This method should **ONLY** be called from the OnStateChange() method during **State.Configure**
- Should your script be the host for other scripts that are creating indicators and series dependent resources in **State.DataLoaded**, please make sure that the host is doing the same **AddDataSeries()** calls as those hosted scripts would. For further reference, please also review the 2nd example below and the 'Adding additional Bars Objects to NinjaScript' section in Multi-Time Frame & Instruments
- Arguments supplied to **AddDataSeries()** should be hardcoded and **NOT** dependent on run-time variables which cannot be reliably obtained during State.Configure (e.g., Instrument, Bars, or user input). Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided. Trying to load bars dynamically may result in an error similar to: **Unable to load bars series. Your NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.**
- When adding multiple Data Series of the same instrument and the same Bar Type, the 'barsToLoad' property will only be effective on the first added series. Subsequent series with a different barsToLoad setting will not load a different number of bars than the first series.
- The `AddDataSeries(string instrumentName)` overload allows loading a different instrument yet using the same BarsPeriod. This could not be supported for Strategy Analyzer use with the 'Optimize Data Series' option enabled, doing so may result in an error similar to: **Unable to load bars series. Your NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.**
- If your NinjaScript object is using `AddDataSeries()` allowing to specify a `tradingHoursName`, please keep in mind that: An indicator / strategy with multiple DataSeries of the same instrument will only process realtime OnBarUpdate() calls when a tick occurs in session of the trading hour template of all added series. Any ticks not processed will be queued and processed as a tick comes in for all subsequent DataSeries.
- When instantiating indicators in a Multi-Series script in OnStateChange, the input any hosted indicator is running on should be explicitly stated

Parameters

instrumentName	A `string` determining instrument name such as "MSFT"
barsPeriod	The BarsPeriod object (period type and interval)
periodType	The BarsType used for the bars period Possible values are: • BarsPeriodType.Tick • BarsPeriodType.Volume • BarsPeriodType.Range • BarsPeriodType.Second • BarsPeriodType.Minute • BarsPeriodType.Day • BarsPeriodType.Week • BarsPeriodType.Month • BarsPeriodType.Year
period	An `int` determining the period interval such as "3" for 3 minute bars
marketDataType	The MarketDataType used for the bars object (last, bid, ask) Possible values are: • MarketDataType.Ask • MarketDataType.Bid • MarketDataType.Last **Note**: Please see the article here on using Bid/Ask series.
tradingHoursName	A `string` determining the trading hours template for the instrument
isResetOnNewTradingDay	A nullable `bool`* determining if the Bars object should Break at EOD

	*Will accept `true`, `false` or `null` as the input. If null is used, the data series will use the settings of the primary data series.
barsToLoad	An `int` determining the number of historical bars to load

Tips:
1. You can optionally add the exchange name as a suffix to the symbol name. This is only advised if the instrument has multiple possible exchanges that it can trade on and it is configured within the Instruments window. For example: `AddDataSeries("MSFT Arca", BarsPeriodType.Minute, 5);`
2. You can add a custom BarsType which is installed on your system by casting the registered enum value for that BarsPeriodType. For example: `AddDataSeries((BarsPeriodType)14, 10);`
3. You can specify optional BarsPeriod values (such as Value2) of a custom BarsType in the BarsPeriod object initializer. For example: `AddDataSeries(new BarsPeriod() { BarsPeriodType = (BarsPeriodType)14, Value = 10, Value2 = 20 });`
4. For the instrument name parameter `null` could be passed in, resulting in the primary data series instrument being used.

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Add a 5 minute Bars object - BarsInProgress index = 1
          AddDataSeries(BarsPeriodType.Minute, 5);

          // Add a 100 tick Bars object for the ES 09-16 contract -
BarsInProgress index = 2
          AddDataSeries("ES 09-16", BarsPeriodType.Tick, 100);
     }
}

protected override void OnBarUpdate()
{
     // Ignore bar update events for the supplementary - Bars
object added above
     if (BarsInProgress == 1 || BarsInProgress == 2)
          return;

     // Go long if we have three up bars on all bars objects
     if (Close[0] > Open[0] && Closes[1][0] > Opens[1][0] &&
Closes[2][0] > Opens[2][0])
          EnterLong();
}
```

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Our hosting script needs to have the AddDataSeries
call included as well, which the Pivots indicator we call in the
2nd statement below
          // also has per default in it's own State.Configure
method. This is required since our Pivots indicator below is
created in State.DataLoaded           // (which is happening after
State.Configure and it depends on the AddDataSeries call to have
the bars available to properly calculate in
          // daily bars mode.
          AddDataSeries(BarsPeriodType.Day, 1);
     }
     else if (State == State.DataLoaded)
     {
          //In this state, we pass the 1 day series to the Pivots
indicator (as BarsArray[1]) and create its instance
          pivots = Pivots(BarsArray[1], PivotRange.Weekly,
HLCCalculationMode.DailyBars, 0, 0, 0, 20);
     }
}
```

11.6.2.1.1 AddHeikenAshi()

Definition

Similar to the AddDataSeries() method for adding Bars objects, this method adds a Heiken Ashi Bars object for multi-series NinjaScript.

Notes:

1. When running NinjaScript, you will be able to choose the first instrument and bar interval to run on. This first Bars object will carry a BarsInProgress index of 0.
2. In a multi-time frame and multi-instrument NinjaScript, supplementary Bars objects are added via this method in State.Configure state of the OnStateChange() method and given an incremented BarsInProgress index value. See additional information on running multi-bars scripts.
3. The BarsInProgress property can be used to filter updates between different bars series
4. If using OnMarketData(), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same). The market data subscription behavior occurs both in real-time and during TickReplay historical
5. For adding regular Bars types please use AddDataSeries()

6. A **Tick Replay** indicator or strategy **CANNOT** use a **MarketDataType.Ask** or **MarketDataType.Bid** series. Please see Developing for Tick Replay for more information.

Syntax

```
AddHeikenAshi(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, Data.MarketDataType marketDataType)
AddHeikenAshi(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, Data.MarketDataType marketDataType, string tradingHoursName)
AddHeikenAshi(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, Data.MarketDataType marketDataType, string tradingHoursName, bool? isResetOnNewTradingDay)
```

Warnings:

- This method should **ONLY** be called from the OnStateChange() method during **State.Configure**
- Should your script be the host for other scripts that are creating indicators and series dependent resources in **State.DataLoaded**, please make sure that the host is doing the same **AddHeikenAshi()** calls as those hosted scripts would. For further reference, please also review the 'Adding additional Bars Objects to NinjaScript' section in Multi-Time Frame & Instruments
- Arguments supplied to **AddHeikenAshi()** should be hardcoded and **NOT** dependent on run-time variables which cannot be reliably obtained during State.Configure (e.g., Instrument, Bars, or user input). Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided. Trying to load bars dynamically may result in an error similar to: **Unable to load bars series. Your NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.**

Parameters

instrumentName	A `string` determining instrument name such as "MSFT"
baseBarsPeriodType	The underlying BarsType used for the Heiken Ashi bars period. Possible values are: • BarsPeriodType.Tick • BarsPeriodType.Volume

	• BarsPeriodType.Range • BarsPeriodType.Second • BarsPeriodType.Minute • BarsPeriodType.Day • BarsPeriodType.Week • BarsPeriodType.Month • BarsPeriodType.Year
baseBarsPeriodTypeValue	An `int` determining the underlying period interval such as "3" for 3 minute bars
marketDataType	The MarketDataType used for the bars object (last, bid, ask) Possible values are: • MarketDataType.Ask • MarketDataType.Bid • MarketDataType.Last **Note**: Please see the article here on using Bid/Ask series.
tradingHoursName	A `string` determining the trading hours template for the instrument
isResetOnNewTradingDay	A nullable `bool`* determining if the Bars object should Break at EOD *Will accept `true`, `false` or `null` as the input. If null is used, the data series will use the settings of the primary data series.

Tip: You can optionally add the exchange name as a suffix to the symbol name. This is only advised if the instrument has multiple possible exchanges that it can trade on and it is configured within the Instruments window. For example: `AddHeikenAshi("MSFT Arca", BarsPeriodType.Minute, 1, MarketDataType.Last);`

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
       Name = "Examples Indicator";
    }
    else if (State == State.Configure)
    {
       // Add a 1 minute Heiken Ashi Bars object for the ES 03-18
contract - BarsInProgress index = 1
       AddHeikenAshi("ES 03-18", BarsPeriodType.Minute, 1,
MarketDataType.Last);
   }
}

protected override void OnBarUpdate()
{
     // Ignore the primary Bars object and only process the Heiken
Ashi object
     if (BarsInProgress == 1)
     {
          // Do something;
     }
}
```

11.6.2.1.2 AddKagi()

Definition

Similar to the AddDataSeries() method for adding Bars objects, this method adds a Kagi Bars object for multi-series NinjaScript.

Notes:

1. When running NinjaScript, you will be able to choose the first instrument and bar interval to run on. This first Bars object will carry a BarsInProgress index of 0.
2. In a multi-time frame and multi-instrument NinjaScript, supplementary Bars objects are added via this method in State.Configure state of the OnStateChange() method and given an incremented BarsInProgress index value. See additional information on running multi-bars scripts.
3. The BarsInProgress property can be used to filter updates between different bars series
4. If using OnMarketData(), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same). The market data subscription behavior occurs both in real-time and during TickReplay historical
5. For adding regular Bars types please use AddDataSeries()

6. A **Tick Replay** indicator or strategy **CANNOT** use a **MarketDataType.Ask** or **MarketDataType.Bid** series. Please see Developing for Tick Replay for more information.

Syntax

```
AddKagi(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, int reversal, Data.ReversalType reversalType,
Data.MarketDataType marketDataType)
AddKagi(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, int reversal, Data.ReversalType reversalType,
Data.MarketDataType marketDataType, string tradingHoursName)
AddKagi(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, int reversal, Data.ReversalType reversalType,
Data.MarketDataType marketDataType, string tradingHoursName, bool?
isResetOnNewTradingDay)
```

Warnings:

- This method should **ONLY** be called from the OnStateChange() method during **State.Configure**
- Should your script be the host for other scripts that are creating indicators and series dependent resources in **State.DataLoaded**, please make sure that the host is doing the same **AddKagi()** calls as those hosted scripts would. For further reference, please also review the 'Adding additional Bars Objects to NinjaScript' section in Multi-Time Frame & Instruments
- Arguments supplied to **AddKagi()** should be hardcoded and **NOT** dependent on run-time variables which cannot be reliably obtained during State.Configure (e.g., Instrument, Bars, or user input). Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided. Trying to load bars dynamically may result in an error similar to: **Unable to load bars series. Your NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.**

Parameters

instrumentName	A `string` determining instrument name such as "MSFT"
baseBarsPeriodType	The underlying BarsType used for the Kagi bars period Possible values are:

	• BarsPeriodType.Day • BarsPeriodType.Minute • BarsPeriodType.Second • BarsPeriodType.Tick • BarsPeriodType.Volume
baseBarsPeriodTypeValue	An `int` determining the underlying period interval such as "3" for 3 minute bars
reversal	An `int` determining the required price movement in the reversal direction before a reversal is identified on the chart
reversalType	An enum determining the mode reversal period is based. Possible values are: • ReversalType.Percent • ReversalType.Tick
marketDataType	The MarketDataType used for the bars object (last, bid, ask) Possible values are: • MarketDataType.Ask • MarketDataType.Bid • MarketDataType.Last **Note**: Please see the article here on using Bid/Ask series.
tradingHoursName	A `string` determining the trading hours template for the instrument
isResetOnNewTradingDay	A nullable `bool`* determining if the Bars object should Break at EOD

	*Will accept `true`, `false` or `null` as the input. If null is used, the data series will use the settings of the primary data series.

Tip: You can optionally add the exchange name as a suffix to the symbol name. This is only advised if the instrument has multiple possible exchanges that it can trade on and it is configured within the Instruments window. For example: `AddKagi("MSFT Arca", PeriodType.Minute, 1, 2, ReversalType.Tick, MarketDataType.Last)`

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
       Name = "Examples Indicator";
    }
    else if (State == State.Configure)
    {
       // Add a 1 minute Kagi Bars object for the ES 03-18 contract
- BarsInProgress index = 1
        AddKagi("ES 03-18",  PeriodType.Minute, 1, 2,
ReversalType.Tick, MarketDataType.Last);
   }
}

protected override void OnBarUpdate()
{
     // Ignore the primary Bars object and only process the Kagi
Bars object
     if (BarsInProgress == 1)
     {
          // Do something;
     }
}
```

11.6.2.1.3 AddLineBreak()

Definition

Similar to the AddDataSeries() method for adding Bars objects, this method adds a Line Break Bars object for multi-series NinjaScript.

Notes:
1. When running NinjaScript, you will be able to choose the first instrument and bar interval to run on. This first Bars object will carry a BarsInProgress index of 0.
2. In a multi-time frame and multi-instrument NinjaScript, supplementary Bars objects are added via this method in State.Configure state of the OnStateChange() method and given an incremented BarsInProgress index value. See additional information on running multi-bars scripts.
3. The BarsInProgress property can be used to filter updates between different bars series
4. If using OnMarketData(), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same). The market data subscription behavior occurs both in real-time and during TickReplay historical
5. For adding regular Bars types please use AddDataSeries()
6. A **Tick Replay** indicator or strategy **CANNOT** use a **MarketDataType.Ask** or **MarketDataType.Bid** series. Please see Developing for Tick Replay for more information.

Syntax

```
AddLineBreak(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, int lineBreakCount, Data.MarketDataType marketDataType)
AddLineBreak(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, int lineBreakCount, Data.MarketDataType marketDataType,
string tradingHoursName)
AddLineBreak(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, int lineBreakCount, Data.MarketDataType marketDataType,
string tradingHoursName, bool? isResetOnNewTradingDay)
```

Warnings:
- This method should **ONLY** be called from the OnStateChange() method during **State.Configure**
- Should your script be the host for other scripts that are creating indicators and series dependent resources in **State.DataLoaded**, please make sure that the host is doing the same **AddLineBreak()** calls as those hosted scripts would. For further reference, please also review the 'Adding additional Bars Objects to NinjaScript' section in Multi-Time Frame & Instruments
- Arguments supplied to **AddLineBreak()** should be hardcoded and **NOT** dependent on run-time variables which cannot be reliably obtained during State.Configure (e.g., Instrument, Bars, or user input). Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided. Trying to load bars dynamically may result in an error similar to: **Unable to load bars series. Your NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.**

Parameters

instrumentName	A `string` determining instrument name such as "MSFT"

baseBarsPeriodType	The underlying BarsType used for the LineBreak bars period
	
	Possible values are:
	
	`BarsPeriodType.Day`
	`BarsPeriodType.Minute`
	`BarsPeriodType.Second`
	`BarsPeriodType.Tick`
	`BarsPeriodType.Volume`

baseBarsPeriodTypeValue	An `int` determining the underlying period interval such as "3" for 3 minute bars
lineBreakCount	An `int` determining the number of bars back used to calculate a line break
marketDataType	The MarketDataType used for the bars object (last, bid, ask) Possible values are: • MarketDataType.Ask • MarketDataType.Bid • MarketDataType.Last **Note**: Please see the article here on using Bid/Ask series.
tradingHoursName	A `string` determining the trading hours template for the instrument
isResetOnNewTradingDay	A nullable `bool`* determining if the Bars object should Break at EOD

	*Will accept `true`, `false` or `null` as the input. If null is used, the data series will use the settings of the primary data series.

Tip: You can optionally add the exchange name as a suffix to the symbol name. This is only advised if the instrument has multiple possible exchanges that it can trade on and it is configured within the Instruments window. For example: `AddLineBreak("MSFT Arca", PeriodType.Minute, 1, 3, MarketDataType.Last)`

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";
   }

   if (State == State.Configure)
   {
      // Add a 1 minute Line Break Bars object for the ES 03-18 -
BarsInProgress index = 1
      AddLineBreak("ES 03-18", BarsPeriodType.Minute, 1, 3,
MarketDataType.Last);
   }
}

protected override void OnBarUpdate()
{
     // Ignore the primary Bars object and only process the Line
Break Bars object
     if (BarsInProgress == 1)
     {
          // Do something;
     }
}
```

11.6.2.1.4 AddPointAndFigure()

Definition

Similar to the AddDataSeries() method for adding Bars objects, this method adds a Point-and-Figure Bars object for multi-series NinjaScript.

Notes:
1. When running NinjaScript, you will be able to choose the first instrument and bar interval to run on. This first Bars object will carry a BarsInProgress index of 0.
2. In a multi-time frame and multi-instrument NinjaScript, supplementary Bars objects are added via this method in State.Configure state of the OnStateChange() method and given an incremented BarsInProgress index value. See additional information on running multi-bars scripts.
3. The BarsInProgress property can be used to filter updates between different bars series
4. If using OnMarketData(), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same). The market data subscription behavior occurs both in real-time and during TickReplay historical
5. For adding regular Bars types please use AddDataSeries()
6. A **Tick Replay** indicator or strategy **CANNOT** use a **MarketDataType.Ask** or **MarketDataType.Bid** series. Please see Developing for Tick Replay for more information.

Syntax

```
AddPointAndFigure(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, int boxSize, int reversal, Data.PointAndFigurePriceType
pointAndFigurePriceType, Data.MarketDataType marketDataType)
AddPointAndFigure(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, int boxSize, int reversal, Data.PointAndFigurePriceType
pointAndFigurePriceType, Data.MarketDataType marketDataType, string tradingHoursName)
AddPointAndFigure(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, int boxSize, int reversal, Data.PointAndFigurePriceType
pointAndFigurePriceType, Data.MarketDataType marketDataType, string tradingHoursName,
bool? isResetOnNewTradingDay)
```

Warnings:
- This method should **ONLY** be called from the OnStateChange() method during **State.Configure**
- Should your script be the host for other scripts that are creating indicators and series dependent resources in **State.DataLoaded**, please make sure that the host is doing the same **AddPointAndFigure()** calls as those hosted scripts would. For further reference, please also review the 'Adding additional Bars Objects to NinjaScript' section in Multi-Time Frame & Instruments
- Arguments supplied to **AddPointAndFigure()** should be hardcoded and **NOT** dependent on run-time variables which cannot be reliably obtained during State.Configure (e.g., Instrument, Bars, or user input). Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided. Trying to load bars dynamically may result in an error similar to: **Unable to load bars series. Your**

NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.

Parameters

instrumentName	A `string` determining instrument name such as "MSFT"
baseBarsPeriodType	The underlying BarsType used for the Point-and-Figure bars period Possible values are: • `BarsPeriodType.Day` • `BarsPeriodType.Minute` • `BarsPeriodType.Second` • `BarsPeriodType.Tick` • `BarsPeriodType.Volume`
baseBarsPeriodTypeValue	An `int` determining the underlying period interval such as "3" for 3 minute bars
boxSize	An `int` determining the price movement signified by the X's and O's of a Point-and-Figure chart
reversal	An `int` determining the number of boxes the price needs to move in the reversal direction before a new column will be built
pointAndFigurePriceType	Determines where to base reversal calculations Possible values are: • `PointAndFigurePriceType.Close` • `PointAndFigurePriceType.HighsAndLows`
marketDataType	The MarketDataType used for the bars object (last, bid, ask) Possible values are: • MarketDataType.Ask

	• MarketDataType.Bid • MarketDataType.Last **Note**: Please see the article here on using Bid/Ask series.
tradingHoursName	A `string` determining the trading hours template for the instrument
isResetOnNewTradingDay	A nullable `bool`* determining if the Bars object should Break at EOD *Will accept `true`, `false` or `null` as the input. If null is used, the data series will use the settings of the primary data series.

Tip: You can optionally add the exchange name as a suffix to the symbol name. This is only advised if the instrument has multiple possible exchanges that it can trade on and it is configured within the Instruments window. For example: `AddPointAndFigure("MSFT Arca", BarsPeriodType.Minute, 1, 2, 3, PointAndFigurePriceType.Close, MarketDataType.Last)`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Add a 1 minute Point-and-Figure Bars object for the ES
03-18 contract - BarsInProgress index = 1
         AddPointAndFigure("ES 03-18", BarsPeriodType.Minute, 1, 2,
 3, PointAndFigurePriceType.Close, MarketDataType.Last);
     }
}

protected override void OnBarUpdate()
{
     // Ignore the primary Bars object and only process the Point-
and-Figure Bars object
     if (BarsInProgress == 1)
     {
          // Do something;
     }
}
```

11.6.2.1.5 AddRenko()

Definition

Similar to the AddDataSeries() method for adding Bars objects, this method adds a Renko Bars object for multi-series NinjaScript.

Notes:

1. When running NinjaScript, you will be able to choose the first instrument and bar interval to run on. This first Bars object will carry a BarsInProgress index of 0.
2. In a multi-time frame and multi-instrument NinjaScript, supplementary Bars objects are added via this method in State.Configure state of the OnStateChange() method and given an incremented BarsInProgress index value. See additional information on running multi-bars scripts.
3. The BarsInProgress property can be used to filter updates between different bars series
4. If using OnMarketData(), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same). The market data subscription behavior occurs both in real-time and during TickReplay historical
5. For adding regular Bars types please use AddDataSeries()
6. A **Tick Replay** indicator or strategy **CANNOT** use a **MarketDataType.Ask** or **MarketDataType.Bid** series. Please see Developing for Tick Replay for more information.

Syntax

```
AddRenko(string instrumentName, int brickSize, Data.MarketDataType marketDataType)
AddRenko(string instrumentName, int brickSize, Data.MarketDataType marketDataType,
string tradingHoursName)
AddRenko(string instrumentName, int brickSize, Data.MarketDataType marketDataType,
string tradingHoursName, bool?isResetOnNewTradingDay)
```

Warnings:

- This method should **ONLY** be called from the OnStateChange() method during **State.Configure**
- Should your script be the host for other scripts that are creating indicators and series dependent resources in **State.DataLoaded**, please make sure that the host is doing the same **AddRenko()** calls as those hosted scripts would. For further reference, please also review the 'Adding additional Bars Objects to NinjaScript' section in Multi-Time Frame & Instruments
- Arguments supplied to **AddRenko()** should be hardcoded and **NOT** dependent on run-time variables which cannot be reliably obtained during State.Configure (e.g., Instrument, Bars, or user input). Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided. Trying to load bars dynamically may result in an error similar to: **Unable to load bars series. Your NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.**

Parameters

`instrumentName`	A `string` determining instrument name such as "MSFT"
`brickSize`	An `int` determining the size (in ticks) of each bar
`marketDataType`	The MarketDataType used for the bars object (last, bid, ask) Possible values are: • MarketDataType.Ask • MarketDataType.Bid • MarketDataType.Last **Note**: Please see the article here on using Bid/Ask series.

tradingHoursName	A string determining the trading hours template for the instrument
isResetOnNewTradingDay	A nullable bool* determining if the Bars object should Break at EOD *Will accept true, false or null as the input. If null is used, the data series will use the settings of the primary data series.

Tip: You can optionally add the exchange name as a suffix to the symbol name. This is only advised if the instrument has multiple possible exchanges that it can trade on and it is configured within the Instruments window. For example: `AddRenko("MSFT Arca", 2, MarketDataType.Last)`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Add a 1 minute Renko Bars object for the ES 03-18
contract - BarsInProgress index = 1
          AddRenko("ES 03-18", 2, MarketDataType.Last);
     }
}

protected override void OnBarUpdate()
{
     // Ignore the primary Bars object and only process the Renko
Bars object
     if (BarsInProgress == 1)
     {
          // Do something;
     }
}
```

11.6.2.1.6 AddVolumetric()

Definition

Similar to the AddDataSeries() method for adding Bars objects, this method adds a Order Flow Volumetric Bars object for multi-series NinjaScript.

Notes:
1. When running NinjaScript, you will be able to choose the first instrument and bar interval to run on. This first Bars object will carry a BarsInProgress index of 0.
2. In a multi-time frame and multi-instrument NinjaScript, supplementary Bars objects are added via this method in State.Configure state of the OnStateChange() method and given an incremented BarsInProgress index value. See additional information on running multi-bars scripts.
3. The BarsInProgress property can be used to filter updates between different bars series
4. If using OnMarketData(), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same). The market data subscription behavior occurs both in real-time and during TickReplay historical
5. For adding regular Bars types please use AddDataSeries()
6. A **Tick Replay** indicator or strategy **CANNOT** use a **MarketDataType.Ask** or **MarketDataType.Bid** series. Please see Developing for Tick Replay for more information.
7. To access additional Volumetric data points programmtically in your NinjaScript studies, please see the example here.

Syntax

```
AddVolumetric(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, Data.VolumetricDeltaType deltaType, int tickPerLevel)
AddVolumetric(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, Data.VolumetricDeltaType deltaType, int tickPerLevel, bool?
isResetOnNewTradingDay)
AddVolumetric(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, Data.VolumetricDeltaType deltaType, int tickPerLevel, string
tradingHoursName, bool? isResetOnNewTradingDay)
AddVolumetric(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int
baseBarsPeriodTypeValue, Data.VolumetricDeltaType deltaType, int tickPerLevel, int
sizeFilter, string tradingHoursName, bool? isResetOnNewTradingDay) (R17 and higher
only)
```

Warnings:
- This method should **ONLY** be called from the OnStateChange() method during **State.Configure**
- Should your script be the host for other scripts that are creating indicators and series dependent resources in **State.DataLoaded**, please make sure that the host is doing the same **AddVolumetric()** calls as those hosted scripts would. For further reference, please also review the 'Adding additional Bars Objects to NinjaScript' section in Multi-Time Frame & Instruments

- Arguments supplied to **AddVolumetric()** should be hardcoded and **NOT** dependent on run-time variables which cannot be reliably obtained during State.Configure (e.g., Instrument, Bars, or user input). Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided. Trying to load bars dynamically may result in an error similar to: **Unable to load bars series. Your** NinjaScript **may be trying to use an additional data series dynamically in an unsupported manner.**

Parameters

instrumentName	A `string` determining instrument name such as "MSFT"
baseBarsPeriodType	The underlying BarsType used for the Volumetric bars period. Possible values are: • BarsPeriodType.Tick • BarsPeriodType.Volume • BarsPeriodType.Range • BarsPeriodType.Second • BarsPeriodType.Minute • BarsPeriodType.Day • BarsPeriodType.Week • BarsPeriodType.Month • BarsPeriodType.Year
baseBarsPeriodTypeValue	An `int` determining the underlying period interval such as "3" for 3 minute bars
deltaType	The DeltaType used for the Volumetric bars object delta calculations Possible values are: • VolumetricDeltaType.BidAsk • VolumetricDetlaType.UpDownTick
ticksPerLevel	An `int` setting the aggregation of price levels for the Volumetric bar, pass in a 1 to analyze each

	price level individually
sizeFilter	An `int` setting the trade size allowed to count in the delta calculations
tradingHoursName	A `string` determining the trading hours template for the instrument
isResetOnNewTradingDay	A nullable `bool`* determining if the Bars object should Break at EOD *Will accept `true`, `false` or `null` as the input. If `null` is used, the data series will use the settings of the primary data series.

Tip: You can optionally add the exchange name as a suffix to the symbol name. This is only advised if the instrument has multiple possible exchanges that it can trade on and it is configured within the Instruments window. For example: `AddVolumetric("MSFT Arca", BarsPeriodType.Minute, 1, VolumetricDeltaType.BidAsk, 1);`

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
       Name = "Examples Indicator";
    }
    else if (State == State.Configure)
    {
       // Add a 1 minute Order Flow Volumetric Bars object for the
ES 03-18 contract - BarsInProgress index = 1
       AddVolumetric("ES 03-18", BarsPeriodType.Minute, 1,
VolumetricDeltaType.BidAsk, 1);
   }
}

protected override void OnBarUpdate()
{
     // Ignore the primary Bars object and only process the Order
Flow Volumetric object
     if (BarsInProgress == 1)
     {
          // Do something;
     }
}
```

11.6.2.1.7 BarsArray

Definition

An array holding Bars objects that are added via the AddDataSeries() method. BarsArray can be used as input for indicator methods. This property is of primary value when working with multi-time frame or multi-instrument scripts.

Property Value

An array of Bars objects.

Warning: This property should **NOT** be accessed within the OnStateChange() method before the State has reached **State.DataLoaded**

Syntax

```
BarsArray[int index]
```

Examples

```
protected override void OnStateChange()
{

   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";
   }
   else if (State == State.Configure)
   {
      // Add a 5 minute Bars object which is added to the BarArray
      // which will take index 1 since the primary Bars object of
the strategy
      // will be index 0
      AddDataSeries(BarsPeriodType.Minute, 5);
   }
}

protected override void OnBarUpdate()
{
   // Ignore bar update events for the supplementary Bars object
added above
   if (BarsInProgress == 1)
     return;

   // Pass in a Bars object as input for the simple moving average
method
   // Evaluates if the 20 SMA of the primary Bars is greater than
   // the 20 SMA of the secondary Bars added above
   if (SMA(20)[0] > SMA(BarsArray[1], 20)[0])
      EnterLong();
}
```

11.6.2.1.8 BarsInProgress

Definition

An index value of the current Bars object that has called the OnBarUpdate() method. In a multi-bars script, the OnBarUpdate() method is called for each Bars object of a script. This flexibility allows you to separate trading logic from different bar events.

Notes:

1. In a single Bars script this property will always return an index value of 0 representing the primary Bars and instrument the script is running on.
2. See additional information on running multi-bars scripts.

Property Value
An `int` value represents the Bars object that is calling the OnBarUpdate() method.

Syntax
```
BarsInProgress
```

Examples
```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Add a 5 minute Bars object: BarsInProgress index = 1
          AddDataSeries(BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Check which Bars object is calling the OnBarUpdate()
method
     if (BarsInProgress == 0)
     {
          // A value of zero represents the primary Bars which is
the ES 09-14
          // 1 minute chart.
          // Do something within the context of the 1 minute Bars
here
     }
     else if (BarsInProgress == 1)
     {
          // A value of 1 represents the secondary 5 minute bars
added in OnStateChange() State.Configure
          // Do something within the context of the 5 minute Bars
     }
}
```

11.6.2.1.9 BarsPeriods

Definition
Holds an array of BarsPeriod objects synchronized to the number of unique Bars objects held within the parent NinjaScript object. If a NinjaScript object holds two Bars series, then BarsPeriods will hold two BarsPeriod objects.

Property Value
An array of BarsPeriod objects.

Warning: This property should NOT be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**

Syntax

```
BarsPeriods[int barSeriesIndex]
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the original data the
strategy is ran on,
          // set by the UI, takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Print out 5, the value of the secondary bars object
     if (BarsInProgress == 1)
          Print(BarsPeriods[1].Value);
}
```

11.6.2.1.10 CurrentBars

Definition

Holds an array of int values representing the number of the current bar in a Bars object. An int value is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the CurrentBar of all Bars objects in a multi-instrument or multi-time frame script.

Note: In multi series processing, the **CurrentBars** starting value will be -1 until all series have processed the first bar.

Property Value

An array of `int` values.

Warning: This property should **NOT** be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**

Syntax

```
CurrentBars[int barSeriesIndex]
```

Examples

Indicator (BarsRequiredToPlot)

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the script. It will
automatically be assigned
          // a Bars object index of 1 since the primary data the
indicator is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Evaluates to make sure we have at least 20 (default value
of BarsRequiredToPlot)
     // or more bars in both Bars objects before continuing.
     if (CurrentBars[0] < BarsRequiredToPlot || CurrentBars[1] <
BarsRequiredToPlot)
          return;

     // Indicator script logic calculation code...
}
```

Strategy (BarsRequiredToTrade)

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the script. It will
automatically be assigned
          // a Bars object index of 1 since the primary data the
indicator is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Evaluates to make sure we have at least 20 (default value
of BarsRequiredToTrade)
     // or more bars in both Bars objects before continuing.
     if (CurrentBars[0] < BarsRequiredToTrade || CurrentBars[1] <
BarsRequiredToTrade)
          return;

     // Strategy script logic calculation code...
}
```

11.6.2.2 Alert, Debug, Share

The following section documents properties and methods used to trigger alerts from a NinjaScript object, send debug messages to the NinjaScript **Output Window**, or utilize **Share Services** to send emails or post to social-media networks.

Alert()	Generates a visual/audible alert for the Alerts Log window
ClearOutputWindow()	Clears all data from the NinjaTrader Output Window
Log()	Generates a NinjaScript category log event record which is output to the Log tab of the NinjaTrader Control Center / Account Data windows
PlaySound()	Plays a .wav file while running on real-time data

Print()	Converts object data to a string format and appends the specified value as text to the NinjaScript Output window
PrintTo	Determines either tab of NinjaScript Output window the Print() and ClearOutputWindow() method targets
RearmAlert()	Rearms an alert created via the Alert() method
SendMail()	Sends an email message through the default email sharing service.
Share()	Sends a message or screen shot to a social network or Share Service.

11.6.2.2.1 Alert()

Definition

Generates a visual/audible alert to display in the Alerts Log window.

Notes:
1. This method can only be called once the State has reached **State.Realtime**. Calls to this method in any other **State** will be silently ignored.
2. For add-ons, please see the AlertCallback() method

Method Return Value

This method does not return a value

Syntax

```
Alert(string id, Priority priority, string message, string soundLocation, int rearmSeconds, Brush backBrush, Brush foreColor)
```

Parameters

id	A `string` representing a unique id for the alert
priority	Sets the precedence of the alert in relation to other alerts Possible values include: Priority.High

	Priority.Low Priority.Medium
message	A `string` representing the Alert message
soundLocation	A `string` representing the absolute file path of the .wav file to play
rearmSeconds	An `int` which sets the number of seconds an alert rearms. **Note**: If the same alert (identified by the id parameter) is called within a time window of the time of last alert + rearmSeconds, the alert will be ignored
backBrush	Sets the background color of the Alerts window row for this alert when triggered (reference)
foreBrush	Sets the foreground color of the Alerts window row for this alert when triggered (reference)

Tip: You can obtain the default NinjaTrader installation directory to access the sounds folder by using `NinjaTrader.Core.Globals.InstallDir` property. Please see the example below for usage.

Example

```
protected override void OnBarUpdate()
{
   // Generate an alert when the RSI value is greater or equal to
20
   if(RSI(14, 3)[0] >= 20)
      Alert("myAlert", Priority.High, "Reached threshold",
NinjaTrader.Core.Globals.InstallDir+@"\sounds\Alert1.wav", 10,
Brushes.Black, Brushes.Yellow);
}
```

11.6.2.2.2 ClearOutputWindow ()

Definition

Clears all data from the NinjaTrader Output Window.

Note: The **ClearOutputWindow()** method only targets the Output tab most recently determined by set PrintTo property.

Method Return Value
This method does not return a value.

Syntax
```
ClearOutputWindow()
```

Examples
```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";
      Description = @"An indicator used to demonstrate various
NinjaScript methods and properties";
   }
   else if (State == State.Configure)
   {
      AddDataSeries(BarsPeriodType.Minute, 5);
   }

   else if(State == State.DataLoaded)
   {
      //clear the output window as soon as the bars data is loaded
      ClearOutputWindow();
   }
}
```

11.6.2.2.3 Log()

Definition
Generates a NinjaScript category log event record and associated time stamp which is output to the Log tab of the NinjaTrader Control Center / Account Data windows. The **Log()** method also writes records to the NinjaTrader log file which can be useful for supporting 3rd party code.

Notes:
1. Log events do **NOT** process to the NinjaScript output window. For temporary logging, please see the Print() method and Output window.

2. The Log event time stamp represents the user configured Time zone from the **Tools** > **Options** > **General** category. This setting could be different from the computer system's time zone.

Method Return Value

This method does not return a value.

Syntax

```
Log(string message, LogLevel logLevel)
```

Warning: Each call to this method creates a log entry which takes memory to keep loaded in the Log tab of the Control Center. Excessive logging can result in huge portions of memory being allocated to display the log messages. Please see the NinjaScript section of the Performance Tips article for more information.

Parameters

message	A `string` value representing the message to be logged
logLevel	Sets the message level for the log event. Different levels are color coded in the NinjaTrader log. • LogLevel.Alert (also generates a pop-up notification window with log message) • LogLevel.Error • LogLevel.Information • LogLevel.Warning

Examples

```
// Generates a log message
Log("This is a log message", LogLevel.Information);

// Generates a log message with a notification window
Log("This will generate a pop-up notification window as well",
LogLevel.Alert);
```

11.6.2.2.4 PlaySound()

Definition

Plays a .wav file while running on real-time data.

Notes:
1. This method will only execute once the State has reached **State.Realtime**. Calls to this method during State.Historical will be ignored (in contrast to the implementation for AddOns)
2. The default behavior is to play the .wav file in an asynchronous manner, which can result in calls to **PlaySound()** to play over one another. Sound files can optionally be configured to execute in a synchronous manner by enabling the **Tools** > **Options** > **Sounds** > **"Play consecutively"** property

Method Return Value

This method does not return a value.

Syntax

```
PlaySound(string fileName)
```

Warning: The underlying framework used to play the sound requires the audio file to be in PCM .wav format. Using another file format such as will generate an error at runtime.

Parameters

fileName	The absolute file path of the .wav file to play

Tip: You can obtain the default NinjaTrader installation directory to access the sounds folder by using `NinjaTrader.Core.Globals.InstallDir` property. Please see the example below for usage.

Examples

```
// Plays the wav file mySound.wav
PlaySound(@"C:\mySound.wav");

// Plays the default Alert1 sound that comes packaged with
NinjaTrader
PlaySound(NinjaTrader.Core.Globals.InstallDir +
@"\sounds\Alert1.wav");
```

11.6.2.2.5 Print()

Definition

Converts object data to a string format and appends the specified value as text to the NinjaScript Output window. Printing data to the NinjaScript Output window is a useful debugging technique to verify values while developing your custom NinjaScript object.

Notes: The **Print()** method only targets the **Output tab** recently specified by set PrintTo property.

Method Return Value

This method does not return a value.

Syntax

`Print(object value)`

Warning: High frequency of Print() method calls can represent a performance hit on your PC. Please see the NinjaScript section of the Performance Tips article for more information.

Parameters

value	The `object` to print to the output window

Tips:

1. You can format prices aligned for easier debugging by using the ToString() method. E.g., Low[0].ToString("0.00") forces the format from 12.5 to 12.50. Low[0].ToString("0.000") forces 12.500.

2. You can format one or more objects in a specified string with the text equivalent of a corresponding object's value for better maintainability using the .NET string.Format() method. Please see the examples below.

Examples

Passing objects directly to Print() method

```
protected override void OnBarUpdate()
{
   // Generates a message
   Print("This is a message");
      //Output:  This is a message

   Print("The high of the current bar is : " + High[0]);
         //Output:  The high of the current bar is : 2112.75

   // Prints the current bar SMA value to the output window
   Print(SMA(Close, 20)[0]);

      //Output: 2110.5;
}
```

Passing string.Format() directly to Print() method

```
protected override void OnBarUpdate()
{
   //Format and Print each bar value to the output window
   Print(string.Format("{0};{1};{2};{3};{4};{5}", Time[0], Open[0],
 High[0], Low[0], Close[0], Volume[0]));
      //Output:  2/24/2015 11:01:00
AM;2110.5;2110.5;2109.75;2110;1702
}
```

Storing and reusing variables in Print() method

```
protected override void OnBarUpdate()
{
   //store the Close[0] value in a variable which can be printed
later*
   double myValue = Close[0];

   //create and store a custom error message
   string myError = string.Format("Error on Bar {0}, value {1} was
not expected", CurrentBar, myValue);

   //*Storing the value adds better reusability of the error
message above for other objects
   //For example later down on line #19 we replace myValue =
Close[0] with another double value Low[0]
   //This allows you to reuse the custom error formatted above on
line #7 without repeating yourself

   //our first test case, if true print our error
   if(myValue > High[0])
      Print(myError);
      //Output: Error on Bar 233, value 1588.25 was not expected

   //reassign myValue
   myValue = Low[0];

   //our second test case (now uses Low[0]), if true print our
error
   if(myValue > Close[0])
      Print(myError);
      //Output: Error on Bar 57, value 1585.5 was not expected
}
```

11.6.2.2.6 PrintTo

Definition

Determines either tab of the NinjaScript Output window the Print() and ClearOutputWindow() method targets

Property Value

An enum value representing the target **Output Tab**. The default value is **PrintTo.OutputTab1**.

Possible values are:

PrintTo.OutputTab1	Output Windows tab named "Output 1"

PrintTo.OutputTab2	Output Windows tab named "Output 2"

Syntax

PrintTo

Examples

Setting the default PrintTo in separate scripts (#1)

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Sample PrintTo Indicator #1";
      Description = @"Used to Print updates to Output 1";

      //Set this scripts Print() calls to the first output tab
      PrintTo = PrintTo.OutputTab1;
   }
}

protected override void OnBarUpdate()
{
   Print("This script will print messages to Output Tab 1");
}
```

Setting the default PrintTo in separate scripts (#2)

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Sample PrintTo Indicator #2";
      Description = "@Used to Print updates to Output 2";

      //Set this scripts Print() calls to the second output tab
      PrintTo = PrintTo.OutputTab2;
   }
}

protected override void OnBarUpdate()
{
   Print("This script will print messages to Output Tab 2");
}
```

Setting PrintTo conditionally in a single script

```
protected override void OnMarketData(MarketDataEventArgs
marketDataUpdate)
{
   if(marketDataUpdate.MarketDataType == MarketDataType.Ask)
   {
      //Print Ask updates to Output 1
      PrintTo = PrintTo.OutputTab1;
      Print("Ask: " + marketDataUpdate.Price);
   }

   else if (marketDataUpdate.MarketDataType == MarketDataType.Bid)
   {
      //Print Bid updates to Output 2
      PrintTo = PrintTo.OutputTab2;
      Print("Bid: " + marketDataUpdate.Price);
   }
}
```

11.6.2.2.7 RearmAlert()

Definition

Rearms an alert created via the Alert() method.

Note: A NinjaScript generated alert by may need to be rearmed after the alert is triggered depending on the **Alert()** methods rearmSeconds parameter.

Method Return Value

This method does not return a value.

Syntax

`RearmAlert(string id)`

Parameters

id	A unique `string` id representing an alert id to rearm

Examples

```
protected override void OnBarUpdate()
{
   //rearms "myAlert" on each new trading session
   if(Bars.IsFirstBarOfSession)
      RearmAlert("myAlert");
}
```

11.6.2.2.8 SendMail()

Definition

Sends an email message through the default email sharing service.

Notes:

1. This method can only be called once the State has reached **State.Realtime**. Calls to this method in any other **State** will be silently ignored (in contrast to the implementation for AddOns)
2. You **MUST** configure an email account as a default **"Mail" Share Service** from the General Options

Method Return Value

This method does not return a value.

Syntax

```
SendMail(string to, string subject, string text)
```

Warning: If mail is not received, please check the Log tab of the control center for any specific errors which could be related to delivering the message.

Parameters

to	The email recipient
subject	Subject line of email
text	Message body of email

Examples

```
// Generates an email message
SendMail("customer@winners.com", "Trade Alert", "Buy ES");
```

11.6.2.2.9 Share()

Definition

Sends a message or screen shot to a social network or Share Service.

Notes:
1. This method can only be called once the State has reached **State.Realtime**. Calls to this method in any other **State** will be silently ignored.
2. You **MUST** configure an account with a **Share Service** provider from the General Options

Method Return Value

This method does not return a value.

Syntax

```
Share(string serviceName, string message)
Share(string serviceName, string message, object[] args)
Share(string serviceName, string message, string screenshotPath)
Share(string serviceName, string message, string screenshotPath, object[] args)
```

Parameters

serviceName	A `string` value representing the share service to be used
message	A `string` value representing the text body sent to the social network or other Share providers. **Note**: The message is what appears in the text box of the Share window
screenshotPath	Optional `string` path to screenshot or other images sent to the social network or other Share providers
args	A generic `object[]` array used to designate additional information sent to the share service

Tips:
1. The "args" parameter differs for each share service used. If you are using a custom developed share adapter, you need to work with the developer of that adapter to understand what the "args" parameter represents for that adapter.
2. For the default NinjaTrader share adapters, the "args" array represents the following:

   - **Mail** share service:
     - args[0] = A string representing the email "*To*" field,
     - args[1] = A string representing the email "*Subject*" field

   - **StockTwits** share service:
     - args[0] = An enum representing the "*StockTwitsSentiment*" parameter

Examples

```
// using "args" as the Mail "To" and "Subject" parameters
Share("Gmail", "Test Message", new object[]{ "example@test.com",
"Test Subject Line" });
```

11.6.2.3 Analytical

NinjaScript provides a number of methods and properties useful for analyzing and identifying specific conditions within Series<T> collections. Some of these methods test a condition and return **true** or **false**, while others return an **int**-based bar index or other numerical value. A list of analytical methods can be found below:

Methods and Properties

CountIf()	Counts the number of occurrences of the test condition
CrossAbove()	Evaluates a cross above condition
CrossBelow()	Evaluates a cross below condition
GetCurrentAsk()	Returns the current Ask price
GetCurrentAskVolume()	Returns the current Ask volume

GetCurrentBid()	Returns the current Bid price
GetCurrentBidVolume()	Returns the current Bid volume
GetMedian()	Returns the median value of the specified series
HighestBar()	Returns the number of bars ago the highest price value occurred
IsFalling()	Evaluates a falling condition
IsRising()	Evaluates a rising condition
Least Recent Occurrence (LRO)	Returns the number of bars ago that the least recent occurrence of a test condition evaluated to true
LowestBar()	Returns the number of bars ago the lowest price value occurred
Most Recent Occurrence (MRO)	Returns the number of bars ago that the most recent occurrence of a test condition evaluated to true
Slope()	Returns a measurement of the steepness of a price series measured by the change over time
TickSize	The value of 1 tick for the corresponding instrument
ToDay()	Calculates an integer value representing a date
ToTime()	Calculates an integer value representing a time

11.6.2.3.1 ApproxCompare()

Definition

Compares two double or float values for equality or being greater than / less than the compared to value.

> **Note**: Comparing for being greater than / less is done using an epsilon value of `1E19`

Method Return Value
An `int` value representing the outcome of the comparison. Returns 0 if values are equal, 1 if value1 is greater than value2. -1 if value1 is less than value2.

Syntax
```
this.ApproxCompare(this double double1, double double2)
this.ApproxCompare(this float float1, double float2)
```

Parameters
double1 / float1	First value to compare against (not actually passed in)
double2 / float2	Second passed in value to compare against

Tip: Main use would be using it for equality comparisons to circumvent running into floating point considerations, value compares for < or > could be usually done more straightforward directly.

Examples
```
// Build the High / Low difference and if 0 sets the indicator
main Value series to 0
if ((High[0] - Low[0]).ApproxCompare(0) == 0)
    Value[0] = 0;
```

11.6.2.3.2 CountIf()

Definition
Counts the number of instances the test condition occurs over the look-back period expressed in bars.

Note: This method does **NOT** work on multi-series strategies and indicators.

Method Return Value
An `int` value representing the number of occurrences found

Syntax
```
CountIf(Func<bool> condition, int period)
```

Parameters

condition	A true/false expression
period	Number of bars to check for the test condition

Tip: The syntax for the "condition" parameter uses lambda expression syntax

Examples

```
// If in the last 10 bars we have had 8 up bars then go long
if (CountIf(() => Close[0] > Open[0], 10) > 8)
     EnterLong();
```

11.6.2.3.3 CrossAbove()

Definition

Evaluates a cross above condition over the specified bar look-back period.

Note: This method does not return `true` if both series being compared have equal values on the current or previous bar with a lookbackPeriod of 1.

Method Return Value

This method returns `true` if a cross above condition occurred; otherwise, `false`.

Syntax

```
CrossAbove(ISeries<double> series1, ISeries<double> series2, int lookBackPeriod)
CrossAbove(ISeries<double> series1, double value, int lookBackPeriod)
```

Parameters

lookBackPeriod	Number of bars back to check the cross above condition
series1 & series2	Any `Series<double>` type object such as an indicator, Close, High, Low, etc...
value	Any `double` value

Examples

```
protected override void OnBarUpdate()
{
   // Go short if CCI crossed above 250 within the last bar
   if (CrossAbove(CCI(14), 250, 1))
        EnterShort();

   // Go long if 10 EMA crosses above 20 EMA within the last bar
   if (CrossAbove(EMA(10), EMA(20), 1))
        EnterLong();

   // Go long we have an up bar and the 10 EMA crosses above 20 EMA
within the last 5 bars
   if (Close[0] > Open[0] && CrossAbove(EMA(10), EMA(20), 5))
        EnterLong();
}
```

11.6.2.3.4 CrossBelow()

Definition

Evaluates a cross below condition over the specified bar look-back period.

> **Note**: This method does not return `true` if both series being compared have equal values on the current or previous bar with a lookbackPeriod of 1.

Method Return Value

This method returns `true` if a cross below condition occurred; otherwise, `false`.

Syntax

```
CrossBelow(ISeries<double> series1, ISeries<double> series2, int lookBackPeriod)
CrossBelow(ISeries<double> series1, double value, int lookBackPeriod)
```

Parameters

lookBackPeriod	Number of bars back to check the cross below condition
series1 & series2	Any `Series<double>` type object such as an indicator, Close, High, Low, etc...
value	Any `double` value

Examples

```
protected override void OnBarUpdate()
{
   // Go long if CCI crossed below -250 within the last bar
   if (CrossBelow(CCI(14), -250, 1))
        EnterLong();

   // Go short if 10 EMA crosses below 20 EMA within the last bar
   if (CrossBelow(EMA(10), EMA(20), 1))
        EnterShort();

   // Go short we have a down bar and the 10 EMA crosses below 20
EMA within the last 5 bars
   if (Close[0] < Open[0] && CrossBelow(EMA(10), EMA(20), 5))
        EnterShort();
}
```

11.6.2.3.5 GetCurrentAsk()

Definition

Returns the current real-time ask price.

Notes:
1. When accessed during **State.Historical**, the Close price of the evaluated bar is substituted. To access historical Ask prices, please see Developing for Tick Replay.
2. The **GetCurrentAsk()** method runs on the bar series currently updating determined by the BarsInProgress property. For multi-instrument scripts, an additional int "barsSeriesIndex" parameter can be supplied which forces the method to run on an supplementary bar series.

Method Return Value

A `double` value representing the current ask price.

Syntax

`GetCurrentAsk()`
`GetCurrentAsk(int barsSeriesIndex)`

Parameters

barsSeriesIndex	An `int` value determining the bar series the method runs. **Note**: This optional parameter is reserved for multi-instrument scripts

Examples

```
protected override void OnBarUpdate()
{
   // Ensure we do not call GetCurrentAsk() on historical data
   if (State == State.Historical)
      return;

   double currentAsk = GetCurrentAsk();
   Print("The Current Ask price is: " + currentAsk);
   // The Current Ask price is: 1924.75
}
```

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Example's Indicator";
   }
   if (State == State.Configure)
   {
      //Add MSFT as our additional data series
      AddDataSeries("MSFT", BarsPeriodType.Minute, 1);
   }
}

protected override void OnBarUpdate()
{
   // Ensure we do not call GetCurrentBid() on historical data
   if (State == State.Historical)
      return;

   if (BarsInProgress == 0)
   {
      double primaryAsk = GetCurrentAsk(0);
      Print("The Primary Ask price is: " + primaryAsk);
      // The Primary Ask price is: 1924.75
   }

   if (BarsInProgress == 1)
   {
      double msftAsk = GetCurrentAsk(1);
      Print("MSFT's Current Ask price is: " + msftAsk);
      // MSFT's Current Ask is: 43.63
   }
}
```

11.6.2.3.6 GetCurrentAskVolume()

Definition

Returns the current real-time ask volume.

Notes:
1. When accessed during **State.Historical**, the Volume of the evaluated bar series is substituted. To access historical Ask Volumes, please see Developing for Tick Replay.
2. The **GetCurrentAskVolume()** method runs on the bar series currently updating determined by the BarsInProgress property. For multi-instrument scripts, an additional int

"barsSeriesIndex" parameter can be supplied which forces the method to run on an supplementary bar series.

Method Return Value

A `long` value representing the current ask volume.

Syntax

```
GetCurrentAskVolume()
GetCurrentAskVolume(int barsSeriesIndex)
```

Parameters

barsSeriesIndex	An `int` value determining the bar series the method runs. **Note**: This optional parameter is reserved for multi-instrument scripts

Examples

```
protected override void OnBarUpdate()
{
   long currentAskVolume = GetCurrentAskVolume();
   Print("The Current Ask volume is: " + currentAskVolume);
   //The Current Ask volume is: 158
}
```

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";
   }
   if (State == State.Configure)
   {
      //Add MSFT as our additional data series
      AddDataSeries("MSFT", BarsPeriodType.Minute, 1);
   }
}

protected override void OnBarUpdate()
{
   if(BarsInProgress == 0)
   {
      long currentAskVolume = GetCurrentAskVolume(0);
      Print("The Current Ask volume is: " + currentAskVolume);
      //The Current Ask volume is: 346
   }

   if(BarsInProgress == 1)
   {
      long msftAskVolume = GetCurrentAskVolume(1);
      Print("MSFT's Current Ask volume is: " + msftAskVolume);
      //MSFT's Current Ask volume is: 1548
   }
}
```

11.6.2.3.7 GetCurrentBid()

Definition

Returns the current real-time bid price.

Notes:
1. When accessed during **State.Historical**, the Close price of the evaluated bar is substituted. To access historical bid prices, please see Developing for Tick Replay.
2. The **GetCurrentBid()** method runs on the bar series currently updating determined by the BarsInProgress property. For multi-instrument scripts, an additional int "barsSeriesIndex" parameter can be supplied which forces the method to run on an supplementary bar series.

Method Return Value

A `double` value representing the current bid price.

Syntax

```
GetCurrentBid()
GetCurrentBid(int barsSeriesIndex)
```

Parameters

barsSeriesIndex	An `int` value determining the bar series the method runs. **Note**: This optional parameter is reserved for multi-instrument scripts

Examples

```
protected override void OnBarUpdate()
{
   // Ensure we do not call GetCurrentBid() on historical data
   if (State == State.Historical)
      return;

   double currentBid = GetCurrentBid();
   Print("The Current Bid price is: " + currentBid);
   // The Current Bid price is: 1924.75
}
```

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Example's Indicator";
   }
   if (State == State.Configure)
   {
      //Add MSFT as our additional data series
      AddDataSeries("MSFT", BarsPeriodType.Minute, 1);
   }
}

protected override void OnBarUpdate()
{
   // Ensure we do not call GetCurrentBid() on historical data
   if (State == State.Historical)
      return;

   if (BarsInProgress == 0)
   {
      double primaryBid = GetCurrentBid(0);
      Print("The Primary Bid price is: " + primaryBid);
      // The Primary Bid price is: 1924.75
   }

   if (BarsInProgress == 1)
   {
      double msftBid = GetCurrentBid(1);
      Print("MSFT's Current Bid price is: " + msftBid);
      // MSFT's Current Bid is: 43.63
   }
}
```

11.6.2.3.8 GetCurrentBidVolume()

Definition

Returns the current real-time bid volume.

Notes:
1. When accessed during **State.Historical**, the Volume of the evaluated bar series is substituted. To access historical Bid Volumes, please see Developing for Tick Replay.
2. The **GetCurrentBidVolume()** method runs on the bar series currently updating determined by the BarsInProgress property. For multi-instrument scripts, an additional int

"barsSeriesIndex" parameter can be supplied which forces the method to run on an supplementary bar series.

Method Return Value

A `long` value representing the current bid volume.

Syntax

```
GetCurrentBidVolume()
GetCurrentBidVolume(int barsSeriesIndex)
```

Parameters

barsSeriesIndex	An `int` value determining the bar series the method runs. **Note**: This optional parameter is reserved for multi-instrument scripts

Examples

```
protected override void OnBarUpdate()
{
   long currentBidVolume = GetCurrentBidVolume();
   Print("The Current Bid volume is: " + currentBidVolume);
   //The Current Bid volume is: 158
}
```

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";
   }
   if (State == State.Configure)
   {
      //Add MSFT as our additional data series
      AddDataSeries("MSFT", BarsPeriodType.Minute, 1);
   }
}

protected override void OnBarUpdate()
{
   if(BarsInProgress == 0)
   {
      long currentBidVolume = GetCurrentBidVolume(0);
      Print("The Current Bid volume is: " + currentBidVolume);
      //The Current Bid volume is: 346
   }

   if(BarsInProgress == 1)
   {
      long msftBidVolume = GetCurrentBidVolume(1);
      Print("MSFT's Current Bid volume is: " + msftBidVolume);
      //MSFT's Current Bid volume is: 1548
   }
}
```

11.6.2.3.9 GetMedian()

Definition

Returns the statistical median value of the specified series over the specified look-back period. This method sorts the values of the specified look back period in ascending order and return the middle value.

Notes:
1. This method should **NOT** be confused with Median prices defined as (High + Low) / 2. This method returns the statistical median of a series.
2. If an even number is passed as the look-back period, the average of the two middle values in the sorted values will be returned.

Method Return Value

A `double` value representing the median value of the series.

Syntax

```
GetMedian(ISeries<double> series, int lookBackPeriod)
```

Parameters

lookBackPeriod	Number of bars back to include in the calculation
series	Any `Series<double>` type object such as an indicator, Close, High, Low, etc...

Examples

```
protected override void OnBarUpdate()
{
   // Print the median price of the last 10 open prices
   //(current open price + look back period's 9 open prices before
that)
   double openMedian = GetMedian(Open, 9);
   Print("The median of the last 10 open prices is: " +
openMedian);
}
```

11.6.2.3.10 HighestBar()

Definition

Returns the number of bars ago the highest price value occurred within the specified look-back period.

Method Return Value

An `int` value representing a value of bars ago.

Syntax

```
HighestBar(ISeries<double> series, int period)
```

Parameters

period	The number of bars to include in the calculation
series	Any `Series<double>` type object such as an indicator, Close, High, Low, etc...

Examples

```
protected override void OnBarUpdate()
{
   // store the highest bars ago value
   int highestBarsAgo = HighestBar(High,
Bars.BarsSinceNewTradingDay);

   //evaluate high price from highest bars ago value
   double highestPrice = High[highestBarsAgo];

   //Printed result:  Highest price of the session: 2095.5 -
occurred 24 bars ago
   Print(string.Format("Highest price of the session: {0} -
occurred {1} bars ago", highestPrice, highestBarsAgo));
}
```

11.6.2.3.11 IsFalling()

Definition

Evaluates a falling condition which is true when the current value is less than the value of 1 bar ago.

Method Return Value

This method returns `true` if a falling condition is present; otherwise, `false`.

Syntax

`IsFalling(ISeries<double> series)`

Parameters

series	Any `Series<double>` type object such as an indicator, Close, High, Low, etc...

Examples

```
protected override void OnBarUpdate()
{
   // If the 20 period SMA is falling (in downtrend) go short
   if (IsFalling(SMA(20)))
        EnterShort();
}
```

11.6.2.3.12 IsRising()

Definition
Evaluates a rising condition which is true when the current value is greater than the value of 1 bar ago.

Method Return Value
This method returns `true` if a rising condition is present; otherwise, `false`.

Syntax
```
IsRising(ISeries<double> series)
```

Parameters

series	Any `Series<double>` type object such as an indicator, Close, High, Low, etc...

Examples

```
protected override void OnBarUpdate()
{
   // If the 20 period SMA is rising (in uptrend) go long
   if (IsRising(SMA(20)))
        EnterLong();
}
```

11.6.2.3.13 Least Recent Occurrence (LRO)

Definition
Returns the number of bars ago that the test condition evaluated to true within the specified look back period expressed in bars. The **LRO()** method start from the furthest bar away and works toward the current bar.

> **Note**: This method does **NOT** work on multi-series strategies and indicators.

Method Return Value
An `int` value representing the number of bars ago. Returns a value of -1 if the specified test condition did not evaluate to true within the look-back period.

Syntax
```
LRO(Func<bool> condition, int instance, int lookBackPeriod)
```

Warnings:
1. The "instance" parameter **MUST** be greater than 0.
2. The "lookBackPeriod" parameter **MUST** be greater than 0.
3. Please check the Log tab for any other exceptions that may be thrown by the condition function parameter.

Parameters

condition	A true/false expression
instance	The occurrence to check for (1 is the least recent, 2 is the 2nd least recent, etc...)
lookBackPeriod	The number of bars to look back to check for the test condition. The test evaluates on the current bar and the bars within the look-back period.

Tip: The syntax for the "condition" parameter uses lambda expression syntax

Examples

```
protected override void OnBarUpdate()
{
   // Prints the high price of the least recent up bar over the
last 10 bars (current bar + look back period's 9 bars before that)
   int barsAgo = LRO(() => Close[0] > Open[0], 1, 9);
   if (barsAgo > -1)
        Print("The bar high was " + High[barsAgo]);
}
```

See Also

Most Recent Occurrence(MRO)

11.6.2.3.14 LowestBar()

Definition

Returns the number of bars ago the lowest price value occurred within the specified look-back period.

Method Return Value
An `int` value representing a value of bars ago.

Syntax
```
LowestBar(ISeries<double> series, int period)
```

Parameters

period	The number of bars to check for the test condition
series	Any `Series<double>` type object such as an indicator, Close, High, Low, etc...

Examples

```
protected override void OnBarUpdate()
{
   // store the lowest bar ago value
   int lowestBar = LowestBar(Low, Bars.BarsSinceNewTradingDay);

   //evaluate low price from lowest bar ago value
   double lowestPrice = Low[lowestBar];

   //Printed result:  Lowest price of the session: 2087.25 -
occurred 362 bars ago
   Print(string.Format("Lowest price of the session: {0} - occurred
{1} bars ago", lowestPrice, lowestBar));
}
```

11.6.2.3.15 Most Recent Occurrence (MRO)

Definition
Returns the number of bars ago that the test condition evaluated to true within the specified look back period expressed in bars. The **MRO()** method starts from the current bar works away (backward) from it.

> **Note**: This method does **NOT** work on multi-series strategies and indicators.

Method Return Value

An `int` value representing the number of bars ago. Returns a value of -1 if the specified test condition did not evaluate to true within the look-back period.

Syntax

```
MRO(Func<bool> condition, int instance, int lookBackPeriod)
```

Warnings:
1. The "instance" parameter **MUST** be greater than 0.
2. The "lookBackPeriod" parameter **MUST** be greater than 0.
3. Please check the Log tab for any other exceptions that may be thrown by the condition function parameter.

Parameters

condition	A true/false expression
instance	The occurrence to check for (1 is the most recent, 2 is the 2nd most recent, etc...)
lookBackPeriod	The number of bars to look back to check for the test condition. The test evaluates on the current bar and the bars within the look-back period.

Tip: The syntax for the "condition" parameter uses lambda expression syntax

Examples

```
protected override void OnBarUpdate()
{
   // Prints the high price of the most recent up bar over the last
10 bars (current bar + look back period's 9 bars before that)
   int barsAgo = MRO(() => Close[0] > Open[0], 1, 9);
   if (barsAgo > -1)
        Print("The bar high was " + High[barsAgo]);
}
```

See Also

Least Recent Occurrence(LRO)

11.6.2.3.16 Slope()

Definition

Returns a measurement of the steepness of a price series (y value) measured by the change over time (x value). The return value can also be thought of as the ratio between the startBarsAgo and endBarsAgo parameters passed to the method.

The formula which is returned from the parameters passed is:

(series[endBarsAgo] - series[startBarsAgo]) / (startBarsAgo - endBarsAgo)

> **Note**: The return value should **NOT** be confused with the angle (or radians) of a line that displays on the chart.

Method Return Value

This method returns a `double` value indicating the slope of a line; A value of 0 returns if the either the startBars or endBars parameters are less than 0 or both parameters are of equal value.

Syntax

```
Slope(ISeries<double> series, int startBarsAgo, int endBarsAgo)
```

> **Warning**: The "startBarsAgo" parameter **MUST** be greater than the "endBarsAgo" parameter

Parameters

series	Any Series<double> type object such as an indicator, Close, High, Low, etc...
startBarsAgo	The starting point of a series to be evaluated
endBarsAgo	The ending point of a series to be evaluated

Tip: Thinking in degrees, for example a 1 to -1 return range would translate to 45 to -45. To convert you could look into working with this formula - Math.Atan(Slope) * 180 / Math.PI

Examples

```
protected override void OnBarUpdate()
{
   // Prints the slope of the 20 period simple moving average of
the last 10 bars
   Print(Slope(SMA(20), 10, 0));

}
```

11.6.2.3.17 TickSize

Definition

The minimum fluctuation value which is always a value of 1-tick for the corresponding master instrument.

Property Value

A `double` value that represents the minimum fluctuation of an instrument.

Syntax

`TickSize`

Warning: This property should **NOT** be accessed during **State.SetDefaults** from within the OnStateChange() method, all bars series would be guaranteed to have loaded in **State.DataLoaded**

Examples

```
// Prints the ticksize to the output window
Print("The ticksize of this instrument is " + TickSize);

// Prints the value of the current bar low less one tick size
double value = Low[0] - TickSize;
Print(value);
```

11.6.2.3.18 ToDay()

Definition
Calculates an integer value representing a date.

> **Note**: Integer representation of day is format as yyyyMMdd where January 8, 2015 would be 20150108.

Method Return Value
An `int` value representing date structure

Syntax
`ToDay(DateTime time)`

Parameters

time	A `DateTime` structure to calculate **Note**: See also the Time property

> **Tip**: NinjaScript uses the .NET DateTime structures which can be complicated for novice programmers. If you are familiar with C# you can directly use DateTime structure properties and methods for date and time comparisons otherwise use this method and the ToTime() method.

Examples

```
protected override void OnBarUpdate()
{
   // Compare the date of the current bar to September 15, 2014
   if (ToDay(Time[0]) > 20140915)
   {
        // Do something
   }
}
```

11.6.2.3.19 ToTime()

Definition
Calculates an integer value representing a time.

Note: Integer representation of time is in the format Hmmss where 7:30 AM would be 73000 and 2:15:12 PM would be 141512.

Method Return Value

An `int` value representing a time structure

Syntax

```
ToTime(DateTime time)
ToTime(int hour, int minute, int second)
```

Parameters

time	A `DateTime` structure to calculate **Note**: See also the Time property
hour	An `int` value representing the hour used for the input
minute	An `int` value representing the minute used for the input
second	An `int` value representing the second used for the input

Tip: NinjaScript uses the .NET DateTime structure which can be complicated for novice programmers. If you are familiar with C# you can directly use DateTime structure properties and methods for date and time comparisons otherwise use this method and the ToDay() method.

Examples

```
// Only trade between 7:45 AM and 1:45 PM
if (ToTime(Time[0]) >= 74500 && ToTime(Time[0]) <= 134500)
{
     // Strategy logic goes here
}
```

```
//store start time as an int variable to be compared
int startTime = ToTime(9, 30, 00);  // 93000

//only trade after 9:30AM
if (ToTime(Time[0]) >= startTime)
{
     // Strategy logic goes here
}
```

11.6.2.4 Attributes

The following section documents both .NET native and NinjaScript custom attributes which are commonly used to define the behavior of a NinjaScript property or object. The **attributes** outlined in the section are primarily used to customize how properties display on the UI, but may also control or how the object is compiled and executed at run time.

Notes:
1. The .NET Framework supplies many other pre-defined system attributes which can technically be used for custom NinjaScript programming, but are **NOT** covered in this section and therefore are considered **unsupported**. 3rd party developers are encourage to explore additional usage, but the resulting behavior **CANNOT** be guaranteed.
2. Not all **attributes** can be applied to all object types. For example, applying an **attribute** that is defined to target an class will **NOT** compile should you attempt to apply this **attribute** to a type of property.

Common Attributes

BrowsableAttribute	Determines if a property should be displays in the NinjaTrader UI's property grid
CategoryOrderAttribute	Determines the sequence in which a NinjaScript object's Display.GroupName categories are arranged in relation to other categories in the UI.
DisplayAttribute	Determines how a property is displays on the NinjaTrader UI's property grid.

NinjaScriptPropertyAttribute	Determines if a property should be included in the NinjaScript object's constructor as a parameter
RangeAttribute	Determines if the value of a property is valid within a specified range
XmlIgnoreAttribute	Determines if a property participates in the XML serialization routines (saving workspaces or templates)

Applying Attributes

Attributes are applied directly before the property, method, or class, and are identified by wrapping brackets:

```
[AnExampleAttribute] // a pseudo-attribute demonstrating how to target an object
public object AnExampleProperty  // the property that is being targeted
{ get; set; }
```

Tip: Conventionally, the suffix "attribute" is provided to the object's name to help determine that is an **attribute**, however C# does not require you to specify the full name of an **attribute**. For example **DisplayAttribute()** will compile the same as **Display()**.

11.6.2.4.1 BrowsableAttribute

Definition

Determines if the following declared property displays in the NinjaTrader UI's property grid. By default, all public properties in a NinjaScript object display, however this behavior can be changed by setting the Browsable attribute to false.

Note: The **BrowsableAttribute** object is a general purpose attribute made available from the .NET Framework. The information on this page is written to demonstrate how you may use this object within NinjaScript conventions used with the NinjaTrader UI's property grid

(e.g., an indicator dialog). There are more methods and properties that you can learn about from **MSDN's** BrowsableAttribute Class which are **NOT** covered in this topic; as such there is **NO** guarantee they will work with the NinjaTrader UI's property grids.

Syntax

```
[Browsable(bool)]
```

Parameters

A `bool` which sets a value indicating if a property is browsable; default value is **true**

Examples

```
#region Properties

// do not show this value on the UI's property grid
[Browsable(false)]
public bool MyBool
{ get; set; }

#endregion
```

11.6.2.4.2 CategoryOrderAttribute

Definition

Determines the sequence in which a NinjaScript object's Display.GroupName categories are arranged in relation to other categories in the UI. The default behavior will display each GroupName of an object in alphabetical order, however this behavior can be changed by defining the **CategoryOrder** attribute before the object's declaration.

Notes:

- The **CategoryOrder** attribute is **ONLY** valid on class-level declarations.
- Categories with values less than 1,000,000 appear at the very top of the property grid (excluding the Strategy Analyzer "General" category)
- NinjaTrader UI reserves using values ending in 000, 500 and the values documented below are subject to change
- If you wish to inject your category between a standard NinjaScript category, please refer to the table below to locate the appropriate position (e.g., to set a property after "Data Series" and before the "Setup" use value of 2,000,001)

NinjaScript Indicators

The follow table applies for Indicators configured from a Chart Indicator, Market Analyzer Indicator Column, or SuperDOM Indicator:

Parameters	1000000
Data Series	2000000
Time Frame	3000000
Setup	4000000
Visual	5000000
Lines	6000000
Plots	7000000

NinjaScript Strategies

The following table applies to Chart Strategies, Control Center Strategies Grid, and the Strategy Analyzer

Parameters	1000000
Data Series	2000000
Time Frame	3000000
Setup	4000000
Historical Fill Processing	5000000
Optimize	6000000
Order Handling	7000000
Order Properties	8000000

Note: The Strategy Analyzer "General" category is purposely excluded from this table and will always show on the top of the parameter grid.

Syntax

```
[Gui.CategoryOrder(string category, int order)]
```

Warning: Attempting to modify the default NinjaScript Category ordering is **NOT** supported. Trying to do so may result in unexpected outcomes.

Parameters

category	A `string` identifying the GroupName to be categorize
order	An `int` determining the sequence the Category displays

Examples

```
[Gui.CategoryOrder("My Strings", 1)]  // display "My Strings" first
[Gui.CategoryOrder("My Bools", 2)]  // then "My Bools"
[Gui.CategoryOrder("My Ints", 3)]  // and finally "My Ints"
public class MyCustomIndicator : Indicator
{
   #region Properties

   [Display(GroupName="My Ints")]
   public int MyCustomInt
   { get; set; }

   [Display(GroupName="My Bools")]
   public bool MyCustomBool
   { get; set; }

   [Display(GroupName="My Strings")]
   public string MyCustomString
   { get; set; }

   #endregion
}
```

11.6.2.4.3 DisplayAttribute

Definition

Determines how the following declared property display on the NinjaTrader UI's property grid.

> **Note**: The **DisplayAttribute** object is a general purpose attribute made available from the .NET Framework. The information on this page is written to demonstrate how you may use this object within NinjaScript conventions used with the NinjaTrader UI's property grid (e.g., an indicator dialog). There are more methods and properties that you can learn about from **MSDN's** DisplayAttribute Class which are **NOT** covered in this topic; as such there is **NO** guarantee they will work with the NinjaTrader UI's property grids.

Syntax

```
[Display(Name=string)]
[Display(Description=string)]
[Display(GroupName=string)]
[Display(Order=int)]
```

> **Warning**: The "**Name**" parameter **MUST** be unique for each property of a particular object. Sharing the same **Name** can have undesirable consequences on various features of the property grid.

Parameters

Name	A `string` which sets the text used to display the property on the UI
Description	A `string` which sets the tool tip used to describe the property from the UI **Note**: Expandable properties will **NOT** display a tool tip (e.g., SimpleFont, Stroke, or any custom component which are a type of an ExpandableObjectConverter)
GroupName	A `string` which sets a name that is used to group various properties in the UI. If no

	GroupName is specified, properties will be listed in the generic "Parameters" section.
Order	An `int` which sets the sequence the property is categorized in relation to other properties in the UI.

Tips:
1. Multiple named parameters can be written separated by a comma during a single declaration as demonstrated in the example below.
2. You may have noticed the default NinjaTrader types such as indicators or strategies use a "`ResourceType = typeof(Custom.Resource)`" property in the DisplayAttribute. This is done for localization purposes, so the default NinjaTrader UI translates to other supported international languages, but is not required for your custom NinjaScript types. The ResourceType property can be safely ignored and left out in your custom development.

Examples

```
#region Properties

// set how the property displays from the UI property grid
[Display(Name="My Period", Order=1, GroupName="My Parameters")]
public int MyPeriod
{ get; set; }

#endregion
```

11.6.2.4.4 NinjaScriptPropertyAttribute

Definition

Determines if the following declared property should be included in the NinjaScript object's constructor as a parameter. This is useful if you plan on calling a NinjaScript object from another (e.g., calling a custom indicator from a strategy) or customizing the display parameter data on a grid or from a chart. This also used to make parameters optimizable in the Strategy Analyzer.

Warning: Only types which can be Xml Serialized should be marked as a **NinjaScriptAttribute**, otherwise you may run into errors when persisting values in various scenarios (e.g., saving workspace, or running Strategy Optimizations). Should you have a property you wish to use as user defined input, you will need to implement a secondary simple type (such as an int or string) as the value to be serialized as user input. Please see the example below which demonstrates using a simple type as the **NinjaScriptProperty** against types which cannot be serialized

Syntax

```
[NinjaScriptProperty]
```

Parameters

This object contains no parameters

Examples

Basic usage of NinjaScriptProperty

```
#region Properties

// set NinjaScriptProperty to ensure this property is used when
calling from another object
[NinjaScriptProperty]
public bool MyBool
{ get; set; }

// do not set NinjaScriptProperty since this property is not
required to call
// nor do we wish to display it on the chart label
public int MyInt
{ get; set; }

#endregion
```

Using a simple type as the NinjaScriptProperty against types which cannot be serialized

```
[NinjaScriptProperty]
[XmlIgnore] // cannot serialize type of TimeSpan, use the
BeginTimeSpanSerialize object to persist properties
public TimeSpan BeginTimeSpan
{ get; set; }

// users will configure this "string" as the TimeSpan which will be
set as a TimeSpan object used in data processing
[Browsable(false)] // prevents this property from showing up on the
UI
[Display(Name = "Begin TimeSpan", GroupName =
"NinjaScriptStrategyParameters", Order = 1)]
public string BeginTimeSpanSerialize
{
   get { return BeginTimeSpan.ToString(); }
   set { BeginTimeSpan = TimeSpan.Parse(value); }
}
```

11.6.2.4.5 RangeAttribute

Definition

Determines if the value of the following declared property is valid within a specified range. These values are checked when the NinjaScript object has reached State.Configure. For configuration through the UI (e.g., the user has selected Apply or OK to configure the value from the indicator dialog box) and determines to be invalid, the value will be automatically rounded to the nearest minimum or maximum value. Should the property be set as a NinjaScriptAttribute and called from a hosting NinjaScript object and determines to be invalid, an exception will be thrown and the hosted indicator will NOT execute.

Note: The **RangeAttribute** object is a general purpose attribute made available from the .NET Framework. The information on this page is written to demonstrate how you may use this object within NinjaScript conventions used for the NinjaTrader UI's property grid (e.g., an indicator dialog). There are more methods and properties that you can learn about from **MSDN's** RangeAttribute Class which are **NOT** covered in this topic; as such there is **NO** guarantee they will work with the NinjaTrader UI's property grids.

Syntax

```
[Range(int minimum, int maximum)]
[Range(double minimum, double maximum)]
[Range(type type, string minimum, string aximum)]
```

Parameters

maximum	Defines the highest allowed value the user can set for the property
minimum	Defines the lowest allowed value the user can set for the property
type	The type of object to test

Examples

```
#region Properties

// set range between 1 and the highest possible integer
[Range(1, int.MaxValue)]
public int Myint
{ get; set; }

//set range between .001 and 1
[Range(.001, 1.0)]
public double MyDouble
{ get; set; }

// set range as a type DateTime between these dates
[Range(typeof(DateTime), "01/01/1990", "12/31/2015")]
public DateTime MyTime
{ get; set; }

#endregion
```

11.6.2.4.6 TypeConverterAttribute

Definition

Binds an object or property to a specific **TypeConverter** implementation. This is commonly used to customize property descriptors on the NinjaTrader property grid.

Notes:

- If you are looking to extend behavior of an Indicator or Strategy (e.g., values of one property influence another), you must implement either an IndicatorBaseTypeConverter or StrategyBaseTypeConveter. This is to ensure default property descriptor behavior works as intended.
- For converting types of a specific property, implementing a standard **TypeConverter** is sufficient

- A working NinjaScript demo can be found through the reference sample on "Using a TypeConverter to Customize Property Grid Behavior"
- The **TypeConverterAttribute** object is a general purpose attribute made available from the .NET Framework. The information on this page is written to demonstrate how you may use this object within NinjaScript conventions used with the NinjaTrader UI's property grid (e.g., an indicator dialog). There are more methods and properties that you can learn about from **MSDN's** TypeConverterAttribute Class which are **NOT** covered in this topic; as such there is **NO** guarantee they will work with the NinjaTrader UI's property grids.

Syntax

```
TypeConverterAttribute(string)
TypeConverterAttribute(type)
```

Examples

```
// Only applied to this property: can just implement a general
TypeConveter
[TypeConverter(typeof(MyCustomBoolConveter))]
public bool CustomBool

// Applied to the entire indicator: must implement an
IndicatorBaseTypeConveter
[TypeConverter("NinjaTrader.NinjaScript.Indicators.MyConverter")]
public class MyCustomIndicator : Indicator
{

}

// Applied to the entire strategy: must implement a
StrategyBaseTypeConveter
[TypeConverter("NinjaTrader.NinjaScript.Strategies.MyCustomConveter
")]
public class MyCustomStrategy : Strategy
{

}
```

11.6.2.4.7 XmlIgnoreAttribute

Definition

Determines if the following declared property participates in the XML serialization routines which are used to save NinjaScript objects to a workspace or template. The default behavior will attempt to serialize all public properties, however there may be some types of objects

which cannot be serialized, or you may not wish for this property to be saved/restored. Should that be the case, you can optionally set the object to be ignored by defining the XmlIgnore attribute.

> **Note**: The **XmlIgnoreAttribute** object is a general purpose attribute made available from the .NET Framework. The information on this page is written to demonstrate how you may use this object within NinjaScript conventions to be used for the NinjaTrader serialization (e.g., saving an indicator property to a workspace). There are more methods and properties that you can learn about from **MSDN's** XmlIgnoreAttribute Class which are **NOT** covered in this topic; as such there is **NO** guarantee they will work with the NinjaTrader serialization.

Syntax

```
[XmlIgnore]
[XmlIgnore(bool)]
```

Parameters

This attribute does not require any parameters; default value is **true** and usage will ensure the property is ignored by XML routines.

Examples

```
#region Properties

[XmlIgnore] // ensures that the property will NOT be
saved/recovered as part of a chart template or workspace
public Brush MyBrush
{ get; set; }

#endregion
```

> **Tip**: A complete example of the usage of **XmlIgnore** attribute and workspace serialization can be found in the tips section of our support forum on User Definable Color Inputs

11.6.2.5 Bars

Definition

Represents the data returned from the historical data repository. The Bars object contain several methods and properties for working with bar data.

Warning: The Bars object and its member should **NOT** be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**

Additional Access Information

Members within the Bars class can be accessed without a null reference check in the OnBarUpdate() event handler. When the OnBarUpdate() event is triggered, there will always be a Bar object which holds the method or property. Should you wish to access these members elsewhere, check for null reference first. e.g. if (Bars != null)

Methods and Properties

BarsSinceNewTradingDay	Number of bars that have elapsed since the start of the trading day
GetAsk()	Returns the Ask price
GetBar()	Returns the bar index based on time
GetBid()	Returns the Bid price
GetClose()	Returns the closing price
GetDayBar()	Returns a Bar object that represents a trading day whose properties for open, high, low, close, time and volume can be accessed.
GetHigh()	Returns the High price
GetLow()	Returns the Low price
GetOpen()	Returns the opening price
GetTime()	Returns the time
GetVolume()	Returns the volume
IsFirstBarOfSession	Returns **true** if the bar is the first bar of a session
IsFirstBarOfSessionByIndex()	Returns **true** if the bar is the first bar of a session

IsLastBarOfSession	Returns **true** if the bar is the last bar of a session
IsResetOnNewTradingDay	Returns **true** if the chart bars should reset on a new trading day
IsTickReplay	Returns **true** if the bars are using tick replay
PercentComplete	Value indicating the completion percent of a bar
TickCount	Total number of ticks of the current bar
ToChartString()	Returns the bars series as a string formatted as the series would be displayed in the user interface

11.6.2.5.1 BarsSinceNewTradingDay

Definition

Returns the number of bars elapsed since the start of the trading day relative to the current bar processing.

Property Value

An `int` value representing the number of bars elapsed. This property cannot be set.

Syntax

```
Bars.BarsSinceNewTradingDay
```

Examples

```
// Only process strategy logic after five bars have posted since
the start of the trading day
protected override void OnBarUpdate()
{
   if (Bars.BarsSinceNewTradingDay >= 5)
   {
      //Strategy logic here
   }
}
```

11.6.2.5.2 GetAsk()

Definition

Returns the ask price value at a selected absolute bar index value.

Notes:

- This method does **NOT** return the current real-time asking price, but rather the historical / real-time asking price at the desired index. For obtaining the current real-time asking price, please use GetCurrentAsk().
- This method returns expected values when 1 tick bid / ask stamped data is used and available from your provider.

Method Return Value

A `double` value that represents the asking price at the desired bar index.

Syntax

```
Bars.GetAsk(int index)
```

Parameters

index	The absolute bar index value used

Examples

```
protected override void OnBarUpdate()
{
    // If the Highs of the two most recent bars are falling, place
a long stop market order
    // at the Ask price
    if (High[0] < High[1] && High[1] < High[2])
    {
        EnterLongStopMarket(Bars.GetAsk(CurrentBar));
    }
}
```

11.6.2.5.3 GetBar()

Definition

Returns the first bar that matches the time stamp of the "time" parameter provided.

Note: If the time parameter provided is older than the first bar in the series, a bar index of 0 is returned. If the time stamp is newer than the last bar in the series, the last absolute bar index is returned.

Method Return Value

An `int` value representing an absolute bar index value.

Syntax

```
Bars.GetBar(DateTime time)
```

Parameters

time	Time stamp to be converted to an absolute bar index

Examples

```
// Check that its past 9:45 AM
if (ToTime(Time[0]) >= ToTime(9, 45, 00))
{
   // Calculate the bars ago value for the 9 AM bar for the current
day
   int barsAgo = CurrentBar - Bars.GetBar(new DateTime(2006, 12,
18, 9, 0, 0));

   // Print out the 9 AM bar closing price
   Print("The close price on the 9 AM bar was: " +
Close[barsAgo].ToString());
}
```

11.6.2.5.4 GetBid()

Definition

Returns the bid price value at a selected absolute bar index value.

Notes:

- This method does **NOT** return the current real-time bid price, but rather the historical / real-time bid price at the desired index. For obtaining the current real-time bid price, please use GetCurrentBid().
- This method returns expected values when 1 tick bid / ask stamped data is used and available from your provider.

Method Return Value

A `double` value that represents the biding price at the desired bar index.

Syntax

```
Bars.GetBid(int index)
```

Parameters

index	The absolute bar index value used

Examples

```
protected override void OnBarUpdate()
{
   // If the Highs of the two most recent bars are falling, place a
long stop market order
   // at the Ask price
   if (Low[0] > Low[1] && Low[1] < Low[2])
   {
      EnterShortStopMarket(Bars.GetBid(CurrentBar));
   }
}
```

11.6.2.5.5 GetClose()

Definition

Returns the closing price at the current bar index value.

Method Return Value

A `double` value that represents the close price at the desired bar index.

Syntax

```
Bars.GetClose(int index)
```

Parameters

index	An `int` representing an absolute bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);
   // loop through only the rendered bars on the chart
   for(int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
   {
      // get the close price at the selected bar index value
      double closePrice = Bars.GetClose(barIndex);
      Print("Bar #" + barIndex + " closing price is " +
closePrice);
   }
}
```

11.6.2.5.6 GetDayBar()

Definition

Returns a virtual historical Bar object that represents a trading day whose properties for open, high, low, close, time and volume can be accessed.

Notes:

1. The bar object returned is a "virtual bar" built from the underlying bar series and its configured session. Since the bar object is virtual, its property values are calculated based on session definitions contained in the trading day only. The returned bar object does **NOT** necessarily represent the actual day. For accessing a true "Daily" bar, please see use AddDataSeries() and use the BarsPeriodType.Day as the bars period.
2. GetDayBar() should **ONLY** be used for accessing prior trading day data. To access current trading day data, use the CurrentDayOHL() method.

Method Return Value

A virtual bar object representing the current configured session. Otherwise null if there is insufficient intraday data

Syntax

The properties below return `double` values:

```
Bars.GetDayBar(int tradingDaysBack).Open
Bars.GetDayBar(int tradingDaysBack).High
Bars.GetDayBar(int tradingDaysBack).Low
Bars.GetDayBar(int tradingDaysBack).Close
```

The property below returns a DateTime structure:

```
Bars.GetDayBar(int tradingDaysBack).Time
```

The property below returns an `int` value:
`Bars.GetDayBar(int tradingDaysBack).Volume`

Warning: You must check for a null reference to ensure there is sufficient intraday data to build a trading day bar.

Parameters

`tradingDaysBack`	An `int` representing the number of the trading day to get OHLCV and time information from

Examples

```
protected override void OnBarUpdate()
{
   // Check to ensure that sufficient intraday data was supplied
   if(Bars.GetDayBar(1) != null)
      Print("The prior trading day's close is: " +
Bars.GetDayBar(1).Close);
}
```

11.6.2.5.7 GetHigh()

Definition
Returns the high price at the selected bar index value.

Method Return Value
A `double` value that represents the high price at the desired bar index.

Syntax
`Bars.GetHigh(int index)`

Parameters

index	An `int` representing an absolute bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);
   // loop through only the rendered bars on the chart
   for(int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
   {
      // get the high price at the selected bar index value
      double highPrice = Bars.GetHigh(barIndex);
      Print("Bar #" + barIndex + " high price is " + highPrice);
   }
}
```

11.6.2.5.8 GetLow()

Definition

Returns the low price at the selected bar index value.

Method Return Value

A `double` value that represents the low price at the desired bar index.

Syntax

`Bars.GetLow(int index)`

Parameters

index	An `int` representing an absolute bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);
   // loop through only the rendered bars on the chart
   for(int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
   {
      // get the low price at the selected bar index value
      double lowPrice = Bars.GetLow(barIndex);
      Print("Bar #" + barIndex + " low price is " + lowPrice);
   }
}
```

11.6.2.5.9 GetOpen()

Definition
Returns the open price at the selected bar index value.

Method Return Value
A `double` value that represents the open price at the desired bar index.

Syntax
```
Bars.GetOpen(int index)
```

Parameters

index	An `int` representing an absolute bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);
   // loop through only the rendered bars on the chart
   for(int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
   {
      // get the open price at the selected bar index value
      double openPrice = Bars.GetOpen(barIndex);
      Print("Bar #" + barIndex + " open price is " + openPrice);
   }
}
```

11.6.2.5.10 GetSessionEndTime()

Definition
Returns the daily bar session ending time stamp relative to the current bar index value.

> **Note**: This method is **ONLY** intended for bars built from daily data. If called on intraday data, **GetSessionEndTime()** will return the Bars.GetTime() value.

Method Return Value
A `DateTime` structure that represents the daily bars ending time stamp at the desired bar index; intraday bars will return the time stamp at the current bar index value.

Syntax
```
Bars.GetSessionEndTime(int index)
```

Parameters

index	An `int` representing an absolute bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);
   // loop through only the rendered bars on the chart
   for (int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
   {
      // get the time stamp at the selected bar index value
      DateTime timeValue = Bars.GetSessionEndTime(barIndex);
      Print("Bar #" + barIndex + " time stamp is " + timeValue);
   }
}
```

11.6.2.5.11 GetTime()

Definition
Returns the time stamp at the current bar index value.

Note: This method will return what is displayed in the chart's data box. For formatting purposes, the value returned is **NOT** guaranteed be equal to the TimeSeries value. If you are using daily bars and need the session end time, you should use Bars.GetSessionEndTime() instead.

Method Return Value
A `DateTime` structure that represents the time stamp at the desired bar index.

Syntax
```
Bars.GetTime(int index)
```

Parameters

index	An `int` representing an absolute bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);
   // loop through only the rendered bars on the chart
   for(int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
   {
      // get the time stamp at the selected bar index value
      DateTime timeValue = Bars.GetTime(barIndex);
      Print("Bar #" + barIndex + " time stamp is " + timeValue);
   }
}
```

11.6.2.5.12 GetVolume()

Definition

Returns the volume at the selected bar index value.

Method Return Value

A `long` value represents the volume at the desired bar index.

Syntax

`Bars.GetVolume(int index)`

Parameters

index	An `int` representing an absolute bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);
   // loop through all the rendered bars on the chart
   for(int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
   {
      // get the volume value at the selected bar index value
      long volumeValue = Bars.GetVolume(barIndex);
      Print("Bar #" + barIndex + " volume value is " +
volumeValue);
   }
}
```

11.6.2.5.13 IsFirstBarOfSession

Definition

Indicates if the current bar processing is the first bar updated in a trading session.

Note: This property always returns **true** on the very first bar processed (i.e., CurrentBar == 0). The represented time of the bar will **NOT** necessarily be equal to the trading hours start time (e.g., if you request 50 1-minute bars at 11:50:00 AM, the first bar processed of the session would be 11:00:00 AM). Loading a data series based on "dates" (Days or custom range) ensures that the first bar processed matches hours defined by the session template.

Property Value

This property returns **true** if the bar is the first processed in a session; otherwise, **false**. This property is read-only.

Warning: This property will always return **false** on non-intraday bar periods (e.g., Day, Month, etc). For checking for new non-intraday bar updates, please see IsFirstTickOfBar

Syntax

`Bars.IsFirstBarOfSession`

Tip: For checking at a specified bar index, please see IsFirstBarOfSessionByIndex()

Examples

```
protected override void OnBarUpdate()
{
   // Print the current bar number of the first bar processed for
each session on a chart
   if (Bars.IsFirstBarOfSession)
      Print(string.Format("Bar number {0} was the first bar
processed of the session at {1}.", CurrentBar, Time[0]));
}
```

11.6.2.5.14 IsFirstBarOfSessionByIndex()

Definition

Indicates if the selected bar index value is the first bar of a trading session.

Property Value

This property returns **true** if the bar is the first bar of a session; otherwise, **false**. This property is read-only.

Syntax

`Bars.IsFirstBarOfSessionByIndex(int index)`

> **Warning**: This property will always return **false** on non-intraday bar periods (e.g., Day, Month, etc)

Parameters

index	An `int` representing an absolute bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);
   //  loop through only the rendered bars on the chart
   for(int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
   {
      // check if the rendered bar is the first bar of the trading
session
      if (Bars.IsFirstBarOfSessionByIndex(barIndex))
      {
         DateTime slotTimeAtBarIndex =
chartControl.GetTimeBySlotIndex(barIndex);
         Print(string.Format("Bar index {0} was the first bar of
the session at slot time {1}.", barIndex, slotTimeAtBarIndex));
      }
   }
}
```

11.6.2.5.15 IsLastBarOfSession

Definition

Indicates if the current bar processing is the last bar updated in a trading session.

Notes:

- This property will always return **false** on non-intraday bar periods (e.g., Day, Month, etc.)
- When running **Calculate.OnEachTick** / **OnPriceChange**, this property will always return **true** on the most current real-time bar since it is the last bar that is updating in the trading session. If you need to find a bar which coincides with the session end time, please use the SessionIterator.ActualSessionEnd.

Property Value

This property returns **true** if the bar is the last processed in a session; otherwise, **false**. This property is read-only.

Syntax

`Bars.IsLastBarOfSession`

Examples

```
protected override void OnBarUpdate()
{
   // Print the current bar number of the first bar processed for
each session on a chart
   if(Bars.IsLastBarOfSession)
      Print(string.Format("Bar number {0} was the last bar
processed of the session at {1}.", CurrentBar, Time[0]));
}
```

11.6.2.5.16 IsResetOnNew TradingDay

Definition

Indicates if the bars series is using the Break EOD data series property.

Property Value

This property returns **true** if the bars series should reset on a new trading day; otherwise, **false**. This property is read-only.

Syntax

`Bars.IsResetOnNewTradingDay`

> **Tip**: This property can be helpful in determine on how to amend new bar data when working with a BarType

Examples

```
protected override void OnDataPoint(Bars bars, double open, double
high, double low, double close, DateTime time, long volume, bool
isBar, double bid, double ask)
{
   // create a session iterator to keep track of session related
information
   if(SessionIterator == null)
      SessionIterator = new SessionIterator(bars);

   // determine if the bars are in a new session
   bool isNewSession = SessionIterator.IsNewSession(time, isBar);

   if(isNewSession)
      SessionIterator.GetNextSession(time, isBar);

   // If bars are using "Break end of day", add a new bar for next
session
   if(bars.IsResetOnNewTradingDay && isNewSession))
      AddBar(bars, open, high, low, close, time, volume);
   else
   {
      // do something with existing bar values
   }
}
```

11.6.2.5.17 IsTickReplay

Definition

Indicates if the bar series is using the Tick Replay data series property.

Property Value

This property returns **true** if the bar series is using tick replay; otherwise, **false**. This property is read-only.

Syntax

```
Bars.IsTickReplay
```

Warning: A **Tick Replay** indicator or strategy **CANNOT** use a **MarketDataType.Ask** or **MarketDataType.Bid** series. Please see Developing for Tick Replay for more information.

Examples

```
private double askPrice;
protected override void OnMarketData(MarketDataEventArgs
marketDataUpdate)
{
   if(Bars.IsTickReplay)
   {
      // if using tick replay, get the current ask price associated
with the tick
      askPrice = marketDataUpdate.Ask;
   }
   else  // otherwise, get the real-time market data price during
MarketDataType.Ask event
      askPrice = marketDataUpdate.MarketDataType ==
MarketDataType.Ask ? marketDataUpdate.Price : double.MinValue;

   // only print if a value is set
   if(askPrice != double.MinValue)
   {
      Print("ask price: " + askPrice);
   }
}
```

11.6.2.5.18 PercentComplete

Definition

Returns a value indicating the percentage complete of the real-time bar processing.

Notes:

1. Since a historical bar is complete, values during **State.Historical** should be ignored (also the case with TickReplay bars)
2. Some BarsTypes may not be compatible with the **PercentComplete** property. In these cases, a value of 0 always returns (e.g., Range, Renko, Point & Figure, Kagi, LineBreak, and some other 3rd party bars types)

Property Value

A `double` value representing a percent e.g. a value of .5 indicates the bar was at 50%. This property is read-only.

Syntax

`Bars.PercentComplete`

Tip: If you are developing a custom **BarsType**, please use the GetPercentComplete()

method used to calculate the value returned by **PercentComplete**

Examples

```
protected override void OnBarUpdate()
{
   if(State == State.Realtime)
   {
      Draw.TextFixed(this, "barstatus",
Bars.PercentComplete.ToString("P2"), TextPosition.BottomRight);
   }
}
```

11.6.2.5.19 TickCount

Definition

Returns the total number of ticks of the current bar processing.

Note: For historical usage, you must use **Calculate.OnEachTick** with TickReplay enabled; otherwise a value of 1 will returned.

Property Value

A `long` value that represents the total number of ticks of the current bar.

Syntax

`Bars.TickCount`

Examples

```
// Prints the tick count to the output window
Print("The tick count of the current bar is " +
Bars.TickCount.ToString());
```

11.6.2.5.20 ToChartString()

Definition

Returns the bars series as a formatted string, including the Instrument.FullName, BarsPeriod Value, and BarsPeriodType name.

Note: To obtain a return value which matches the user configured ChartBars Label property, please see the ChartBars.ToChartString() method

Syntax
`Bars.ToChartString()`

Return Value
A `string` value that represents the bars series

Parameters
This method does not accept any parameters

Examples

```
protected override void OnBarUpdate()
{
   // print the chart string on start up
   if(CurrentBar == 0)
      Print(Bars.ToChartString()); // ES 09-15 (60 Minute)
}
```

11.6.2.6 Charts

The following section covers information related to accessing chart related data, such as ChartControl, ChartBars, ChartScales, and ChartPanels, and advanced Indicator Rendering.

In this section

1. ChartBars	The Chart's Primary Data Series which the NinjaScript object is running
2. ChartControl	The entire grid hosting the chart including the X-axis, additional panels, and chart related properties
3. ChartPanel	The Panel that the indicator object is running
4. ChartSc	The Y-axis of the indicator object's panel

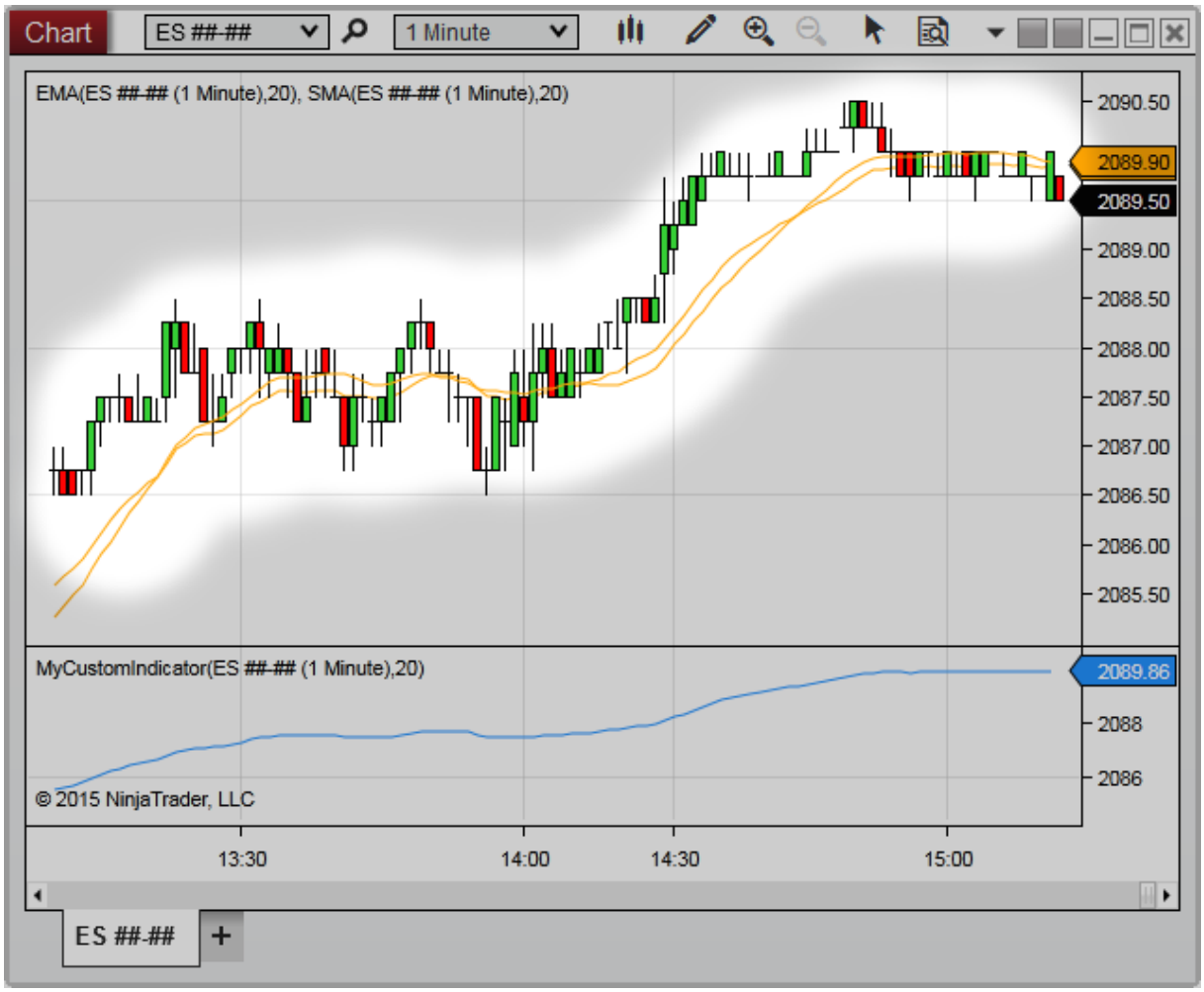
ale	

A chart's objects can be broken down into the four following areas:

11.6.2.6.1 ChartBars

The **ChartBars** class provides GUI access related methods and properties to the primary bars series configured on the Chart through the Data Series menu. For data access information related to the NinjaScript input's bars series, please use the Bars Series object (or the BarsArray for multi-series input)

Note: A **ChartBars** object will **ONLY** exist should the hosting NinjaScript type be loaded through a Chart. For example, a **Strategy** would have access to a **ChartBars** property when running on a **Chart**, but would **NOT** when loaded through the Strategies Grid or Strategy analyzer.



Warning: It is crucial to check for object references before accessing the **ChartBars** otherwise possible null reference errors can be expected depending on where the NinjaScript object was started. See example below

Methods and Properties

Bars	Data returned from the historical data repository.
Count	The total number of ChartBars that exist on the chart

FromIndex	An index value representing the first bar painted on the chart.
GetBarIdxByTime()	An ChartBar index value calculated from a time value on the chart.
GetBarIdxByX()	Returns the ChartBar index value at a specified x-coordinate relative to the ChartControl.
GetTimeByBarIdx()	The ChartBars time value calculated from a bar index value on the chart.
Panel	The Panel index value that the ChartBars eside.
Properties	Various ChartBar properties that have been configured from the Chart's Data Series menu.
ToChartString()	A string formatted for the Chart's Data Series Label as well as the period.
ToIndex	An index value representing the last bar painted on the chart.

Example

```
protected override void OnStateChange()
{
   if (State == State.DataLoaded)
   {
      if(ChartBars != null)
      {
         Print("The starting number of bars on the chart is " +
ChartBars.Bars.Count);
      }
      else
      {
         Print("Strategy was not loaded from a chart, exiting
strategy...");
         return;
      }
   }
}
```

11.6.2.6.1.1 Bars

Definition

Represents the data returned from the historical data repository in relation to the primary ChartBars object configured on the chart. See also Bars

Property Value

A Bars object

Syntax

ChartBars.Bars

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   if(ChartBars != null && ChartBars.Bars != null)
   {
      Print("The configured bars period type represented on the
chart is" + ChartBars.Bars.BarsPeriod.BarsPeriodType);
   }
}
```

11.6.2.6.1.2 Count

Definition

The total number of ChartBars in the charts primary data series

Property Value
An `int` value representing the the total number of bars.

Syntax
```
ChartBars.Count
```

Examples
```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   if(ChartBars != null)
   {
      Print("ChartBars contain " + ChartBars.Count + " bars");
      //Output:  ChartBars contain 73 bars
   }
}
```

11.6.2.6.1.3 FromIndex

Definition
An index value representing the first bar rendered on the chart. See also ToIndex.

> **Note**: This value is **NOT** the first value that exists on the ChartBars, but rather the first bar index that is within the viewable range of the chart canvas area. This value changes as the user interacts with the ChartControl time-scale (x-axis).

Property Value
An `int` representing the first bar index painted on the chart

Syntax
```
ChartBars.FromIndex
```

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   if (ChartBars != null)
   {
      // loop through all of the viewable range of the chart
      for (int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
      {
         // print the High value for each index within the viewable
range
         Print(High.GetValueAt(barIndex));
      }
   }
}
```

11.6.2.6.1.4 GetBarIdxByTime()

Definition

Returns the ChartBars index value calculated from the time parameter provided.

Method Return Value

An `int` representing the bar index value at a specific time

Syntax

`ChartBars.GetBarIdxByTime(ChartControl chartControl, DateTime time)`

Method Parameters

`chartControl`	The ChartControl object used to determine the chart's time axis
`time`	The DateTime value used to convert to a **ChartBar** index value

Examples

```
protected override void OnBarUpdate()
{
   if (ChartBars != null)
   {
      Print(ChartBars.GetBarIdxByTime(ChartControl, Time[0]));
   }
}
```

11.6.2.6.1.5 GetBarIdxByX()

Definition

Returns the ChartBars index value at a specified x-coordinate relative to the ChartControl.

Method Return Value

An `int` value representing the bar index

Syntax

```
ChartBars.GetBarIdxByX(ChartControl chartControl, int x)
```

Method Parameters

`chartControl`	The ChartControl object used to determine the chart's time axis
`x`	The x-coordinate used to find a bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // get the users mouse down point and convert to device pixels
for DPI accuracy
   int mousePoint =
chartControl.MouseDownPoint.X.ConvertToHorizontalPixels(chartContro
l.PresentationSource);

   // convert mouse point to bar index
   int barIdx = ChartBars.GetBarIdxByX(chartControl, mousePoint);

   Print("User clicked on Bar #" + barIdx);
}
```

11.6.2.6.1.6 GetTimeByBarIdx()

Definition
Returns the ChartBars time value calculated from a bar index parameter provided.

Method Return Value
A DateTime struct representing a bar time value at a specific bar index value

Syntax
```
ChartBars.GetTimeByBarIdx(ChartControl chartControl, int barIndex)
```

Method Parameters

chartControl	The ChartControl object used to determine the chart's time axis
barIndex	An int value representing a bar index used to convert to a ChartBar index value

Examples

```
protected override void OnBarUpdate()
{
   if (ChartBars != null)
   {
      Print(ChartBars.GetTimeByBarIdx(ChartControl,
50));  //8/11/2015 4:30:00 AM
   }
}
```

11.6.2.6.1.7 Panel

Definition
A zero-based index value that represents the ChartPanel where the ChartBars reside.

> **Note**: This is **NOT** the same as the PanelUI property displays on the Chart's Data Series menu. A **ChartBars.Panel** value of 0 represents the first panel on the chart.

Property Value
An int indicating the panel of the **ChartBars**

Syntax
Bars.Panel

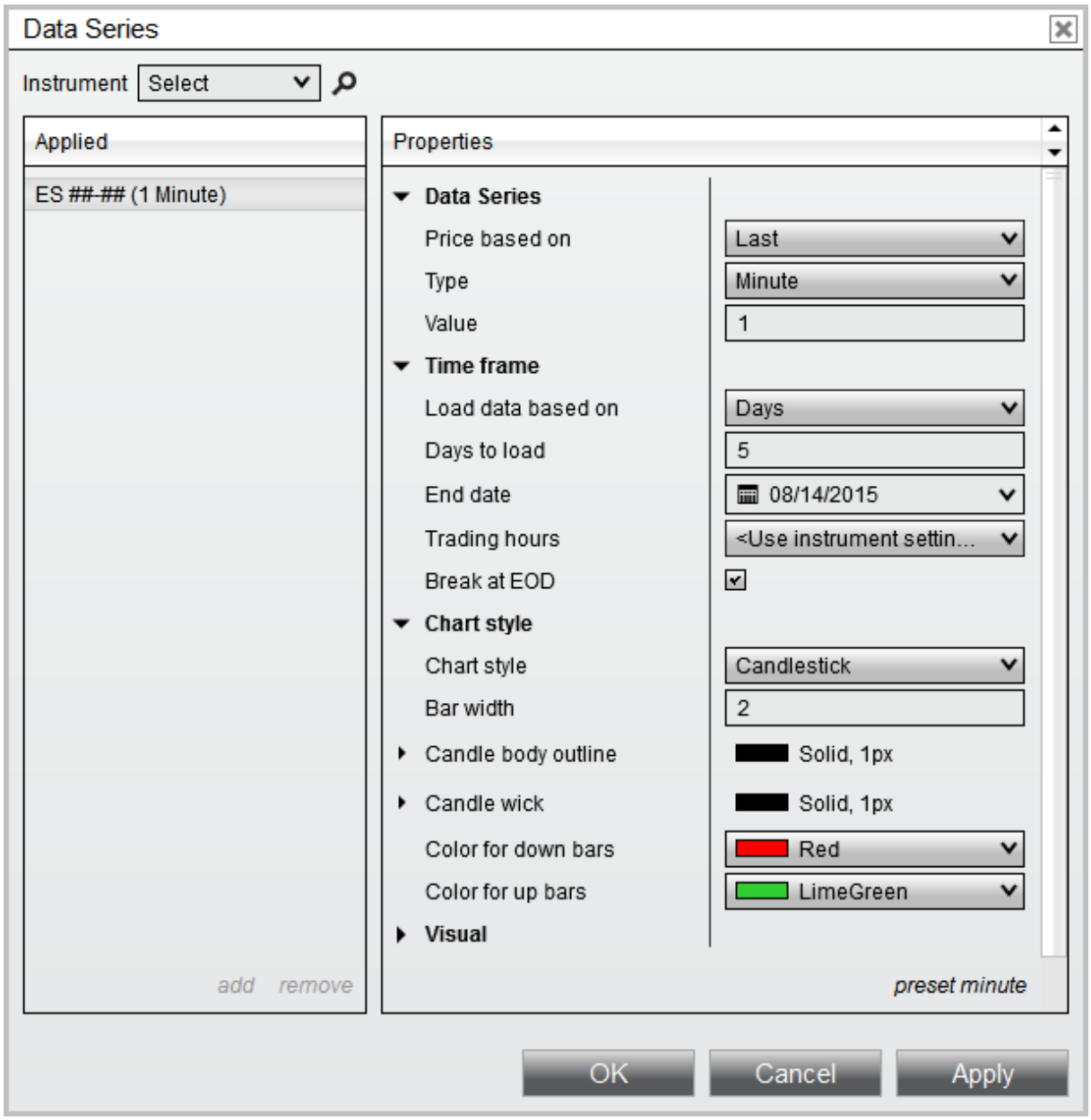
Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   Print("ChartBars reside on panel index: " + ChartBars.Panel);
   // Output:  ChartBars reside on panel index: 0
}
```

11.6.2.6.1.8 Properties

Definition
Represents various ChartBar properties configured from the Chart's Data Series menu.

Note: The properties on this page indicate what have been *configured* by the user, and is **NOT** necessarily representative of what is actually contained on the chart. For example, a user may have a requested 120 days of chart data, however only 60 days of bar data actually returned from their provider.



Warning: These are UI properties which are designed to be set by a user. Attempting to modify these values through a custom script is **NOT** guaranteed to take effect.

Properties

AutoScale	A `bool` indicating if the Chart Data Series participates in the chart's auto scaling methods

BarsBack	An `int` representing the Chart's Data Series configured "Bars to load" when the `RangeType.Bars` is selected
BarsPeriod	The BarsPeriod object configured for Chart's Data Series
CenterPriceOnScale	A `bool` indicating if the Chart's Data Series should center the last traded price on the chart scale
ChartStyle	The ChartStyle object configured for the Chart's Data Series
ChartStyleType	A `ChartStyleType enum` indicating the type of chart style configured. System defaults include: • ChartStyleType.Box, • ChartStyleType.CandleStick, • ChartStyleType.LineOnClose, • ChartStyleType.OHLC, • ChartStyleType.PointAndFigure, • ChartStyleType.KagiLine, • ChartStyleType.OpenClose, • ChartStyleType.Mountain
DaysBack	An `int` representing the Chart's Data Series configured "Days to load" when the `RangeType.Days` is selected
DisplayInDataBox	A `bool` indicating if the Chart's Data Series value should display in the Chart's Data Box
DisplayName	A `string` representing the Chart's Data Series instrument and period
From	A `DateTime` representing the Chart's Data Series configured

	"Start Date" when the `RangeType.CustomRange` configured.
Instrument	A `string` representing the Chart's Data Series instrument
IsStableSession	A `bool` indicating the Chart's Data Series Break EOD option is configured
IsTickReplay	A `bool` indicating the Chart's Data Series Tick Replay option is configured
Label	A `string` representing the configured Chart's Data Series "Label"
LongExecutionBrush	A Brush object representing the Chart's Data Series "Color for execution - buy" brush configured
PaintPriceMarker	A `bool` indicating the Chart's Data Series Price Marker "Visible" option is configured
Panel	An `int` indicating whichChart's Data Series "Panel" the ChartBars are configured
PlotExecutions	A `ChartExecutionStyle enum` representing "Plot executions" option. Possible values include: • ChartExecutionStyle.DoNotPlot, • ChartExecutionStyle.MarkersOnly, • ChartExecutionStyle.TextAndMarker

PositionPenLoser	A Stroke object representing the Chart's Data Series "NinjaScript strategy unprofitable trade line"
PositionPenWinner	A Stroke object representing the Chart's Data Series "NinjaScript strategy profitable trade line"
PriceMarker	A `PriceMarker` object representing various brushes used to paint the Chart's Data Series "Price marker"
RangeType	A `RangeType` `enum` indicating the "Load data based on" value configured on the Data Series. Possible values include: • RangeType.Bars, • RangeType.Days, • RangeType.CustomRange
ScaleJustification	A ScaleJustification enum indicating the "Scale justification" option configured on the Chart's Data Series. Possible values include: • ScaleJustification.Right, • ScaleJustification.Left, • ScaleJustification.Overlay
ShortExecutionBrush	A Brush object representing the Chart's Data Series "Color for execution - sell" brush configured
ShowGlobalDrawObjects	A `bool` indicating the Chart's Data Series "Show global draw object" option is configured
To	A `DateTime` representing the configured "End Date" used with any `RangeType`

TradingHoursBreakLine	A TradingHoursBreakLine object representing the stroke used and `TradingHoursBreakLineVisible enum` used for the Chart's Data Series "Trading hours break line". Possible `TradingHoursBreakLine.TradingHoursBreakLineVisible` values include: • TradingHoursBreakLineVisible.AllSessions, • TradingHoursBreakLineVisible.EodOnly, • TradingHoursBreakLineVisible.Off
TradingHoursData	A `string` representing the Chart's Data Series configured "Trading hours" option

11.6.2.6.1.9 ToChartString()

Definition

Returns a formatted string representing the ChartBars.Properties.Label property, BarsPeriod Value, and BarsPeriodType name.

> **Note**: The property returned is dependent on a user configured Data Series property, and results may return differently than expected. See also Bars.ToChartString() for a return value which is not subject to user-defined variables.

Syntax

```
ChartBars.ToChartString()
```

Return Value

A `string` value that represents the **ChartBars** label and configured bars period

Parameters

This method does not accept any parameters

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   if (ChartBars != null)
      Print(ChartBars.ToChartString()); // My Favorite Instrument
(1 Minute)
}
```

11.6.2.6.1.10 ToIndex

Definition

An index value representing the last bar rendered on the chart. See also FromIndex.

> **Note**: This value is **NOT** the last value that exists on the ChartBars, but rather the last bar index that is within the viewable range of the chart canvas area. This value changes as the user interacts with the ChartControl time-scale (x-axis).

Property Value

An `int` representing the last bar index painted on the chart

Syntax

`ChartBars.ToIndex`

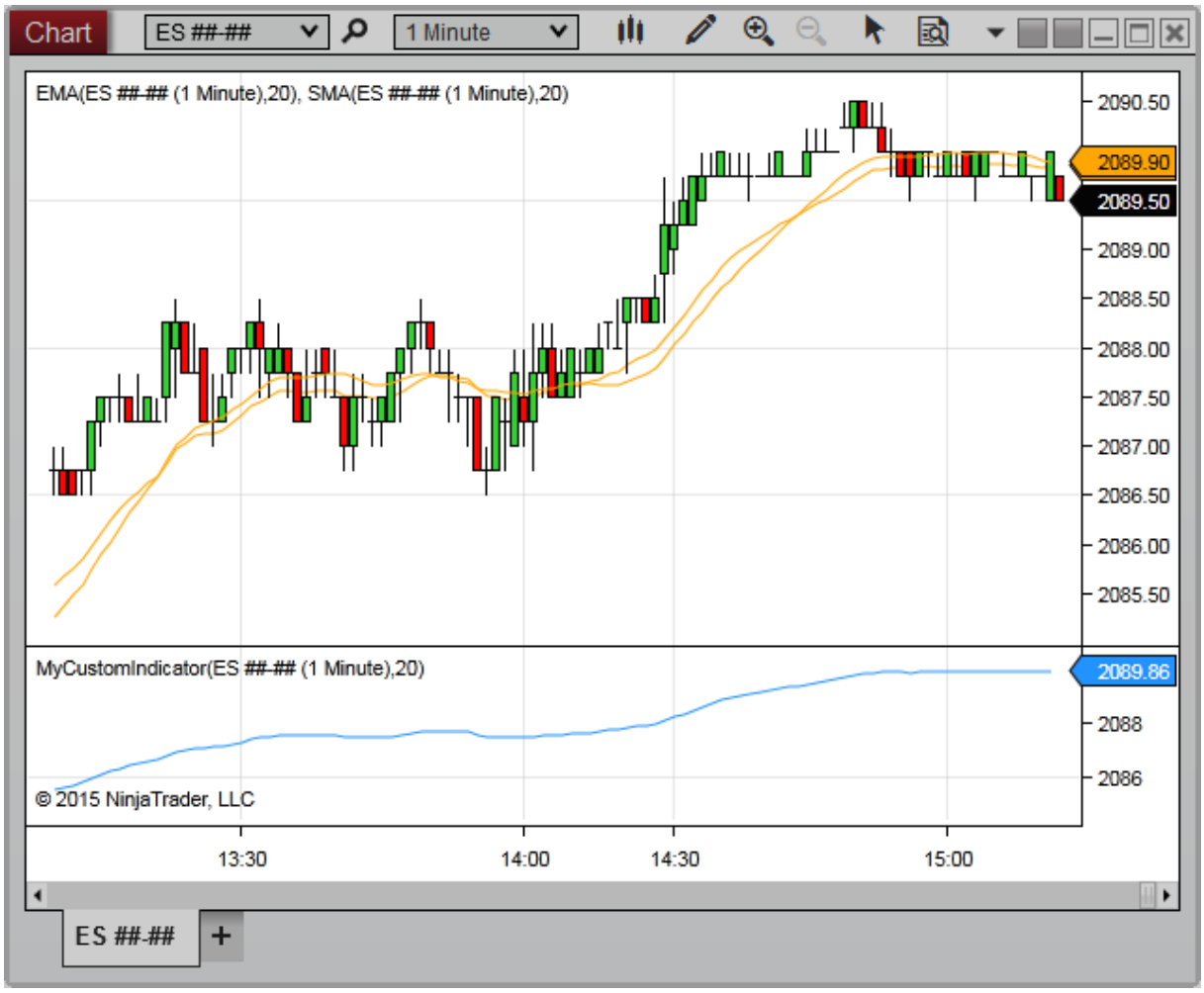
Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   if (ChartBars != null)
   {
      // loop through all of the viewable range of the chart
      for (int barIndex = ChartBars.FromIndex; barIndex <=
ChartBars.ToIndex; barIndex++)
      {
         // print the High value for each index within the viewable
range
         Print(High.GetValueAt(barIndex));
      }
   }
}
```

11.6.2.6.2 ChartControl

The **ChartControl** class provides access to a wide range of properties and methods related to the location of objects on a chart and other chart-related properties. The ChartControl object provides information related to the entire hosting grid of the chart, which overlap with the ChartPanel, ChartScale and ChartBars.

Note: The ChartControl object is **ONLY** guaranteed to be available when a NinjaScript type initiates from a Chart Window. There are situations where an indicator or strategy starts from another Windows (such as the Control Center's Strategies Grid, or from a Strategy Analyzer), where the **ChartContol** object is **NOT** accessible. Therefore, the **ChartControl** object should always be safely accessed (e.g., from within a try-catch, or conditionally using null reference checks)



Warning: The **ChartControl** and its methods and properties should **ONLY** be access once the State has reached **State.Historical**

Methods and Properties

AxisXHeight	Measures the distance (in pixels) between the x-axis and the top of the horizontal scroll bar
AxisYLeftWidth	Measures the distance (in pixels) between the y-axis and the left margin of a chart
AxisYRightWidth	Measures the distance (in pixels) between the y-axis and the right margin of a chart
BarMarginLeft	Measures the margin to the left of each bar on the chart, in pixels
BarsArray	Provides a collection of ChartBars objects currently configured on the chart
BarSpacingType	Provides the type of bar spacing used for the primary Bars object on the chart
BarsPeriod	Provides the period (interval) used for the primary Bars object on the chart
BarWidth	Measures the value of the bar width set for the primary Bars object on the chart
BarWidthArray	An array containing the values of the BarWidth properties of all Bars objects on the chart
CanvasLeft	Indicates the x-coordinate (in pixels) of the beginning of the chart canvas area
CanvasRight	Indicates the x-coordinate (in pixels) of the end of the chart canvas area
CanvasZoomState	Indicates the current state of the Zoom tool on the chart

ChartPanels	Holds a collection of ChartPanel objects
CrosshairType	Indicates the Cross Hair type currently enabled on the chart
FirstTimePainted	Indicates a time value of the first bar painted on the chart
GetBarPaintWidth()	Returns the width of the bars in the primary Bars object on the chart, in pixels
GetSlotIndexByTime()	Returns the slot index of the primary Bars object on the chart corresponding to a specified time value
GetSlotIndexByX()	Returns the slot index of the primary Bars object on the chart corresponding to a specified x-coordinate on the visible chart canvas
GetTimeBySlotIndex()	Returns a time value corresponding to a specified slot index of the primary Bars object on the chart
GetTimeByX()	Returns a time value related to the primary Bars' slot index at a specified x-coordinate on the chart canvas
GetXByBarIndex()	Returns the chart-canvas x-coordinate of the bar at a specified index of a specified ChartBars object on the chart
GetXByTime()	Returns the chart-canvas x-coordinate of the slot index of the primary Bars object corresponding to a specified time
Indicators	Returns a collection of indicators currently configured on the chart
IsScrollArrowVisible	Indicates the time-axis scroll arrow is visible in the top-right corner of the chart
IsStayInDrawMode	Indicates the Stay in Draw Mode is currently enabled on the chart

IsYAxisDisplayedLeft	Indicates the y-axis displays (in any chart panel) to the left side of the chart canvas
IsYAxisDisplayedOverlay	Indicates an object on the chart is using the Overlay scale justification
IsYAxisDisplayedRight	Indicates the y-axis displays (in any chart panel) to the right side of the chart canvas
LastSlotPainted	Indicates the slot index of the most recently painted bar on the primary Bars object configured on the chart
LastTimePainted	Indicates the time of the most recently painted bar on the primary Bars object configured on the chart
MouseDownPoint	Indicates the x- and y-coordinates of the mouse cursor at the most recent OnMouseDown() event
Properties	A collection of properties related to the configuration of the Chart
SlotsPainted	Indicates the number of index slots in which bars are painted within the chart canvas area
Strategies	A collection of strategies configured on the chart
TimePainted	Indicates the range of time in which bars are painted on the visible chart canvas

11.6.2.6.2.1 AxisXHeight

Definition

Measures the distance (in pixels) between the x-axis and the top of the horizontal scroll bar near the bottom of the chart.

Property Value

A `double` representing the number of pixels separating the x-axis and the top of the horizontal scroll bar on the chart.

Syntax

```
<ChartControl>.AxisXHeight
```

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
      // Print the number of pixels between the x-axis and the top
of the horizontal scrollbar
      double height = chartControl.AxisXHeight;
      Print(height);
}
```

Based on the image below, AxisXHeight reveals that the space between the x-axis and the top of the horizontal scrollbar is 31 pixels on this chart.

11.6.2.6.2.2 AxisYLeftWidth

Definition

Measures the distance (in pixels) between the y-axis and the left edge of a chart.

Property Value

A `double` representing the number of pixels separating the y-axis and the left edge of the chart.

Syntax

```
<ChartControl>.AxisYLeftWidth
```

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
      // Print the number of pixels between the y-axis and the left
edge of the chart
      double leftWidth = chartControl.AxisYLeftWidth;
      Print(leftWidth);
}
```

Based on the image below, AxisYLeftWidth reveals that the space between the y-axis and the left edge of the chart is 53 pixels on this chart.

Note: When there are no left-justified data series on a chart, AxisYLeftWidth will return 0, as there will be no space between the y-axis and the left margin.

11.6.2.6.2.3 AxisYRightWidth

Definition

Measures the distance (in pixels) between the y-axis and the right edge of a chart.

Property Value

A `double` representing the number of pixels separating the y-axis and the right edge of the chart.

Syntax

`<ChartControl>.AxisYRightWidth`

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
      // Print the number of pixels between the y-axis and the
right edge of the chart
      double rightWidth = chartControl.AxisYRightWidth;
      Print(rightWidth);
}
```

Based on the image below, AxisYRightWidth reveals that the space between the y-axis and the right edge of the chart is 53 pixels on this chart.

Note: When there are no right-justified data series on a chart, AxisYRightWidth will return 0, as there will be no space between the y-axis and the right edge.

11.6.2.6.2.4 BarMarginLeft

Definition

A hard-coded minimum bar margin value, set to 8 pixels, which can be used as a base value when creating custom Chart Styles.

Property Value

A value representing the minimum margin applied to the left edge of bars. This value is hard-coded to 8 pixels, and it can be used as a base value when setting the bar margin in custom Chart Styles.

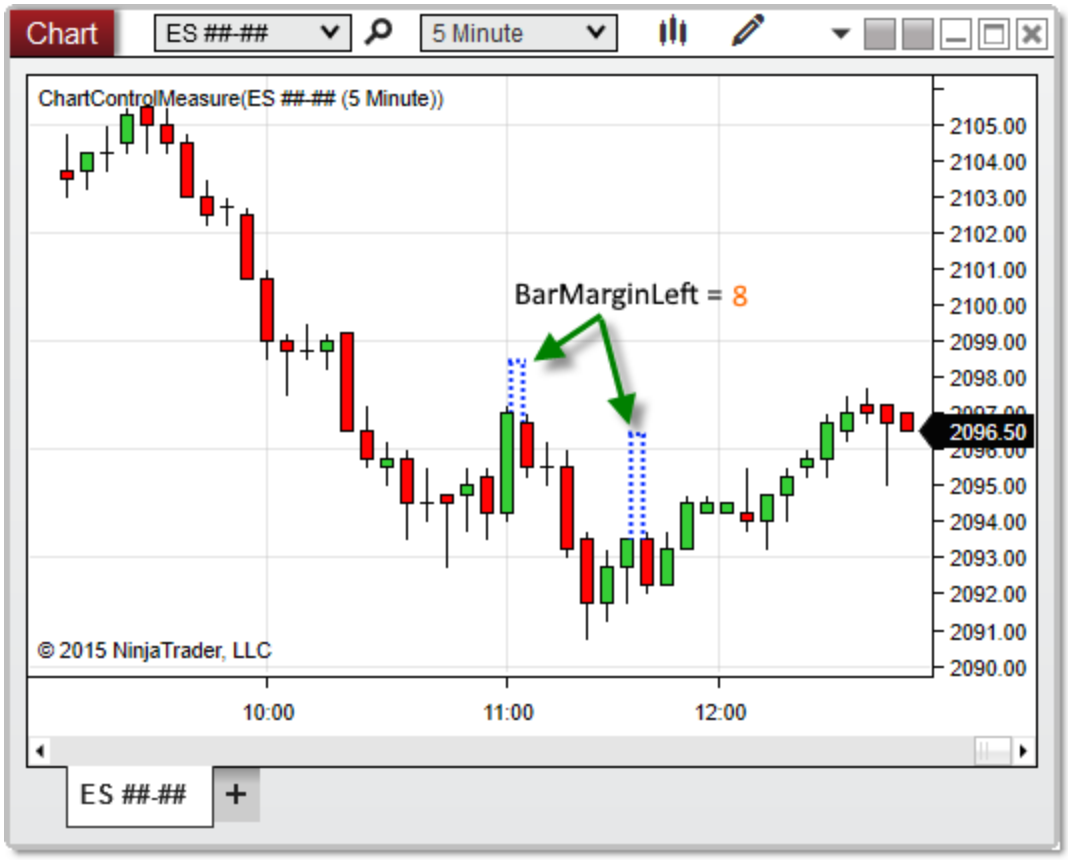
Syntax

<ChartControl>.BarMarginLeft

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
      // Print the number of pixels maintained as a margin to the
left of bars
      double barMargin = chartControl.BarMarginLeft;
      Print(barMargin);
}
```

Based on the image below, BarMarginLeft reveals that the minimum margin maintained to the left of each bar is 8 pixels on this chart.



11.6.2.6.2.5 BarsArray

Definition

Provides a collection of ChartBars objects currently configured on the chart.

Property Value

An ObservableCollection of `ChartBars` objects

Syntax
<ChartControl>.BarsArray

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Instantiate a new <ChartControl>.BarsArray collection
    System.Collections.ObjectModel.ObservableCollection<ChartBars>
myChartBars = chartControl.BarsArray;

    // Print the number of bars in each Bars object within the
<ChartControl>.BarsArray collection
    foreach(ChartBars bars in myChartBars)
    {
        Print(bars.Bars.Count);
    }
}
```

11.6.2.6.2.6 BarSpacingType

Definition
Indicates the type of bar spacing used for the primary Bars object on the chart.

Property Value
An enum representing one of the values below:

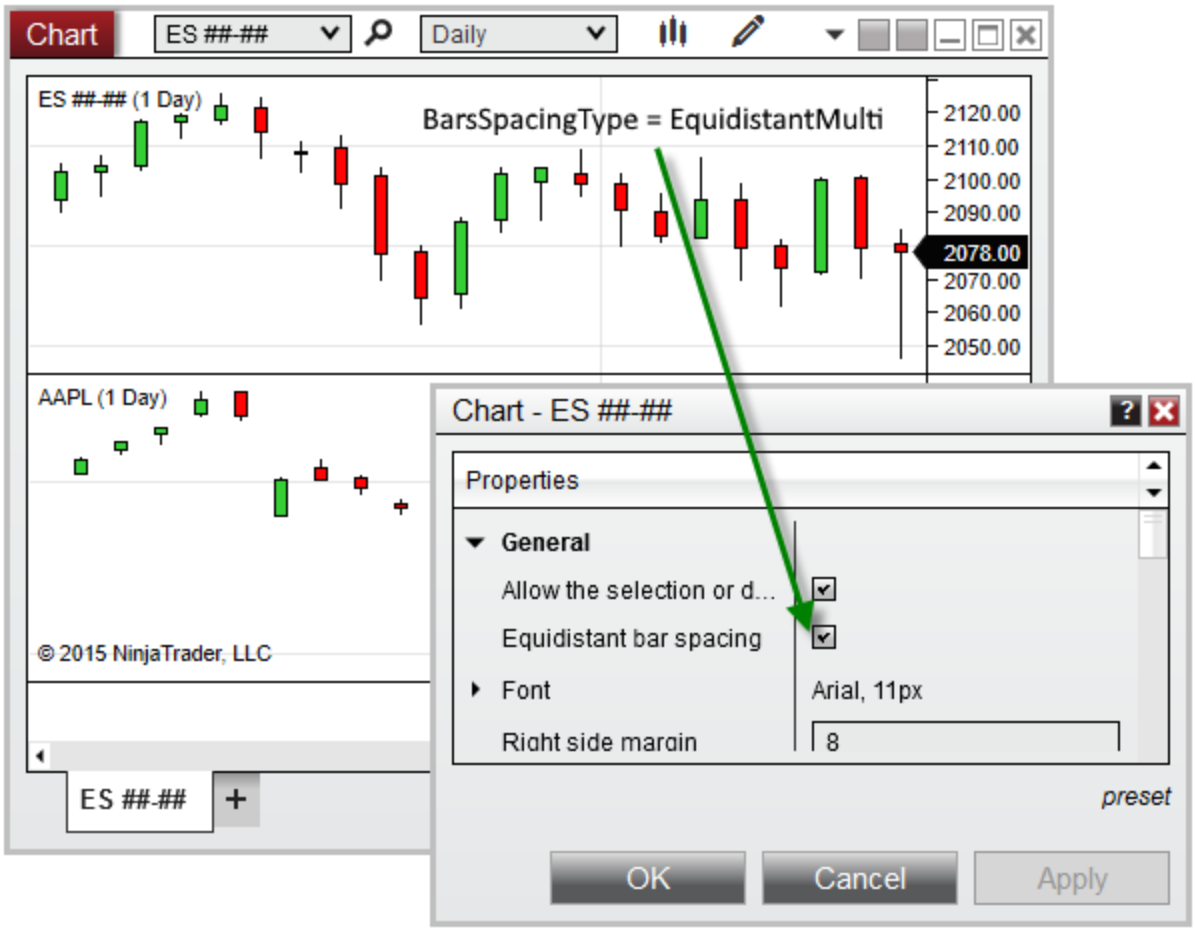
EquidistantSingle	Indicates Equidistant Bar Spacing is used, and only one Bars object exists on the chart
EquidistantMulti	Indicates Equidistant Bar Spacing is used, and more than one Bars objects exist on the chart
TimeBased	Indicates Time-Based bar spacing is used

Syntax
<ChartControl>.BarSpacingType

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Print the type of bar spacing used on the chart
    Print(chartControl.BarSpacingType);
}
```

Based on the image below, BarSpacingType confirms that there are multiple Bars objects configured on the chart, and that the chart is set to Equidistant Bar Spacing:



11.6.2.6.2.7 BarsPeriod

Definition

Provides the period (interval) used for the primary Bars object on the chart.

Property Value

A `NinjaTrader.Data.BarsPeriod` object containing information on the period used by the Bars object on the chart.

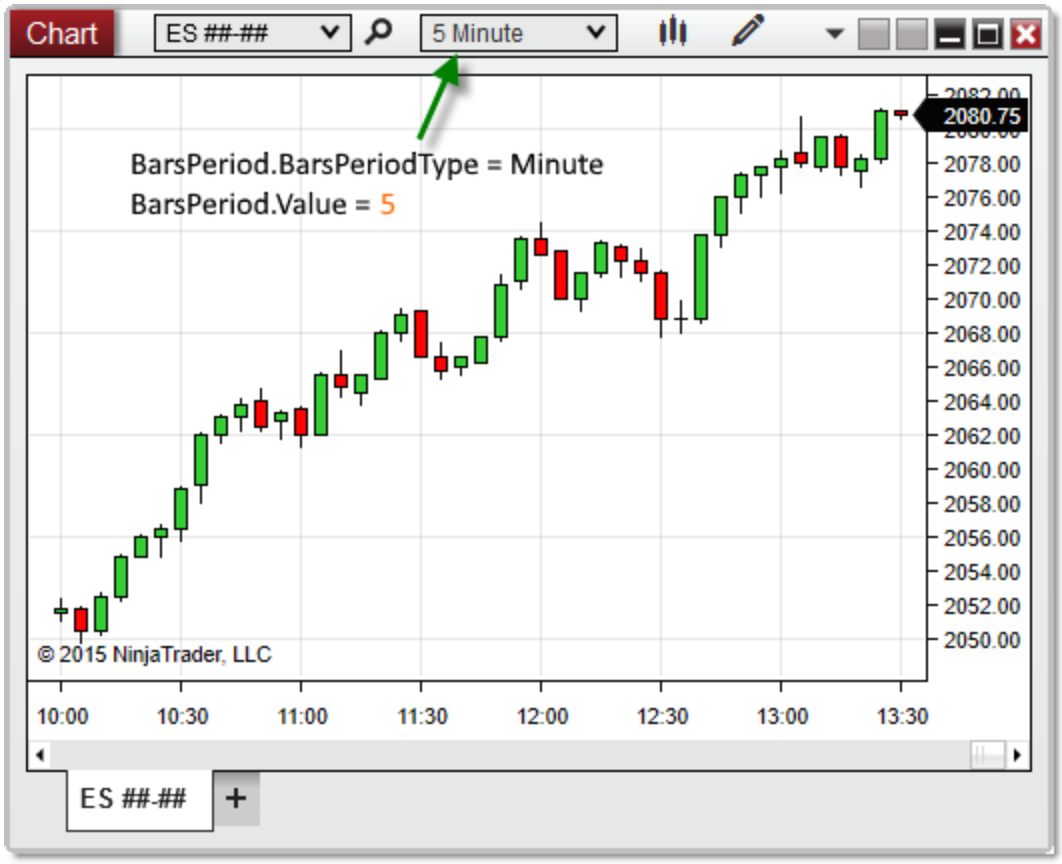
Syntax

`<ChartControl>.BarsPeriod`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    BarsPeriod period = chartControl.BarsPeriod;

    // Print the period (interval) of the Bars object on the chart
    Print(period);
}
```

Based on the image below, BarsPeriod confirms that the primary Bars object on the chart is configured to a 5-minute interval.



11.6.2.6.2.8 BarWidth

Definition

Measures the value of the bar width set for the primary Bars object on the chart.

> **Note**: This property value is not stated in pixels. To obtain the pixel-width of bars on the chart, use GetBarPaintWidth() instead.

Property Value

A `double` representing the value of the bar width.

Syntax

```
<ChartControl>.BarWidth
```

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    double barWidth = chartControl.BarWidth;

    // Prints the width of bars on the chart
    Print(barWidth);
}
```

Based on the image below, BarWidth reveals that the bars on the chart are 4.02 pixels wide.

11.6.2.6.2.9 BarWidthArray

Definition

An array containing the values of the BarWidth properties of all Bars objects applied to the chart.

Property Value

An `array` of `double` variables containing the values of the BarWidth properties of Bars objects on the chart.

Syntax

<ChartControl>.BarWidthArray[]

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Assign BarWidthArray to a new array
    double[] barWidths = chartControl.BarWidthArray;

    double referenceWidth = barWidths[0];

    // Trigger an alert if bar widths on the chart differ
    foreach (double width in barWidths)
    {
        if (width != referenceWidth)
            Alert("mismatchWidths", Priority.Low, "Bar widths on
the chart do not match!", " ", 20, Brushes.White, Brushes.Black);
    }
}
```

11.6.2.6.2.10 CanvasLeft

Definition
Indicates the x-coordinate (in pixels) of the beginning of the chart canvas area.

Property Value
A `double` representing the beginning of the chart canvas area.

Syntax
<ChartControl>.CanvasLeft

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Store the beginning and ending x-coordinates of the canvas
area
    double canvasBeginCoordinate = chartControl.CanvasLeft;
    double canvasEndCoordinate = chartControl.CanvasRight;

    // Print the stored values
    Print(String.Format("Chart canvas begins at x-coordinate {0}
and ends at x-coordinate {1}", canvasBeginCoordinate,
canvasEndCoordinate));
}
```

Based on the image below, CanvasLeft reveals that the chart canvas area begins at x-coordinate 53.

Note: When no data series are left-aligned on a chart, CanvasLeft will return 0, representing the x-coordinate origin, because the chart canvas will begin at coordinate 0.

11.6.2.6.2.11 CanvasRight

Definition

Indicates the x-coordinate (in pixels) of the end of the chart canvas area.

Property Value

A `double` representing the end of the chart canvas area.

Syntax

<ChartControl>.CanvasRight

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Store the beginning and ending x-coordinates of the canvas
area
    double canvasBeginCoordinate = chartControl.CanvasLeft;
    double canvasEndCoordinate = chartControl.CanvasRight;

    // Print the stored values
    Print(String.Format("Chart canvas begins at x-coordinate {0}
and ends at x-coordinate {1}", canvasBeginCoordinate,
canvasEndCoordinate));
}
```

Based on the image below, CanvasRight reveals that the chart canvas ends at x-coordinate 526.

Yes,

11.6.2.6.2.12 CanvasZoomState

Definition

Indicates the current state of the Zoom tool on the chart. This property reveals the state of the tool while it is in use, and does not indicate a chart is zoomed in on or not. As soon as a zoom action is completed, the tool is considered to be no longer in use.

Property Value

An enum representing the state of the Zoom tool on the chart. Possible values are listed below:

None	The Zoom tool is not currently being used

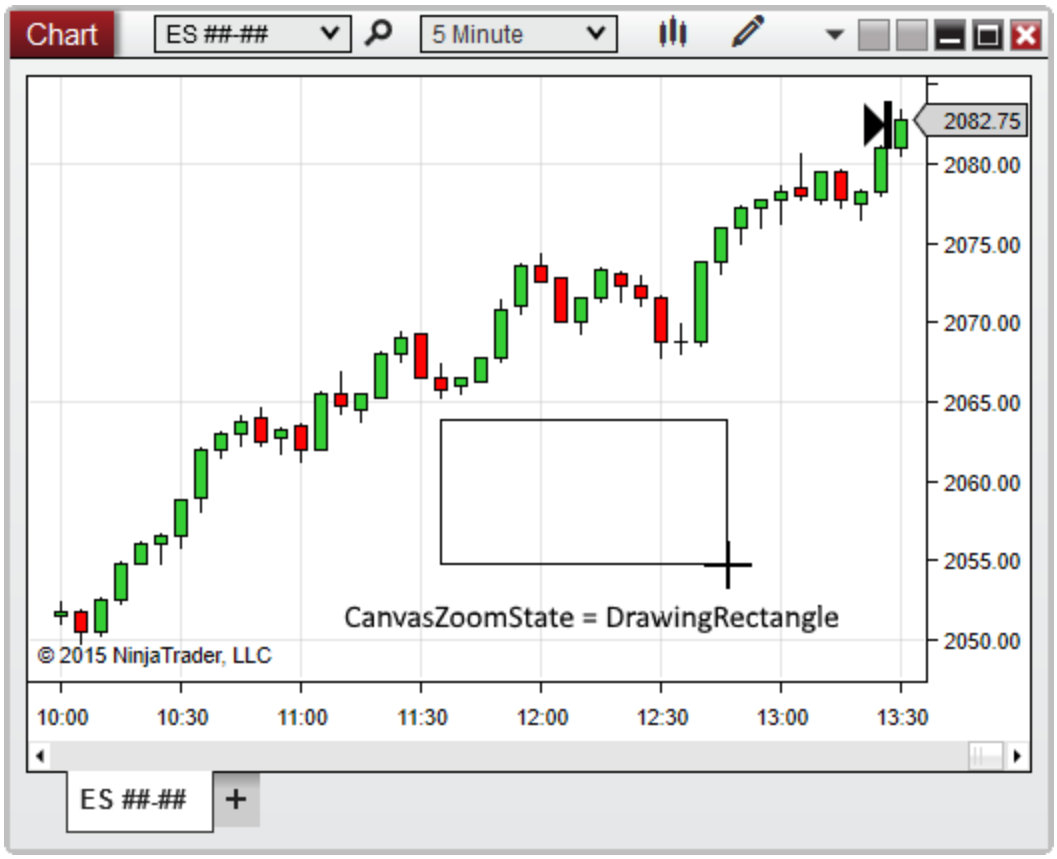
Selected	The Zoom tool is selected, but has not yet been used to zoom in
Drawing Rectangle	The Zoom tool is currently in use (User is currently drawing the rectangle in which to zoom)

Syntax
<ChartControl>.CanvasZoomState

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    CanvasZoomState zoomState = chartControl.CanvasZoomState;

    // Trigger an alert while a user is zooming in on a chart
    if (zoomState == CanvasZoomState.DrawingRectangle)
        Alert("zoomAlert", Priority.Medium, "Make sure to zoom in
on the entire chart pattern!", " ", 60, Brushes.White,
Brushes.Black);
}
```

Based on the image below, CanvasZoomState confirms that the Zoom rectangle is currently being drawn:



11.6.2.6.2.13 ChartPanels

Definition

Holds a collection of ChartPanel objects containing information about the panels active on the chart.

Property Value

An ObservableCollection of `ChartPanel` objects

Syntax

<ChartControl>.ChartPanels

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // Print the number of panels currently displayed on the chart
   Print(String.Format("There are {0} panels on the chart",
chartControl.ChartPanels.Count));
}
```

Based on the image below, there are three ChartPanel objects in the ChartPanels collection, as seen by ChartPanels.Count in the code above.

11.6.2.6.2.14 CrosshairType

Definition

Indicates the Cross Hair type currently enabled on the chart.

Property Value

An `enum` specifying the type of Cross Hair currently enabled on the chart. Possible values are listed below:

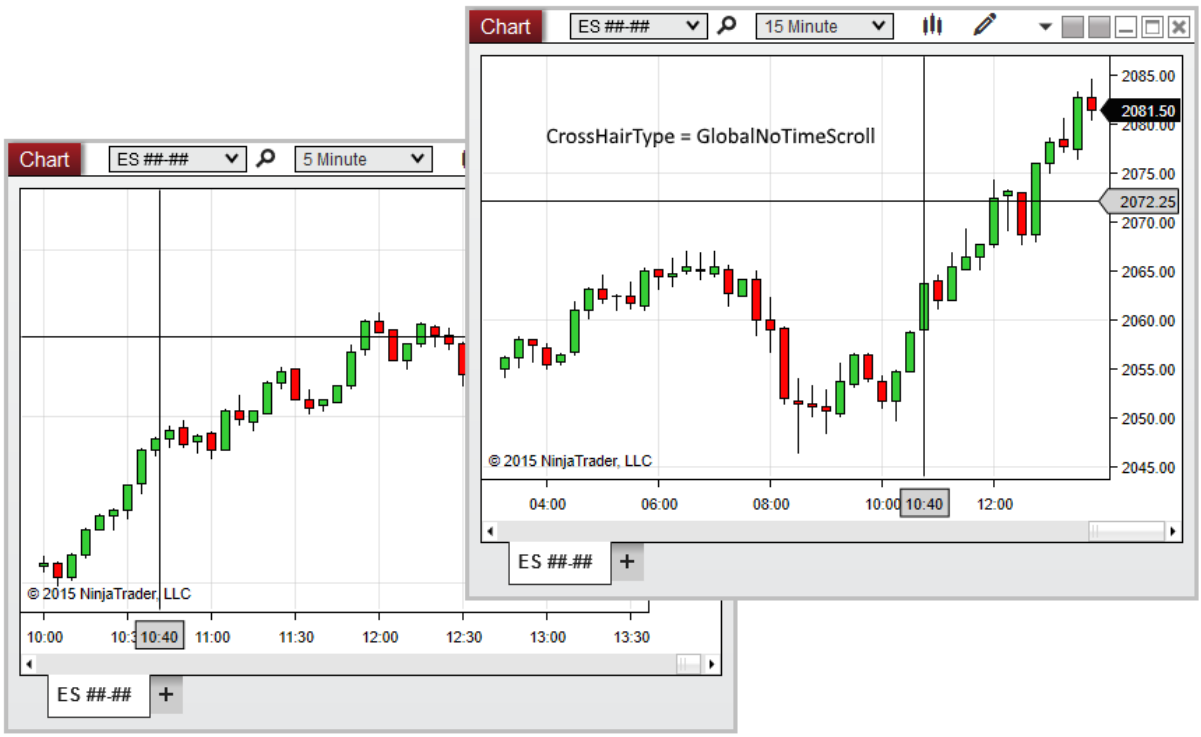
Local	The local (single-chart) Cross Hair is enabled
Global	Global Cross Hair
GlobalNoTimeScroll	Global Cross Hair (No Time Scroll) is enabled

Syntax
<ChartControl>.CrosshairType

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // Print a message if the user enables the Global Cross Hair
without time scrolling
    if (chartControl.CrosshairType ==
CrosshairType.GlobalNoTimeScroll)
        Print("It is recommended to enable Global Cross Hair time
scrolling with this indicator");
}
```

In the image below, CrosshairType reveals that Global Cross Hair (No Time Scroll) is enabled on the chart.



11.6.2.6.2.15 FirstTimePainted

Definition
Indicates a DateTime value of the first bar painted on the chart.

FirstTimePainted provides the timestamp of the first bar, NOT the time at which the bar was painted. For example, if a chart was opened and historical bars drawn on August 2nd at 5:00 pm, but the first bar on the chart is painted at a time-axis value of July 31st at 1:00 am, then FirstTimePainted will return the July 31st date and time.

Property Value

A DateTime object containing information on the timestamp of the first bar of the chart.

Syntax

<ChartControl>.FirstTimePainted

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Draw text to display the first timestamp of a bar on the
chart
    Draw.Text(this, "firstTimeText", String.Format("The first bar
of {0} is drawn at {1}", Instrument.MasterInstrument.Name,
chartControl.FirstTimePainted), 1, High[0],Brushes.Black);
}
```

In the image below, FirstTimePainted reveals that the first painted slot corresponds to 8/12/17 at 10:40:00 AM.

11.6.2.6.2.16 GetBarPaintWidth()

Definition

Returns the width of the bars in the primary Bars object on the chart, in pixels.

Method Return Value

A `double` representing the pixel width of bars on the chart

Syntax

```
<ChartControl>.GetBarPaintWidth(ChartBars chartBars)
```

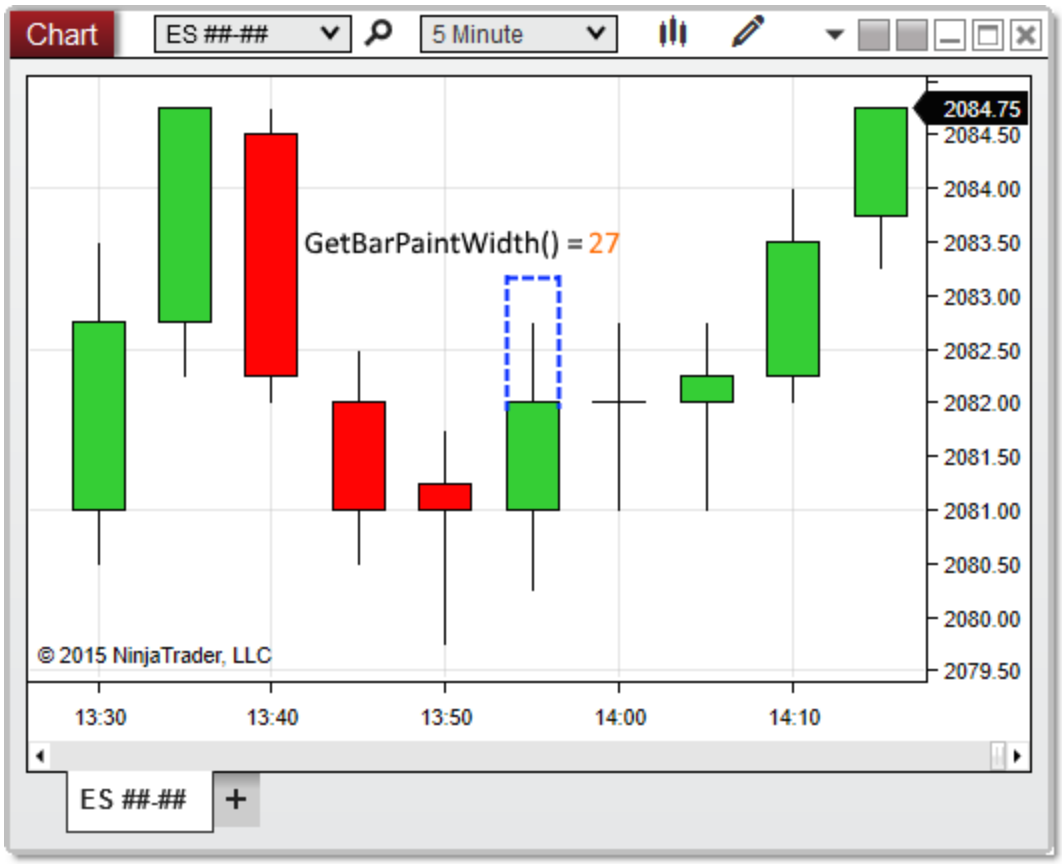
Method Parameters

chartBars	A ChartBars object to measure

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Use BarsArray[0] to pass in a ChartBars object representing
the primary Bars object on the chart
    double barPixelWidth =
chartControl.GetBarPaintWidth(chartControl.BarsArray[0]);

    // Print the pixel width of bars painted on the chart
    Print(String.Format("Bars on the chart are {0} pixels wide",
barPixelWidth));
}
```

In the image below, GetBarPaintWidth() reveals that the bars are being drawn 27 pixels wide on the chart:



11.6.2.6.2.17 GetSlotIndexByTime()

Definition

Returns the slot index relative to the chart control corresponding to a specified time value.

Notes:
- A "**Slot**" is used in **Equidistant** bar spacing and represents a position on the chart canvas background which may or may not contain a bar. The concept of "**Slots**" does **NOT** exist on a **TimeBased** bar spacing type.
- If you are looking for information on a bar series, please see ChartBars.GetBarIdxByTime()

Method Return Value
A `double` representing a slot index

Syntax
```
<ChartControl>.GetSlotIndexByTime(DateTime time)
```

Warning: This method **CANNOT** be called on **BarSpacingType.TimeBased** charts. You will need to ensure an **Equidistant** bar spacing type is used, otherwise errors will be thrown.

Method Parameters

time	A DateTime Structure used to determine a slot index

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // ensure that GetSlotIndexByTime is called on TimeBased charts
   if(chartControl.BarSpacingType != BarSpacingType.TimeBased)
   {
      // get the slot index of the first time painted on the chart
      double slotIndex =
chartControl.GetSlotIndexByTime(chartControl.FirstTimePainted);

      Print(slotIndex);
   }
}
```

11.6.2.6.2.18 GetSlotIndexByX()

Definition

Returns the slot index relative to the chart control corresponding to a specified x-coordinate

Notes:

- A "**Slot**" is used in **Equidistant** bar spacing and represents a position on the chart canvas background which may or may not contain a bar. The concept of "**Slots**" does **NOT** exist on a **TimeBased** bar spacing type.
- If you are looking for information on a bar series, please see ChartBars.GetBarIdxByX()
- Since the slot index is based on the chart canvas, the value returned by GetSlotIndexByX() can be expected to change as new bars are painted, or as the chart is scrolled backward or forward on the x-axis.

Method Return Value

A `double` representing a slot index; returns -1 on a time based bar spacing type

Syntax

```
<ChartControl>.GetSlotIndexByX(int x)
```

Method Parameters

x	An `int` used to determine a slot index

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Find the index of the bar painted at x-coordinate 35
    double slotIndex = chartControl.GetSlotIndexByX(35);

    // Print the slot index of the specified time
    Print(slotIndex);
}
```

11.6.2.6.2.19 GetTimeBySlotIndex()

Definition

Returns a time value relative to the chart control corresponding to a specified slot index.

Notes:

- A "**Slot**" is used in **Equidistant** bar spacing and represents a position on the chart canvas background which may or may not contain a bar. The concept of "**Slots**" does **NOT** exist on a **TimeBased** bar spacing type.
- If you are looking for information on a bar series, please see ChartBars.GetTimeByBarIdx()
- For slot index values in the future, an estimation of time will be returned. It is not possible to predict the future time of a bar for all bar series (i.e., tick/volume based bars)

Method Return Value

A **DateTime** object corresponding the a specified slot index; returns **DateTime** value for 'now' on a time based bar spacing type

Syntax

```
<ChartControl>.GetTimeBySlotIndex(double slotIndex)
```

Method Parameters

slotIndex	The slot index used to determine a time value

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Find the timestamp of the bar at index 150
    DateTime slotTime = chartControl.GetTimeBySlotIndex(150);

    // Print the date of slotTime
    Print(slotTime.Date);
}
```

11.6.2.6.2.20 GetTimeByX()

Definition

Returns a time value related to the primary Bars' slot index at a specified x-coordinate relative to the ChartControl.

Note: Since the time is based upon a coordinate of the chart canvas, the value returned by GetTimeByX() can be expected to change as new bars are painted on the chart, or as

the chart is scrolled backward or forward on the x-axis.

Method Return Value
A DateTime object corresponding to a slot index at a specified x-coordinate

Syntax
```
<ChartControl>.GetTimeByX(int x)
```

Method Parameters

x	The x-coordinate used to find a time value

Example
```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Find the timestamp of the bar at x-coordinate 100
    DateTime slotTime = chartControl.GetTimeByX(100);

    // Print the date of slotTime
    Print(slotTime);
}
```

11.6.2.6.2.21 GetXByBarIndex()

Definition
Returns the chart-canvas x-coordinate of the bar at a specified index of a specified ChartBars object on the chart.

Note: Since the index is based upon bars that move across the chart canvas as new bars are painted, the value returned by GetXByBarIndex() can be expected to change as new bars are painted on the chart, or as the chart is scrolled backward or forward on the x-axis.

Method Return Value
An `int` representing a chart-canvas x-coordinate

Syntax

```
<ChartControl>.GetXByBarIndex(ChartBars chartBars, int barIndex)
```

Method Parameters

chartBars	The ChartBars object to check
barIndex	The slot index used to determine an x-coordinate

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    double xCoordinate = chartControl.GetXByBarIndex(ChartBars,
100);

    // Print the x-coordinate value
    Print(xCoordinate);
}
```

11.6.2.6.2.22 GetXByTime()

Definition

Returns the chart-canvas x-coordinate of the slot index of the primary Bars object corresponding to a specified time.

Note: Since the time correlates with a specific bar index, and since bars move on the chart canvas as new bars are painted, the value returned by GetXByTime() can be expected to change as new bars are painted on the chart, or as the chart is scrolled backward or forward on the x-axis.

Method Return Value

An `int` representing a chart-canvas x-coordinate

Syntax

```
<ChartControl>.GetXByTime(DateTime time)
```

Method Parameters

time	A DateTime object used to determine an x-coordinate

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    DateTime timeToCheck = new DateTime(2017, 8, 6, 11, 0, 0);

    // Find the chart-canvas x-coordinate of the bar at the
specified time
    int xCoordinate = chartControl.GetXByTime(timeToCheck);

    // Print the x-coordinate value
    Print(xCoordinate);
}
```

11.6.2.6.2.23 Indicators

Definition

Contains a collection of indicators currently configured on the chart.

Property Value

A `ChartObjectCollection` of `NinjaTrader.Gui.NinjaScript.IndicatorRenderBase` objects representing the indicators on the chart

Syntax

`<ChartControl>.Indicators`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Instantiate a ChartObjectCollection to hold
chartControl.Indicators
    ChartObjectCollection<NinjaTrader.Gui.NinjaScript.IndicatorRend
erBase> indicatorCollection = chartControl.Indicators;

    // Print the Calculate setting for any configured indicators
not using Calculate.OnBarClose
    foreach (NinjaTrader.Gui.NinjaScript.IndicatorRenderBase
indicator in indicatorCollection)
    {
        if(indicator.Calculate != Calculate.OnBarClose)
            Print(String.Format("{0} is using Calculate.{1}",
indicator.Name, indicator.Calculate.ToString()));
    }
}
```

11.6.2.6.2.24 IsScrollArrow Visible

Definition

Indicates the time-axis scroll arrow is visible in the top-right corner of the chart.

Property Value

A `bool` value. When **True**, indicates that the scroll arrow is visible on the chart; otherwise **False**.

Syntax

`<ChartControl>.IsScrollArrowVisible`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Print a message if the scroll arrow is visible on the chart
    if(chartControl.IsScrollArrowVisible);
        Print("The chart is currently not set to auto-scroll. Click
the scroll arrow to return to auto-scrolling");
}
```

Based on the image below, IsScrollArrowVisible confirms that the scroll arrow is currently visible on the chart.

11.6.2.6.2.25 IsStayInDraw Mode

Definition

Indicates Stay in Draw Mode is currently enabled on the chart.

Property Value

A `bool` value. When **True**, indicates that **Stay in Draw Mode** is enabled on the chart; otherwise **False**.

Syntax

`<ChartControl>.IsStayInDrawMode`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Print a message if Stay in Draw Mode is enabled
    if(chartControl.IsStayInDrawMode);
        Print("Stay in Draw Mode is currently enabled");
}
```

11.6.2.6.2.26 IsYAxisDisplayedLeft

Definition

Indicates the y-axis displays (in any chart panel) to the left side of the chart.

Property Value

A `boolean` value. When **True**, indicates that the y-axis displays to the left of the chart canvas; otherwise **False**.

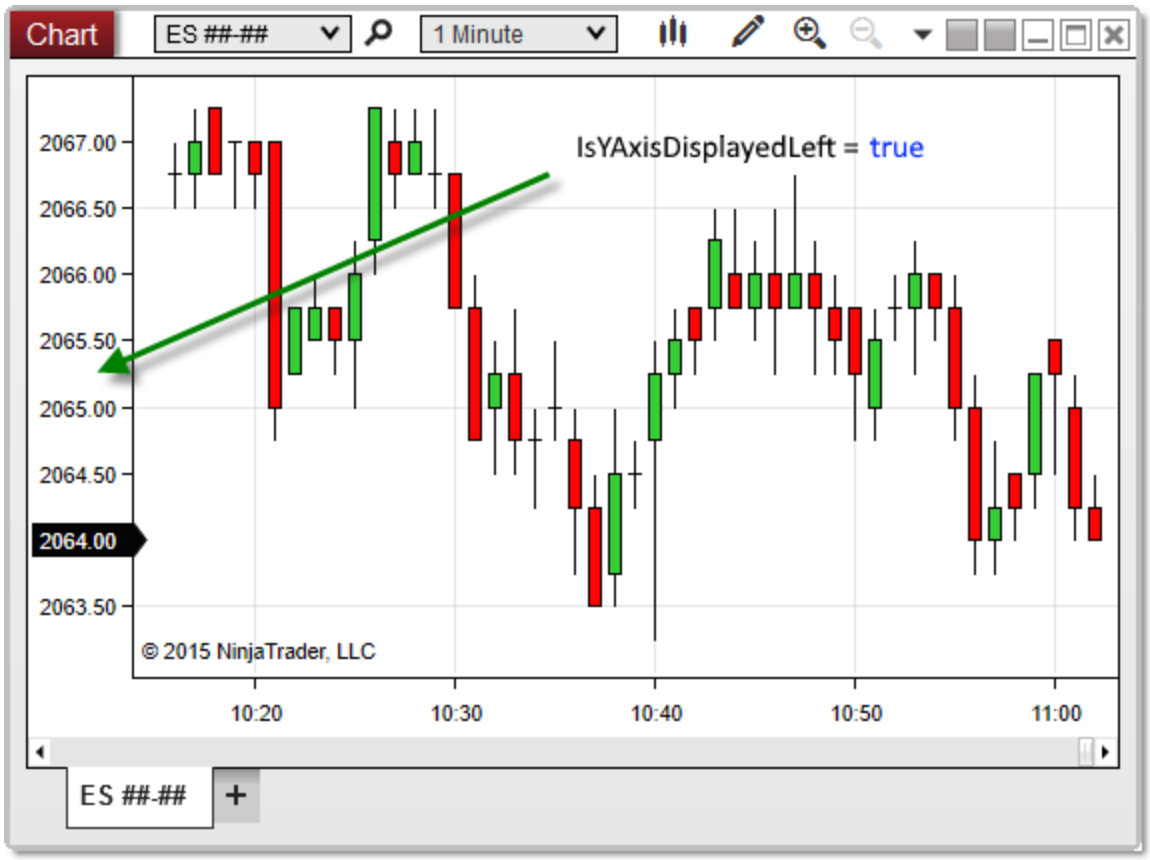
Syntax

`<ChartControl>.IsYAxisDisplayedLeft`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Print the value of IsYAxisDisplayedLeft
    Print("Y-Axis visible to the left of the chart canvas? " +
chartControl.IsYAxisDisplayedLeft);
}
```

Based on the image below, IsYAxisDisplayedLeft confirms that the y-axis displays to the left of the chart canvas.



11.6.2.6.2.27 IsYAxisDisplayedOverlay

Definition

Indicates an object on the chart is using the Overlay scale justification.

Property Value

A `boolean` value. When **True**, indicates that one or more objects on the chart are using the Overlay scale justification; otherwise **False**.

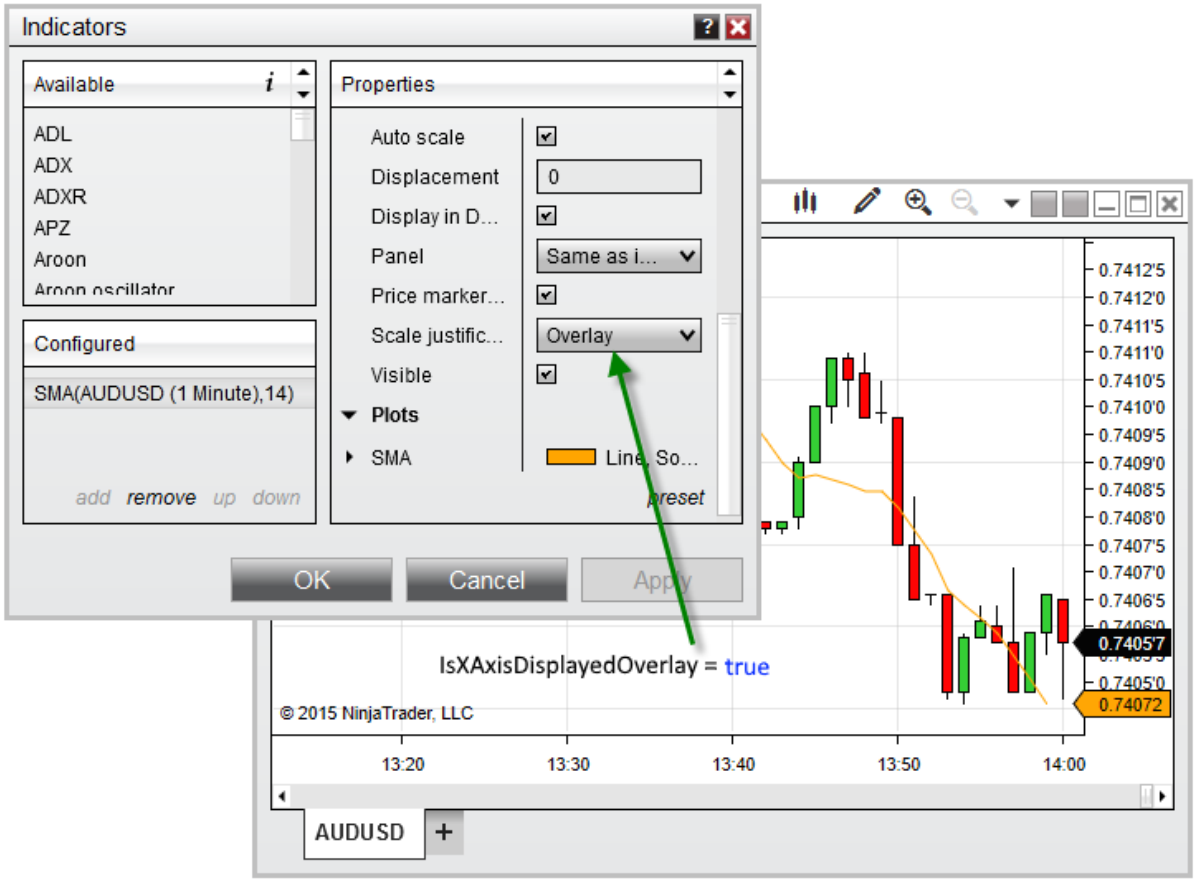
Syntax

`<ChartControl>.IsYAxisDisplayedOverlay`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Print the value of IsYAxisDisplayedOverlay
    Print("Is Overlay used? " +
chartControl.IsYAxisDisplayedOverlay);
}
```

Based on the image below, `IsYAxisDisplayedOverlay` confirms that the an object on the chart, in this case an SMA indicator, is using the Overlay scale justification.



11.6.2.6.2.28 IsYAxisDisplayedRight

Definition

Indicates the y-axis displays (in any chart panel) to the right side of the chart.

Property Value

A `boolean` value. When **True**, indicates that the y-axis displays to the right of the chart canvas; otherwise **False**.

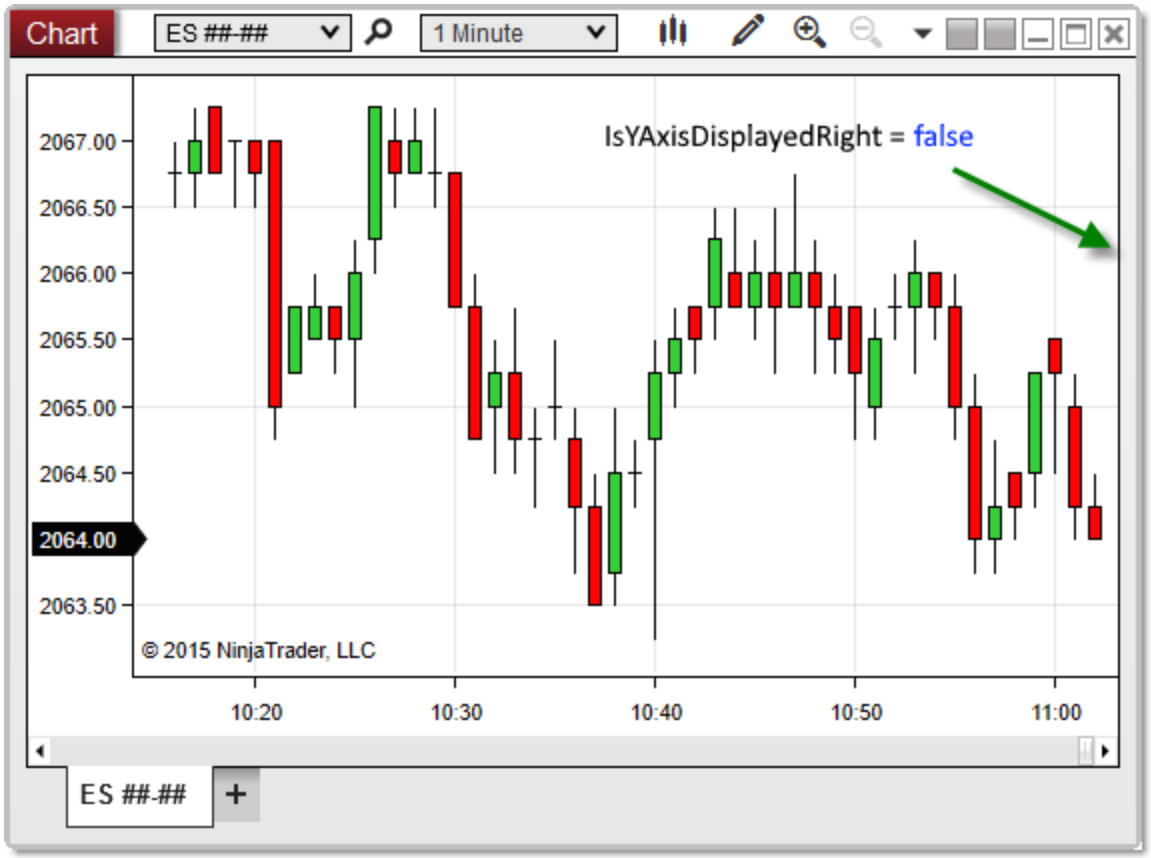
Syntax

```
<ChartControl>.IsYAxisDisplayedRight
```

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Print the value of IsYAxisDisplayedRight
    Print("Y-Axis visible to the right of the chart canvas? " +
chartControl.IsYAxisDisplayedRight);
}
```

Based on the image below, IsYAxisDisplayedRight confirms that the y-axis is not displayed to the right of the chart canvas.



11.6.2.6.2.29 LastSlotPainted

Definition

Indicates the most recent (last) slot index of the Data Series on the chart, regardless if a bar is actually painted in that slot.

Note: LastSlotPainted differs from ChartBars.ToIndex, which returns the last index containing a bar painted in the visible area of the chart.

Property Value
A `int` representing the most recent (last) slot index on the chart

Syntax
`<ChartControl>.LastSlotPainted`

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    int lastSlot = chartControl.LastSlotPainted;

    // Print the index of the last slot on the chart
    Print(lastSlot);
}
```

11.6.2.6.2.30 LastTimePainted

Definition
Indicates the time of the most recently painted bar on the primary Bars object configured on the chart.

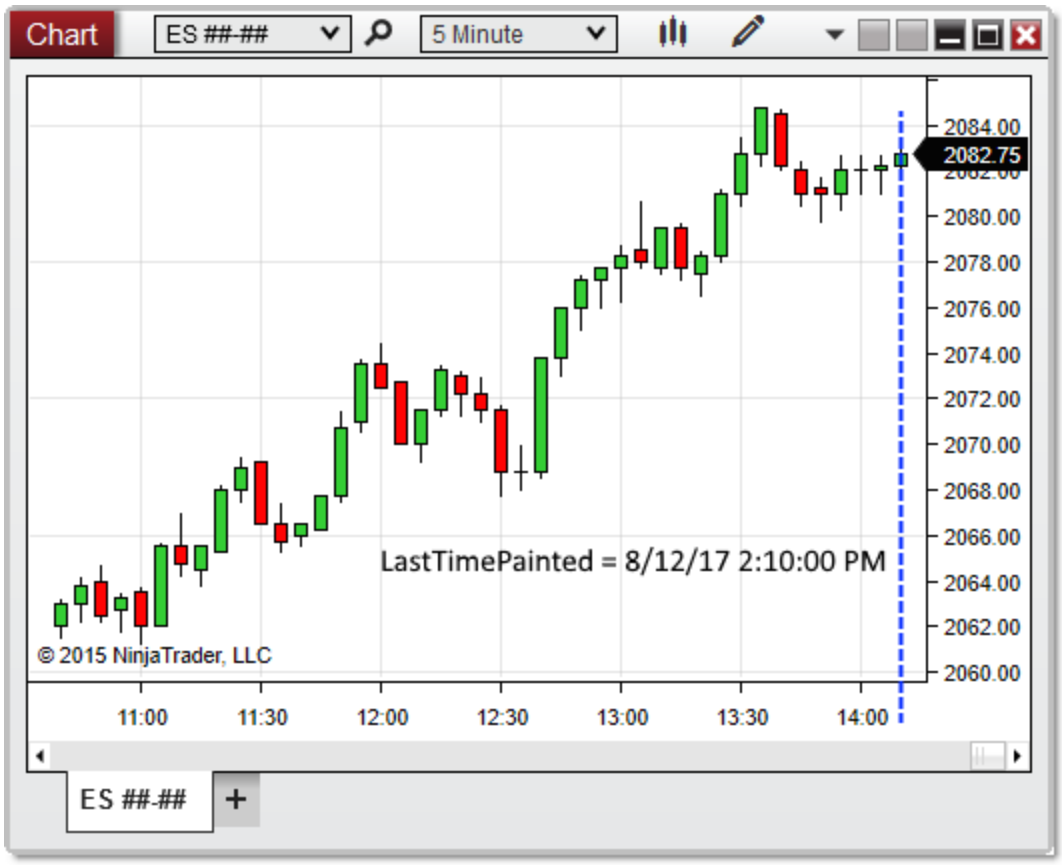
Property Value
A DateTime object corresponding to the slot index of the most recently painted bar

Syntax
`<ChartControl>.LastTimePainted`

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    DateTime lastSlotTime = chartControl.LastTimePainted;

    // Print the index of the last slot painted on the chart
    Print(lastSlotTime);
}
```

In the image below, LastTimePainted reveals that the last index painted on the chart corresponds to 8/12/17 at 2:10:00 PM.



11.6.2.6.2.31 MouseDow nPoint

Definition

Indicates the WPF x- and y-coordinates of the mouse cursor at the most recent OnMouseDown() event.

Property Value

A Point object containing x- and y-coordinates of the mouse cursor when the left mouse button is clicked or held

Syntax

```
<ChartControl>.MouseDownPoint
```

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    Point cursorPoint = chartControl.MouseDownPoint;

    // Print the x- and y-coordinates of the mouse cursor when
clicked
    Print(String.Format("Mouse clicked at coordinates {0},{1}",
cursorPoint.X, cursorPoint.Y));
}
```

11.6.2.6.2.32 PresentationSource

Definition

Provides a reference to the base window in which the chart is rendered. PresentationSource can be used when converting application pixels to/from device pixels via the helper methods in the ChartingExtensions class.

Property Value

A PresentationSource object representing the base window in which the chart is rendered.

Syntax

```
ChartControl.PresentationSource
```

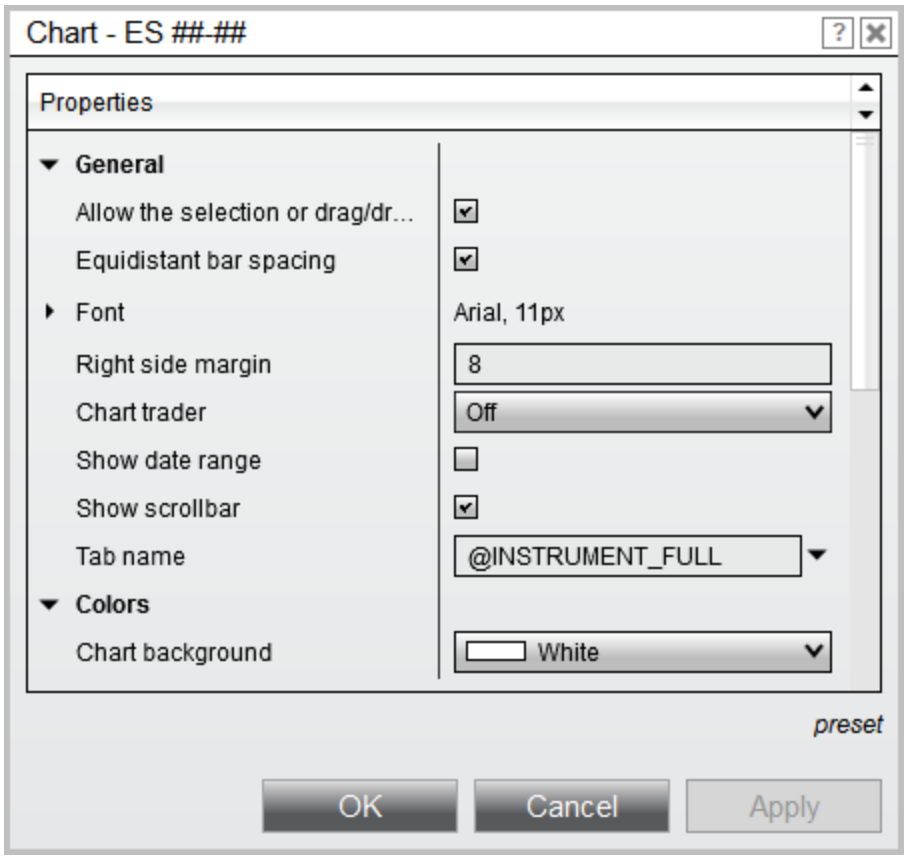
Example

```
int devicePixelX;

protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // Obtain the device-pixel coordinate corresponding to an
application-pixel X value of 500
   devicePixelX = ChartingExtensions.ConvertToHorizontalPixels(500,
 ChartControl.PresentationSource);
}
```

11.6.2.6.2.33 Properties

Definition

A collection of properties related to the configuration of the Chart



Warning: These are UI properties which are designed to be set by a user. Attempting to modify these values through a custom script is **NOT** guaranteed to take effect.

Property Value

A `ChartControlProperties` object containing values for all properties configured on the specified `ChartBars` object.

Property	Return Type and Description
AllowSelectionDragging	A `bool` indicating selected chart objects can be moved within a chart panel or dragged to a new chart panel
AlwaysOnTop	A `bool` indicating "Always on Top" is enabled for the chart window
AreHGridLinesVisible	A `bool` indicating the horizontal grid lines are visible on the chart

AreTabsVisible	A `bool` indicating tabs are visible in the chart window
AreVGridLinesVisible	A `bool` indicating the vertical grid lines are visible on the chart
AxisPen	A `Stroke` object used in painting the x- and y-axis
BarDistance	A `float` measuring the distance (in pixels) between the left or right edge of one bar and the corresponding edge of the previous or subsequent bar
BarMarginRight	An `int` representing the "Right Margin" property value configured on the chart
ChartBackground	A `Brush` object used to paint the chart background
ChartText	A `Brush` object used to paint text on the chart
ChartTraderVisibility	An `enum` indicating the visibility status of **Chart Trader**. Possible values are `Collapsed`, `Visible`, and `VisibleCollapsed`
CrosshairCrosshairType	An `enum` indicating the type of **Cross Hair** enabled on the chart. Possible values are `Off`, `Local`, `Global`, and `GlobalNoTimeScroll`
CrosshairIsLocked	A `bool` indicating the **Cross Hair's** vertical line is locked in place
CrosshairLabelBackground	A `Brush` object used to paint the **Cross Hair's** price and time markers in the x- and y-axis
CrosshairLabelForeground	A `Brush` object used to paint the text in the **Cross Hair's** price and time markers
CrosshairPen	A `string` representing the `Pen` used within the `Stroke` that is used to draw the **Cross Hair**

CrosshairStroke	A `CrosshairStroke` object containing information on the **Cross Hair's** `Stroke`, `CrosshairType`, and `isLocked` property
GridLineHPen	A `GridLine` object containing information on the horizontal grid lines' `Stroke` and `isVisible` property
GridLineVPen	A `GridLine` object containing information on the vertical grid lines' `Stroke` and `isVisible` property
InactivePriceMarkersBackground	A `Brush` object used to paint the background of inactive price markers on the chart
InactivePriceMarkersForeground	A `Brush` object used to paint the display text of inactive price markers on the chart
LabelFont	A `NinjaTrader.Gui.Tools.SimpleFont` object containing information on the font used in text labels throughout the chart
PanelSplitterPen	A `Stroke` object used to paint the lines between chart panels
ShowDateRange	A `bool` indicating the date range of the bars painted on the visible chart canvas will be displayed within the chart
ShowScrollBar	A `bool` indicating the horizontal scroll bar is visible beneath the x-axis
SnapMode	An `enum` indicating the currently enabled Snap Mode. Possible values are `None`, `Bar`, `Price`, and `BarAndPrice`
TabName	A `string` representing the name of the current tab

Syntax

```
<ChartControl>.Properties
```

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Instantiate a ChartControlProperties object to hold a
reference to chartControl.Properties
    ChartControlProperties myProperties = chartControl.Properties;

    // Set the AllowSelectionDragging property to false
    myProperties.AllowSelectionDragging = false;
}
```

11.6.2.6.2.34 SlotsPainted

Definition

Indicates the number of index slots in which bars are painted within the chart canvas area. This covers the visible portion of the chart only, and does not include historical painted bars outside of the visible area.

Property Value

An `int` representing the number of index slots in which bars are painted

Syntax

`<ChartControl>.SlotsPainted`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    int painted = chartControl.SlotsPainted;

    // Print the number of bars painted on the visible chart canvas
    Print(painted);
}
```

In the image below, SlotsPainted reveals that there are 17 bars painted on the chart canvas.

11.6.2.6.2.35 Strategies

Definition

A collection of strategies configured on the chart.

Property Value

A `ChartObjectCollection` of `StrategyRenderBase` objects containing information on all configured strategies on the chart.

Syntax

`<ChartControl>.Strategies`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Print the number of strategies configured on the chart
    if (chartControl.Strategies.Count > 0)
        Print(chartControl.Strategies[0].Name);
}
```

11.6.2.6.2.36 TimePainted

Definition
Indicates the range of time in which bars are painted on the visible chart canvas.

Property Value
A `TimeSpan` measuring the difference between the earliest and latest times at which bars are painted on the chart

Syntax
`<chartControl>.TimePainted`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // Print a message if less than three hours' worth of data is
painted on the chart canvas
    if(chartControl.TimePainted.Hours < 3)
        Print(String.Format("It is recommended to view at least
three hours worth of data on your chart with this indicator. You
are currently viewing {0}", chartControl.TimePainted));
}
```

Note: TimePainted is intended to be used when Non-Equidistant (time-based) bar spacing is enabled on the chart. Otherwise, it will have a value of 0.

11.6.2.6.3 ChartingExtensions

The **ChartingExtensions** class provides helper methods useful for converting a pixel coordinate from application-specific pixels (i.e., WPF coordinates) to Device Independent Pixels.

Note: More information about the differences between application pixels and **device** pixels can be found on the Working with Pixel Coordinates page.

ChartingExtensions Helper Methods

ConvertFromHorizontalPixels	Converts a horizontal coordinate (x) from device pixels to application pixels

ConvertFromVerticalPixels	Converts a vertical coordinate (y) from device pixels to application pixels
ConvertToHorizontalPixels	Converts a horizontal coordinate (x) in application pixels to device pixels
ConvertToVerticalPixels	Converts a vertical coordinate (y) in application pixels to device pixels

11.6.2.6.3.1 ConvertFromHorizontalPixels

Definition

Converts an x-axis pixel coordinate from device pixels to application pixels.

> **Note**: For more information concerning the differences between **application pixels** and **device pixels**, please see the Working with Pixel Coordinates educational resource.

Method Return Value

A `double` representing an x-coordinate value in terms of application pixels

Syntax

```
ChartingExtensions.ConvertFromHorizontalPixels(this int x, PresentationSource target)
<int>.ConvertFromHorizontalPixels(PresentationSource target)
```

Parameters

x	The horizontal `int` coordinates in device pixels to convert
`target`	The PresenationSource representing the display surface used for the conversion **Note**: For Charts, see ChartControl.PresentationSource

Example

```
int applicationPixelX;

protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // Obtain the application-pixel coordinate corresponding to a
device-pixel X value of 500
   applicationPixelX =
ChartingExtensions.ConvertFromHorizontalPixels(500,
ChartControl.PresentationSource);
}
```

11.6.2.6.3.2 ConvertFromVerticalPixels

Definition

Converts a y-axis pixel coordinate from device pixels to application pixels.

> **Note**: For more information concerning the differences between **application pixels** and **device pixels**, please see the Working with Pixel Coordinates educational resource.

Method Return Value

A `double` representing a y-coordinate value in terms of application pixels

Syntax

```
ChartingExtensions.ConvertFromVerticalPixels(this int x, PresentationSource target)
<int>.ConvertFromVerticalPixels(PresentationSource target)
```

x	The vertical `int` coordinates in device pixels to convert
`target`	The PresenationSource representing the display surface used for the conversion **Note**: For Charts, see ChartControl.PresentationSource

Example

```
int applicationPixelY;

protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // Obtain the application-pixel coordinate corresponding to a
device-pixel Y value of 500
   applicationPixelY =
ChartingExtensions.ConvertFromVerticalPixels(500,
ChartControl.PresentationSource);
}
```

11.6.2.6.3.3 ConvertToHorizontalPixels

Definition

Converts an x-axis pixel coordinate from application pixels to device pixels.

> **Note**: For more information concerning the differences between **application pixels** and **device pixels**, please see the Working with Pixel Coordinates educational resource.

Method Return Value

An `int` representing an x-coordinate value in terms of device pixels

Syntax

```
ChartingExtensions.ConvertToHorizontalPixels(this double x, PresentationSource target)
<double>.ConvertToHorizontalPixels(PresentationSource target)
```

x	The horizontal `double` coordinates in application pixels to convert
`target`	The PresenationSource representing the display surface used for the conversion **Note**: For Charts, see ChartControl.PresentationSource

Example

```
int devicePixelX;

protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // Obtain the device-pixel coordinate corresponding to an
application pixel X-value of 500
   devicePixelX = ChartingExtensions.ConvertToHorizontalPixels(500,
 ChartControl.PresentationSource);
}
```

11.6.2.6.3.4 ConvertToVerticalPixels

Definition

Converts a y-axis pixel coordinate from application pixels to device pixels.

> **Note**: For more information concerning the differences between **application pixels** and **device pixels**, please see the Working with Pixel Coordinates educational resource.

Method Return Value

An `int` representing a y-coordinate value in terms of device pixels

Syntax

```
ChartingExtensions.ConvertToVerticalPixels(this double x, PresentationSource target)
<double>.ConvertToVerticalPixels(PresentationSource target)
```

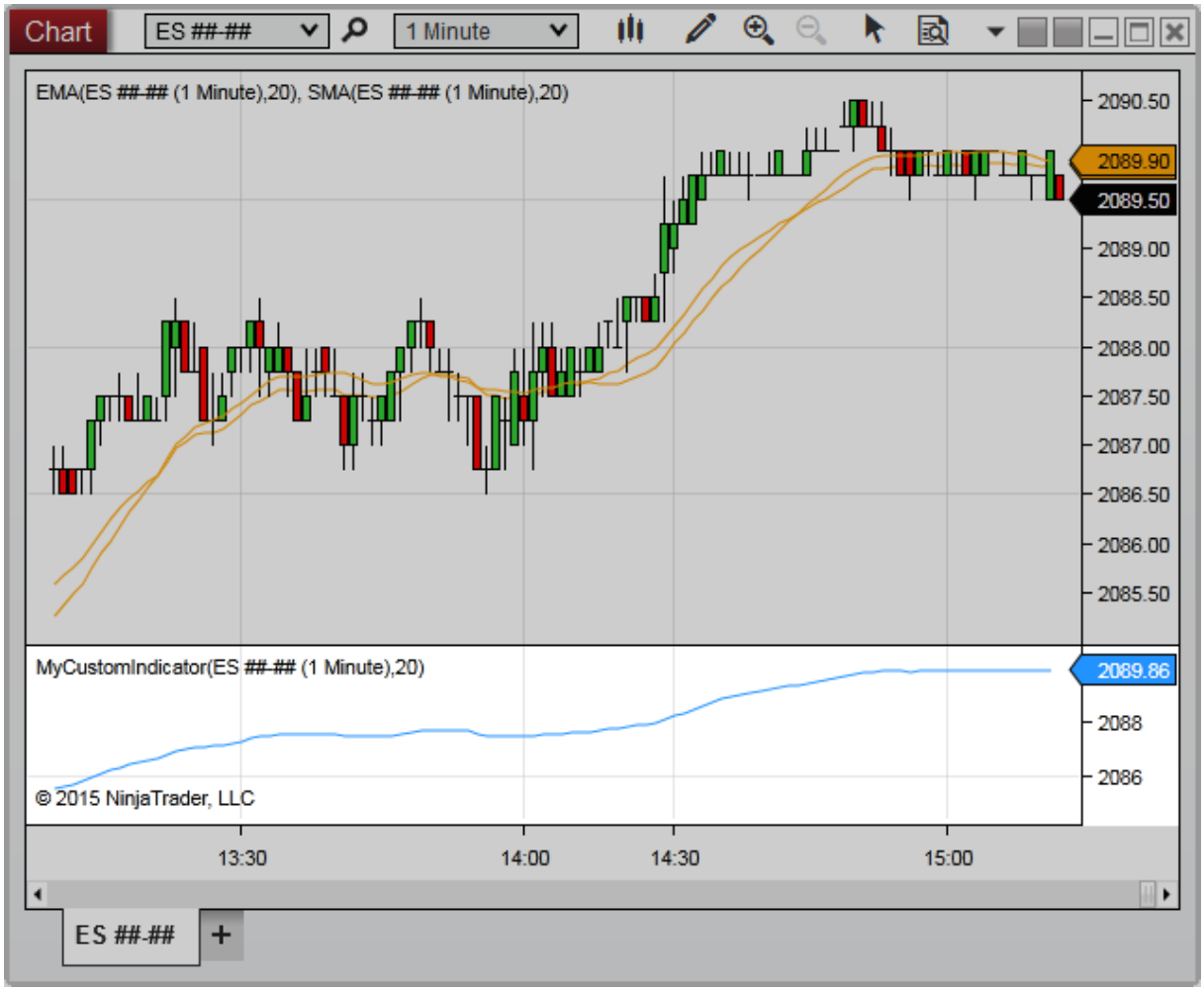
x	The vertical `double` coordinates in application pixels to convert
`target`	The PresenationSource representing the display surface used for the conversion **Note**: For Charts, see ChartControl.PresentationSource

Example

```
int devicePixelY;

protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // Obtain the device-pixel coordinate corresponding to an
application-pixel Y value of 500
   devicePixelY = ChartingExtensions.ConvertToVerticalPixels(500,
ChartControl.PresentationSource);
}
```

11.6.2.6.4 ChartPanel

The ChartPanel class includes a range of properties related to the panel on which the calling script resides. Each Panel has 3 independent ChartScales: Left, Right, and Overlay.



Methods and Properties

ChartObjects	A collection of objects configured on the chart panel
H	Indicates the height (in pixels) of the chart panel
IsFocused	Indicates the chart panel is currently in focus in the window
IsWaitingForBars	Indicates one or more objects in the chart panel are waiting for Bars objects to load or refresh
IsYAxisDisplayedLeft	Indicates the y-axis is visible on the left side of the chart panel
IsYAxisDisplayedOverlay	Indicates any objects configured in the panel are using the Overlay scale justification
IsYAxisDisplayedRight	Indicates the y-axis is visible on the right side of the chart panel
MaxValue	Indicates the maximum Y value of objects within the chart panel
MinValue	Indicates the minimum Y value of objects within the chart panel
PanelIndex	Indicates the index of the chart panel in the collection of configured panels
Scales	A collection of ChartScale objects corresponding to objects within the chart panel
W	Indicates the width (in pixels) of the chart panel
X	Indicates the x-coordinate on the chart canvas at which the chart panel begins
Y	Indicates the y-coordinate on the chart canvas at which the chart panel begins

11.6.2.6.4.1 ChartObjects

Definition

A collection of objects configured on the chart panel

Property Value

An IList of `Gui.NinjaScript.IChartObject` instances containing references to the objects configured on the panel

Syntax

`ChartPanel.ChartObjects`

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);

   IList<Gui.NinjaScript.IChartObject> myObjects =
ChartPanel.ChartObjects;

   foreach (Gui.NinjaScript.IChartObject thisObject in myObjects)
   {
       Print(String.Format("{0} is of type {1}", thisObject.Name,
thisObject.GetType()));
   }
}
```

The image below shows the output of the code example above, while applied in a chart panel with three objects.

11.6.2.6.4.2 H (Height)

Definition

Indicates the height (in pixels) of the rendered area of the chart panel.

> **Note**: The paintable area does not extend all the way to the top edge of the panel itself, as seen in the image below.

Property Value

A `int` representing the height of the panel in pixels

Syntax

```
ChartPanel.H
```

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    base.OnRender(chartControl, chartScale);

    // Print the height of the panel
    Print(ChartPanel.H);
}
```

Based on the image below, H reveals that the paintable area of the chart panel is 69 pixels high.

11.6.2.6.4.3 IsYAxisDisplayedLeft

Definition

Indicates the y-axis is visible on the left side of the chart panel.

Property Value

A `bool` indicating the y-axis is visible to the left

Syntax

ChartPanel.IsYAxisDisplayedLeft

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);

    // Print a message if the y-axis is visible on the left
   if (ChartPanel.IsYAxisDisplayedLeft)
       Print("The y-axis is visible on the left");
}
```

Based on the image below, IsYAxisDisplayedLeft confirms that the y-axis displays to the left. In this image, the property would be set to `true` when applied to either chart panel.

11.6.2.6.4.4 IsYAxisDisplayedOverlay

Definition
Indicates any objects configured in the panel are using the Overlay scale justification.

Property Value
A `bool` indicating any objects use the Overlay scale justification

Syntax
`ChartPanel.IsYAxisDisplayedOverlay`

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);

    // Trigger an alert when the Overlay scale justification is
used
   if (ChartPanel.IsYAxisDisplayedOverlay)
       Alert("overlayAlert", Priority.Low, "It is not recommended
to use 'Overlay' with this indicator", "", 300, Brushes.Yellow,
Brushes.Black);
}
```

Based on the image below, IsYAxisDisplayedOverlay is set to True, since the SMA indicator is using the Overlay scale justification.

11.6.2.6.4.5 IsYAxisDisplayedRight

Definition

Indicates the y-axis is visible on the right side of the chart panel.

Property Value

A `bool` indicating the y-axis is visible to the right

Syntax

```
ChartPanel.IsYAxisDisplayedRight
```

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);

    // Print a message if the y-axis is visible on the right
   if (ChartPanel.IsYAxisDisplayedRight)
       Print("The y-axis is visible on the right");
}
```

Based on the image below, IsYAxisDisplayedRight confirms that the y-axis is not displayed on the right. The property would be set to false when applied in either chart panel in this instance.

11.6.2.6.4.6 MaxValue

Definition

Indicates the maximum Y value of objects within the chart panel, based on the current y-axis scale. The scale of the y-axis is dependent upon the values of objects in the panel which have Auto Scale enabled.

Property Value
A `double` representing the maximum Y value in the panel's vertical scale

Syntax
`ChartPanel.MaxValue`

Example


```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    base.OnRender(chartControl, chartScale);

    // Print the minimum and maximum Y values for objects in the
panel
    Print(String.Format("Min value: {0}, Max value:
{1}",ChartPanel.MinValue, ChartPanel.MaxValue));
}
```

11.6.2.6.4.7 MinValue

Definition
Indicates the minimum Y value of objects within the chart panel, based on the current y-axis scale. The scale of the y-axis is dependent upon the values of objects in the panel which have Auto Scale enabled.

Property Value
A `double` representing the minimum Y value in the panel's vertical scale

Syntax
`ChartPanel.MinValue`

Example


```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    base.OnRender(chartControl, chartScale);

    // Print the minimum and maximum Y values for objects in the
panel
    Print(String.Format("Min value: {0}, Max value:
{1}",ChartPanel.MinValue, ChartPanel.MaxValue));
}
```

11.6.2.6.4.8 PanelIndex

Definition

Indicates the index of the chart panel in the collection of configured panels.

> **Note**: This property comes from a zero-based index, which is not the same as the panel number displayed in the Indicators window opened from within the chart. The panel number displayed in the Indicators window will equate to `ChartPanel.PanelIndex + 1`.

Property Value

A `int` representing the zero-based index of the panel

Syntax

```
ChartPanel.PanelIndex
```

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   base.OnRender(chartControl, chartScale);

   // Print the panel's zero-based index
   Print(String.Format("This panel sits at index
{0}",ChartPanel.PanelIndex));
}
```

Notice three things in the image below:

1) An indicator containing the example code above is configured on the second chart panel
2) In the Indicators window, the "Panel" property is set to 2
3) The output of the example code displays the zero-based index of Panel #2, which is at index 1

11.6.2.6.4.9 Scales

Definition

A collection of ChartScale objects corresponding to objects within the chart panel.

Property Value

A `ChartScaleCollection` containing `ChartScale` objects

Syntax

`ChartPanel.Scales`

Example

```
protected override void OnStateChange()
{
   if (State == State.Historical)
   {
      // loop through each panel which is currently configured on
the hosting chart
      foreach (ChartPanel chartPanel in ChartControl.ChartPanels)
      {
         // there are multiple scale per panel
         // i.e., Right, Left, Overlay
         foreach (ChartScale scale in chartPanel.Scales)
         {
            // get the right scale margin type
            if (scale.ScaleJustification ==
ScaleJustification.Right)
            {
               Print(string.Format("The Right Scale of panel #{0}'s
margin type is {1}",
                                   scale.PanelIndex,
scale.Properties.AutoScaleMarginType));
            }
         }
      }
   }
}
```

```
protected override void OnStateChange()
{
    if (State == State.Historical)
    {
        // Shows us at which index in the Scales collection the
individual panel scales reside [0: Right, 1: Left, 2: Overlay]
        // The Scale collection gets accessed via passing the
ScaleJustification enum in as index
        Print("Scales index " + 0 + " " +
ChartPanel.Scales[ScaleJustification.Right]);
        Print("Scales index " + 1 + " " +
ChartPanel.Scales[ScaleJustification.Left]);
        Print("Scales index " + 2 + " " +
ChartPanel.Scales[ScaleJustification.Overlay]);
    }
}
```

11.6.2.6.4.10 W (Width)

Definition

Indicates the width (in pixels) of the paintable area of the chart panel.

> **Note**: The paintable area does not extend all the way to the right edge of the panel itself, as seen in the image below.

Property Value

A `int` representing the width of the panel in pixels

Syntax

```
ChartPanel.W
```

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    base.OnRender(chartControl, chartScale);

    // Print the width of the panel
    Print(ChartPanel.W);
}
```

Based on the image below, W reveals that the chart panel is 451 pixels wide.

11.6.2.6.4.11 X (Coordinate)

Definition

Indicates the x-coordinate on the chart canvas at which the chart panel begins.

Property Value

A `int` representing the x-coordinate at which the panel begins. This property will only contain a value greater than zero if the y-axis displays to the left of the paintable chart canvas area in the panel (if an object in the panel is using the "Left" scale justification).

Syntax

```
ChartPanel.X
```

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    base.OnRender(chartControl, chartScale);

    // Print the coordinates of the top-left corner of the panel
    Print(String.Format("The panel begins at coordinates {0},
{1}",ChartPanel.X ,ChartPanel.Y));
}
```

Based on the image below, X reveals that the chart panel begins at x-coordinate 52.

11.6.2.6.4.12 Y (Coordinate)

Definition

Indicates the y-coordinate on the chart canvas at which the chart panel begins.

Property Value

A `int` representing the y-coordinate at which the panel begins.

Syntax

ChartPanel.Y

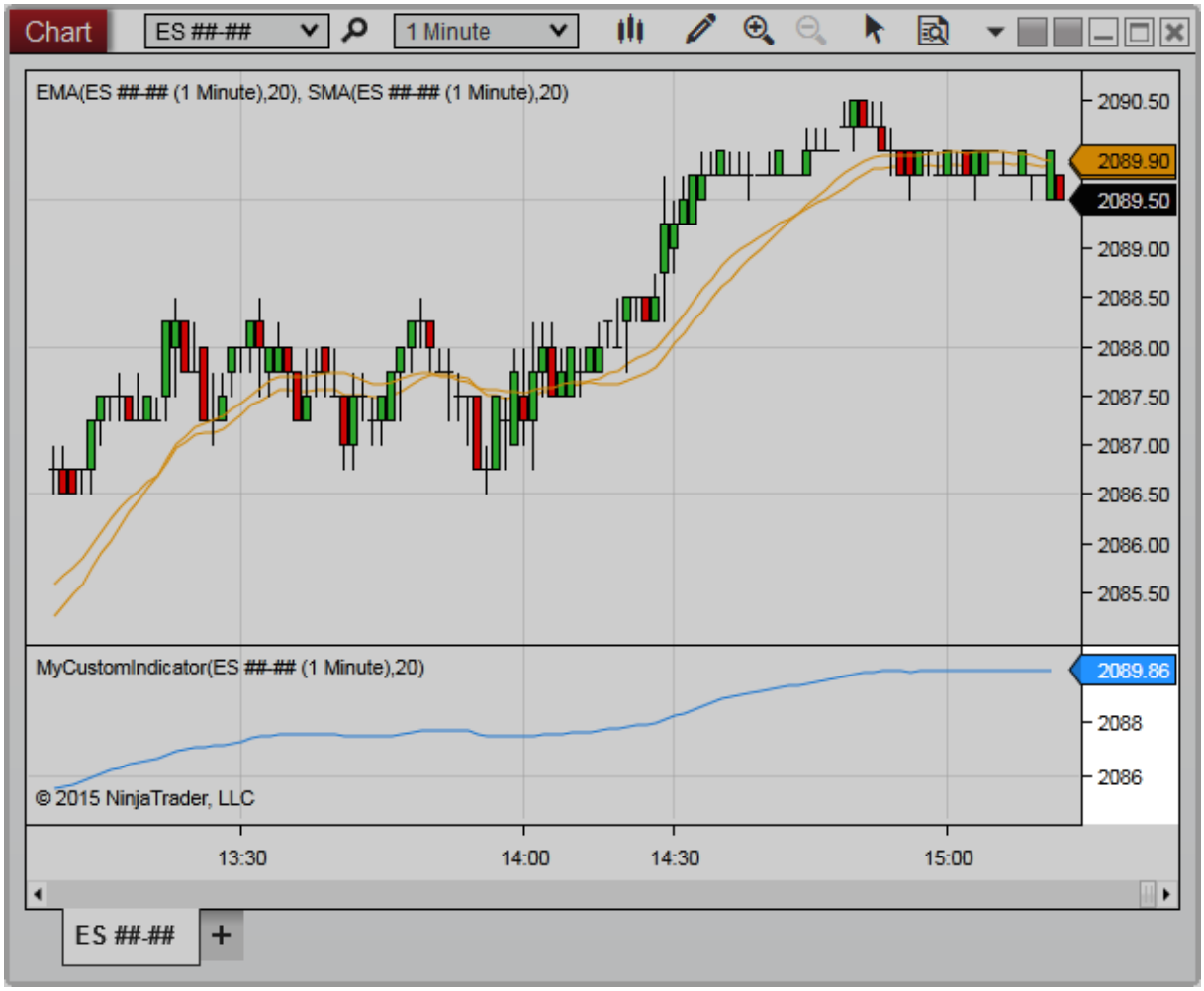
Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    base.OnRender(chartControl, chartScale);

    // Print the coordinates of the top-left corner of the panel
    Print(String.Format("The panel begins at coordinates {0},
{1}",ChartPanel.X ,ChartPanel.Y));
}
```

Based on the image below, Y reveals that the chart panel begins at y-coordinate 232.

11.6.2.6.5 ChartScale

The ChartScale class includes a range of properties related to the Y-Axis values of the ChartPanel on which the calling script resides. The ChartScale can be configured to Right, Left, or Overlay.



Methods and Properties

GetPixelsForDistance()	Returns the number of device pixels between the value passed to the method representing a series point value on the chart scale
GetValueByY()	Returns the series value on the chart scale determined by a y pixel coordinate on the chart
GetValueByYWpf()	Returns the series value on the chart scale determined by a WPF coordinate on the chart

GetYByValue()	Returns the chart's y-pixel coordinate on the chart determined by a series value represented on the chart scale
GetYByValueWpf()	Returns a WPF coordinate on the chart determined by a series value represented on the chart scale
Height	Indicates the overall distance (from top to bottom) of the chart scale in device pixels
IsVisible	Indicates if the chart scale is viewable on the UI
MaxMinusMin	The difference between the chart scale's MaxValue and MinValue represented as a y value
MaxValue	The highest displayed value on the chart scale
MinValue	The lowest rendered value on the chart scale
PanelIndex	The panel on which the chart scale resides
Properties	Represents a number of properties available to the Chart Scale which can be configured to change the appearance of the scale
ScaleJustification	Indicates the location of the chart scale relative to the chart control
Width	Indicates the overall distance (from left to right) of the chart scale in device pixels

11.6.2.6.5.1 GetPixelsForDistance()

Definition

Returns the number of device pixels between the value passed to the method representing a series point value on the chart scale.

Method Return Value

A `float` representing the number of pixels between a value.

Syntax

```
<chartScale>.GetPixelsForDistance(double distance)
```

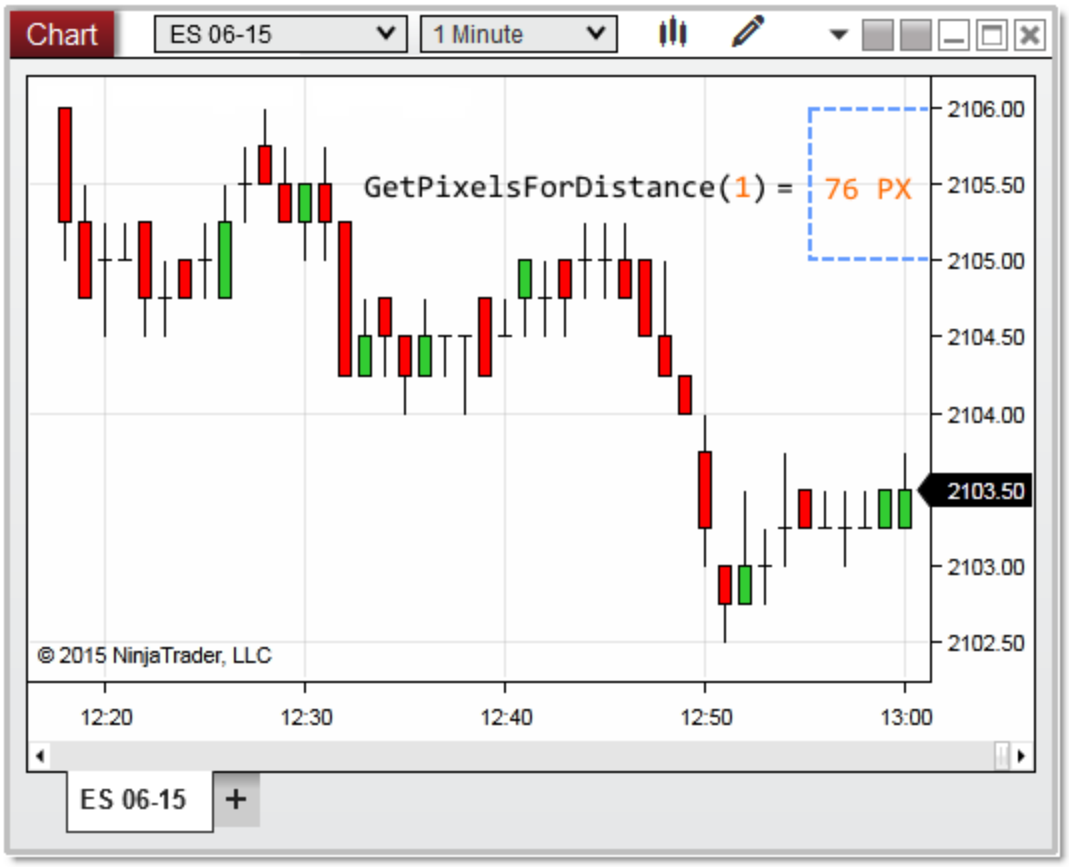
Method Parameters

distance	A `double` value representing the distance in points to be measured

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // the number of pixels between the point value passed as a
distance to the method
   float   pixelForDistance =
chartScale.GetPixelsForDistance(0.25);

   Print("pixelForDistance: " + pixelForDistance);  //20 pixels per
every 1 tick on the chart scale
}
```

In the image below, we pass a value of 1 for the distance, which tells us there are 76 pixels for every 1 point on the ES 06-15 chart scale.



11.6.2.6.5.2 GetValueByY()

Definition

Returns the series value on the chart scale determined by a y pixel coordinate on the chart.

Method Return Value

A `double` value representing a series value on the chart scale. This is normally a price value, but can represent indicator plot values as well.

Syntax

```
<chartScale>.GetValueByY(float y)
```

Method Parameters

y	A `float` value representing a pixel coordinate on the chart scale

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // the price value of the pixel coordinate passed in the method
   double  valueByY =    chartScale.GetValueByY(1);

   Print("valueByY: " + valueByY);  //2106.19693333
}
```

In the image below, we pass a value of 1 for the y value, which tells us the pixel coordinate of 1 is located at a price of 2106.19 on the chart scale

11.6.2.6.5.3 GetValueByYWpf()

Definition

Returns the series value on the chart scale determined by a WPF coordinate on the chart.

Method Return Value

A `double` value representing a series value on the chart scale. This is normally a price value, but can represent indicator plot values as well.

Syntax

```
<chartScale>.GetValueByYWpf(double y)
```

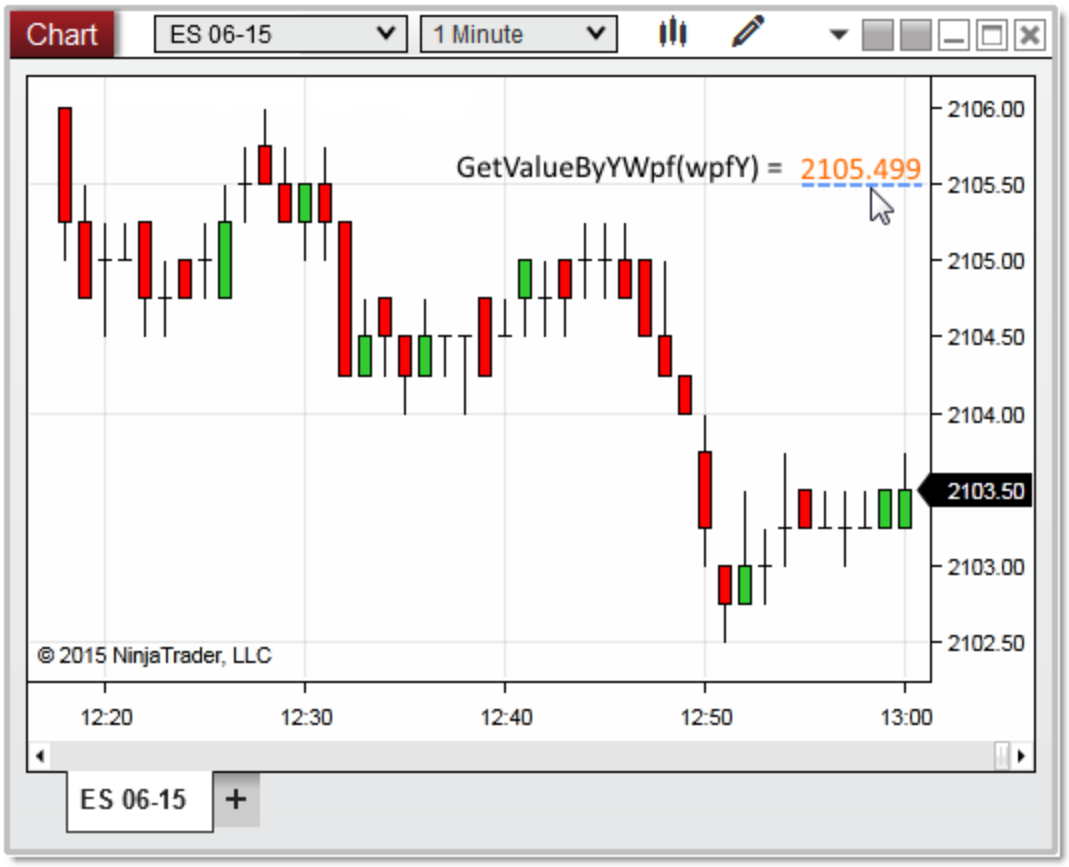
Method Parameters

y	A `double` value representing a WPF coordinate on the chart scale

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // store the y location the user clicked
   double    wpfY = chartControl.MouseDownPoint.Y;

   // gets price value of the WPF coordinate passed to the method
   double    valueByYWpf = chartScale.GetValueByYWpf(wpfY);

   Print("valueByYWpf: " + valueByYWpf);  //2105.49995215
}
```

In the image below, we used the Chart Control property MouseDownPoint as the "`wpfy`" variable, which in return tells us the user clicked on a Y value of 2105.499 on the chart scale.



11.6.2.6.5.4 GetYByValue()

Definition

Returns the chart's y-pixel coordinate on the chart determined by a series value represented on the chart scale.

Method Return Value

An `int` value representing a y pixel coordinate on the chart scale.

Syntax

```
<chartScale>.GetYByValue(double val)
```

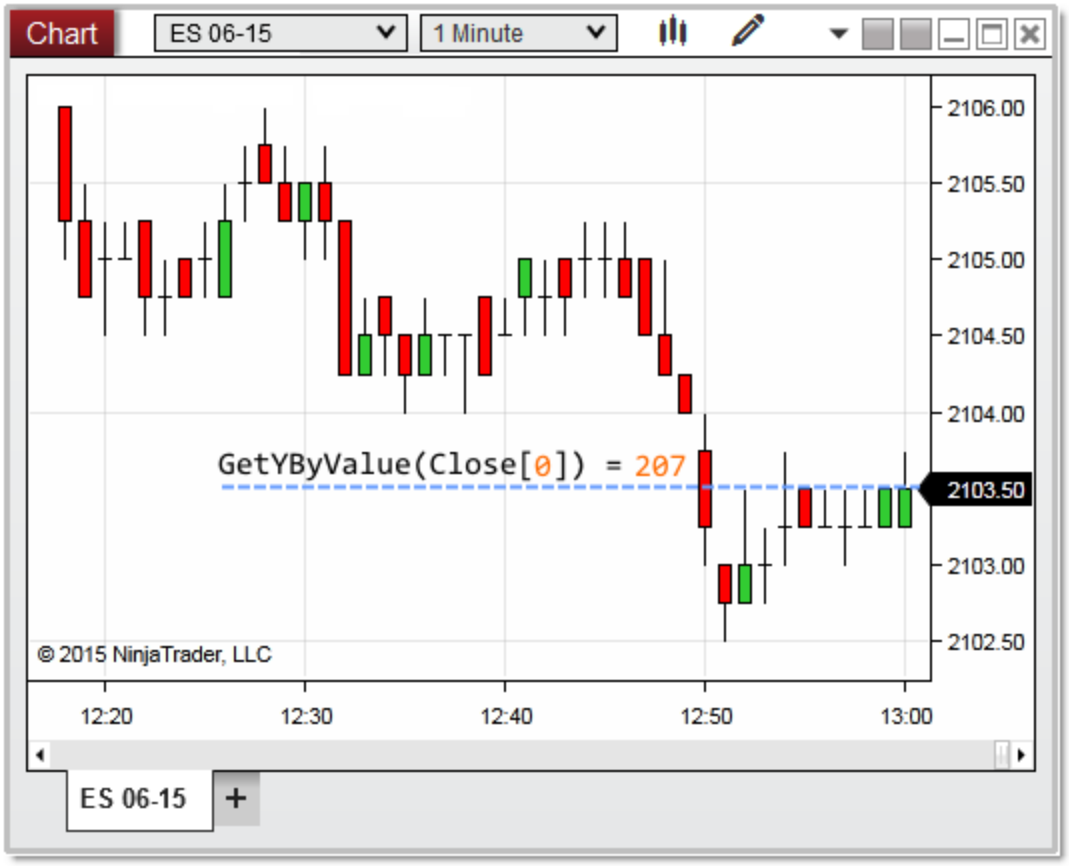
Method Parameters

val	A `double` value which usually represents a price or indicator value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // gets the pixel coordinate of the price value passed to the
method
   int      yByValue =
chartScale.GetYByValue(Bars.GetClose(Bars.Count - 1));

   Print("yByValue: " + yByValue); // 207
}
```

In the image below, we pass the last bar close as the value (example logic avoids using a bars ago index, see also OnRender() note #5), which in return tells us the last price displayed on the chart is at a y location of 207 pixels.



11.6.2.6.5.5 GetYByValueWpf()

Definition

Returns a WPF coordinate on the chart determined by a series value represented on the chart scale.

Method Return Value

An `double` value representing a WPF coordinate on the chart scale

Syntax

```
<chartScale>.GetYByValueWpf(double val)
```

Method Parameters

val	A `double` value which usually represents a price or indicator value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // gets the wpf coordinate of the price value passed to the
method
   int      valueByYWpf =
chartScale.GetYByValueWpf(Bars.GetClose(Bars.Count - 1));

   Print("valueByYWpf: " + valueByYWpf); // 207
}
```

In the image below, we pass the last bar close as the value (example logic avoids using a bars ago index, see also OnRender() note #5), which in return tells us the last price displayed on the chart is at a WPF location of 207.30998 pixels.

11.6.2.6.5.6 Height

Definition

Indicates the overall distance (from top to bottom) of the chart scale.

> **Note**: Height does not return its value in terms of device pixels. However, using Height.ConvertToVerticalPixels or Height.ConvertToHorizontalPixels will convert the Height value to device pixels. Alternatively, RenderTarget.PixelSize.Height or ChartPanel.H will also provide the height in terms of device pixels.

Property Value

A `double` value representing the height of the chart scale.

Syntax

```
<chartScale>.Height
```

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // the height of the entire chart scale
   double   height       = chartScale.Height;
   Print("the height of the chart scale is: " + height);
}
```

In the image below, the entire of height of the chart scale is represented by the blue line which is calculated at 300 pixels.

11.6.2.6.5.7 IsVisible

Definition

Indicates if the chart scale is viewable on the UI. If the bar series, indicator, or strategy which uses the chart scale is not in view, the chart scale IsVisible property will return false.

Property Value

A bool value, which when **true** the series used to build the scale is viewable; otherwise **false**. This property is read-only.

Syntax
```
<chartScale>.IsVisible
```

Examples
```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // do not process render info chart scale is not visible
   if(!chartScale.IsVisible)
      return;
}
```

11.6.2.6.5.8 MaxMinusMin

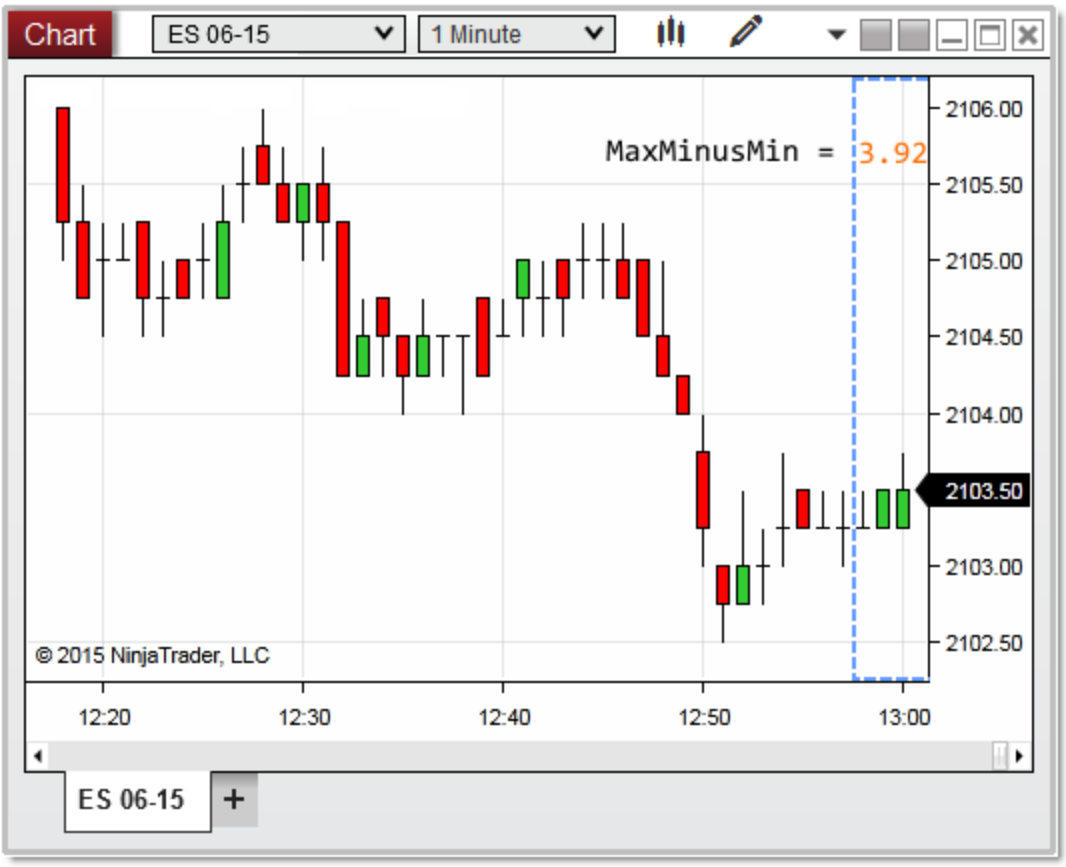
Definition
The difference between the chart scale's MaxValue and MinValue represented as a y value.

Property Value
A double value representing the difference in scale as a y value.

Syntax
```
<chartScale>.MaxMinusMin
```

Examples
```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // the difference between the scales maximum and minimum value
   double   maxMinusMin = chartScale.MaxMinusMin;

   Print("maxMinusMin: " + maxMinusMin);  // maxMinusMin: 3.92
}
```

In the image below, the highest calculated value on the chart scale is 2106.21, with the lowest value being 2102.29; the MaxMinusMin property therefore provides us calculated value of 3.92.



11.6.2.6.5.9 MaxValue

Definition
The highest displayed value on the chart scale.

Property Value
A `double` value representing highest value on the chart scale as a y value.

Syntax
`<chartScale>.MaxValue`

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // the maximum value of the chart scale
   double  maxValue   = chartScale.MaxValue;

   Print("maxValue: " + maxValue);
}
```

In the image below, the highest value displayed as text on the y-axis reads 2106.00, however as you can see, there are a few pixels on the chart scale above this tick. The absolute rendered MaxValue on the chart scale is calculated as 2106.21

11.6.2.6.5.10 MinValue

Definition

The lowest rendered value on the chart scale.

Property Value

A `double` value representing lowest value on the chart scale as a y value.

Syntax

```
<chartScale>.MinValue
```

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // the minimum value of the chart scale
   double  minValue   = chartScale.MinValue;

   Print("minValue: " + minValue);
}
```

In the image below, the lowest value displayed as text on the y-axis reads 2102.50, however as you can see, there are a few pixels on the chart scale below this tick. The absolute rendered MinValue on the chart scale is calculated as 2102.29.

11.6.2.6.5.11 PanelIndex

Definition

The panel on which the chart scale resides.

Note: This value is **NOT** the same value as the indicator's PanelUI. **PanelIndex** will provide the actual indexed value of the chart panel used for this chart scale.

Property Value

An `int` value representing the panel as an index value which starts at 0 and will increment for each panel configured on the chart. This property is read-only.

Syntax

`<chartScale>.PanelIndex`

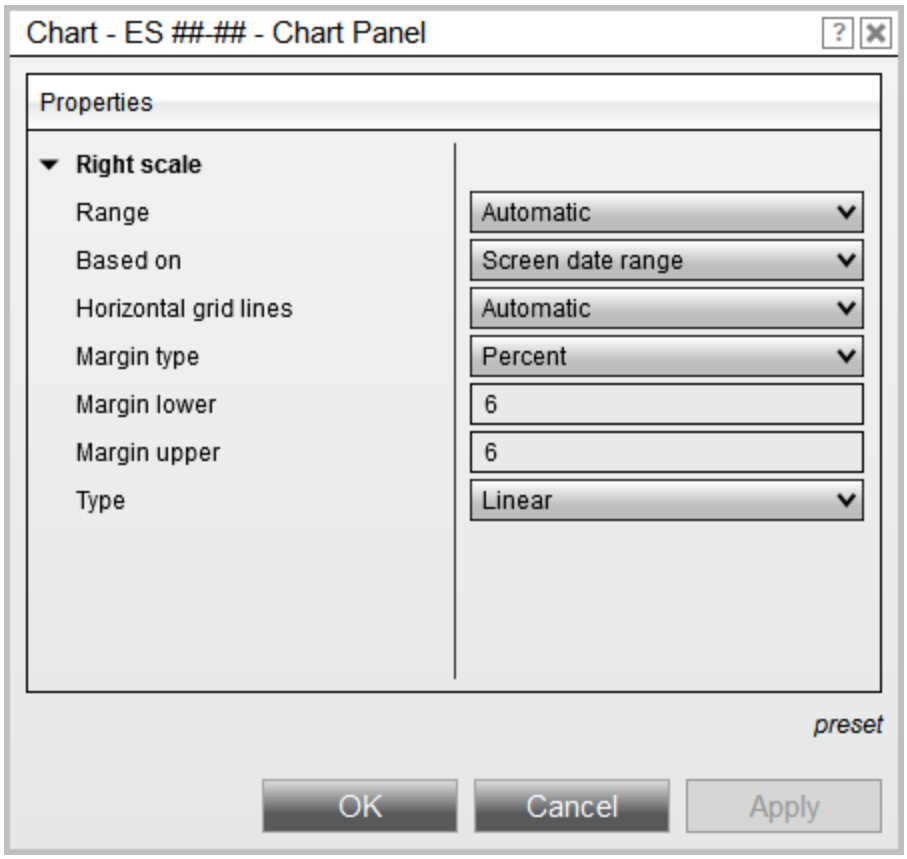
Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // the index value of the panel (not the same as the panelUI)
   int      panel      = chartScale.PanelIndex;
   Print("panel: " + panel);
}
```

11.6.2.6.5.12 Properties

Definition

Represents a number of properties available to the Chart Scale which can be configured to change the appearance of the scale.



Warning: These are UI properties which are designed to be set by a user. Attempting to modify these values through a custom script is **NOT** guaranteed to take effect.

Property Values

YAxisRangeType	An `YAxisRangeType` `enum`, possible values are: • Automatic • Fixed
AutoScaleDateRangeType	An `AutoScaleDateRangeType` `enum`, possible values are: • ScreenDateRange • EntireDateRangeSeriesOnly
HorizontalGridlinesCalculation	An `YAxisRangeType` `enum`, possible values are:

	• Automatic • Fixed
HorizontalGridlinesIntervalType	A `HorizontalGridlinesIntervalType enum`, possible values are: • Ticks • Points • Pips
HorizontalGridlinesInterval	A `double value` representing the vertical interval of the horizontal axis
AutoScaleMarginType	An `AutoScaleMarginType enum`, possible values are: • Percent • Price
AutoScaleMarginLower	A `double` value representing the lowest margin used for the chart scale
AutoScaleMarginUpper	A `double` value representing the highest margin used for the chart scale
YAxisScalingType	An `YAxisScalingType enum`, possible values are: • Linear • Logarithmic
FixedScaleMax	A `double` representing the highest series value used for the chart scale when the scale is fixed
FixedScaleMin	A `double` representing the lowest series value used for the chart scale when the scale is fixed

Syntax

```
<chartScale>.Properties
```

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   if (chartScale.Properties.YAxisScalingType ==
YAxisScalingType.Linear)
   {
      // do something
   }
}
```

11.6.2.6.5.13 ScaleJustification

Definition

Indicates the location of the chart scale relative to the chart control.

Property Value

A ScaleJustification `enum`. Possible values are:

- Right
- Left
- Overlay

Syntax

`ScaleJustification`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   if (chartScale.ScaleJustification == ScaleJustification.Right)
   {
      // do something
   }
}
```

11.6.2.6.5.14 Width

Definition

Indicates the overall distance (from left to right) of the chart scale.

Note: Width does not return its value in terms of device pixels. However, using

Width.ConvertToVerticalPixels or Width.ConvertToHorizontalPixels will convert the Width value to device pixels. Alternatively, RenderTarget.PixelSize.Width or ChartPanel.W will also provide the width in terms of device pixels.

Property Value

A `double` value representing the width of the chart scale.

Syntax

```
<chartScale>.Width
```

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // the width of the entire chart scale
   double   width       = chartScale.Width;
   Print("the width of the chart scale is: " + Width);
}
```

In the image below, the entire of width of the chart scale is represented by the blue line which is calculated at 450 pixels.

11.6.2.6.6 Rendering

Rendering methods and properties can be useful when carrying out custom drawing tasks for chart objects. Event handlers such as OnCalculateMinMax() and OnRender() allow you to override behavior at key points in the rendering process.

Notse:
1. Some rendering methods and properties make use of SharpDX libraries, which provide a managed framework for working with DirectX technology. Please see the SharpDX SDK Reference for more information.
2. For a walk through for using the **SharpDX**, please see the educational resource Using SharpDX for Custom Chart Rendering

Methods and Properties

RenderTarget	Creates objects and exposes methods used for drawing in the chart area.

ForceRefresh()	Forces OnRender() to be called, which will re-paint the chart
IsInHitTest	Qualifies if object drawn in chart object should be selectable in the hit test procedure
IsSelected	Indicates a chart object is currently selected
IsVisibleOnChart()	Indicates a chart object is visible on the chart canvas
MaxValue	The maximum value used for the automatic scaling of the y axis
MinValue	The minimum value used for the automatic scaling of the y axis
OnCalculateMinMax()	An event driven method which is called while the chart scale is being updated
OnRender()	Used to render custom drawing to a chart from various chart objects
OnRenderTargetChanged()	Used for efficient handling of SharpDX resources
PanelUI	The chart panel that is configured on the chart's UI
ZOrder	A unique identifier used to control the order in which chart objects are drawn on the chart's Z-axis

11.6.2.6.6.1 D2DFactory

Definition

Provides a default **Direct2D1** factory used for creating SharpDX.Direct2D1 components.

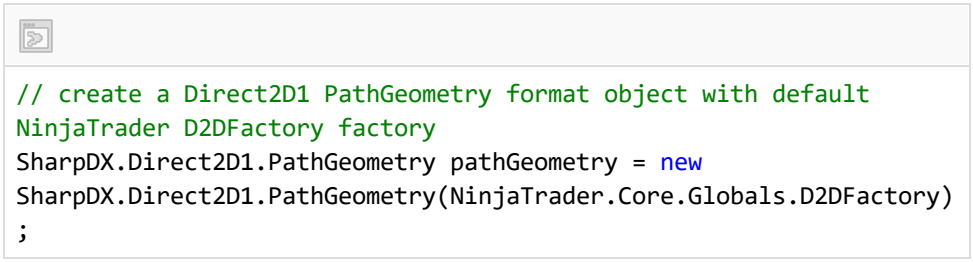
Property Value

A read-only **SharpDX.Direct2D1.Factory** to create **Direct2D1** objects compatible with NinjaTrader rendering

Syntax

```
NinjaTrader.Core.Globals.D2DFactory
```

Warning: Please ensure this property would only be accessed from OnRender() or OnRenderTargetChanged() (which run in the UI thread), as access from other threads outside those methods could cause a degradation in performance.



```
// create a Direct2D1 PathGeometry format object with default
NinjaTrader D2DFactory factory
SharpDX.Direct2D1.PathGeometry pathGeometry = new
SharpDX.Direct2D1.PathGeometry(NinjaTrader.Core.Globals.D2DFactory)
;
```

11.6.2.6.6.2 DirectWriteFactory

Definition

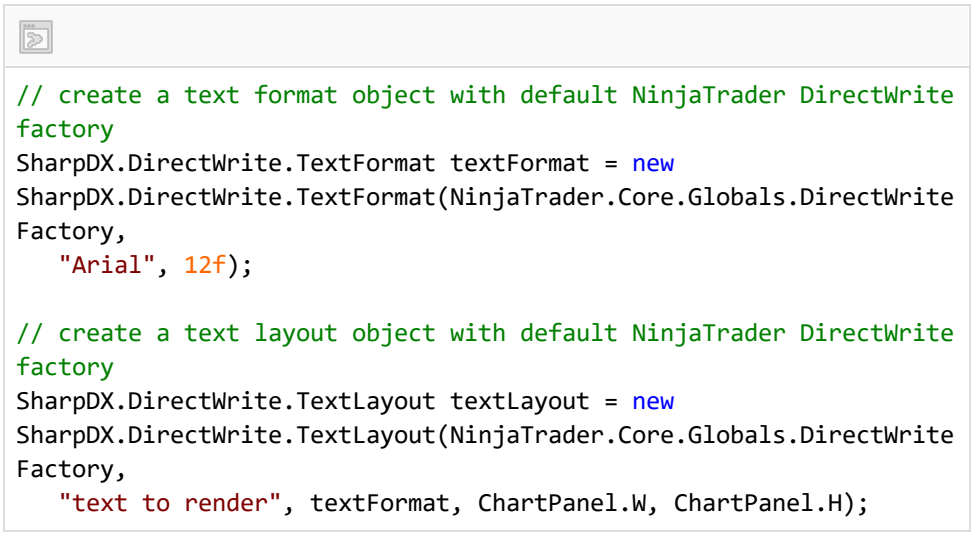
Provides an default **DirectWrite** factory used for creating SharpDX.DirectWrite components.

Property Value

A read-only **SharpDX.DirectWrite.Factory** used to create **DirectWrite** objects compatible with NinjaTrader rendering

Syntax

```
NinjaTrader.Core.Globals.DirectWriteFactory
```



```
// create a text format object with default NinjaTrader DirectWrite
factory
SharpDX.DirectWrite.TextFormat textFormat = new
SharpDX.DirectWrite.TextFormat(NinjaTrader.Core.Globals.DirectWrite
Factory,
    "Arial", 12f);

// create a text layout object with default NinjaTrader DirectWrite
factory
SharpDX.DirectWrite.TextLayout textLayout = new
SharpDX.DirectWrite.TextLayout(NinjaTrader.Core.Globals.DirectWrite
Factory,
    "text to render", textFormat, ChartPanel.W, ChartPanel.H);
```

11.6.2.6.6.3 DxExtensions

The **DxExtensions** class provides helper methods useful for converting **WPF** resources to **SharpDX** resources

Note: For more information on SharpDX Resources, please see the educational resource Using SharpDX for Custom Chart Rendering

DxExtensions Helper Methods

ToDxBrush()	Converts a **WPF Brush** to a **SharpDX Brush**
ToVector2()	Converts a **System.Windows.Point** structure to a **SharpDX.Vector2**

Definition

Converts a **WPF Brush** to a **SharpDX Brush** used for SharpDX rendering. Supports **SolidColorBrush**, **LinearGradientBrush**, and **RadialGradientBrush** types.

Note: If you are using a large number of brushes, and are not tied to WPF resources, you should favor creating the **SharpDX Brush** directly since the **ToDxBrush()** method can lead to performance issues if called too frequently during a single render pass.

Method Return Value

A new SharpDX.Direct2D1.Brush constructed colors and brush properties of the WPF brush

Syntax

```
DxExtensions.ToDxBrush(this System.Windows.Media.Brush brush, RenderTarget
renderTarget)
<WPFBrush>.ToDxBrush(RenderTarget renderTarget)
```

Parameters

brush	The System.Windows.Media.Brush to convert
renderTarget	The RenderTarget associated with the brush resource

Example

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Example ToDXBrush";

      // pushes the WPF brush to the UI for user to configure
      TextBrush = System.Windows.Media.Brushes.DodgerBlue;
   }

}
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // convert user WPF selection to a DX brush
   SharpDX.Direct2D1.Brush dxBrush =
TextBrush.ToDxBrush(RenderTarget);

   using (dxBrush)
   {
      RenderTarget.FillRectangle(new RectangleF(ChartPanel.X,
ChartPanel.Y, ChartPanel.W, ChartPanel.H), dxBrush);
   }
}

// the WPF exposed to the UI which the user defines
[XmlIgnore]
public System.Windows.Media.Brush TextBrush { get; set; }

[Browsable(false)]
public string TextBrushSerialize
{
   get { return Serialize.BrushToString(TextBrush); }
   set { TextBrush = Serialize.StringToBrush(value); }
}
```

Definition

Converts a **System.Windows.Point** structure to a **SharpDX.Vector2** used for SharpDX rendering.

Method Return Value

A new SharpDX.Vector2 constructed with the point parameters X and Y values

Syntax

```
DxExtensions.ToVector2(this System.Windows.Point point)
<point>.ToVector2()
```

Parameters

point	The System.Windows.Point point to convert

Example

```
// gets the application/user WPF point and converts to a SharpDX
Vector
System.Windows.Point wpfPoint = ChartControl.MouseDownPoint;

SharpDX.Vector2 dxVector2 = wpfPoint.ToVector2();
```

11.6.2.6.6.4 ForceRefresh()

Definition

Resets an internal marker used to determine if the chart visuals need to re-render.

ChartControl runs a timed event every 250ms to determine the chart needs to be updated. If it does, the OnRender() method is called. Under normal circumstances, the marker used to call **OnRender()** will be reset after the following conditions:

- OnBarUpdate() event
- OnConnectionStatusUpdate() event
- User clicks on the chart
- Drawing object(s) have been removed from the chart
- Strategy enabled/disabled on chart
- ChartTrader enabled/disabled

In most cases, the conditions listed above should be satisfactory for rendering standard and custom chart objects; however for more advance programming concepts, there may be other situations you run into which would **NOT** force the chart to refresh (e.g., a user interacting with a custom control). In these special cases, you can use the **ForceRefresh()** method to re-queue the render event.

Note: As the chart is optimized on a timer, calling **ForceRefresh()** will **NOT** immediately trigger a render event. Calling **ForceRefesh()** simply re-queues the render event to trigger during the next timed event. In other words, it may take up to 250ms for the render event to function.

Method Return Value

This method does not return a value

Syntax
ForceRefresh()

> **Warning**: Excessive calls to **ForceRefresh()** and **OnRender()** can carry an impact on general application performance. You should only call **ForceRefresh()** if the chart truly needs to be visually updated. It is **NOT** recommended to invalidate the chart control directly as this could cause issues with threading which result in dead locks.

Method Parameters
This method does not accept any parameters

Examples

```
DateTime lastTimeCalled = DateTime.MinValue;

private void MyCustomMethod()
{
   // if it has been longer than one second since the last time
   // this method was called update the chart visually
   if (Core.Globals.Now.Subtract(lastTimeCalled).Seconds >= 1)
   {
      ForceRefresh();
      lastTimeCalled = Core.Globals.Now;
   }
}
```

11.6.2.6.6.5 IsInHitTest

Definition
Indicates a user is currently clicking in the chart panel in which the calling script resides.

> **Note**: In addition to the example below, IsInHitTest can also be tested directly on chart objects (for example, myHorizontalLine.IsInHitTest). In this case, the IsInHitTest property of a specific object will refer to the panel in which the calling script resides, even if the calling script resides in a different panel than the object itself.

Property Value
This property returns **true** to indicate that the chart panel in which the script resides is being clicked on; otherwise, **false**. Default set to **false**.

Syntax
IsInHitTest

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   if(IsInHitTest)
    {
        Print("user clicked on object");

        // do something
   }
}
```

11.6.2.6.6.6 IsSelected

Definition
Indicates a chart object is currently selected. When this property is set to true in a DrawingTool, the GetSelectionPoints() will be called.

Property Value
This property returns **true** to indicate that the chart object is selected; otherwise, **false**. Default set to **false**.

Warning: This property value is **ONLY** guaranteed to be settable by the object to which it belongs (e.g., from within a DrawingTool). Modifying its value from an external object (such as attempting to set a **DrawingTool.IsSelected** from an indicator) can result in the property automatically returning the value handled by its source. In other words, unless you are working with a chart object type directly (e.g., building a custom drawing tool), the **IsSelected** property should be considered read-only.

Syntax
IsSelected

Examples

Reading the IsSelected property from an indicator

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   foreach(DrawingTool drawTool in DrawObjects)
   {
      // only apply logic below to types of "Rectangle")
      if(drawTool.GetType().ToString().Contains("Rectangle"))
      {
         // safely cast as dynamic type at run-time
         dynamic myRect = drawTool;

         // Changes the brush to pink to indicating selected
         if(drawTool.IsSelected)
         {
            myRect.AreaBrush = Brushes.Pink;
         }
         // otherwise, set back to default value on next render
pass
         else myRect.AreaBrush =  Brushes.CornflowerBlue;

      }
   }
}
```

Explicitly setting the IsSelected property from a DrawingTool type

```
public override void OnMouseDown(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   if(DrawingState == DrawingState.Building)
   {
      if(dataPoint.IsEditing)
      {
         // do something
      }

      // when done editing anchor, set the state to normal and
unselect the drawing object
      else if(dataPoint.IsEditing)
      {
         DrawingState = DrawingState.Normal;
         IsSelected = false;
      }
   }
}
```

11.6.2.6.6.7 IsVisibleOnChart()

Definition

Indicates a chart object is visible on the chart. When the IsVisibleOnChart() method determines a chart object is not visible and returns false, the object will not be used in a render pass, will not be considered in a hit test, and will not be used for alerting. The base implementation is to always return true on all chart objects, however this behavior can be overridden for your custom object if desired.

Method Return Value

A virtual `bool` value which when **true**, the object will be rendered and can be interacted with by a user; otherwise **false**. Default value is **true**.

Syntax

You must override this method using the following syntax:

```
public override bool IsVisibleOnChart(ChartControl chartControl, ChartScale
chartScale, DateTime firstTimeOnChart, DateTime lastTimeOnChart)
{
   return true;
}
```

Method Parameters

chartControl	A ChartControl representing the x-axis
chartScale	A ChartScale representing the y-axis
firstTimeOnChart	A DateTime representing the first painted bar displayed on the chart
lastTimeOnChart	A DateTime representing the last painted bar displayed on the chart

Examples

```
public override bool IsVisibleOnChart(ChartControl chartControl,
ChartScale chartScale, DateTime firstTimeOnChart, DateTime
lastTimeOnChart)
{
   // check if any chart anchors are visible
   foreach (ChartAnchor anchor in Anchors)
   {
      if (anchor.Time >= firstTimeOnChart && anchor.Time <=
lastTimeOnChart)
         return true;
   }
   return false;  // otherwise the object should not be displayed

}
```

11.6.2.6.6.8 MaxValue

Definition

The maximum value used for the automatic scaling of the y axis. This property will only be used when the chart object is set to IsAutoScale

Property Value

A `double` value

Syntax

`MaxValue`

Examples

```
public override void OnCalculateMinMax()
{
   if (DrawingState != DrawingState.Building)
   {
      //set the maximum value to the chart anchors price
      MaxValue = Anchor.Price;
   }
}
```

11.6.2.6.6.9 MinValue

Definition

The minimum value used for the automatic scaling of the y axis. This property will only be used when the chart object is set to IsAutoScale

Property Value

A `double` value

Syntax

`MinValue`

Examples

```
public override void OnCalculateMinMax()
{
   if (DrawingState != DrawingState.Building)
   {
      //set the minimum value to the chart anchors price
      MinValue = Anchor.Price;
   }
}
```

11.6.2.6.6.10 OnCalculateMinMax()

Definition

An event driven method which is called while the chart scale is being updated. This method is used to determine the highest and lowest value that can be used for the chart scale. It is only called when the chart object is either set to IsAutoScale while there are multiple charts objects rendered or only a single object would be rendered on the chart.

Note: The indexer used to look up a Series<T> value through barsAgo is **NOT** guaranteed to be in sync when the OnCalculateMinMax() method is called. You will need to use GetValueAt() to obtain a historical value at a specified absolute index.

Method Return Value
This method does not return a value.

Syntax
You must override the method in your NinjaScript object with the following syntax:

```
public override void OnCalculateMinMax()
{

}
```

Method Parameters
This method does not accept any parameters.

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name        = "Example Indicator";
      IsOverlay   = true;

      // set this to true to ensure CalculateMinMix() is called
      IsAutoScale = true;
   }
}

public override void OnCalculateMinMax()
{
   // make sure to always start fresh values to calculate new
min/max values
   double tmpMin = double.MaxValue;
   double tmpMax = double.MinValue;

   // For performance optimization, only loop through what is
viewable on the chart
   for (int index = ChartBars.FromIndex; index <=
ChartBars.ToIndex; index++)
   {
      // since using Close[0] is not guaranteed to be in sync
      // retrieve "Close" value at the current viewable range index
      double plotValue = Close.GetValueAt(index);

      // return min/max of close value
      tmpMin = Math.Min(tmpMin, plotValue);
      tmpMax = Math.Max(tmpMax, plotValue);
   }

   // Finally, set the minimum and maximum Y-Axis values to +/- 50
ticks from the primary close value
   MinValue = tmpMin - 50 * TickSize;
   MaxValue = tmpMax + 50 * TickSize;
}
```

11.6.2.6.6.11 OnRender()

Definition

Used to render custom drawing to a chart from various chart objects, such as an Indicator, DrawingTool or Strategy.

Notes:

1. This method uses the 3rd party SharpDX library to render custom Direct2D Text and Shapes. For a walk through for using the **SharpDX**, please see the educational resource Using SharpDX for Custom Chart Rendering
2. The **OnRender()** method frequently runs once the State has reached **State.Realtime** in response to market data updates or a user interacting with the chart (e.g., clicking, resizing, rescaling, etc.)
3. For performance optimizations, the timing of the calls to **OnRender()** are buffered to at least 250ms, and re-renders once internal logic determines that values may be out-of-date. See also ForceRefresh() for more details
4. When using the Strategy Analyzer, **OnRender()** does **NOT** call until you switch to the "Chart" display and renders from **State.Terminated**. As a result, this method should **NOT** be relied on for historical Strategy backtesting logic and should **ONLY** be used for rendering purposes
5. Unlike market data events and strategy order related events, there is **NO** guarantee that the *barsAgo* indexer used for Series<T> objects are in sync with the current bars in progress. As a result, you should favor using an absolute index method to look up values (e.g., <series>.GetValueAt(), Bars.GetOpen(), etc)
6. While **OnRender()** is an excellent means for customizing and enhancing indicators and strategies, its application can easily be abused, resulting in unforeseen performance issues which you may not catch until the right conditions (e.g., in the hands of your users during an FOMC event)
7. Please limit any calculations or algorithms you may be tempted run in OnRender() simply to rendering. You should always favor precomputed values and store them for rendering later as the preferred approach to working with the OnRender() method (e.g., reusing brushes, passing values from OnBarUpdate(), etc.). See also OnRenderTargetChanged() method for more information on reusing Brushes
8. If you are using this method as an opportunity to "hook" onto a user related event, such as when a user selects a 3rd party control, you should alternatively consider using the events of that control independent of official NinjaScript events. See also TriggerCustomEvent()

Method Return Value

This method does not return a value

Syntax

```
protected override void OnRender(ChartControl chartControl, ChartScale chartScale)
{

}
```

Warning: Each DirectX render target requires its own brushes. You must create a

brushes directly in **OnRender()** or using OnRenderTargetChanged(). If you do not you will receive an error at run time similar to:

"A direct X error has occured while rendering the chart: HRESULT: [0x88990015], Module: [SharpDX.Direct2D1], ApiCode: [D2DERR_WRONG_RESOURCE_DOMAIN/WrongResourceDomain], Message: The resource was realized on the wrong render target. : Each DirectX render target requires its own brushes. You must create brushes directly in OnRender() or using OnRenderTargetChanged().

Please see OnRenderTargetChanged() for examples of a brush that needs to be recalculated, or the example below of recreating a static brush.

Method Parameters

`chartControl`	A ChartControl object (the chart's bar-related properties and x-axis)
`chartScale`	A ChartScale object (the chart's y-axis)

Tips:

- Please see the help guide topic on Working with Brushes for general information on using brushes and advanced brush concepts
- If you are using standard Plots along with custom rendering from an indicator or strategy, you will need to ensure to call the **base.OnRender()** method for those plots to display.

Examples

Using a static SharpDX Brush to render a rectangle on the chart panel

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // implicitly recreate and dispose of brush on each render pass
   using (SharpDX.Direct2D1.SolidColorBrush dxBrush = new
SharpDX.Direct2D1.SolidColorBrush(RenderTarget,
SharpDX.Color.Blue))
   {
      RenderTarget.FillRectangle(new
SharpDX.RectangleF(ChartPanel.X, ChartPanel.Y, ChartPanel.W,
ChartPanel.H), dxBrush);
   }
}
```

Calling the base.OnRender() method to ensure Plots are rendered along with custom render logic

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // call the base.OnRender() to ensure standard Plots work as
designed
    base.OnRender(chartControl, chartScale);

    // custom render logic
}
```

Using multiple SharpDX objects to override the default plot appearance

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // get the starting and ending bars from what is rendered on the
chart
   float startX = chartControl.GetXByBarIndex(ChartBars,
ChartBars.FromIndex);
   float endX = chartControl.GetXByBarIndex(ChartBars,
ChartBars.ToIndex);

   // Loop through each Plot Values on the chart
   for (int seriesCount = 0; seriesCount < Values.Length;
seriesCount++)
   {
      // get the value at the last bar on the chart (if it has been
set)
      if
(Values[seriesCount].IsValidDataPointAt(ChartBars.ToIndex))
      {
         double plotValue =
Values[seriesCount].GetValueAt(ChartBars.ToIndex);

         // convert the plot value to the charts "Y" axis point
         float chartScaleYValue =
chartScale.GetYByValue(plotValue);

         // calculate the x and y values for the line to start and
end
         SharpDX.Vector2 startPoint = new SharpDX.Vector2(startX,
chartScaleYValue);
         SharpDX.Vector2 endPoint = new SharpDX.Vector2(endX,
chartScaleYValue);

         // draw a line between the start and end point at each
plot using the plots SharpDX Brush color and style
         RenderTarget.DrawLine(startPoint, endPoint,
Plots[seriesCount].BrushDX,
            Plots[seriesCount].Width,
Plots[seriesCount].StrokeStyle);

         // use the chart control text form to draw plot values
along the line
         SharpDX.DirectWrite.TextFormat textFormat =
chartControl.Properties.LabelFont.ToDirectWriteTextFormat();

         // calculate the which will be rendered at each plot using
it the plot name and its price
         string textToRender = Plots[seriesCount].Name + ": " +
plotValue;

         // calculate the layout of the text to be drawn
         SharpDX.DirectWrite.TextLayout textLayout = new
```

11.6.2.6.6.12 OnRenderTargetChanged()

Definition

Called whenever a Chart's RenderTarget is created or destroyed. OnRenderTargetChanged() is used for creating / cleaning up resources such as a **SharpDX.Direct2D1.Brush** used throughout your NinjaScript class.

Notes:
1. A RenderTarget will be created and destroyed several times during the lifetime of a chart. For example, a user resizing the chart would cause the **RenderTarget** to be re-created as the chart is rendered to reflect the new dimensions. Another example is when a user clicks on the chart as a **RenderTarget** is used during hit testing. Since there are multiple **RenderTargets**, you **MUST** ensure the resource being used belongs to the destination target. In practice, all you need to understand is if you are using a device resource (e.g., custom SharpDX Brush) throughout different event methods, you should recreate these resource during **OnRenderTargetChanged()** which ensures the device resource is updated correctly as the devices context changes.
2. During initialization your NinjaScript indicators and strategies are guaranteed to see State.Configure before OnRenderTargetChanged() would be called.

Method Return Value

This method does not return a value.

Syntax

You may override the method in your indicator with the following syntax:

```
public override void OnRenderTargetChanged()
{
}
```

Warning: Each DirectX render target requires its own brushes. You must create a brushes directly in OnRender() or using **OnRenderTargetChanged()**. If you do not you will receive an error at run time similar to:

"A direct X error has occured while rendering the chart: HRESULT: [0x88990015], Module: [SharpDX.Direct2D1], ApiCode: [D2DERR_WRONG_RESOURCE_DOMAIN/WrongResourceDomain], Message: The resource was realized on the wrong render target. : Each DirectX render target requires its own brushes. You must create brushes directly in OnRender() or using OnRenderTargetChanged().

Please see the example below on using **OnRenderTargetChanged()** with brush that needs to be recalculated, or OnRender() for an example of recreating a static brush.

Parameters

This method does not accept any parameters

Tips:

1. If you are exclusively using resources in **OnRender()** (e.g., not passing values from **OnStateChange()** or other events) you only need to create and dispose of the resource in **OnRender()**. The **OnRenderTargetChanged()** concepts illustrated below would not need to be applied.
2. For a walk through for using the **SharpDX RenderTarget**, please see the educational resource Using SharpDX for Custom Chart Rendering

Examples

Recalculating a SharpDX Brush conditionally in OnBarUpdate()

```
private SharpDX.Direct2D1.Brush dxBrush = null; // the SharpDX
brush used for rendering
private System.Windows.Media.SolidColorBrush brushColor; // used to
determine the color of the brush conditionally

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "OnRenderTargetChanged Example";
      IsOverlay = false;
   }
}

protected override void OnBarUpdate()
{
   if (Close[0] > Open[0])
   {
      brushColor = Brushes.Green;
   }

   else if (Close[0] < Open[0])
   {
      brushColor = Brushes.Red;
   }

   else brushColor = Brushes.Blue;
}

public override void OnRenderTargetChanged()
{
   // if dxBrush exists on first render target change, dispose of
it
   if (dxBrush != null)
   {
      dxBrush.Dispose();
   }

   // recalculate dxBrush from value calculated in OnBarUpdated
when RenderTarget is recreated
   if (RenderTarget != null)
      dxBrush = brushColor.ToDxBrush(RenderTarget);
}

protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // fill a custom SharpDX rectangle using the dx brush
   RenderTarget.FillRectangle(new SharpDX.RectangleF(ChartPanel.X,
ChartPanel.Y, ChartPanel.W, ChartPanel.H), dxBrush);
}
```

Recalculating a SharpDX Brush based on user input

```
private SharpDX.Direct2D1.Brush dxBrush = null; // the SharpDX
brush used for rendering

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "OnRenderTargetChanged Example";
      IsOverlay = false;
      UserBrush = Brushes.Red;  // user selection pushed to the UI
   }
}
public override void OnRenderTargetChanged()
{
   // if dxBrush exists on first render target change, dispose of
it
   if (dxBrush != null)
   {
      dxBrush.Dispose();
   }

   // recalculate dxBrush from user defined brush when RenderTarget
is recreated
   if (RenderTarget != null)
      dxBrush = UserBrush.ToDxBrush(RenderTarget);
}

protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // fill a custom SharpDX rectangle using the dx brush
   RenderTarget.FillRectangle(new SharpDX.RectangleF(ChartPanel.X,
ChartPanel.Y, ChartPanel.W, ChartPanel.H), dxBrush);
}

[XmlIgnore]
public Brush UserBrush { get; set; } // brush selection set by user
in UI

[Browsable(false)]
public string MyBrushSerialize // string used to serialize
selection set by user in UI
{
   get { return Serialize.BrushToString(UserBrush); }
   set { UserBrush = Serialize.StringToBrush(value); }
}
```

11.6.2.6.6.13 PanelUI

Definition

The zero-based index of the chart panel in which the calling script is configured.

> **Note**: The "Panel" property configured in the Indicators or Strategies window on a chart is non-zero-based, while PanelUI is zero-based. For example, if an indicator is configured in Panel # 1 in the Indicators window, PanelUI will return an index of 0. If the indicator were configured in Panel # 4 in the Indicators window, PanelUI would return an index of 3.

Property Value

An `int` value representing the panel the object is configured. This property is **read-only**.

Syntax

`PanelUI`

Examples

```
protected override void OnBarUpdate()
{
    // Print the zero-based panel index on which the script is
configured
    Print("My object is on is on panel # " + PanelUI);
}
```

11.6.2.6.6.14 RenderTarget

Definition

A **SharpDX Direct2D1 RenderTarget** creates objects and exposes methods used for drawing in the chart area.

> **Notes**:
> 1. There are two **RenderTarget**'s used in a chart. This is important to understand when creating/destroying device resources. Please see the OnRenderTargetChanged() page for more information
> 2. For a walk through for using the **SharpDX RenderTarget**, please see the educational resource Using SharpDX for Custom Chart Rendering

Property Value

A SharpDX.Direct2D1.RenderTarget

SharpDX.Direct2D1.WindowRenderTarget	Used to render the actual contents of the chart to the window
SharpDX.Direct2D1.WicRenderTarget	Used to render a bitmap for a few scenarios: 1. A user clicks on a chart area; a bitmap is used to do any hit detection to determine where the user clicked 2. User clicks on the Windows task bar; a bitmap is used to rendered the preview the contents of the chart display through a thumbnail on the task bar 3. A user re-sizes the chart; a bitmap is used to render the current contents of the chart, which is redrawn using the **WindowRenderTarget** after the desired changes have been set

Syntax

```
RenderTarget
```

Warning: Each DirectX render target requires its own brushes. You must create a brushes directly in OnRender() or using OnRenderTargetChanged(). If you do not you will receive an error at run time similar to:

"A direct X error has occured while rendering the chart: HRESULT: [0x88990015], Module: [SharpDX.Direct2D1], ApiCode: [D2DERR_WRONG_RESOURCE_DOMAIN/WrongResourceDomain], Message: The resource was realized on the wrong render target. : Each DirectX render target requires its own brushes. You must create brushes directly in OnRender() or using OnRenderTargetChanged().

Please see OnRenderTargetChanged() for examples with brush that needs to be recalculated, or OnRender() for an example of recreating a static brush.

11.6.2.6.6.15 SetZOrder

Definition

Used to assign a unique identifier representing the index in which chart objects are drawn on the chart's Z-axis (front to back ordering). Objects with a higher ZOrder are drawn first.

Note:

1. To check on which ZOrder index the object gets drawn use the ZOrder property.
2. Assigning specific ZOrder indices to draw at should be done once the State has reached **State.Historical**
3. If you want to draw your object behind the bars, assign to use index **-1** (like in the example below)
4. If you want to draw your object topmost, assign to use index **int.MaxValue**
5. Any levels in between can be directly assigned, the starting / default levels used by NinjaTrader can be seen here.
6. You can see the highest ZOrder currently in a chart with code such our second example below - setting higher values than this value will result in the ZOrder to be set to this value, so this can be thought of as the current 'top'.

Method Return Value

This method does not return a value

Syntax

```
SetZOrder(int DesiredZOrderLevel)
```

Examples

```
protected override void OnStateChange()
{
    if (State == State.Historical)
    {
        // Make sure our object plots behind the chart bars
        SetZOrder(-1);
    }
}
```

```
protected override void OnRender(ChartControl cc, ChartScale cs)
{
    Print(ChartPanel.ChartObjects.Max(co => co.ZOrder));
}
```

11.6.2.6.6.16 ZOrder

Definition

A unique identifier representing the index in which chart objects are drawn on the chart's Z-axis (front to back ordering). Objects with a higher ZOrder are drawn first.

> **Note**: The **ZOrder** index should **NOT** be set using this property. Please use the dedicated SetZOrder() for this purpose.

Property Value

A `int` value representing the order that the object is drawn. Default value is categorized by the type of object drawn, which will then increment for each instance of the chart object that is drawn. Each type of object will have a different default starting value to keep these objects separate:

Chart Bars	1
NinjaScript Objects	10001
Global Draw Objects	20001
Draw Objects	30001

Syntax

`ZOrder`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    // call the base.OnRender() to ensure standard Plots work as
designed
    base.OnRender(chartControl, chartScale);

    // Print the currently assigned ZOrder index for this
NinjaScript object
    Print("Current ZOrder level is: " + ZOrder);
}
```

11.6.2.6.7 FormatPriceMarker()

Definition
Used to override the default string format of a NinjaScript's price marker values.

Method Return Value
A virtual `string` which is overridden from the default price marker value

Syntax
You must override the method in your indicator with the following syntax
```
public override string FormatPriceMarker(double price)
{

}
```

Parameters

price	A `double` value representing the value to be overridden.

Tip: Standard Numeric Format Strings examples can be found on Microsoft's Developer Network (MSDN article)

Examples
```
// FormatPriceMarker method of a custom indicator
public override string FormatPriceMarker(double price)
{
     // Formats price marker values to 4 decimal places
     return price.ToString("N4");
}

protected override void OnBarUpdate()
{
   // overriding FormatPriceMarker will ensure display of 4 decimal
places
   MyPlot[0] = (Close[0] + Open[0] * .0025);
}
```

11.6.2.6.8 IsAutoScale

Definition
If true, the object will call CalculateMinMax() in order to determine the object's MinValue and MaxValue value used to scale the Y-axis of the chart.

Property Value
This property returns **true** if the object's are included in the y-scale; otherwise, **false**. Default set to **false** for DrawingTools, but set to **true** for Indicators.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax
`IsAutoScale`

Example
```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name                    = "Example Indicator";
      // set this to true to call CalculateMinMix() to ensure
drawing tool is fully rendered in chart scale
      IsAutoScale = true;
   }
   else if (State == State.Configure)
   {
   }
}
```

11.6.2.6.9 IsOverlay

Definition
Determines if indicator plot(s) are drawn on the chart panel over top of price. Setting this value to true will also allow an Indicator to be used as a SuperDOM Indicator.

Property Value
This property returns **true** if any indicator plot(s) are drawn on the chart panel; otherwise, **false**. Default set to **false**.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults**

Syntax
`IsOverlay`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IsOverlay = true; // Indicator plots are drawn on the
chart panel on top of price
          AddPlot(Brushes.Orange, "SMA");
     }
}
```

11.6.2.6.10 IsSeparateZOrder

Definition

Determines the ZOrder of the drawing object will be different than the NinjaScript object that drew it. When false the drawing object will share the same ZOrder.

Property Value

This property returns **true** if the object is drawn on a separate **ZOrder**; otherwise, **false**. Default set to **false**.

Syntax

`IsSeparateZOrder`

Example

```
protected override void OnBarUpdate()
{
    // Instantiate a Dot object
    Dot myDot = Draw.Dot(this, "NewDot", true, 5, High[5],
Brushes.Black);

    // Set the Dot object to use a separate Z-Order than the
indicator that created it
    myDot.IsSeparateZOrder = true;
}
```

11.6.2.6.11 ScaleJustification

Definition

Determines which scale an indicator will be plotted on.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Property Value
This property returns a ScaleJustification value of either:

```
NinjaTrader.Gui.Charts.ScaleJustification.Left;
NinjaTrader.Gui.Charts.ScaleJustification.Overlay;
NinjaTrader.Gui.Charts.ScaleJustification.Right;
```

Syntax
```
ScaleJustification
```

Examples


```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";

      // force "My Plot" to be plotted on the left scale
      ScaleJustification = ScaleJustification.Left;
   }
   else if (State == State.Configure)
   {
      AddPlot(Brushes.Orange, "My Plot");
   }
}
```

11.6.2.6.12 Stroke Class

Definition
Objects derived from the Stroke class are used to characterize how a plot is visually displayed (plotted) on a chart.

Syntax
```
Stroke(Stroke stroke)
Stroke(Brush brush)
Stroke(Brush brush, float width)
Stroke(Brush brush, DashStyle dashStyleHelper, float width)
```

Parameters

brush	The brush used to draw the plot (reference)
dashStyleHelper	Possible values: DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid
stroke	The stroke object
width	The width of the stroke

Properties

Brush	The System.Windows.Media.Brush used to construct the stroke (reference)
BrushDX	A SharpDX.Direct2D1.Brush used to actually render the stroke **Note**: To avoid and resolve access violation exceptions, please see Warning and examples remarked below
DashStyleDX	A SharpDX.Direct2D1.DashStyle used to render the stroke style **Note**: To avoid and resolve access violation exceptions, please see Warning and examples remarked below
DashStyleHelper	A dashstyle used to construct the stroke. Possible values are: • DashStyleHelper.Dash

	• DashStyleHelper.DashDot • DashStyleHelper.DashDotDot • DashStyleHelper.Dot • DashStyleHelper.Solid
RenderTarget	The RenderTarget drawing context used for the stroke. **Note**: This property must be set before accessing a stroke's BrushDX property. Please see Warning and examples remarked below
StrokeStyle	A SharpDX.Direct2D1.StrokeStyle
Width	A float representing the width in pixels

Warning: There may be situations where a **RenderTarget** has not been set, and to prevent access violation exception before accessing the **BrushDX** or **DashStyleDX** properties, you should explicitly set the **RenderTarget** before attempting to access that property. Please see the example below.

Examples

See the AddPlot() method for additional examples.

Using a Stroke SharpDX Brush for Custom Rendering

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      IsOverlay = true;
      // set the Stroke default to red brush
      MyStroke = new Stroke(Brushes.Red);
   }
   else if (State == State.Configure)
   {
   }
}

public override void OnRenderTargetChanged()
{
   // Explicitly set the Stroke RenderTarget
   if (RenderTarget != null)
      MyStroke.RenderTarget = RenderTarget;
}

protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // create two points from the top left corner
   SharpDX.Vector2 pointA = new SharpDX.Vector2(0, 0);
   // to 300 pixels offset X and Y to create a diagonal line
   SharpDX.Vector2 pointB = new SharpDX.Vector2(300, 300);

   // Draw the line using the Stroke SharpDX brush
   RenderTarget.DrawLine(pointA, pointB, MyStroke.BrushDX,
MyStroke.Width, MyStroke.StrokeStyle);

}

[NinjaScriptProperty]
[Description("My Stroke")]
public Stroke MyStroke { get; set; }
```

Convert the Windows Media Brush to a SharpDX Brush

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      IsOverlay = true;
      // set stroke default to blue brush
      MyStroke = new Stroke(Brushes.Blue);
   }
   else if (State == State.Configure)
   {
   }
}

protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // create two points from the top left corner
   SharpDX.Vector2 pointA = new SharpDX.Vector2(0, 0);
   // to 300 pixels offset X and Y to create a diagonal line
   SharpDX.Vector2 pointB = new SharpDX.Vector2(300, 300);

   NinjaTrader.Gui.Stroke MyStroke = new Stroke(Brushes.Blue);

   // if BrushDX is null, convert the constructed brush to a DX
brush
   SharpDX.Direct2D1.Brush myBrush = MyStroke.BrushDX ??
MyStroke.Brush.ToDxBrush(RenderTarget);
   RenderTarget.DrawLine(pointA, pointB, myBrush, MyStroke.Width,
MyStroke.StrokeStyle);

   myBrush.Dispose();
}

[NinjaScriptProperty]
[Description("My Stroke")]
public Stroke MyStroke { get; set; }
```

11.6.2.6.13 UserControlCollection

Definition

An observable collection of 3rd party framework elements, the purpose of which is to allow developers to add a custom control to the chart (e.g., add a button or create your own data grid). This framework collection resides on top of the ChartControl in order to prevent 3rd party custom controls from interfering with native NinjaTrader chart framework members. For example, if you wish to add a button to a chart, it is recommended to add it to this UserControlCollection rather than attempting to modify or add to any pre-existing NinjaTrader chart elements.

Notes:
1. This collection is provided "as-is" and does **NOT** contain any automatic layout options. By default, the last added framework element will reside on top of any previously added controls. This means it is possible for a user to install two NinjaScript objects which may be competing for an area of a chart.
2. Once the NinjaScript object is removed from the chart by the user, the custom control will be automatically removed from the collection.

Warnings:
1. This property should **ONLY** be accessed once your NinjaScript object has reached **State.Historical** or later
2. You **MUST** use a Dispatcher in order to account for any UI threading errors. Please see the example below for proper usage
3. It is imperative that you dispose of any custom control resources in **State.Terminated** to ensure there are no leaks between instances of the object

Property Value

```
ObservableCollection<System.Windows.FrameworkElement>
```

Syntax

```
UserControlCollection[int idx]
```

Examples

```
private System.Windows.Controls.Button  myBuyButton;
private System.Windows.Controls.Button  mySellButton;
private System.Windows.Controls.Grid    myGrid;

// Define a custom event method to handle our custom task when the
button is clicked
private void OnMyButtonClick(object sender, RoutedEventArgs rea)
{
   System.Windows.Controls.Button button = sender as
System.Windows.Controls.Button;
   if (button != null)
      Print(button.Name + " Clicked");
}

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name          = "SampleAddButton";
      Description   = "Adds a custom control to the chart";
      IsOverlay     = true;
   }
   else if (State == State.Configure)
   {
   }

   // Once the NinjaScript object has reached State.Historical, our
custom control can now be added to the chart
   else if (State == State.Historical)
   {
      // Because we're dealing with UI elements, we need to use the
Dispatcher which created the object
      // otherwise we will run into threading errors...
      // e.g, "Error on calling 'OnStateChange' method: You are
accessing an object which resides on another thread."
      // Furthermore, we will do this operation Asynchronously to
avoid conflicts with internal NT operations
      ChartControl.Dispatcher.InvokeAsync((() =>
      {
         // Grid already exists
         if (UserControlCollection.Contains(myGrid))
            return;

         // Add a control grid which will host our custom buttons
         myGrid = new System.Windows.Controls.Grid
         {
            Name = "MyCustomGrid",
            // Align the control to the top right corner of the
chart
            HorizontalAlignment = HorizontalAlignment.Right,
            VerticalAlignment = VerticalAlignment.Top,
         };
```

11.6.2.7 Drawing

You can use NinjaScript to draw custom shapes, lines, text and colors on price and indicator panels from both Indicators and Strategies.

Draw Methods and Associated Return Types

Draw Method	Return Type
Draw.AndrewsPitchfork()	AndrewsPitchfork
Draw.Arc()	Arc
Draw.ArrowDown()	ArrowDown
Draw.ArrowLine()	ArrowLine

Draw.ArrowUp()	ArrowUp
Draw.Diamond()	Diamond
Draw.Dot()	Dot
Draw.Ellipse()	Ellipse
Draw.ExtendedLine()	ExtendedLine
Draw.FibonacciCircle()	FibonacciCircle
Draw.FibonacciExtensions()	FibonacciExtensions
Draw.FibonacciRetracements()	FibonacciRetracements
Draw.FibonacciTimeExtensions()	FibonacciTimeExtensions
Draw.GannFan()	GannFan
Draw.HorizontalLine()	HorizontalLine
Draw.Line()	Line
Draw.Pathtool()	Pathtool
Draw.Polygon()	Polygon
Draw.Ray()	Ray
Draw.Rectangle()	Rectangle
Draw.Region()	Region
Draw.RegionHighlightX()	RegionHighlightX

Draw.RegionHighlightY()	RegionHighlightY
Draw.RegressionChannel()	RegressionChannel
Draw.RiskReward()	RiskReward
Draw.Ruler()	Ruler
Draw.Square()	Square
Draw.Text()	Text
Draw.TextFixed()	TextFixed
Draw.TimeCycles()	TimeCycles
Draw.TrendChannel()	TrendChannel
Draw.Triangle()	Triangle
Draw.TriangleDown()	TriangleDown
Draw.TriangleUp()	TriangleUp
Draw.VerticalLine()	VerticalLine

Drawing Methods and Properties

Property	Description
AllowRemovalOfDrawObjects	Determines if programmatically drawn DrawObjects can be manually removed from the chart

BackBrush	Sets the brush used for painting the chart panel's background color for the current bar
BackBrushAll	Sets the brush used for painting the chart's background color for the current bar
BackBrushes	A collection of historical brushes used for the background colors for the chart panel
BackBrushesAll	A collection of historical brushes used for the background colors for all chart panels
BarBrush	Sets the brush used for painting the color of a price bar's body
BarBrushes	A collection of historical brushes used for painting the color of a price bar's body
Brushes	A collection of static, predefined Brushes supplied by the .NET Framework
CandleOutlineBrush	Sets the outline Brush of a candlestick
CandleOutlineBrushes	A collection of historical outline brushes for candlesticks
DrawObjects	A collection holding all of the drawn chart objects for the primary bar series
IDrawingTool	Represents an interface that exposes information regarding a drawn chart object
RemoveDrawObject()	Removes a draw object from the chart based on its tag value
RemoveDrawObjects()	Removes all draw objects originating from the indicator or strategy from the chart
SimpleFont Class	Defines a particular font configuration

1. Custom graphics for custom indicators can be painted on either the price panel or indicator panel. You could for example have a custom indicator displayed in an indicator panel yet have associated custom graphics painted on the price panel. The "DrawOnPricePanel" property is set to **true** by default, which means that custom graphics will always be painted on the price panel, even if the indicator is plotted in a separate panel. If you want your custom graphics to be plotted on the indicator panel, set this property to **false** in the OnStateChange() method of your custom indicator.
2. Set unique tag values for each draw object, unless you intend for new draw objects to replace existing objects with the same tag. A common trick is to incorporate the bar number as part of the unique tag identifier. For example, if you wanted to draw a dot that indicated a buying condition above a bar, you could express it:

```
Draw.Dot(this, CurrentBar.ToString() + "Buy", false, 0, High[0] + TickSize,
Brushes.ForestGreen);
```

3. Draw methods will not work if they are called from the OnStateChange() method.

11.6.2.7.1 Draw.AndrewsPitchfork()

Definition

Draws an Andrew's Pitchfork.

Method Return Value

An AndrewsPitchfork object that represents the draw object.

Syntax

```
Draw.AndrewsPitchfork(NinjaScriptBase owner, string tag, bool isAutoScale, int
anchor1BarsAgo, double anchor1Y, int anchor2BarsAgo, double anchor2Y, int
anchor3BarsAgo, double anchor3Y, Brush brush, DashStyleHelper dashStyle, int width)
Draw.AndrewsPitchfork(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
anchor1Time, double anchor1Y, DateTime anchor2Time, double anchor2Y, DateTime
anchor3Time, double anchor3Y, Brush brush, DashStyleHelper dashStyle, int width)
Draw.AndrewsPitchfork(NinjaScriptBase owner, string tag, bool isAutoScale, int
anchor1BarsAgo, double anchor1Y, int anchor2BarsAgo, double anchor2Y, int
anchor3BarsAgo, double anchor3Y, bool isGlobal, string templateName)
Draw.AndrewsPitchfork(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
anchor1Time, double anchor1Y, DateTime anchor2Time, double anchor2Y, DateTime
anchor3Time, double anchor3Y, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method

	Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale
anchor1BarsAgo	The number of bars ago (x value) of the 1st anchor point
anchor1Time	The time of the 1st anchor point
anchor1Y	The y value of the 1st anchor point
anchor2BarsAgo	The number of bars ago (x value) of the 2nd anchor point
anchor2Time	The time of the 2nd anchor point
anchor2Y	The y value of the 2nd anchor point
anchor3BarsAgo	The number of bars ago (x value) of the 3rd anchor point
anchor3Time	The time of the 3rd anchor point
anchor3Y	The y value of the 3rd anchor point
brush	The brush used to color draw object (reference)
dashStyle	DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid

	Note: Drawing objects with y values very far off the visible canvas can lead to performance hits. Fancier DashStyles like DashDotDot will also require more resources than simple DashStyles like Solid.
width	The width of the draw object
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws an Andrew's Pitchfork
Draw.AndrewsPitchfork(this, "tag1", true, 4, Low[4], 3, High[3], 1,
 Low[1], Brushes.Blue, DashStyleHelper.Solid, 3);
```

11.6.2.7.1.1 AndrewsPitchfork

Definition

Represents an object that exposes information regarding an Andrews Pitchfork IDrawingTool.

The *Standard Pitchfork* creates a trend channel out of the 3 user defined extreme price anchor points by connecting the first 2 points to form the anchor, and the next 2 points to form the retracement handle. From the first point then a trendline is drawn through the 50% midpoint of the retracement handle, parallel lines originating at the other 2 points forming the channel, while multiple further price levels could be set to allow for finer analysis.

In contrast the *Schiff Pitchfork* variant is constructed then by shifting the first anchor of the Standard Pitchfork one-half the vertical distance between the first 2 anchor points.

As further alternation the *Modified Schiff Pitchfork* variant is found by moving the first anchor to the midpoint of the original pitchfork's anchor handle, the trend-line connecting our first 2 anchor points.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
ExtensionAnchor	An IDrawingTool's ChartAnchor representing the extension point of the drawing object
PriceLevels	A collection of prices calculated by the drawing object
CalculationMethod	The AndrewsPitchforkCalculationMethod property determining which method is used to calculate the pitchfork. Possible values are: • ModifiedSchiff • Schiff • StandardPitchfork
IsTextDisplayed	A `bool` value determining if the draw object should display text on the chart.
RetracementLineStroke	A Stroke object used to draw the center retracement line of the object
AnchorLineStroke	A Stroke object used to draw the object

Example

```
// Instantiate an Andrews Pitchfork object
AndrewsPitchfork myFork = Draw.AndrewsPitchfork(this, "tag1",
false, 7, Low[7], 5, High[5], 1, Low[1], false, "ForkTemplate");

// Print the tag used to draw the object
Print(myFork.Tag);
```

11.6.2.7.2 Draw.Arc()

Definition
Draws an arc.

Method Return Value
An Arc object that represents the draw object.

Syntax
```
Draw.Arc(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush)
Draw.Arc(NinjaScriptBase owner, string tag, DateTime startTime, double startY, DateTime endTime, double endY, Brush brush)
Draw.Arc(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush, DashStyleHelper dashStyle, int width)
Draw.Arc(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime, double startY, DateTime endTime, double endY, Brush brush, DashStyleHelper dashStyle, int width)
Draw.Arc(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush, DashStyleHelper dashStyle, int width, bool drawOnPricePanel)
Draw.Arc(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime, double startY, DateTime endTime, double endY, Brush brush, DashStyleHelper dashStyle, int width, bool drawOnPricePanel)
Draw.Arc(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int endBarsAgo, double endY, bool isGlobal, string templateName)
Draw.Arc(NinjaScriptBase owner, string tag, DateTime startTime, double startY, DateTime endTime, double endY, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.

isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn.
startY	The starting y value co-ordinate where the draw object will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
endY	The end y value co-ordinate where the draw object will terminate
brush	The brush used to color draw object (reference)
dashStyle	DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid **Note**: Drawing objects with y values very far off the visible canvas can lead to performance hits. Fancier DashStyles like DashDotDot will also require more resources than simple DashStyles like Solid.
width	The width of the draw object
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel

isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a dotted lime green arc
Draw.Arc(this, "tag1", false, 10, 1000, 0, 1001, Brushes.LimeGreen,
DashStyleHelper.Dot, 2);
```

11.6.2.7.2.1 Arc

Definition

Represents an interface that exposes information regarding an Arc IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
AreaBrush	A Brush object representing the fill color of the draw object
AreaOpacity	An `int` value representing the opacity of the area color
ArcStroke	The Stroke object used to draw the arc line of the object's outline
Stroke	The Stroke object used to draw the straight line of the object's outline

Example

```
// Draw an Arc object
Arc myArc = Draw.Arc(this, "myArc", Time[10], Close[10], Time[0],
Close[0], Brushes.Blue);

// Set the opacity of the shading between the arc and the chord
myArc.AreaOpacity = 100;
```

11.6.2.7.3 Draw.ArrowDown()

Definition

Draws an arrow pointing down.

Method Return Value

An ArrowDown object that represents the draw object.

Syntax

```
Draw.ArrowDown(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo,
double y, Brush brush)
Draw.ArrowDown(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, Brush brush)
Draw.ArrowDown(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo,
double y, Brush brush, bool drawOnPricePanel)
Draw.ArrowDown(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, Brush brush, bool drawOnPricePanel)
Draw.ArrowDown(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo,
double y, bool isGlobal, string templateName)
Draw.ArrowDown(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, bool isGlobal, string templateName)
```

Parameters

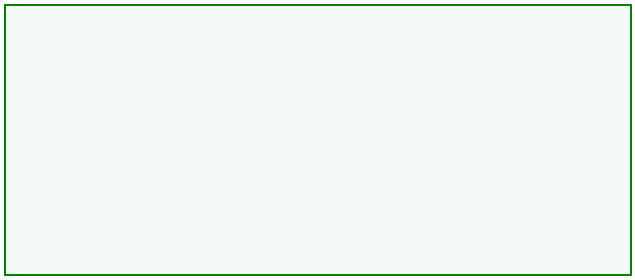
owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.

isAutoScale	Determines if the draw object will be included in the y-axis scale
barsAgo	The bar the object will be drawn at. A value of 10 would be 10 bars ago.
time	The time the object will be drawn at.
y	The y value
brush	The brush used to color draw object (reference)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Tip: The size of the

arrowistiedtothechart'sBarWidthandthuswills

c
a
l
e
a
u
t
o
m
a
t
i
c
a
l
l
y
a
s
t
h
e
c
h
a
r
t
i
s
r
e
s
i
z
e
d



Examples

```
// Paints a red down arrow on the current bar 1 tick above the high
   Draw.ArrowDown(this, "tag1", true, 0, High[0] + TickSize,
Brushes.Red);

// Paints a blue down arrown on a three bar reversal pattern
if (High[2] > High[3] && High[1] > High[2] && Close[0] < Open[0])
     Draw.ArrowDown(this, CurrentBar.ToString(), true, 0, High[0] +
 TickSize, Brushes.Blue);
```

11.6.2.7.3.1 Arrow Down

Definition

Represents an interface that exposes information regarding an Arrow Down IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object
AreaBrush	A Brush object representing the fill color of the draw object
OutlineBrush	A Brush object representing the color of the draw object's outline

Example

```
// Instantiate an ArrowDown object
ArrowDown myArrow = Draw.ArrowDown(this, "tag1", true, Time[0],
High[0] + (2 * TickSize), Brushes.Green);

// Set the outline color of the Arrow
myArrow.OutlineBrush = Brushes.Black;
```

11.6.2.7.4 Draw .Arrow Line()

Definition

Draws an arrow line.

Method Return Value

An ArrowLine object that represents the draw object.

Syntax

```
Draw.ArrowLine(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
 endBarsAgo, double endY, Brush brush)
Draw.ArrowLine(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime endTime, double endY, Brush brush)
Draw.ArrowLine(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
 endBarsAgo, double endY, Brush brush, DashStyleHelper dashStyle, int width)
Draw.ArrowLine(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int endBarsAgo, double endY, Brush brush, DashStyleHelper dashStyle,
int width, bool drawOnPricePanel)
Draw.ArrowLine(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
startTime, double startY, DateTime endTime, double endY, Brush brush, DashStyleHelper
dashStyle, int width, bool drawOnPricePanel)
Draw.ArrowLine(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
 endBarsAgo, double endY, bool isGlobal, string templateName)
Draw.ArrowLine(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime endTime, double endY,  bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object.

	For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn.
startY	The starting y value co-ordinate where the draw object will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
endY	The end y value co-ordinate where the draw object will terminate
brush	The brush used to color draw object (reference)
dashStyle	DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid **Note**: Drawing objects with y values very far off the visible canvas can lead to performance hits. Fancier DashStyles like DashDotDot will also require more resources than simple DashStyles like Solid.
width	The width of the draw object

drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a dotted lime green arrow line
Draw.ArrowLine(this, "tag1", 10, 1000, 0, 1001, Brushes.LimeGreen,
DashStyleHelper.Dot, 2);
```

11.6.2.7.4.1 Arrow Line

Definition

Represents an interface that exposes information regarding an Arrow Line IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
Stroke	A Stroke object used to draw the object

Example

```
// Draw an ArrowLine object
ArrowLine myArrow = Draw.ArrowLine(this, "myArrowLine", 3, High[3],
 1, High[1], Brushes.Blue, DashStyleHelper.DashDot, 3);

// Disable the arrow's visibility
myArrow.IsVisible = false;
```

11.6.2.7.5 Draw.ArrowUp()

Definition
Draws an arrow pointing up.

Method Return Value
An ArrowUp object that represents the draw object.

Syntax
```
Draw.ArrowUp(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double y, Brush brush)
Draw.ArrowUp(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time, double y, Brush brush)
Draw.ArrowUp(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double y, Brush brush, bool drawOnPricePanel)
Draw.ArrowUp(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time, double y, Brush brush, bool drawOnPricePanel)
Draw.ArrowUp(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double y, bool isGlobal, string templateName)
Draw.ArrowUp(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time, double y, bool isGlobal, string templateName)
```

Parameters
owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale
barsAgo	The bar the object will be drawn at. A value of 10 would be 10 bars ago.
time	The time the object will be drawn at.

y	The y value
brush	The brush used to color draw object (reference)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Tip: The size of the arrow is tied

d
t
o
t
h
e
c
h
a
r
t
'
s
B
a
r
W
i
d
t
h
a
n
d
t
h
u
s
w
i
l
l
s
c
a
l
e
a
u
t
o
m
a

```
t
i
c
a
l
l
y
a
s
t
h
e
c
h
a
r
t
i
s
r
e
s
i
z
e
d
```

Examples

```
// Paints a red up arrow on the current bar 1 tick below the low
Draw.ArrowUp(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);
```

11.6.2.7.5.1 Arrow Up

Definition

Represents an interface that exposes information regarding an Arrow Up IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object
AreaBrush	A Brush object representing the fill color of the draw object
OutlineBrush	A Brush object representing the color of the draw object's outline

Example

```
// Instantiate an ArrowDown object
ArrowUp myArrow = Draw.ArrowUp(this, "tag1", true, Time[0], Low[0]
- (2 * TickSize), Brushes.Green);

// Set the outline color of the Arrow
myArrow.OutlineBrush = Brushes.Black;
```

11.6.2.7.6 Draw .Diamond()

Definition

Draws a diamond.

Method Return Value

A Diamond object that represents the draw object.

Syntax

```
Draw.Diamond(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double
y, Brush brush)
Draw.Diamond(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, Brush brush)
Draw.Diamond(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
```

```
double y, Brush brush, bool drawOnPricePanel)
Draw.Diamond(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double
y, Brush brush, bool drawOnPricePanel)
Draw.Diamond(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double
y, bool isGlobal, string templateName)
Draw.Diamond(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, bool isGlobal, string templateName)
```

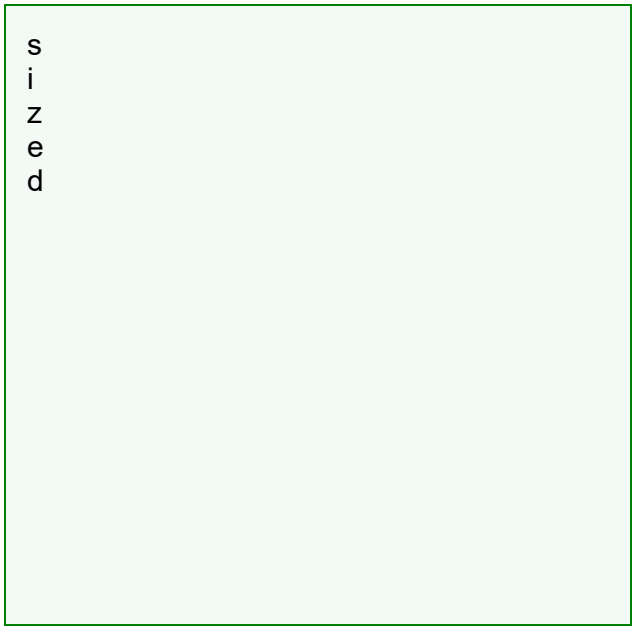
Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale
barsAgo	The bar the object will be drawn at. A value of 10 would be 10 bars ago.
time	The time the object will be drawn at.
y	The y value
brush	The brush used to color draw object (reference)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties

	(empty string could be used to just use the UI default visuals instead)

Tip: The size of the diamond distributed to the chart's Ba

r
W
i
d
t
h
a
n
d
t
h
u
s
w
i
l
l
s
c
a
l
e
a
u
t
o
m
a
t
i
c
a
l
l
y
a
s
t
h
e
c
h
a
r
t
i
s
r
e



Examples

```
// Paints a red diamond on the current bar 1 tick below the low
Draw.Diamond(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);
```

11.6.2.7.6.1 Diamond

Definition

Represents an interface that exposes information regarding a Diamond IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object
AreaBrush	A Brush object representing the fill color of the draw object
OutlineBrush	A Brush object representing the color of the draw object's outline

Example

```
// Instantiates a red diamond on the current bar 1 tick below the
low
Diamond myDiamond = Draw.Diamond(this, "tag1", true, 0, Low[0] -
TickSize, Brushes.Red);

// Set the area fill color to Red
myDiamond.AreaBrush = Brushes.Red;
```

11.6.2.7.7 Draw .Dot()

Definition

Draws a dot.

Method Return Value

A Dot object that represents the draw object.

Syntax

```
Draw.Dot(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time, double y,
 Brush brush)
Draw.Dot(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double y,
Brush brush)
Draw.Dot(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time, double y,
 Brush brush, bool drawOnPricePanel)
Draw.Dot(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double y,
Brush brush, bool drawOnPricePanel)
Draw.Dot(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time, double y,
 bool isGlobal, string templateName)
Draw.Dot(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double y,
bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.

isAutoScale	Determines if the draw object will be included in the y-axis scale
barsAgo	The bar the object will be drawn at. A value of 10 would be 10 bars ago.
time	The time the object will be drawn at.
y	The y value
brush	The brush used to color draw object (reference)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Tip: These sizeof the dodo

tistiedtothechart'sBarWidthandthushswillscale

automatically as the chart is resized

Examples

```
// Paints a red dot on the current bar 1 tick below the low
Draw.Dot(this, "tag1", true, 0, Low[0] - TickSize, Brushes.Red);
```

11.6.2.7.7.1 Dot

Definition

Represents an interface that exposes information regarding a Dot IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object
AreaBrush	A Brush object representing the fill color of the draw object
OutlineBrush	A Brush object representing the color of the draw object's outline

Example

```
// Instantiates a red dot on the current bar 1 tick below the low
Dot myDot = Draw.Dot(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);

// Disable the dot's Auto Scale property
myDot.IsAutoScale = false;
```

11.6.2.7.8 Draw.Ellipse()

Definition

Draws an ellipse.

Method Return Value

An Ellipse object that represents the draw object.

Syntax

```
Draw.Ellipse(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush)
Draw.Ellipse(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush, Brush areaBrush, int areaOpacity)
Draw.Ellipse(NinjaScriptBase owner, string tag, DateTime startTime, double startY, DateTime endTime, double endY, Brush brush)
Draw.Ellipse(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime, double startY, DateTime endTime, double endY, Brush brush, Brush areaBrush, int areaOpacity)
Draw.Ellipse(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush, bool drawOnPricePanel)
Draw.Ellipse(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush, Brush areaBrush, int areaOpacity, bool drawOnPricePanel)
Draw.Ellipse(NinjaScriptBase owner, string tag, DateTime startTime, double startY, DateTime endTime, double endY, Brush brush, bool drawOnPricePanel)
Draw.Ellipse(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime, double startY, DateTime endTime, double endY, Brush brush, Brush areaBrush, int areaOpacity, bool drawOnPricePanel)
Draw.Ellipse(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int endBarsAgo, double endY, bool isGlobal, string templateName)
Draw.Ellipse(NinjaScriptBase owner, string tag, DateTime startTime, double startY, DateTime endTime, double endY, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.

startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back
startTime	The starting time where the draw object will be drawn
startY	The starting y value co-ordinate where the draw object will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
endY	The end y value co-ordinate where the draw object will terminate
brush	The brush used to color the outline of draw object (reference)
areaBrush	The brush used to color the fill area of the draw object (reference)
areaOpacity	Sets the level of transparency for the fill color. Valid values between 0 - 100. (0 = completely transparent, 100 = no opacity)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobalDrawingTool	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Paints a red ellipse on the current bar
Draw.Ellipse(this, "tag1", true, 5, Close[5], 0, Close[0],
Brushes.Red, Brushes.Red, 5);
```

11.6.2.7.8.1 Ellipse

Definition

Represents an interface that exposes information regarding an Ellipse IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
AreaBrush	A Brush class representing the fill color of the draw object
AreaOpacity	An `int` value representing the opacity of the area color
OutlineStroke	The Stroke object used to draw the object's outline

Example

```
// Paint a red ellipse on the current bar
Ellipse myEllipse = Draw.Ellipse(this, "tag1", true, 5, Close[5],
0, Close[0], Brushes.Red, Brushes.Red, 5);

// Change the AreaOpacity of the Ellipse
myEllipse.AreaOpacity = 0;
```

11.6.2.7.9 Draw .ExtendedLine()

Definition

Draws a line with infinite end points.

Method Return Value

An ExtendedLine object that represents the draw object.

Syntax

```
Draw.ExtendedLine(NinjaScriptBase owner, string tag, int startBarsAgo, double startY,
int endBarsAgo, double endY, Brush brush)
Draw.ExtendedLine(NinjaScriptBase owner, string tag, DateTime startTime, double
startY, DateTime endTime, double endY, Brush brush)
Draw.ExtendedLine(NinjaScriptBase owner, string tag, bool isAutoScale, int
startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush, DashStyleHelper
 dashStyle, int width)
Draw.ExtendedLine(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
startTime, double startY, DateTime endTime, double endY, Brush brush, DashStyleHelper
dashStyle, int width)
Draw.ExtendedLine(NinjaScriptBase owner, string tag, bool isAutoScale, int
startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush, DashStyleHelper
 dashStyle, int width, bool drawOnPricePanel)
Draw.ExtendedLine(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
startTime, double startY, DateTime endTime, double endY, Brush brush, DashStyleHelper
dashStyle, int width, bool drawOnPricePanel)
Draw.ExtendedLine(NinjaScriptBase owner, string tag, int startBarsAgo, double startY,
int endBarsAgo, double endY, bool isGlobal, string templateName)
Draw.ExtendedLine(NinjaScriptBase owner, string tag, DateTime startTime, double
startY, DateTime endTime, double endY, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back

startTime	The starting time where the draw object will be drawn
startY	The starting y value co-ordinate where the draw object will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
endY	The end y value co-ordinate where the draw object will terminate
brush	The brush used to color draw object (reference)
dashStyle	DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid **Note**: Drawing objects with y values very far off the visible canvas can lead to performance hits. Fancier DashStyles like DashDotDot will also require more resources than simple DashStyles like Solid.
width	The width of the draw object
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a dotted lime green
Draw.ExtendedLine(this, "tag1", 10, Close[10], 0, Close[0],
Brushes.LimeGreen, DashStyleHelper.Dot, 2);
```

11.6.2.7.9.1 ExtendedLine

Definition

Represents an interface that exposes information regarding an Extended Line IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
Stroke	A Stroke object used to draw the object

Example

```
// Instantiate a dotted lime green Extended Line
ExtendedLine myLine = Draw.ExtendedLine(this, "tag1", 10,
Close[10], 0, Close[0], Brushes.LimeGreen, DashStyleHelper.Dot, 2);

// Make the line a Global Drawing Object
myLine.IsGlobalDrawingTool = true;
```

11.6.2.7.10 Draw .FibonacciCircle()

Definition

Draws a fibonacci circle.

Method Return Value

A FibonacciCircle object that represents the draw object.

Syntax

```
Draw.FibonacciCircle(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
startTime, double startY, DateTime endTime, double endY)
Draw.FibonacciCircle(NinjaScriptBase owner, string tag, bool isAutoScale, int
startBarsAgo, double startY, int endBarsAgo, double endY)
```

```
Draw.FibonacciCircle(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
startTime, double startY, DateTime endTime, double endY, bool isGlobal, string
templateName)
Draw.FibonacciCircle(NinjaScriptBase owner, string tag, bool isAutoScale, int
startBarsAgo, double startY, int endBarsAgo, double endY, bool isGlobal, string
templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn
startY	The starting y value co-ordinate where the draw object will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate

endY	The end y value co-ordinate where the draw object will terminate
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a Fibonacci circle
Draw.FibonacciCircle(this, "tag1", true, 10, Low[10], 0, High[0]);
```

11.6.2.7.10.1 FibonacciCircle

Definition

Represents an interface that exposes information regarding a Fibonacci Circle IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
PriceLevels	A collection of prices calculated by the drawing object
IsTimePriceDividedSeparately	A `bool` value which when true determines if the time and price are calculated together as a ratio, or independently
IsTextDisplayed	A `bool` value determining if the draw object should display text on the chart.

Example

```
// Instantiate a Fibonacci circle
FibonacciCircle myFibCirc = Draw.FibonacciCircle(this, "tag1",
true, 10, Low[10], 0, High[0]);

// Ensure that text is being displayed on the Drawing Object
myFibCirc.IsTextDisplayed = true;
```

11.6.2.7.11 Draw .FibonacciExtensions()

Definition

Draws a fibonacci extension.

Method Return Value

A FibonacciExtensions object that represents the draw object.

Syntax

```
Draw.FibonacciExtensions(NinjaScriptBase owner, string tag, bool isAutoScale, int
startBarsAgo, double startY, int endBarsAgo, double endY, int extensionBarsAgo, double
 extensionY)
Draw.FibonacciExtensions(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
 startTime, double startY, DateTime endTime, double endY, DateTime extensionTime,
double extensionY)
Draw.FibonacciExtensions(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
 startTime, double startY, DateTime endTime, double endY, DateTime extensionTime,
double extensionY, bool isGlobal, string templateName)
Draw.FibonacciExtensions(NinjaScriptBase owner, string tag, bool isAutoScale, int
startBarsAgo, double startY, int endBarsAgo, double endY, int extensionBarsAgo, double
 extensionY, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.

isAutoScale	Determines if the draw object will be included in the y-axis scale
startBarsAgo	The number of bars ago (x value) of the 1st anchor point
startTime	The time of the 1st anchor point
startY	The y value of the 1st anchor point
endBarsAgo	The number of bars ago (x value) of the 2nd anchor point
endTime	The time of the 2nd anchor point
endY	The y value of the 2nd anchor point
extensionBarsAgo	The number of bars ago (x value) of the 3rd anchor point
extensionTime	The time of the 3rd anchor point
extensionY	The y value of the 3rd anchor point
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a fibonnaci extension
Draw.FibonacciExtensions(this, "tag1", true, 4, Low[4], 3, High[3],
 1, Low[1]);
```

11.6.2.7.11.1 FibonacciExtensions

Definition

Represents an interface that exposes information regarding a Fibonacci Extensions IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
ExtensionAnchor	An IDrawingTool's ChartAnchor representing the extension point of the drawing object
PriceLevels	A collection of prices calculated by the drawing object
TextLocation	An `enum` determining the text location; can be set to TextLocation.Off to remove text
IsExtendedLinesLeft	A `bool` value determining if the draw object should draw lines to the far left side of the screen
IsExtendedLinesRight	A `bool` value determining if the draw object should draw lines to the far right side of the screen

Example

```
// Instantiates a Fibonnaci Extension
FibonacciExtensions myFibExt = Draw.FibonacciExtensions(this,
"tag1", true, 4, Low[4], 3, High[3], 1, Low[1]);

// Extend the Fibonacci Extension oject's lines to the right
myFibExt.IsExtendedLinesRight = true;
```

11.6.2.7.12 Draw .FibonacciRetracements()

Definition

Draws a fibonacci retracement.

Method Return Value

A FibonacciRetracements object that represents the draw object.

Syntax

```
Draw.FibonacciRetracements(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo, double startY, int endBarsAgo, double endY)
Draw.FibonacciRetracements(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime, double startY, DateTime endTime, double endY)
Draw.FibonacciRetracements(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime, double startY, DateTime endTime, double endY, bool isGlobal, string templateName)
Draw.FibonacciRetracements(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo, double startY, int endBarsAgo, double endY, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn.
startY	The starting y value co-ordinate where the draw object will be drawn

endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
endY	The end y value co-ordinate where the draw object will terminate
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a fibonnaci retracement
Draw.FibonacciRetracements(this, "tag1", true, 10, Low[10], 0,
High[0]);
```

11.6.2.7.12.1 FibonacciRetracements

Definition

Represents an interface that exposes information regarding a Fibonacci Retracements IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
PriceLevels	A collection of prices calculated by the drawing object
TextLocation	An enum determining the text location; can be set to TextLocation.Off to remove text

IsExtendedLinesLeft	A `bool` value determining if the draw object should draw lines to the far left side of the screen
IsExtendedLinesRight	A `bool` value determining if the draw object should draw lines to the far right side of the screen

Example

```
// Instantiate a FibonacciRetracements object
FibonacciRetracements myFibRet = Draw.FibonacciRetracements(this,
"tag1", true, 10, Low[10], 0, High[0]);

// Set the object's lines to extend to the right
myFibRet.IsExtendedLinesRight = true;
```

11.6.2.7.13 Draw .FibonacciTimeExtensions()

Definition

Draws a fibonacci time extension.

Method Return Value

A FibonacciTimeExtensions object that represents the draw object.

Syntax

```
Draw.FibonacciTimeExtensions(NinjaScriptBase owner, string tag, bool isAutoScale,
DateTime startTime, double startY, DateTime endTime, double endY)
Draw.FibonacciTimeExtensions(NinjaScriptBase owner, string tag, bool isAutoScale, int
startBarsAgo, double startY, int endBarsAgo, double endY)
Draw.FibonacciTimeExtensions(NinjaScriptBase owner, string tag, bool isAutoScale,
DateTime startTime, double startY, DateTime endTime, double endY, bool isGlobal,
string templateName)
Draw.FibonacciTimeExtensions(NinjaScriptBase owner, string tag, bool isAutoScale, int
startBarsAgo, double startY, int endBarsAgo, double endY, bool isGlobal, string
templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")

tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn.
startY	The starting y value co-ordinate where the draw object will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
endY	The end y value co-ordinate where the draw object will terminate
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a fibonacci time extension object
Draw.FibonacciTimeExtensions(this, "tag1", false, 10, Low[10], 0,
High[0]);
```

11.6.2.7.13.1 FibonacciTimeExtensions

Definition

Represents an interface that exposes information regarding a Fibonacci Time Extensions IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
PriceLevels	A collection of prices calculated by the drawing object
IsTextDisplayed	A `bool` value determining if the draw object should display text on the chart.
IsExtendedLinesLeft	A `bool` value determining if the draw object should draw lines to the far left side of the screen
IsExtendedLinesRight	A `bool` value determining if the draw object should draw lines to the far right side of the screen

Example

```
// Instantiate a FibonacciTimeExtensions object
FibonacciTimeExtensions myFibTime =
Draw.FibonacciTimeExtensions(this, "tag1", false, 10, Low[10], 0,
High[0]);

// Instantiate a new PriceLevel to be used in the step below
PriceLevel myLevel = new PriceLevel(99, Brushes.Black);

// Change the object's price level at index 3
myFibTime.PriceLevels[3] = myLevel;
```

11.6.2.7.14 Draw .GannFan()

Definition

Draws a Gann Fan.

Method Return Value

A GannFan object that represents the draw object.

Syntax

```
Draw.GannFan(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double
y)
Draw.GannFan(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y)
Draw.GannFan(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double
y, bool isGlobal, string templateName)
Draw.GannFan(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.

isAutoScale	Determines if the draw object will be included in the y-axis scale
barsAgo	The bar the object will be drawn at. A value of 10 would be 10 bars ago.
time	The time the object will be drawn at.
y	The y value
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a Gann Fan at the current bar low
Draw.GannFan(this, "tag1", true, 0, Low[0]);
```

11.6.2.7.14.1 GannFan

Definition

Represents an interface that exposes information regarding a Gann Fan IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
PriceLevels	A collection of prices calculated by the drawing object
GannFanDirection	Possible values: GannFanDirection.DownLeft GannFanDirection.DownRight GannFanDirection.UpLeft

	GannFanDirection.UpRight
PointsPerBar	A `double` value representing the number of points per bar
IsTextDisplayed	A `bool` value representing if text will be drawn along with the draw object

Example

```
// Instantiate a GannFan object
GannFan myFan = Draw.GannFan(this, "tag1", true, 0, Low[0]);

// Instantiate a new PriceLevel to be used in the step below
PriceLevel myLevel = new PriceLevel(99, Brushes.Black);

// Change the object's price level at index 3
myFan.PriceLevels[3] = myLevel;
```

11.6.2.7.15 Draw .HorizontalLine()

Definition

Draws a horizontal line.

Method Return Value

A HorizontalLine object that represents the draw object.

Syntax

```
Draw.HorizontalLine(NinjaScriptBase owner, string tag, double y, Brush brush)
Draw.HorizontalLine(NinjaScriptBase owner, string tag, bool isAutoScale, double y,
Brush brush, DashStyleHelper dashStyle, int width)
Draw.HorizontalLine(NinjaScriptBase owner, string tag, bool isAutoscale, double y,
Brush brush, bool drawOnPricePanel)
Draw.HorizontalLine(NinjaScriptBase owner, string tag, double y, Brush brush,
DashStyleHelper dashStyle, int width, bool drawOnPricePanel)
Draw.HorizontalLine(NinjaScriptBase owner, string tag, double y, bool isGlobal, string
 templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method

	Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
y	The y value
brush	The brush used to color draw object (reference)
dashStyle	DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid **Note**: Fancier DashStyles like DashDotDot will require more resources than simple DashStyles like Solid.
width	The width of the draw object
isDrawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a horizontal line
Draw.HorizontalLine(this, "tag1", 1000, Brushes.Black);
```

11.6.2.7.15.1 HorizontalLine

Definition

Represents an interface that exposes information regarding a Horizontal Line IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
Stroke	A Stroke object used to draw the object

Example

```
// Instantiate a HorizontalLine object
HorizontalLine myLine = Draw.HorizontalLine(this, "tag1", 1000,
Brushes.Black);

// Set a new Stroke for the object
myLine.Stroke = new Stroke(Brushes.Green, DashStyleHelper.Dash, 5);
```

11.6.2.7.16 Draw .Line()

Definition

Draws a line between two points.

Method Return Value

A Line object that represents the draw object.

Syntax

```
Draw.Line(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
endBarsAgo, double endY, Brush brush)
Draw.Line(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int endBarsAgo, double endY, Brush brush, DashStyleHelper dashStyle,
int width)
Draw.Line(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime,
double startY, DateTime endTime, double endY, Brush brush, DashStyleHelper dashStyle,
int width)
Draw.Line(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int endBarsAgo, double endY, Brush brush, DashStyleHelper dashStyle,
```

```
int width, bool drawOnPricePanel)
Draw.Line(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime,
double startY, DateTime endTime, double endY, Brush brush, DashStyleHelper dashStyle,
int width, bool drawOnPricePanel)
Draw.Line(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime,
double startY, DateTime endTime, double endY, string templateName)
Draw.Line(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int endBarsAgo, double endY, string templateName)
Draw.Line(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int endBarsAgo, double endY, bool isGlobal, string templateName)
Draw.Line(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime,
double startY, DateTime endTime, double endY, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn
startY	The starting y value co-ordinate where the draw object will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate

endTime	The end time where the draw object will terminate
endY	The end y value co-ordinate where the draw object will terminate
brush	The brush used to color draw object (reference)
dashStyle	DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid **Note**: Drawing objects with y values very far off the visible canvas can lead to performance hits. Fancier DashStyles like DashDotDot will also require more resources than simple DashStyles like Solid.
width	The width of the draw object
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a dotted lime green line from 10 bars back to the current
bar
// with a width of 2 pixels
Draw.Line(this, "tag1", false, 10, 1000, 0, 1001,
Brushes.LimeGreen, DashStyleHelper.Dot, 2);
```

11.6.2.7.16.1 Line

Definition
Represents an interface that exposes information regarding a Line IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
Stroke	A Stroke object used to draw the object

Example

```
// Instantiate a Line object
NinjaTrader.NinjaScript.DrawingTools.Line myLine = Draw.Line(this,
"tag1", false, 10, 1000, 0, 1001, Brushes.LimeGreen,
DashStyleHelper.Dot, 2);

// Set a new Stroke for the object
myLine.Stroke = new Stroke(Brushes.Green, DashStyleHelper.Dash, 5);
```

Note: To differentiate between NinjaTrader.NinjaScript.DrawingTools.Line and NinjaTrader.Gui.Line when assigning a Line object, you will need to invoke the former path explicitly, as seen in the example above.

11.6.2.7.17 Draw .PathTool()

Definition
Draws a path which can have a user defined set of anchors.

Method Return Value
A PathTool object that represents the draw object.

Syntax
```
Draw.PathTool(NinjaScriptBase owner, string tag, bool isAutoScale, int anchor1BarsAgo,
double anchor1Y, int anchor2BarsAgo, double anchor2Y, int anchor3BarsAgo, double
anchor3Y)
```

```
Draw.PathTool(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
Anchor1Time, double anchor1Y, DateTime Anchor2Time, double anchor2Y, DateTime
Anchor3Time, double anchor3Y)
Draw.PathTool(NinjaScriptBase owner, string tag, bool isAutoScale, int anchor1BarsAgo,
double anchor1Y, int anchor2BarsAgo, double anchor2Y, int anchor3BarsAgo, double
anchor3Y, int anchor4BarsAgo, double anchor4Y)
Draw.PathTool(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
Anchor1Time, double anchor1Y, DateTime Anchor2Time, double anchor2Y, DateTime
Anchor3Time, double anchor3Y, DateTime Anchor4Time, double anchor4Y)
Draw.PathTool(NinjaScriptBase owner, string tag, bool isAutoScale, int anchor1BarsAgo,
double anchor1Y, int anchor2BarsAgo, double anchor2Y, int anchor3BarsAgo, double
anchor3Y, int anchor4BarsAgo, double anchor4Y, int anchor5BarsAgo, double anchor5Y)
Draw.PathTool(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
Anchor1Time, double anchor1Y, DateTime Anchor2Time, double anchor2Y, DateTime
Anchor3Time, double anchor3Y, DateTime Anchor4Time, double anchor4Y, DateTime
Anchor5Time, double anchor5Y)
Draw.PathTool(NinjaScriptBase owner, string tag, bool isAutoScale, List<ChartAnchor>
chartAnchors, Brush brush, DashStyleHelper dashStyle)
Draw.PathTool(NinjaScriptBase owner, string tag, bool isAutoScale, List<ChartAnchor>
chartAnchors, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
chartAnchors	A list of the chart anchors
anchor1BarsAgo	The bar the first anchor of the object will be drawn at. A value of 10 would be 10 bars ago.

anchor2BarsAgo	The bar the second anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor3BarsAgo	The bar the third anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor4BarsAgo	The bar the forth anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor5BarsAgo	The bar the fifth anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor1Y	The first anchor y value
anchor2Y	The second anchor y value
anchor3Y	The third anchor y value
anchor4Y	The forth anchor y value
anchor5Y	The fifth anchor y value
Anchor1Time	The time the first anchor of the object will be drawn at
Anchor2Time	The time the second anchor of the object will be drawn at
Anchor3Time	The time the third anchor of the object will be drawn at
Anchor4Time	The time the forth anchor of the object will be drawn at
Anchor5Time	The time the fifth anchor of the object will be drawn at
Brush	The brush used to color draw object (reference)
templateName	The name of the drawing tool template the object will use to determine various visual properties

	(empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a PathTool object based on bars ago and y anchors
Draw.PathTool(this, "tag1", false, 20, 194, 10, 184, 13, 176, 25,
182);

// Draws a PathTool object based on a list of anchors with
specified times
List<ChartAnchor> anchors = new List<ChartAnchor>();
anchors.Add(new ChartAnchor(new DateTime(2018, 5, 25), 194,
ChartControl));
anchors.Add(new ChartAnchor(new DateTime(2018, 6, 12), 184,
ChartControl));
anchors.Add(new ChartAnchor(new DateTime(2018, 6, 7), 176,
ChartControl));
anchors.Add(new ChartAnchor(new DateTime(2018, 5, 21), 182,
ChartControl));

Draw.PathTool(this, "tag1", false, anchors, Brushes.CornflowerBlue,
DashStyleHelper.Solid);
```

11.6.2.7.17.1 PathTool

Definition

Represents an interface that exposes information regarding a PathTool IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
Stroke	A Stroke object used to draw the object

Example

```
// Instantiate a PathTool object
PathTool myPathTool = Draw.PathTool(this, "tag1", false, 20, 194,
10, 184, 13, 176, 25, 182);
```

11.6.2.7.18 Draw .Polygon()

Definition

Draws a polygon which can have a user defined set of anchors.

Method Return Value

A Polygon object that represents the draw object.

Syntax

```
Draw.Polygon(NinjaScriptBase owner, string tag, bool isAutoScale, List<ChartAnchor>
chartAnchors, bool isGlobal, string templateName)
Draw.Polygon(NinjaScriptBase owner, string tag, bool isAutoScale, List<ChartAnchor>
chartAnchors, Brush brush, DashStyleHelper dashStyle, Brush areaBrush, int
areaOpacity)
Draw.Polygon(NinjaScriptBase owner, string tag, bool isAutoScale, int anchor1BarsAgo,
double anchor1Y, int anchor2BarsAgo, double anchor2Y, int anchor3BarsAgo, double
anchor3Y, int anchor4BarsAgo, double anchor4Y)
Draw.Polygon(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
Anchor1Time, double anchor1Y, DateTime Anchor2Time, double anchor2Y, DateTime
Anchor3Time, double anchor3Y, DateTime Anchor4Time, double anchor4Y)
Draw.Polygon(NinjaScriptBase owner, string tag, bool isAutoScale, int anchor1BarsAgo,
double anchor1Y, int anchor2BarsAgo, double anchor2Y, int anchor3BarsAgo, double
anchor3Y, int anchor4BarsAgo, double anchor4Y, int anchor5BarsAgo, double anchor5Y)
Draw.Polygon(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
Anchor1Time, double anchor1Y, DateTime Anchor2Time, double anchor2Y, DateTime
Anchor3Time, double anchor3Y, DateTime Anchor4Time, double anchor4Y, DateTime
Anchor5Time, double anchor5Y)
Draw.Polygon(NinjaScriptBase owner, string tag, bool isAutoScale, int anchor1BarsAgo,
double anchor1Y, int anchor2BarsAgo, double anchor2Y, int anchor3BarsAgo, double
anchor3Y, int anchor4BarsAgo, double anchor4Y, int anchor5BarsAgo, double anchor5Y,
int anchor6BarsAgo, double anchor6Y)
Draw.Polygon(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
Anchor1Time, double anchor1Y, DateTime Anchor2Time, double anchor2Y, DateTime
Anchor3Time, double anchor3Y, DateTime Anchor4Time, double anchor4Y, DateTime
Anchor5Time, double anchor5Y, DateTime Anchor6Time, double anchor6Y)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method

	Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
chartAnchors	A list of the chart anchors
anchor1BarsAgo	The bar the first anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor2BarsAgo	The bar the second anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor3BarsAgo	The bar the third anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor4BarsAgo	The bar the forth anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor5BarsAgo	The bar the fifth anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor6BarsAgo	The bar the sixth anchor of the object will be drawn at. A value of 10 would be 10 bars ago.
anchor1Y	The first anchor y value
anchor2Y	The second anchor y value
anchor3Y	The third anchor y value
anchor4Y	The forth anchor y value

anchor5Y	The fifth anchor y value
anchor6Y	The sixth anchor y value
Anchor1Time	The time the first anchor of the object will be drawn at
Anchor2Time	The time the second anchor of the object will be drawn at
Anchor3Time	The time the third anchor of the object will be drawn at
Anchor4Time	The time the forth anchor of the object will be drawn at
Anchor5Time	The time the fifth anchor of the object will be drawn at
Anchor6Time	The time the sixth anchor of the object will be drawn at
areaBrush	The brush used to color draw object (reference)
areaOpacity	Sets the level of transparency for the fill color. Valid values between 0 - 100. (0 = completely transparent, 100 = no opacity)
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a Polygon object based on bars ago and y anchors
Draw.Polygon(this, "tag1", false, 20, 194, 10, 184, 13, 176, 25,
182);

// Draws a Polygon object based on a list of anchors with specified
times
List<ChartAnchor> anchors = new List<ChartAnchor>();
anchors.Add(new ChartAnchor(new DateTime(2018, 5, 25), 194,
ChartControl));
anchors.Add(new ChartAnchor(new DateTime(2018, 6, 12), 184,
ChartControl));
anchors.Add(new ChartAnchor(new DateTime(2018, 6, 7), 176,
ChartControl));
anchors.Add(new ChartAnchor(new DateTime(2018, 5, 21), 182,
ChartControl));

Draw.Polygon(this, "tag1", false, anchors, Brushes.CornflowerBlue,
DashStyleHelper.Solid, Brushes.CornflowerBlue, 40);
```

11.6.2.7.18.1 Polygon

Definition

Represents an interface that exposes information regarding a Polyon IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
Stroke	A Stroke object used to draw the object

Example

```
// Instantiate a Polygon object
Polygon myPolygon = Draw.Polygon(this, "tag1", false, 20, 194, 10,
184, 13, 176, 25, 182);

// Set a new area brush for the object
myPolygon.AreaBrush = Brushes.Green;
```

11.6.2.7.19 Draw.Ray()

Definition

Draws a line which has an infinite end point in one direction.

Method Return Value

A Ray object that represents the draw object.

Syntax

```
Draw.Ray(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
endBarsAgo, double endY, Brush brush)
Draw.Ray(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo, double
 startY, int endBarsAgo, double endY, Brush brush, DashStyleHelper dashStyle, int
width)
Draw.Ray(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime endTime, double endY, Brush brush)
Draw.Ray(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime endTime, double endY, Brush brush, DashStyleHelper dashStyle, int width)
Draw.Ray(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo, double
 startY, int endBarsAgo, double endY, Brush brush, DashStyleHelper dashStyle, int
width, bool drawOnPricePanel)
Draw.Ray(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime endTime, double endY, Brush brush, DashStyleHelper dashStyle, int width, bool
 drawOnPricePanel)
Draw.Ray(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
endBarsAgo, double endY, bool isGlobal, string templateName)
Draw.Ray(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime endTime, double endY, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.

isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The number of bars ago (x value) of the 1st anchor point
startTime	The time of the 1st anchor point
startY	The y value of the 1st anchor point
endBarsAgo	The number of bars ago (x value) of the 2nd anchor point
endTime	The time of the 2nd anchor point
endY	The y value of the 2nd anchor point
brush	The brush used to color draw object (reference)
dashStyle	DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid **Note**: Drawing objects with y values very far off the visible canvas can lead to performance hits. Fancier DashStyles like DashDotDot will also require more resources than simple DashStyles like Solid.
width	The width of the draw object
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties

	(empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a lime green ray from 10 bars back through the current bar
Draw.Ray(this, "tag1", 10, 1000, 0, 1001, Brushes.LimeGreen);
```

11.6.2.7.19.1 Ray

Definition

Represents an interface that exposes information regarding a Ray IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
Stroke	A Stroke object used to draw the object

Example

```
// Instantiate a Ray object
Ray myRay = Draw.Ray(this, "tag1", 10, 1000, 0, 1001,
Brushes.LimeGreen);

// Set a new Stroke for the object
myRay.Stroke = new Stroke(Brushes.Green, DashStyleHelper.DashDot,
3);
```

11.6.2.7.20 Draw .Rectangle()

Definition

Draws a rectangle.

Method Return Value

A Rectangle object that represents the draw object.

Syntax

```
Draw.Rectangle(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
 endBarsAgo, double endY, Brush brush)
Draw.Rectangle(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime endTime, double endY, Brush brush)
Draw.Rectangle(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int endBarsAgo, double endY, Brush brush, Brush areaBrush, int
areaOpacity)
Draw.Rectangle(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
startTime, double startY, DateTime endTime, double endY, Brush brush, Brush areaBrush,
 int areaOpacity)
Draw.Rectangle(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
 endBarsAgo, double endY, Brush brush, bool drawOnPricePanel)
Draw.Rectangle(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int endBarsAgo, double endY, Brush brush, Brush areaBrush, int
areaOpacity, bool drawOnPricePanel)
Draw.Rectangle(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
startTime, double startY, DateTime endTime, double endY, Brush brush, Brush areaBrush,
 int areaOpacity, bool drawOnPricePanel)
Draw.Rectangle(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
 endBarsAgo, double endY, bool isGlobal, string templateName)
Draw.Rectangle(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime endTime, double endY, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.

startTime	The starting time where the draw object will be drawn
startY	The starting y value co-ordinate where the draw object will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
endY	The end y value co-ordinate where the draw object will terminate
brush	The brush used to color the outline of draw object (reference)
areaBrush	The brush used to color the fill area of the draw object (reference)
areaOpacity	Sets the level of transparency for the fill color. Valid values between 0 - 100. (0 = completely transparent, 100 = no opacity)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a blue rectangle from the low 10 bars back to the high of
5 bars back
Draw.Rectangle(this, "tag1", 10, Low[10] - TickSize, 5, High[5] +
TickSize, Brushes.Blue);

// Draws a blue rectangle from the low 10 bars back to the high of
5 bars back with
// a fill color or pale green with a transparency level of 2
Draw.Rectangle(this, "tag1", false, 10, Low[10] - TickSize, 5,
High[5] + TickSize, Brushes.PaleGreen, Brushes.PaleGreen, 2);
```

11.6.2.7.20.1 Rectangle

Definition

Represents an interface that exposes information regarding a Rectangle IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
AreaBrush	A Brush object representing the fill color of the draw object
AreaOpacity	An `int` value representing the opacity of the area color
OutlineStroke	The Stroke object used to draw the object's outline

Example

```
// Instantiate a Rectangle object
Rectangle myRec = Draw.Rectangle(this, "tag1", 10, Low[10] -
TickSize, 5, High[5] + TickSize, Brushes.Blue);

// Set the object's AreaBrush to Blue
myRec.AreaBrush = Brushes.Blue;
```

11.6.2.7.21 Draw .Region()

Definition

Draws a region on a chart.

Method Return Value

A Region object that represents the draw object.

Syntax

```
Draw.Region(NinjaScriptBase owner, string tag, int startBarsAgo,
         int endBarsAgo, ISeries<double> series, double price, Brush areaBrush, int
areaOpacity,  int displacement = 0)
Draw.Region(NinjaScriptBase owner, string tag, int startBarsAgo,
         int endBarsAgo, ISeries<double> series1, ISeries<double> series2, Brush
outlineBrush,
         Brush areaBrush, int areaOpacity, [int displacement])
Draw.Region(NinjaScriptBase owner, string tag, DateTime startTime,
         DateTime endTime, ISeries<double> series, double price, Brush areaBrush, int
areaOpacity)
Draw.Region(NinjaScriptBase owner, string tag, DateTime startTime,
         DateTime endTime, ISeries<double> series1, ISeries<double> series2, Brush
outlineBrush, Brush areaBrush, int areaOpacity)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn.

endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
series, series1, series2	Any `Series<double>` type object such as an indicator, Close, High, Low etc.. The value of the object will represent a y value.
price	Any `double` value
outlineBrush	The brush used to color the region outline of draw object (reference)
areaBrush	The brush used to color the fill region area of the draw object (reference)
areaOpacity	Sets the level of transparency for the fill color. Valid values between 0 - 100. (0 = completely transparent, 100 = no opacity)
displacement	An optional parameter which will offset the barsAgo value for the Series<double> value used to match the desired Displacement. Default value is 0.

Example

```
// Draw a region between upper and lower Bollinger bands
Draw.Region(this, "tag1", CurrentBar, 0, Bollinger(2, 14).Upper,
Bollinger(2, 14).Lower, null, Brushes.Blue, 50);
```

Tips:
1. Pass in `null` to the "outlineColor" parameter if you do not want to have an outline color.
2. If you wanted to fill a region between a value (20 period simple moving average) and the upper edge of the chart, pass in an extreme value to the "y" parameter such as 1000000.
3. Should you be drawing regions based on `Series<double>` objects instead of indicator plots, be sure to create the `Series<double>` with the MaximumBarsLookBack.Infinite

parameter if the region you are drawing would be maintained on the chart for more than 256 bars back.

11.6.2.7.21.1 Region

Definition

Represents an interface that exposes information regarding a Region IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
AreaOpacity	An `int` value representing the opacity of the area color
AreaBrush	A Brush object representing the fill color of the draw object
OutlineStroke	A Stroke used for the outline of the region

Example

```
// Instantiate a Region object
Region myRegion = Draw.Region(this, "tag1", CurrentBar, 0,
Bollinger(2, 14).Upper, Bollinger(2, 14).Lower, null, Brushes.Blue,
 50);

// Set the object's OutlineStroke to a new Stroke
myRegion.OutlineStroke = new Stroke(Brushes.Red,
DashStyleHelper.Solid, 3);
```

11.6.2.7.22 Draw .RegionHighlightX()

Definition

Draws a region highlight x on a chart.

Method Return Value

A RegionHighlightX object that represents the draw object.

Syntax

```
Draw.RegionHighlightX(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
endTime, Brush brush)
Draw.RegionHighlightX(NinjaScriptBase owner, string tag, int startBarsAgo, int
endBarsAgo, Brush brush)
Draw.RegionHighlightX(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
endTime, Brush brush, Brush areaBrush, int areaOpacity)
Draw.RegionHighlightX(NinjaScriptBase owner, string tag, int startBarsAgo, int
endBarsAgo, Brush brush, Brush areaBrush, int areaOpacity)
Draw.RegionHighlightX(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
endTime, bool isGlobal, string templateName)
Draw.RegionHighlightX(NinjaScriptBase owner, string tag, int startBarsAgo, int
endBarsAgo, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn.
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate

brush	The brush used to color the outline of draw object (reference)
areaBrush	The brush used to color the fill area of the draw object (reference)
areaOpacity	Sets the level of transparency for the fill color. Valid values between 0 - 100. (0 = completely transparent, 100 = no opacity)
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Fills in the region between the startBar and endBar
Draw.RegionHighlightX(this, "tag1", 10, 0, Brushes.Blue);
```

11.6.2.7.22.1 RegionHighlightX

Definition

Represents an interface that exposes information regarding a Region Highlight X IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
AreaBrush	A Brush class representing the fill color of the draw object

AreaOpacity	An `int` value representing the opacity of the area color
OutlineStroke	The Stroke object used to draw the object's outline

Example

```
// Instantiate a RegionHighlightX object
RegionHighlightX myReg = Draw.RegionHighlightX(this, "tag1", 10, 0,
 Brushes.Blue);

// Change the object's opacity
myReg.AreaOpacity = 25;
```

11.6.2.7.23 Draw .RegionHighlightY()

Definition

Draws a region highlight y on a chart.

Method Return Value

A RegionHighlightY object that represents the draw object.

Syntax

```
Draw.RegionHighlightY(NinjaScriptBase owner, string tag, double startY, double endY,
Brush brush)
Draw.RegionHighlightY(NinjaScriptBase owner, string tag, bool isAutoScale, double
startY, double endY, Brush brush, Brush areaBrush, int areaOpacity)
Draw.RegionHighlightY(NinjaScriptBase owner, string tag, double startY, double endY,
bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object.

	For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startY	The starting y value co-ordinate where the draw object will be drawn
endY	The ending y value co-ordinate where the draw object will be drawn
brush	The brush used to color the outline of draw object (reference)
areaBrush	The brush used to color the fill area of the draw object (reference)
areaOpacity	Sets the level of transparency for the fill color. Valid values between 0 - 100. (0 = completely transparent, 100 = no opacity)
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Fills in the region between the startY and endY
Draw.RegionHighlightY(this, "tag1",
true, High[0], Low[0], Brushes.Blue, Brushes.Green, 20);
```

11.6.2.7.23.1 RegionHighlightY

Definition

Represents an interface that exposes information regarding a Region Highlight Y IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
AreaBrush	A Brush class representing the fill color of the draw object
AreaOpacity	An `int` value representing the opacity of the area color
OutlineStroke	The Stroke object used to draw the object's outline

Example

```
// Instantiate a RegionHighlightX object
RegionHighlightY myReg = Draw.RegionHighlightY(this, "tag1", 10, 0,
 Brushes.Blue);

// Change the object's opacity
myReg.AreaOpacity = 25;
```

11.6.2.7.24 Draw .RegressionChannel()

Definition

Draws a regression channel.

Method Return Value

A RegressionChannel object that represents the draw object.

Syntax

```
Draw.RegressionChannel(NinjaScriptBase owner, string tag, int startBarsAgo, int
endBarsAgo, Brush brush)
Draw.RegressionChannel(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
 endTime, Brush brush)
```

```
Draw.RegressionChannel(NinjaScriptBase owner, string tag, bool isAutoScale, int
startBarsAgo, int endBarsAgo, Brush upperBrush, DashStyleHelper upperDashStyleHelper,
int upperWidth, Brush middleBrush, DashStyleHelper middleDashStyleHelper, int
middleWidth, Brush lowerBrush, DashStyleHelper lowerDashStyleHelper, int lowerWidth)
Draw.RegressionChannel(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
startTime, DateTime endTime, Brush upperBrush, DashStyleHelper upperDashStyleHelper,
int upperWidth, Brush middleBrush, DashStyleHelper middleDashStyleHelper, int
middleWidth, Brush lowerBrush, DashStyleHelper lowerDashStyleHelper, int lowerWidth)
Draw.RegressionChannel(NinjaScriptBase owner, string tag, int startBarsAgo, int
endBarsAgo, bool isGlobal, string templateName)
Draw.RegressionChannel(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
 endTime, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn.
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate

brush	The brush used to color the outline of draw object (reference)
upperDashStyle, middleDashStyle, lowerDashStyle	DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid **Note**: Fancier DashStyles like DashDotDot will require more resources than simple DashStyles like Solid.
upperBrush, middleBrush, lowerBrush	The line colors (reference)
upperWidth, middleWidth, lowerWidth	The line width
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a regression channel from the low 10 bars back to the high of 5 bars back
Draw.RegressionChannel(this, "tag1", 10, 0, Brushes.Blue);
```

11.6.2.7.24.1 RegressionChannel

Definition

Represents an interface that exposes information regarding a Regression Channel IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
RegressionStroke	The Stroke object used to draw the middle line of the object
LowerChannelStroke	The Stroke object used to draw the lower line of the object
UpperChannelStroke	The Stroke object used to draw the upper line of the object
PriceType	Possible values are: PriceType.Close PriceType.High PriceType.Low PriceType.Median PriceType.Open PriceType.Typical
ChannelType	An `enum` value representing if the object will use standard deviations calculations for the upper/lower lines. Possible values are • RegressionChannelType.Segment, • RegressionChannelType.StandardDeviation
ExtendLeft	A `bool` value representing if the object will extend to the left
ExtendRight	A `bool` value representing if the object will extend to the right
StandardDeviationLowerDistance	A `double` value representing the standard deviation distance to the lower line
StandardDeviationUpperDistance	A `double` value representing the standard deviation distance to the upper line

Example

```
// Instantiate a RegressionChannel object
NinjaTrader.NinjaScript.DrawingTools.RegressionChannel myRegChan =
Draw.RegressionChannel(this, "tag1", 10, 0, Brushes.Blue);

// Change the object's PriceType
myRegChan.PriceType = PriceType.Median;
```

Note: To differentiate between DrawingTools.RegressionChannel and Indicators.RegressionChannel when assigning a RegressionChannel object, you will need to invoke the former path explicitly, as seen in the example above.

11.6.2.7.25 Draw.RiskReward()

Definition

Draws a risk/reward on a chart.

Method Return Value

A RiskReward object that represents the draw object.

Syntax

```
Draw.RiskReward(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
entryTime, double entryY, DateTime endTime, double endY, double ratio, bool isStop)
Draw.RiskReward(NinjaScriptBase owner, string tag, bool isAutoScale, int entryBarsAgo
, double entryY, int endBarsAgo, double endY, double ratio, bool isStop)
Draw.RiskReward(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
entryTime, double entryY, DateTime endTime, double endY, double ratio, bool isStop,
bool isGlobal, string templateName)
Draw.RiskReward(NinjaScriptBase owner, string tag, bool isAutoScale, int entryBarsAgo
, double entryY, int endBarsAgo, double endY, double ratio, bool isStop, bool
isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")

tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
entryTime	The time where the draw object's entry will be drawn.
entryBarsAgo	The starting bar (x axis co-ordinate) where the draw object's entry will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
entryY	The y value co-ordinate where the draw object's entry price will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
endY	The starting y value co-ordinate where the draw object will be drawn
ratio	An `double` value determining the calculated ratio between the risk and reward based on the entry point. Example: reward : risk is ratio of 1.0
isStop	A `bool` value, when true will use the endTime / endBarsAgo and endY to set the stop and will automatically calculate the target based off the ratio value. When false, will set the target and will automatically calculate the stop based off the ratio value.

isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// draw a risk/reward tool starting from the current bar to 10 bars
ago
// with calcuate a ratio of 2 based on stop level
Draw.RiskReward(this, "tag1", false, 0, High[0], 10, Low[0], 2,
true);
```

11.6.2.7.25.1 RiskReward

Definition

Represents an interface that exposes information regarding a Risk Reward IDrawingTool.

Methods and Properties

EntryAnchor	An IDrawingTool's ChartAnchor representing the entry point of the drawing object
RiskAnchor	An IDrawingTool's ChartAnchor representing the stop loss point of the drawing object
RewardAnchor	An IDrawingTool's ChartAnchor representing the profit target point of the drawing object
Ratio	An `int` value determining the calculated ratio between the risk or reward based on the entry point

Example

```
// Instantiate a RiskReward object
RiskReward myRR = Draw.RiskReward(this, "tag1", false, 0, High[0],
10, Low[0], 2, true);

// Change the object's risk/reward ratio to 2:1
myRR.Ratio = 2;
```

11.6.2.7.26 Draw .Ruler()

Definition

Draws a ruler.

Method Return Value

A Ruler object that represents the draw object.

Syntax

```
Draw.Ruler(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int endBarsAgo, double endY, int textBarsAgo, double textY)
Draw.Ruler(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime,
double startY, DateTime endTime, double endY, DateTime textTime, double textY)
Draw.Ruler(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int endBarsAgo, double endY, int textBarsAgo, double textY, bool
isGlobal, string templateName)
Draw.Ruler(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime,
double startY, DateTime endTime, double endY, DateTime textTime, double textY, bool
isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.

isAutoScale	Determines if the draw object will be included in the y-axis scale
startBarsAgo	The number of bars ago (x value) of the 1st anchor point
startTime	The time of the 1st anchor point
startY	The y value of the 1st anchor point
endBarsAgo	The number of bars ago (x value) of the 2nd anchor point
endTime	The time of the 2nd anchor point
endY	The y value of the 2nd anchor point
textBarsAgo	The number of bars ago (x value) of the 3rd anchor point
textTime	The time of the 3rd anchor point
textY	The y value of the 3rd anchor point
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Example

```
// Draws a ruler measuring the primary bar series
Draw.Ruler(this, "tag1", true, 4, Low[4], 3, High[3], 1,
Low[1]);
```

11.6.2.7.26.1 Ruler

Definition

Represents an interface that exposes information regarding a Ruler IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
TextAnchor	An IDrawingTool's ChartAnchor representing the text point of the drawing object
TextColor	A Brush class representing the fill color of the draw object's text area
LineColor	A Stroke object used to draw the object

Example

```
// Instantiate a Ruler object
Ruler myRuler = Draw.Ruler(this, "tag1", true, 4, Low[4], 3,
High[3], 1, Low[1]);

// Change the object's text color to white
myRuler.TextColor = Brushes.White;
```

11.6.2.7.27 Draw.Square()

Definition

Draws a square.

Method Return Value

A Square object that represents the draw object.

Syntax

```
Draw.Square(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time, double
 y, Brush brush)
Draw.Square(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double
y, Brush brush)
```

```
Draw.Square(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time, double
 y, Brush brush, bool drawOnPricePanel)
Draw.Square(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double
y, Brush brush, bool drawOnPricePanel)
Draw.Square(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time, double
 y, bool isGlobal, string templateName)
Draw.Square(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo, double
y, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale
barsAgo	The bar the object will be drawn at. A value of 10 would be 10 bars ago.
time	The time the object will be drawn at.
y	The y value
brush	The brush used to color draw object (reference)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument

templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Tip: The size of the square is tied to the char

t
'
s
B
a
r
W
i
d
t
h
a
n
d
t
h
u
s
w
i
l
l
s
c
a
l
e
a
u
t
o
m
a
t
i
c
a
l
l
y
a
s
t

h
e
c
h
a
r
t
i
s
r
e
s
i
z
e
d

Examples

```
// Paints a red square on the current bar 1 tick below the low
Draw.Square(this, "tag1", true, 0, Low[0] - TickSize, Brushes.Red);
```

11.6.2.7.27.1 Square

Definition

Represents an interface that exposes information regarding a Square IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object

OutlineBrush	A Brush used for the outline of the square
AreaBrush	A Brush object representing the fill color of the draw object

Example

```
// Instantiate a Square object
Square mySquare = Draw.Square(this, "tag1", true, 0, Low[0] -
TickSize, Brushes.Red);

// Change the object's OutlineBrush
mySquare.OutlineBrush = Brushes.Blue;
```

11.6.2.7.28 Draw .Text()

Definition

Draws text.

Method Return Value

A Text object that represents the draw object.

Syntax

```
Draw.Text(NinjaScriptBase owner, string tag, string text, int barsAgo, double y)
Draw.Text(NinjaScriptBase owner, string tag, string text, int barsAgo, double y, Brush
 textBrush)
Draw.Text(NinjaScriptBase owner, string tag, string text, int barsAgo, double y, bool
isGlobal, string templateName)
Draw.Text(NinjaScriptBase owner, string tag, bool isAutoScale, string text, int
barsAgo, double y, int yPixelOffset, Brush textBrush, SimpleFont font, TextAlignment
alignment, Brush outlineBrush, Brush areaBrush, int areaOpacity)
Draw.Text(NinjaScriptBase owner, string tag, bool isAutoScale, string text, DateTime
time, double y, int yPixelOffset, Brush textBrush, SimpleFont font, TextAlignment
alignment, Brush outlineBrush, Brush areaBrush, int areaOpacity)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")

tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale. Default value is `false`.
text	The text you wish to draw
barsAgo	The bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
time	The time where the draw object will be drawn.
y	The y co-ordinate location the object will be drawn
yPixelOffset	The offset value in pixels from within the text box area
textBrush	The brush used to color the text of the draw object (reference)
font	A Simple Font object
alignment	TextAlignment.Center, TextAlignment.Left, TextAlignment.Right, TextAlignment.Justify (reference)
outlineBrush	The brush used to color the text box outline (reference)
areaBrush	The brush used to color the text box fill area (reference)

areaOpacity	Sets the level of transparency for the fill color. Valid values between 0 - 100. (0 = completely transparent, 100 = no opacity)
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws text
Draw.Text(this, "tag1", "Text to draw", 10, 1000, Brushes.Black);
```

Tip: In some cases, it may be useful to pass in the ChartControl.Properties **TextFont** brush as well as the **LabelFont** SimpleFont object to render your custom text . This will help ensure that the text will be visible and match what a user has configured for their chart label display settings.

```
// match the text brush to what the user has configured on their
chart
Draw.Text(this, "tag1", "Text to draw", 10, 1000,
ChartControl.Properties.ChartText);
```

11.6.2.7.28.1 Text

Definition

Represents an interface that exposes information regarding a Text IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object

YPixelOffset	An `int` value representing the offset value in pixels from within the text box area
Alignment	Possible values are: TextAlignment.Center, TextAlignment.Left, TextAlignment.Right, TextAlignment.Justify (reference)
AreaOpacity	An `int` value representing the opacity of the area color
AreaBrush	A Brush class representing the fill color of the text box
Text	A `string` value representing the text to be drawn
TextBrush	A Brush class representing the color of the text
Font	A Font object representing the font for the text
OutlineStroke	The Stroke object used to outline the text box

Example

```
// Instantiate a Text object
Text myText = Draw.Text(this, "tag1", "Text to draw", 10, High[10]
+ (5 * TickSize), Brushes.Black);

// Change the object's DisplayText
myText.DisplayText = "New Display Text";
```

11.6.2.7.29 Draw .TextFixed()

Definition

Draws text in one of 5 available pre-defined fixed locations on panel 1 (price panel) of a chart. Please note the Z-Order is internally set for the method to always be drawn on top.

Method Return Value

A TextFixed object that represents the draw object.

Syntax

```
Draw.TextFixed(NinjaScriptBase owner, string tag, string text, TextPosition
textPosition, Brush textBrush, SimpleFont font, Brush outlineBrush, Brush areaBrush,
int areaOpacity)
Draw.TextFixed(NinjaScriptBase owner, string tag, string text, TextPosition
textPosition)
Draw.TextFixed(NinjaScriptBase owner, string tag, string text, TextPosition
textPosition, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
text	The text you wish to draw
TextPosition	TextPosition.BottomLeft TextPosition.BottomRight TextPosition.Center TextPosition.TopLeft TextPosition.TopRight
textBrush	The brush used to color the text of the draw object (reference)
font	A Simple Font object
outlineBrush	The brush used to color the text box outline (reference)

areaBrush	The brush used to color the text box fill area (reference)
areaOpacity	Sets the level of transparency for the fill color. Valid values between 0 - 100. (0 = completely transparent, 100 = no opacity)
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws text in the upper right corner of panel 1
Draw.TextFixed(this, "tag1", "Text to draw",
TextPosition.TopRight);
```

Tip: In some cases, it may be useful to pass in the ChartControl.Properties **TextFont** brush as well as the **LabelFont** SimpleFont object to render your custom text . This will help ensure that the text will be visible and match what a user has configured for their chart label display settings.

```
// match the text brush to what the user has configured on their
chart
Draw.TextFixed(this, "myTextFixed", "Hello world!",
TextPosition.BottomRight, ChartControl.Properties.ChartText,
   ChartControl.Properties.LabelFont, Brushes.Blue,
Brushes.Transparent, 0);
```

11.6.2.7.29.1 TextFixed

Definition

Represents an interface that exposes information regarding a Text Fixed IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object
YPixelOffset	An `int` value representing the offset value in pixels from within the text box area
Alignment	Possible values are: TextAlignment.Center TextAlignment.Far TextAlignment.Near TextAlignment.Justify (reference)
AreaOpacity	An `int` value representing the opacity of the area color
AreaBrush	A Brush class representing the fill color of the text box
DisplayText	A `string` value representing the text to be drawn
TextBrush	A Brush class representing the color of the text
Font	A Font object representing the font for the text
OutlineStroke	The Stroke object used to outline the text box
TextPosition	Possible values are: TextPosition.BottomLeft TextPosition.BottomRight TextPosition.Center TextPosition.TopLeft TextPosition.TopRight

Example

```
// Instantiate a TextFixed object
TextFixed myTF = Draw.TextFixed(this, "tag1", "Text to draw",
TextPosition.TopRight);

// Change the object's TextPosition
myTF.TextPosition = TextPosition.Center;
```

11.6.2.7.30 Draw.TimeCycles()

Definition

Draws a time cycle based on two points.

Method Return Value

A TimeCycles object that represents the draw object.

Syntax

```
Draw.TimeCycles(NinjaScriptBase owner, string tag, int startBarsAgo, int endBarsAgo,
double endY, Brush brush, bool drawOnPricePanel)
Draw.TimeCycles(NinjaScriptBase owner, string tag, int startBarsAgo, int endBarsAgo,
bool isGlobal, string templateName)
Draw.TimeCycles(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
endTime, Brush brush, bool drawOnPricePanel)
Draw.TimeCycles(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
endTime, bool isGlobal, string templateName)
Draw.TimeCycles(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
endTime, Brush brush, Brush areaBrush, int areaOpacity)
Draw.TimeCycles(NinjaScriptBase owner, string tag, int startBarsAgo, int endBarsAgo,
Brush brush, Brush areaBrush, int areaOpacity)
Draw.TimeCycles(NinjaScriptBase owner, string tag, int startBarsAgo, int endBarsAgo,
Brush brush, Brush areaBrush, int areaOpacity, bool drawOnPricePanel)
Draw.TimeCycles(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
endTime, Brush brush, Brush areaBrush, int areaOpacity, bool drawOnPricePanel)
Draw.TimeCycles(NinjaScriptBase owner, string tag, DateTime startTime, DateTime
endTime, Brush brush)
Draw.TimeCycles(NinjaScriptBase owner, string tag, int startBarsAgo, int endBarsAgo,
Brush brush)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")

tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
startBarsAgo	The starting bar (x axis co-ordinate) where the draw object will be drawn. For example, a value of 10 would paint the draw object 10 bars back.
startTime	The starting time where the draw object will be drawn
endBarsAgo	The end bar (x axis co-ordinate) where the draw object will terminate
endTime	The end time where the draw object will terminate
brush	The brush used to color draw object (reference)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a Time Cycles object based on 10 bars back to the current bar that is cornflower blue with an opacity of 40
Draw.TimeCycles(this, "tag1", 0, 10, Brushes.CornflowerBlue, Brushes.CornflowerBlue, 40);
```

11.6.2.7.30.1 TimeCycles

Definition

Represents an interface that exposes information regarding a TimeCyles IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object
OutlineStroke	A Stroke used for the outline of the region
AreaBrush	A Brush object representing the fill color of the draw object

Example

```
// Instantiate a Time Cycles object
TimeCycles myTimeCycles = (this, "tag1", 0, 10,
Brushes.CornflowerBlue, Brushes.CornflowerBlue, 40);

// Change the object's OutlineBrush
myTimeCycles.OutlineStroke = new  Stroke(Brushes.Red);
```

11.6.2.7.31 Draw .TrendChannel()

Definition

Draws a trend channel.

Method Return Value

A TrendChannel object that represents the draw object.

Syntax

```
Draw.TrendChannel(NinjaScriptBase owner, string tag, bool isAutoScale, int
anchor1BarsAgo, double anchor1Y, int anchor2BarsAgo, double anchor2Y, int
anchor3BarsAgo, double anchor3Y)
Draw.TrendChannel(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
anchor1Time, double anchor1Y, DateTime anchor2Time, double anchor2Y, DateTime
anchor3Time, double anchor3Y)
Draw.TrendChannel(NinjaScriptBase owner, string tag, bool isAutoScale, int
anchor1BarsAgo, double anchor1Y, int anchor2BarsAgo, double anchor2Y, int
anchor3BarsAgo, double anchor3Y, bool isGlobal, string templateName)
Draw.TrendChannel(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime
```

```
anchor1Time, double anchor1Y, DateTime anchor2Time, double anchor2Y, DateTime
anchor3Time, double anchor3Y, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale
anchor1BarsAgo	The number of bars ago (x value) of the 1st anchor point
anchor1Time	The time of the 1st anchor point
anchor1Y	The y value of the 1st anchor point
anchor2BarsAgo	The number of bars ago (x value) of the 2nd anchor point
anchor2Time	The time of the 2nd anchor point
anchor2Y	The y value of the 2nd anchor point
anchor3BarsAgo	The number of bars ago (x value) of the 3rd anchor point
anchor3Time	The time of the 3rd anchor point
anchor3Y	The y value of the 3rd anchor point

isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a trend channel
Draw.TrendChannel(this, "tag1", true, 10, Low[10], 0, High[0], 10,
High[10] + 5 * TickSize);
```

11.6.2.7.31.1 TrendChannel

Definition

Represents an interface that exposes information regarding a Trend Channel IDrawingTool.

Methods and Properties

TrendStartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
TrendEndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
ParallelStartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the second line used in the trend channel
PriceLevels	A collection of prices calculated by the drawing object

Example

```
// Instantiate a TrendChannel object
TrendChannel myTC = Draw.TrendChannel(this, "tag1", true, 10,
Low[10], 0, High[0], 10, High[10] + 5 * TickSize);

// Increase the y-axis position of the object's TrendEndAnchor
myTC.TrendEndAnchor.Price += 15;
```

11.6.2.7.32 Draw .Triangle()

Definition

Draws a triangle.

Method Return Value

A Triangle object that represents the draw object.

Syntax

```
Draw.Triangle(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
middleBarsAgo, double middleY, int endBarsAgo, double endY, Brush brush)
Draw.Triangle(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime middleTime, double middleY, DateTime endTime, double endY, Brush brush)
Draw.Triangle(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int middleBarsAgo, double middleY, int endBarsAgo, double endY, Brush
brush, Brush areaBrush, int areaOpacity)
Draw.Triangle(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime,
 double startY, DateTime midTime, double middleY, DateTime endTime, double endY, Brush
 brush, Brush areaBrush, int areaOpacity)
Draw.Triangle(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
middleBarsAgo, double middleY, int endBarsAgo, double endY, Brush brush, bool
drawOnPricePanel)
Draw.Triangle(NinjaScriptBase owner, string tag, bool isAutoScale, int startBarsAgo,
double startY, int middleBarsAgo, double middleY, int endBarsAgo, double endY, Brush
brush, Brush areaBrush, int areaOpacity, bool drawOnPricePanel)
Draw.Triangle(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime startTime,
 double startY, DateTime midTime, double middleY, DateTime endTime, double endY, Brush
 brush, Brush areaBrush, int areaOpacity, bool drawOnPricePanel)
Draw.Triangle(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int
middleBarsAgo, double middleY, int endBarsAgo, double endY, bool isGlobal, string
templateName)
Draw.Triangle(NinjaScriptBase owner, string tag, DateTime startTime, double startY,
DateTime middleTime, double middleY, DateTime endTime, double endY, bool isGlobal,
string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method

	Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale
startBarsAgo	The number of bars ago (x value) of the 1st anchor point
startTime	The time of the 1st anchor point
startY	The y value of the 1st anchor point
middleBarsAgo	The number of bars ago (x value) of the 2nd anchor point
midTime	The time of the 2nd anchor point
middleY	The y value of the 2nd anchor point
endBarsAgo	The number of bars ago (x value) of the 3rd anchor point
endTime	The time of the 3rd anchor point
endY	The y value of the 3rd anchor point
brush	The brush used to color the outline of draw object (reference)
areaBrush	The brush used to color the fill area of the draw object (reference)

areaOpacity	Sets the level of transparency for the fill color. Valid values between 0 - 100. (0 = completely transparent, 100 = no opacity)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Paints a blue triangle on the chart
Draw.Triangle(this, "tag1", 4, Low[4], 3, High[3], 1, Low[1],
Brushes.Blue);
```

11.6.2.7.32.1 Triangle

Definition

Represents an interface that exposes information regarding a Triangle IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
MiddleAnchor	An IDrawingTool's ChartAnchor representing the middle point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
AreaBrush	A Brush class representing the fill color of the draw object
AreaOpacity	An `int` value representing the opacity of the area color

OutlineStroke	The Stroke object used to draw the object's outline

Example

```
// Instantiate a Triangle object
Triangle myTri = Draw.Triangle(this, "tag1", 4, Low[4], 3, High[3],
 1, Low[1], Brushes.Blue);

// Change the object's AreaOpacity
myTri.AreaOpacity = 100;
```

11.6.2.7.33 Draw.TriangleDown()

Definition

Draws a triangle pointing down.

Method Return Value

A TriangleDown object that represents the draw object.

Syntax

```
Draw.TriangleDown(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, Brush brush)
Draw.TriangleDown(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo,
double y, Brush brush)
Draw.TriangleDown(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, Brush brush, bool drawOnPricePanel)
Draw.TriangleDown(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo,
double y, Brush brush, bool drawOnPricePanel)
Draw.TriangleDown(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, bool isGlobal, string templateName)
Draw.TriangleDown(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo,
double y, bool isGlobal, string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")

tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
isAutoScale	Determines if the draw object will be included in the y-axis scale
barsAgo	The bar the object will be drawn at. A value of 10 would be 10 bars ago.
time	The time the object will be drawn at.
y	The y value
brush	The brush used to color draw object (reference)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Tip: Thesiz

e
o
f
t
h
e
t
r
i
a
n
g
l
e
i
s
t
i
e
d
t
o
t
h
e
c
h
a
r
t
'
s
B
a
r
W
i
d
t
h
a
n
d

thuswillscaleautomaticallyasthecharti sresiz

e
d

Examples

```
// Paints a red triangle pointing down on the current bar 1 tick
below the low
Draw.TriangleDown(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);
```

11.6.2.7.33.1 TriangleDow n

Definition

Represents an interface that exposes information regarding a Triangle Down IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object
AreaBrush	A Brush class representing the fill color of the draw object
OutlineBrush	A Brush class representing the outline color of the draw object

Example

```
// Instantiate a TriangleDown object
TriangleDown myTri = Draw.TriangleDown(this, "tag1", true, 0,
Low[0] - TickSize, Brushes.Red);

// Change the object's AreaBrush
myTri.AreaBrush = Brushes.Beige;
```

11.6.2.7.34 Draw.TriangleUp()

Definition

Draws a triangle pointing up.

Method Return Value

A TriangleUp object that represents the draw object.

Syntax

```
Draw.TriangleUp(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, Brush brush)
Draw.TriangleUp(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo,
double y, Brush brush)
Draw.TriangleUp(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, Brush brush, bool drawOnPricePanel)
Draw.TriangleUp(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo,
double y, Brush brush, bool drawOnPricePanel)
Draw.TriangleUp(NinjaScriptBase owner, string tag, bool isAutoScale, DateTime time,
double y, bool isGlobal, string templateName)
Draw.TriangleUp(NinjaScriptBase owner, string tag, bool isAutoScale, int barsAgo,
double y, bool isGlobal, string templateName)
```

Parameters

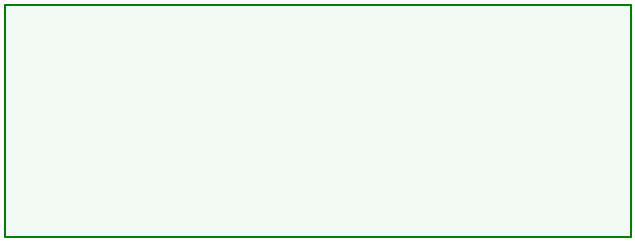
owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.

isAutoScale	Determines if the draw object will be included in the y-axis scale
barsAgo	The bar the object will be drawn at. A value of 10 would be 10 bars ago.
time	The time the object will be drawn at.
y	The y value
brush	The brush used to color draw object (reference)
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Tip: The size of the sheet

i
a
n
g
l
e
i
s
t
i
e
d
t
o
t
h
e
c
h
a
r
t
'
s
B
a
r
W
i
d
t
h
a
n
d
t
h
u
s
w
i
l
l

scaleautomaticallyasthechartisresized



Examples

```
// Paints a red triangle pointing up on the current bar 1 tick
below the low
Draw.TriangleUp(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);
```

11.6.2.7.34.1 TriangleUp

Definition

Represents an interface that exposes information regarding a Triangle Up IDrawingTool.

Methods and Properties

Anchor	An IDrawingTool's ChartAnchor representing the point of the drawing object
AreaBrush	A Brush class representing the fill color of the draw object
OutlineBrush	A Brush class representing the outline color of the draw object

Examples

```
// Instantiate a TriangleUp object
TriangleUp myTri = Draw.TriangleUp(this, "tag1", true, 0, Low[0] -
TickSize, Brushes.Red);

// Change the object's AreaBrush
myTri.AreaBrush = Brushes.Beige;
```

11.6.2.7.35 Draw .VerticalLine()

Definition

Draws a vertical line.

Method Return Value

A VerticalLine object that represents the draw object.

Syntax

```
Draw.VerticalLine(NinjaScriptBase owner, string tag, DateTime time, Brush brush)
Draw.VerticalLine(NinjaScriptBase owner, string tag, DateTime time, Brush brush,
DashStyleHelper dashStyle, int width, bool drawOnPricePanel)
Draw.VerticalLine(NinjaScriptBase owner, string tag, int barsAgo, Brush brush)
Draw.VerticalLine(NinjaScriptBase owner, string tag, int barsAgo, Brush brush,
DashStyleHelper dashStyle, int width, bool drawOnPricePanel)
Draw.VerticalLine(NinjaScriptBase owner, string tag, int barsAgo, bool isGlobal,
string templateName)
Draw.VerticalLine(NinjaScriptBase owner, string tag, DateTime time, bool isGlobal,
string templateName)
```

Parameters

owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "this")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.
barsAgo	The bar the object will be drawn at. A value of 10 would be 10 bars ago.
time	The time the object will be drawn at.
brush	The brush used to color draw object (reference)
dashStyle	DashStyleHelper.Dash DashStyleHelper.DashDot DashStyleHelper.DashDotDot DashStyleHelper.Dot DashStyleHelper.Solid

	Note: Fancier DashStyles like DashDotDot will require more resources than simple DashStyles like Solid.
width	The width of the draw object
drawOnPricePanel	Determines if the draw-object should be on the price panel or a separate panel
isGlobal	Determines if the draw object will be global across all charts which match the instrument
templateName	The name of the drawing tool template the object will use to determine various visual properties (empty string could be used to just use the UI default visuals instead)

Examples

```
// Draws a vertical line
Draw.VerticalLine(this, "tag1", 10, Brushes.Black);
```

11.6.2.7.35.1 VerticalLine

Definition

Represents an interface that exposes information regarding a Vertical Line IDrawingTool.

Methods and Properties

StartAnchor	An IDrawingTool's ChartAnchor representing the starting point of the drawing object
EndAnchor	An IDrawingTool's ChartAnchor representing the end point of the drawing object
Stroke	A Stroke object used to draw the object

Examples

```
// Instantiate a VerticalLine object
VerticalLine myLine = Draw.VerticalLine(this, "tag1", 10,
Brushes.Black);

// Change the object's Stroke
myLine.Stroke = new Stroke(Brushes.BlanchedAlmond,
DashStyleHelper.Dot, 5);
```

11.6.2.7.36 Brushes

For detailed information on using Brushes for Drawing please see the Working with Brushes educational resource.

11.6.2.7.37 AllowRemovalOfDrawObjects

Definition

Determines if programmatically drawn DrawObjects are allowed to remove manually from the chart

Property Value

When set to **true**, the draw objects from the indicator or strategy can be deleted from the chart manually by a user. If **false**, draw objects from the indicator or strategy can only be removed from the chart if the script removes the drawing object, or the script is terminates. Default set to **false**.

Syntax

```
AllowRemovalOfDrawObjects
```

Examples

```
protected override void OnStateChange()
{
     Add(new Plot(Brushes.Orange, "SMA"));
     AllowRemovalOfDrawObjects = true; // Draw objects can be
removed separately from the script
}
```

11.6.2.7.38 BackBrush

Definition

Sets the brush used for painting the chart panel's background color for the current bar.

> **Note**: This property will only set the back color for the panel the indicator is running. To set background color for all panels, please see the BackBrushAll property.

Property Value

A Brush object that represents the color of the current chart bar.

Syntax

`BackBrush`

Warning: You may have up to 65,535 unique BackBrush instances, therefore, using static predefined brushes should be favored. Alternatively, in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created.

Examples

```
protected override void OnBarUpdate()
{
     // Sets the chart panel back color to pale green
     BackBrush = Brushes.PaleGreen;

     // Sets the back color to to null which will use the default
color set in the chart properties dialog window
     BackBrush = null;

     // Sets the back color to maroon when the closing price is
less than the 20 period SMA // and to lime green when above (see
image below)
     BackBrush = SMA(20)[0] >= Close[0] ? Brushes.Maroon :
Brushes.LimeGreen;
}
```

11.6.2.7.39 BackBrushAll

Definition

A collection of prior back brushes used for the background colors for all chart panels.

Property Value

A Brush object that represents the color of the current chart bar.

> **Tip**: To reset the Chart background color to the default background color property, set the **BackBrushAll** to `null` for that bar.

Syntax

```
BackBrushAll
```

> **Warning**: You may have up to 65,535 unique BackBrushAll instances, therefore, using

static predefined brushes should be favored. Alternatively, in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created.

Examples

```
protected override void OnBarUpdate()
{
     // Sets the back color to pale green
     BackBrushAll = Brushes.PaleGreen;

     // Sets the back color to null to use the default color set in
the chart properties dialog window
     BackBrushAll = null;

     // Sets the back color to pink when the closing price is less
than the 20 period SMA
     // and to lime green when above (see image below)
     BackBrushAll = SMA(20)[0] >= Close[0] ? Brushes.Pink :
Brushes.PaleGreen;
}
```

11.6.2.7.40 BackBrushes

Definition

A collection of prior back brushes used for the background colors of the chart panel.

Property Value

A brush series type object. Accessing this property via an index value `[int barsAgo]` returns a Brush object representing the color of the background color on the referenced bar.

Syntax

```
BackBrushes
BackBrushes[int barsAgo]
```

Warning: You may have up to 65,535 unique BackBrushes instances, therefore, using static predefined brushes should be favored. Alternatively, in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 1)
         return;

     // Sets the color of the background on the current bar as blue
     BackBrushes[0] = Brushes.Blue;

     // Sets the color of the background on the previous bar as
orange
     BackBrushes[1] = Brushes.Orange;
}
```

11.6.2.7.41 BackBrushesAll

Definition

A collection of historical brushes used for the background colors for all chart panels.

Property Value

A brush series type object. Accessing this property via an index value `[int barsAgo]` returns a Brush object representing the color of the background color on the referenced bar for all chart panels.

Syntax

```
BackBrushesAll
BackBrushesAll[int barsAgo]
```

Warning: You may have up to 65,535 unique BackBrushAll instances, therefore, using static predefined brushes should be favored. Alternatively, in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 1)
         return;

     // Sets the color of the background on the current bar as blue
on all chart panels.
     BackBrushesAll[0] = Brushes.Blue;

     // Sets the color of the background on the previous bar as
orange on all chart panels.
     BackBrushesAll[1] = Brushes.Orange;
}
```

11.6.2.7.42 BarBrush

Definition

Sets the brush used for painting the color of a price bar's body.

Property Value

A Brush object that represents the color of this price bar.

> **Tip**: To set the price bar color to an empty color which uses the default bar color property, set the **BarBrush** to `null` for that bar.

Syntax

`BarBrush`

> **Warning**: You may have up to 65,535 unique BarBrush instances, therefore, using static predefined brushes should be favored. Alternatively, in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created.

Examples

```
protected override void OnBarUpdate()
{
     // Sets the bar color to yellow
     BarBrush = Brushes.Yellow;

     // Sets the brush used for the bar color to its default color
as defined in the chart properties dialog
     BarBrush = null;

     // Sets the bar color to yellow if the 20 SMA is above the 50
SMA and the closing
     // price is above the 20 SMA (see image below)
     if (SMA(20)[0] > SMA(50)[0] && Close[0] > SMA(20)[0])
          BarBrush = Brushes.Yellow;
}
```

11.6.2.7.43 BarBrushes

Definition

A collection of historical brushes used for painting the color of a price bar's body.

Property Value

A brush series type object. Accessing this property via an index value `[int barsAgo]` returns a Brush object representing the referenced bar's color.

> **Note**: This will only return the color of a bar in which an explicit color overwrite was used. Otherwise it will return `null`.

Syntax

```
BarBrushes
BarBrushes[int barsAgo]
```

> **Warning**: You may have up to 65,535 unique BarBrushes instances, therefore, using static predefined brushes should be favored. Alternatively, in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 1)
         return;

     // Sets the color of the current bar to blue.
     BarBrushes[0] = Brushes.Blue;

     // Sets the color of the previous bar to orange.
     BarBrushes[1] = Brushes.Orange;

}
```

11.6.2.7.44 CandleOutlineBrush

Definition
Sets the outline Brush of a candlestick.

Property Value
A brush object that represents the color of this price bar.

Syntax
`CandleOutlineBrush`

Warning: You may have up to 65,535 unique CandleOutlineBrushes instances, therefore, using static predefined brushes should be favored. Alternatively, in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created.

Examples

```
// Sets the candle outline color to black
CandleOutlineBrush = Brushes.Black;
```

11.6.2.7.45 CandleOutlineBrushes

Definition
A collection of historical outline brushes for candlesticks.

Property Value
A brush series type object. Accessing this property via an index value `[int barsAgo]` returns a brush structure representing the referenced bar's outline color.

Note: This will only return the color of a candlestick outline in which an explicit color overwrite was used. Otherwise it will return `null`.

Syntax

```
CandleOutlineBrushes
CandleOutlineBrushes[int barsAgo]
```

Warning: You may have up to 65,535 unique CandleOutlineBrushes instances, therefore, using static predefined brushes should be favored. Alternatively, in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created.

Examples

```
// Sets the outline color of the current bar to black.
CandleOutlineBrushes[0] = Brushes.Black;

// Sets the outline color of the previous bar to blue.
CandleOutlineBrushes[1] = Brushes.Blue;
```

11.6.2.7.46 Draw Objects

Definition

A collection holding all of the drawn chart objects on the chart, for all series. The draw objects can be manually drawn or script generated objects.

Notes:

- When reloading NinjaScript, all objects (including manual drawing tools) are reloaded at the same time. There is no guarantee a manually drawn object will be added to the **DrawObjects** collection before an indicator starts processing data.
- DrawObjects.ToList() is thread safe. DrawObjects collection itself is still dynamic (meaning it updates live) and as a result you can still run the risk of the collection being modified while you try to read it (and thus would see the related C# log entry) However, DrawObjects.ToList() is a snapshot of DrawObjects collection at the time the call is made.
- Also please keep in mind that iterating over a large DrawObjects collection could have an impact on performance

- Draw objects are disposed (for example on chart closing) after State.Terminated is seen for your custom NinjaScript studies potentially working with those

Property Value
A collection of IDrawingTool objects.

Syntax
```
DrawObjects
DrawObjects[string tag]
DrawObjects.Count
```

Examples
Finding the draw object of a specific tag
```
protected override void OnBarUpdate()
{
   if (DrawObjects["someTag"] != null && DrawObjects["someTag"] is
DrawingTools.Line)
   {
      // Do something with the drawing tool line
   }

   // An alternative approach to find the draw object by a tag
   if (DrawObjects["someTag"] as DrawingTools.Line != null)
   {
      // Do something drawing tool line
   }

   // Yet another way to find a drawing tool by a tag
   if (DrawObjects["someTag"].GetType().Name == "Line")
   {
      // Do something drawing tool line
   }
}
```

Get the number of draw objects on a chart

```
protected override void OnBarUpdate()
{
    if (DrawObjects.Count == 3)
    {
        // Do something
    }
}
```

Looping through the collection to find specific draw objects

```
protected override void OnBarUpdate()
{
   // Loops through the DrawObjects collection via a threadsafe
list copy
   foreach (DrawingTool draw in DrawObjects.ToList())
   {
      // Finds line objects that are attached globally to all
charts of the same instrument
      if (draw.IsGlobalDrawingTool && draw is DrawingTools.Line)
      {
         DrawingTools.Line globalLine = draw as DrawingTools.Line;

         // Changes the line color and prints its starting and end
points
         globalLine.Stroke.Brush = Brushes.Black;

         Print("Start: " + globalLine.StartAnchor.SlotIndex + "
End: " + globalLine.EndAnchor.SlotIndex);
      }

      // Finds non-global line objects
      else if (draw is DrawingTools.Line)
      {
         // Indicates if this is a manually drawn or script
generated line
         Print("Line Object: " + draw.Tag + " Manually Drawn: " +
draw.IsUserDrawn);
      }
   }
}
```

Note: Typecasting as in the example above will not function the same way in a compiled assembly (DLL). For an alternative approach, see the Considerations For Compiled

Assemblies page.

11.6.2.7.47 IDrawingTool

Definition

Represents an interface that exposes information regarding a drawn chart object.

IDrawingTool Properties are standard properties that are shared by all drawing tools.

Each specific **IDrawingTool** will have its own uniquely named **ChartAnchor** representing where the object was drawn on the chart. The name and number of **ChartAnchors** will be specific to that drawing tool (e.g., StartAnchor, EndAnchor, etc), however the fields available will be the same (e.g., BarsAgo, DrawnOnBar, etc). Details on those shared fields are outlined in the **ChartAnchor Properties** section toward the bottom of this topic.

Note: For implementing a custom Drawing Tool project, please see the DrawingTools section of this help guide.

IDrawingTool Properties

Anchors	A read-only collection of all of the IDrawingTool's ChartAnchors
AttachedTo	An `enum` determining where the drawing tool is attached. Possible values are: • AttachedToType.Bars, • AttachedToType.GlobalInstrument, • AttachedToType.Indicator, • AttachedToType.Strategy
DrawingState	The current DrawingState of the drawing tool
DrawnBy	An object value indicating which type of NinjaScript the drawing tool originated (`null` if user drawn)
IsAttachedToNinjaScript	A read-only `bool` indicating if the drawing tool is attached to an

	indicator or strategy
IgnoresUserInput	A read-only `bool` determining if the drawing tool can be interacted with by the user.
IsGlobalDrawingTool	A `bool` determining if the drawing tool displays on all charts of the instrument
IsLocked	A `bool` determining if the drawing tool can be moved
IsSeparateZOrder	A `bool` determining if the drawing tool will reside on a different ZOrder from the NinjaScript object it was drawn
IsUserDrawn	A read-only `bool` indicating if drawing tool was manually drawn by a user
PanelIndex	An `int` value representing the panel the drawing tool resides
SupportsAlerts	A read-only `bool` indicating if the drawing tool can be used for creating an alert
Tag	A `string` value representing the unique ID of the draw object. (Global draw objects will have an "@" added as a prefix to the string)
ZOrderType	An `enum` indicating the order in which the drawing tool will be drawn. Possible values are: • DrawingToolZOrder.Normal, • DrawingToolZOrder.AlwaysDrawnFirst,

	• DrawingToolZOrder.AlwaysDrawnLast

ChartAnchor Properties

<ChartAnchor>.BarsAgo	An `int` representing the "barsAgo" value that was passed to the Draw method **Note**: This value will **NOT** be set for objects drawn manually
<ChartAnchor>.DisplayName	A `string` representing name of the DrawingTool's chart anchor that is displaying on the UI
<ChartAnchor>.DrawingTool	The `IDrawingTool` object which created the DrawingTool's chart anchor object
<ChartAnchor>.DrawnOnBar	An `int` representing the CurrentBar value that the DrawingTool's chart anchor was drawn
<ChartAnchor>.IsNinjaScriptDrawn	A `bool` indicating the object was drawn programmatically
<ChartAnchor>.Price	A `double` representing the price the DrawingTool's chart anchor was drawn
<ChartAnchor>.SlotIndex	A `double` representing the DrawingTool's chart anchor index value the anchor was drawn
<ChartAnchor>.Time	A `DateTime` representing the time value the DrawingTool's chart anchor was drawn

Examples

```
Text myText;
protected override void OnBarUpdate()
{
   if(CurrentBar == 50)
      myText = Draw.Text(this, "tag", "test", 0, High[0]);

   if(myText != null)
   {
      Print(myText.Anchor.DrawnOnBar); // drawn on bar 50
   }

}
```

11.6.2.7.48 PriceLevels

Definition

A collection of PriceLevel objects defining lines for multi-price-level Drawing Tools (Fibonacci tools, etc.). Each PriceLevel within the collection can be configured programmatically or analyzed to obtain the parameters of user-drawn objects.

Note: PriceLevels is only used with the following pre-built Drawing Tools, but it can be used with custom Drawing Tools, as well:

- AndrewsPitchfork
- FibonacciCircle
- FibonacciExtensions
- FibonacciRetracements
- FibonacciTimeExtensions
- GannFan
- TrendChannel

Syntax

```
PriceLevels[int idx]
PriceLevels[int idx].GetPrice(double startPrice, double totalPriceRange, bool
isInverted)
PriceLevels[int idx].GetY(ChartScale chartScale, double startPrice, double
totalPriceRange, bool isInverted)
```

Methods and Properties

GetPrice()	Returns a `double` which repents the price value at the specified price level

GetY()	Returns a `float` representing the y-pixel coordinate at the specified price level
Name	The Name property of the specified PriceLevel. Set to a formatted version of Value by default.
Stroke	The Stroke used to draw the line associated with the specified PriceLevel
Tag	A tag used to identify the specified PriceLevel. Null by default.
Value	The value of the PriceLevel in percentage terms

Examples

```
// Define a FibonacciRetracements object outside of OnBarUpdate(), so the same object can be re-used
FibonacciRetracements myRetracements;

protected override void OnBarUpdate()
{

   if (CurrentBar < 20)
      return;

   // Instantiate myRetracements
   myRetracements = Draw.FibonacciRetracements(this, "fib", true, 20, High[20], 2, Low[2]);

   // Print each price level and the corresponding value in the PriceLevels collection contain in myRetracements
   // setting isInverted correctly is important for the Fibonacci Retracements since it will define which starting point is used, as it changes based     // on the anchors, i.e. if the Fibonacci is drawn from 100% to 0% (default) or the other inverted way (0% to 100%).
   foreach (PriceLevel p in myRetracements.PriceLevels)
   {
      Print(p.Value);
      Print(p.GetPrice(myRetracements.StartAnchor.Price, myRetracements.EndAnchor.Price - myRetracements.StartAnchor.Price, false));
   }
}
```

```
// Define a TrendChannel object outside of OnBarUpdate(), so the
same object can be re-used
TrendChannel myTCh;

protected override void OnBarUpdate()
{

   if (CurrentBar < 20)
      return;

   // Instantiate myTrendChannel
   myTCh = Draw.TrendChannel(this, "tc", true, 10, Low[10], 0,
High[0], 10, High[10] + 5 * TickSize);

   // Print each price level and the corresponding value in the
PriceLevels collection contain in myTrendChannel
   // For the TrendChannel the 0% is the Trend anchor, the 100% the
Parallel anchor
   foreach (PriceLevel p in myTCh.PriceLevels)
   {
      Print(p.Value);
      Print(p.GetPrice(myTCh.TrendStartAnchor.Price,
myTCh.ParallelStartAnchor.Price - myTCh.TrendStartAnchor.Price,
false));
   }
}
```

11.6.2.7.49 RemoveDraw Object()

Definition

Removes a draw object from the chart based on its tag value.

> **Note**: This method will **ONLY** remove DrawObjects which were created by a NinjaScript object. User drawn objects **CANNOT** be removed from via NinjaScript

Method Return Value

This method does not return a value

Syntax

`RemoveDrawObject(string tag)`

Parameters

tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag", each time this tag is used, the same draw object is modified. If unique tags are used each time, a new draw object will be created each time.

Examples

```
// Removes a draw object with the tag "tag1"
RemoveDrawObject("tag1");
```

11.6.2.7.50 RemoveDraw Objects()

Definition
Removes all draw objects originating from the indicator or strategy from the chart.

Note: This method will **ONLY** remove DrawObjects which were created by a NinjaScript object. User drawn objects **CANNOT** be removed from via NinjaScript

Method Return Value
This method does not return a value

Syntax
`RemoveDrawObjects()`

Examples

```
// Removes all draw objects
RemoveDrawObjects();
```

11.6.2.8 Instruments

Definition
A collection of Instrument objects currently used by a script.

Property Value
An array of Instrument objects

Syntax
Instruments[]

Examples

```
protected override void OnStateChange()
{
    if (State == State.DataLoaded)
    {
        // Print all instruments which have been loaded
        foreach (Instrument i in Instruments)
        {
            Print(i.FullName);
        }
    }
}
```

11.6.2.8.1 Instrument

Definition
A tradable symbol. Represents an instance of a Master Instrument

Warning: The properties in this class should **NOT** be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**

Methods and Properties

Exchange	Exchange of the current instrument
Expiry	Expiration date of the futures contract
FullName	Full name of the instrument
GetInstrument()	Returns an **Instrument** object by the master instrument name configured in the database.
MasterInstrument	An instrument's configuration settings. These are settings and

	properties which are defined in the Instrument window.
FundmentalData	Instrument thread specific FundamentalData events
MarketData	Instrument thread specific MarketData events
MarketDepth	Instrument thread specific MarketDepth events
Dispatcher	A Dispatcher used for subscribing to Instrument related events See Multi-Threading Considerations

11.6.2.8.1.1 Exchange

Definition
Indicates the current exchange of an instrument

Property Value
Represents the exchange which is selected for the current instrument.

Syntax
`Instrument.Exchange`

Examples

```
protected override void OnBarUpdate()
{
    // Print the exchange of the currently configured instrument
    Print(String.Format("Configured instrument is on the {0}
exchange", Instrument.Exchange));
}
```

Additional Access Information
This property can be accessed without a null reference check in the OnBarUpdate() event handler. When the OnBarUpdate() event is triggered, there will always be an Instrument object. Should you wish to access this property elsewhere, check for null reference first. e.g. if (Instrument != null)

11.6.2.8.1.2 Expiry

Definition
Indicates the expiration month of a futures contract.

Property Value
A DateTime structure representing the expiration month of a futures contract.

Syntax
```
Instrument.Expiry
```

Examples
```
protected override void OnBarUpdate()
{
    // Print the expiry of the currently configured futures
instrument
    Print(String.Format("You are viewing the {0} contract",
Instrument.Expiry));
}
```

Additional Access Information
This property can be accessed without a null reference check in the OnBarUpdate() event handler. When the OnBarUpdate() event is triggered, there will always be an Instrument object. Should you wish to access this property elsewhere, check for null reference first. e.g. if (Instrument != null)

11.6.2.8.1.3 FullName

Definition
Indicates the full NinjaTrader name of an instrument. For futures, this would include the expiration date. The September S&P 500 Emini contract full name is "ES 09-16".

Property Value
A `string` representing the full name of the instrument.

Syntax
```
Instrument.FullName
```

Examples

```
protected override void OnBarUpdate()
{
    // Print the full name (including contract month) of the
configured instrument
    Print(String.Format("{0} is being used as the input series",
Instrument.FullName));
}
```

Additional Access Information

This property can be accessed without a null reference check in the OnBarUpdate() event handler. When the OnBarUpdate() event is triggered, there will always be an Instrument object. Should you wish to access this property elsewhere, check for null reference first. e.g. if (Instrument != null)

11.6.2.8.1.4 GetInstrument()

Definition

Returns an Instrument object by the master instrument name configured in the database.

Note: This method does **NOT** add additional data for real-time or historical processing. For adding an additional data to your script, please see the AddDataSeries() method.

Method Return Value

An Instrument object

Syntax

```
Instrument.GetInstrument(string instrumentName)
```

Parameters

instrumentName	A string value representing a name of an instrument

Examples

```
protected override void OnStateChange()
{
   if (State == State.Historical)
   {
      Instrument myInstrument = Instrument.GetInstrument("AAPL");

      Print("AAPL's tick size is " +
myInstrument.MasterInstrument.TickSize.ToString());
   }
}
```

11.6.2.8.1.5 MasterInstrument

Definition

An instrument's configuration settings. These are settings and properties which are defined in the Instrument window.

Warning: The properties in this class should **NOT** be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**.

Methods and Properties

Compare()	Returns an `int` value compares two price values with respect to the Instrument tick size
Currency	The currency that the instrument traded in
Description	A written representation of a given instrument
Dividends	A collection of dividends for stock instruments
Exchanges	A collection of exchanges configured for an instrument
FormatPrice()	Returns a `string` representing the price formatted to the nearest tick size

InstrumentType	The type of instrument
MergePolicy	The Merge Policy that is configured for the current master instrument.
Name	The name of the instrument.
GetNextExpiry()	Returns a `DateTime` structure representing the next futures expiry for a given date
PointValue	Currency value of 1 full point of movement
RolloverCollection	A collection of expiration dates and offsets for futures instruments
RoundToTickSize()	Rounds the value up to the nearest valid value
RoundDownToTickSize()	Rounds the value down to the nearest valid value
Splits	A collection of splits for stock instruments
TickSize	The smallest movement in price configured
Url	A web url where contract details have been collected

Definition

Compares two price values with respect to the Instrument TickSize to ensure accuracy when dealing with floating point math.

Method Return Value

An `int` value.

A value of "1" is returned if price1 is greater than price2
A value of "-1" is returned if price1 is less than price2
A value of "0" if price1 is equal to price2

Syntax
`Instrument.MasterInstrument.Compare(double price1, double price2)`

Parameters

price1	A double value representing a price
price2	A double value representing a price

Examples

```
double newPrice = Close[0] + High[0] + Open[0];
if (Instrument.MasterInstrument.Compare(newPrice, Close[1]) == 1)
     // Do something since price1 is greater than price2
```

Definition
Indicates the currency configured for the Master Instrument properties.

Property Value
A type of `Currency` which is configured for the current master instrument.

Syntax
`Bars.Instrument.MasterInstrument.Currency`

Examples

```
if (Bars.Instrument.MasterInstrument.Currency != Currency.UsDollar)
{
       //Prints if the currency is not UsDollar and indicates what
currency it is
       Print ("Warning: Instruments base currency is not UsDollar,
it is " + Bars.Instrument.MasterInstrument.Currency);
}
```

Definition
Indicates the description configured for the Master Instrument properties.

Property Value
A `string` value which is configured for the current master instrument.

Syntax
Bars.Instrument.MasterInstrument.Description

Examples

```
protected override void OnBarUpdate()
		{
			// Displays the master instrument's
description at the bottom right of the chart
			Draw.TextFixed(this, "tag1",
Bars.Instrument.MasterInstrument.Description,
TextPosition.BottomRight);
		}
```

Definition
An collection of Dividends configured for the Master Instrument properties used in for stocks.

Property Value
A collection of `Dividends` configured for the current instrument.

Possible values are:

Amount	A double value representing the amount in dollars which was paid on the date of the dividend
Date	A `DateTime` structure representing the date of the dividend

Syntax
Bars.Instrument.MasterInstrument.Dividends

Examples

```
foreach(Dividend dividends in
Bars.Instrument.MasterInstrument.Dividends)
{
    Print(dividends.Amount);
    Print(dividends.Date);
}
```

Definition
A collection of exchange(s) configured for the Master Instrument properties.

Property Value

A collection of Exchanges which represent the exchanges configured for the current instrument.

Syntax
Bars.Instrument.MasterInstrument.Exchanges

Examples
```
foreach(Exchange exchange in
Bars.Instrument.MasterInstrument.Exchanges)
{
  Print(exchange);  // Default, Nasdaq, NYSE
}
```

Definition
Returns a price value as a string which will be formatted to the nearest tick size.

Note: This is useful as the standard format specifier will only use the minimum number of digits for a decimal by default; however you can use this method to ensure that your data is always formatted per the instrument tick size for easier readability. For example, a value of 1985.50 would Print() as 1985.5, while using FormatPrice(), we can expect the value to be formatted as 1985.50.

Property Value
A string value which will ensure the price data is always formatted to the nearest tick size.

Syntax
Bars.Instrument.MasterInstrument.FormatPrice(double price, [bool round])

Parameters
price	A double value representing a price
round	An optional bool when true (default) will round the price value to the nearest tick size

Examples

```
protected override void OnBarUpdate()
{
   // called without setting the optional bool parameter, which is
defaulted to true then
   Print(Bars.Instrument.MasterInstrument.FormatPrice(Close[0]));

}
```

```
protected
 override void OnMarketData(MarketDataEventArgs marketDataUpdate)
{
   Print(marketDataUpdate.Instrument.MasterInstrument.FormatPrice(m
arketDataUpdate.Price));
}
```

Definition

Returns the type of instrument.

Property Value

An `InstrumentType` representing the type of an instrument.

Possible values are:

InstrumentType.Future
InstrumentType.Stock
InstrumentType.Index
InstrumentType.Forex
InstrumentType.Cfd
InstrumentType.Cryptocurrency

Syntax

`Instrument.MasterInstrument.InstrumentType`

Examples

```
if (Instrument.MasterInstrument.InstrumentType ==
InstrumentType.Future)
{
	// Do something
}
else
{
	// Do something else
}
```

Additional Access Information

This property can be accessed without a null reference check in the OnBarUpdate() event handler. When the OnBarUpdate() event is triggered, there will always be an Instrument object. Should you wish to access this property elsewhere, check for null reference first. e.g. if (Instrument != null)

Definition

Indicates the Merge Policy configured for the Master Instrument properties.

Property Value

Represents the MergePolicy that is configured for the current master instrument.

Possible values are:

DoNotMerge	No merge policy is applied
MergeBackAdjusted	Merge policy is applied between contracts along with rollover offsets
MergeNonBackAdjusted	Merge policy is applied between contracts without offsets
UseGlobalSettings	Uses the value configured from **Tools** -> **Options** -> **Market Data**

Syntax

`Bars.Instrument.MasterInstrument.MergePolicy`

Examples

```
//Prints a warning, indicating what merge policy is in use if not
using global settings
if (Bars.Instrument.MasterInstrument.MergePolicy !=
MergePolicy.UseGlobalSettings)
{
	Print("Warning: Instrument has merge policy of " +
Bars.Instrument.MasterInstrument.MergePolicy);
}
```

Definition

Indicates the NinjaTrader database name of an instrument. For example, "MSFT", "ES", "NQ" etc...

Property Value

A `string` representing the name of the instrument.

Syntax

```
Instrument.MasterInstrument.Name
```

Examples

```
protected override void OnBarUpdate()
		{
			// Displays the master instrument's name at
the bottom right of the chart
			Draw.TextFixed(this, "tag1",
Bars.Instrument.MasterInstrument.Name, TextPosition.BottomRight);
		}
```

Additional Access Information

This property can be accessed without a null reference check in the OnBarUpdate() event handler. When the OnBarUpdate() event is triggered, there will always be an Instrument object. Should you wish to access this property elsewhere, check for null reference first. e.g. if (Instrument != null)

Definition

Returns the current futures expiry compared to the time of the input value used for the method.

Method Return Value

A DateTime structure

Syntax
Bars.Instrument.MasterInstrument.GetNextExpiry(DateTime *afterDate*)

Parameters

afterDate	A DateTime value representing to be compared

Examples

```
// Indicates what the current expiry is in the bottom right of the
chart
Draw.TextFixed(this, "tag1", "The current expiry is " +
Bars.Instrument.MasterInstrument.GetNextExpiry(DateTime.Now).ToStri
ng("MM-yy"), TextPosition.BottomRight);
```

Definition
Indicates the currency value of 1 full point of movement. For example, 1 point in the S&P 500 Emini futures contract (ES) is $50 USD which is equal to $12.50 USD per tick.

Property Value
A double value representing the currency value of 1 point of movement.

Syntax
Instrument.MasterInstrument.PointValue

Examples

```
protected override void OnBarUpdate()
		{
			// Displays the master instrument's point
value at the bottom right of the chart
			Draw.TextFixed(this, "Point value: ",
Bars.Instrument.MasterInstrument.PointValue.ToString(),
TextPosition.BottomRight);
		}
```

Additional Access Information
This property can be accessed without a null reference check in the OnBarUpdate() event handler. When the OnBarUpdate() event is triggered, there will always be an Instrument object. Should you wish to access this property elsewhere, check for null reference first. e.g. if (Instrument != null)

Definition
Indicates the rollovers that have been configured for the Master Instrument properties used in for futures.

Property Value
A `RolloversCollection` configured for the current instrument.

Possible values are:

ContractMonth	A `DateTime` structure representing the expiry month of a futures contract
Date	A `DateTime` structure representing the date of the rollover
Offset	A `double` value representing the number of points between contracts

Syntax
```
Bars.Instrument.MasterInstrument.RolloverCollection
```

Examples


```
foreach(var rollover in
Bars.Instrument.MasterInstrument.RolloverCollection)
{
     Print(rollover.ContractMonth);
     Print(rollover.Date);
     Print(rollover.Offset);
}
```

Definition
Returns a value that is rounded up to the nearest valid value evenly divisible by the instrument's tick size.

Method Return Value
A `double` value.

Syntax
```
Instrument.MasterInstrument.RoundToTickSize(double price)
```

Parameters
price	A double value representing a price

Examples

```
//Takes the last 3 closes, divides them by 3, and rounds the value
up to the nearest valid tick size
Value[0] = Instrument.MasterInstrument.RoundToTickSize((Close[0] +
Close[1] + Close[2]) / 3);
```

Definition

Returns a value that is rounded down to the nearest valid value evenly divisible by the instrument's tick size.

Method Return Value

A `double` value.

Syntax

`Instrument.MasterInstrument.RoundDownToTickSize(double price)`

Parameters

price	A double value representing a price

Examples

```
//Takes the last 3 closes, divides them by 3, and rounds the value
down to the nearest valid tick size
Value[0] =
Instrument.MasterInstrument.RoundDownToTickSize((Close[0] +
Close[1] + Close[2]) / 3);
```

Definition

Indicates the Splits that have been configured for the Master Instrument properties used in for stocks.

Property Value

A collection of `Splits` configured for the current instrument.

Possible values are:

Date	A `DateTime` structure representing the date of the split
Factor	A `double` value representing the number of points the stock split

Syntax

```
Bars.Instrument.MasterInstrument.Splits
```

Examples

```
foreach (Split split in Bars.Instrument.MasterInstrument.Splits)
{
     Print(split.Date);
     Print(split.Factor);
}
```

Definition

Indicates the tick size configured for the Master Instrument properties.

Property Value

A `double` value representing the tick size configured for the current master instrument.

Syntax

```
Bars.Instrument.MasterInstrument.TickSize
```

Examples

```
protected override void OnBarUpdate()
              {
                     // Displays the master instrument's tick size
at the bottom right of the chart
                     Draw.TextFixed(this, "tag1",
Bars.Instrument.MasterInstrument.TickSize.ToString(),
TextPosition.BottomRight);
              }
```

Definition

Indicates the Url configured for the Master Instrument properties.

Property Value

A `string` value representing the Url that is configured for the current master instrument.

Syntax

```
Bars.Instrument.MasterInstrument.Url
```

Examples

```
protected override void OnBarUpdate()
	{
		// Displays the master instrument's URL at the bottom
right of the chart
		Draw.TextFixed(this, "tag1", "Instruments URL is " +
Bars.Instrument.MasterInstrument.Url, TextPosition.BottomRight);
	}
```

11.6.2.9 ISeries<T>

Definition

ISeries<T> is an interface that is implemented by all NinjaScript classes that manage historical data as an ISeries<double> (Open, High, Low, Close, etc), used for indicator input, and other object data. Please see the help guide article on Working with Price Series for a basic overview on how to access this information.

Types of ISeries

Series<T>	Represents a generic custom data structure for custom development
PriceSeries	Historical price data structured as an `ISeries<double>` interface (`Close[0]`, `High[0]`, `Low[0]`, etc)
TimeSeries	Historical time stamps structured as an `ISeries<DateTime>` interface (`Time[0]`)
VolumeSeries	Historical volume data structured as an `ISeries<double>` interface (`Volume[0]`)

Methods and Properties

GetValueAt()	Returns the underlying input value at a specified bar index value.
IsValidDataPoint()	Indicates if the specified input is set at a barsAgo value relative to the current bar.

IsValidDataPointAt()	Indicates if the specified input is set at a specified bar index value.
Count	Return the number total number of values in the `ISeries` array

Tips: (see examples below)
1. By specifying a parameter of type `ISeries<double>`, you can then pass in an array of closing prices, an indicator, or a user defined data series.
2. When working with `ISeries<double>` objects in your code you may come across situations where you are not sure if the value being accessed is a valid value or just a "placeholder" value. To check if you are using valid values for your logic calculations that have been explicitly set, please use `.IsValidDataPoint(int barsAgo)` to check.

Examples

Using ISeries as a method parameter

```
//create custom a method named DoubleTheValue that accepts any
object that implements
// the ISeries<double> interface as a parameter
private double DoubleTheValue(ISeries<double> priceData)
{
     return priceData[0] * 2;
}

protected override void OnBarUpdate()
{
   // This custom method is then used twice,
   //the first time passing in an array of closing prices
     Print(DoubleTheValue(Close));
   //and the second time passing in a 20 period simple moving
average.
     Print(DoubleTheValue(SMA(20)));
}
```

Checking ISeries value before accessing

```
protected override void OnBarUpdate()
{
     // Only set our plot if the input is a valid value
     if (Input.IsValidDataPoint(0))
          Plot0[0] = Input[0];
}
```

11.6.2.9.1 Series<T>

Definition

A Series<T> is a special generic type of data structure that can be constructed with any chosen data type and holds a series of values equal to the same number of elements as bars in a chart. If you have 200 bars loaded in your chart with a moving average plotted, the moving average itself holds a Series<double> object with 200 historical values of data, one for each bar. Series<double> objects can be used as input data for all indicator methods. The Series<T> class implements the ISeries<T> interface.

Note: By default NinjaTrader limits the number of values stored for `Series<T>` objects to 256 from the current bar being processed. This drastically improves memory performance by not holding onto old values that are generally not needed. Should you need more values than the last 256 please be sure to create the `Series<T>` object so that it stores all values instead through the use of the MaximumBarsLookBack property.

Constructors

`Series<T>(ninjaScriptBase)`	Creates a **Series<T>** object synchronized to the primary data series of the provided NinjaScript
`Series<T>(ninjaScriptBase, maximumBarsLookBack)`	Creates a **Series<T>** object synchronized to the primary data series of the provided NinjaScript. This constructor also allows controlling the Series<T>'s MaximumBarsLookBack
`Series<T>(bars)`	Creates a **Series<T>** object synchronized to the provided Bars object, for Multi Time Frame scripts, this could be given from BarsArray

`Series<T>(bars, maximumBarsLookBack)`	Creates a **Series<T>** object synchronized to the provided Bars object, for Multi Time Frame scripts, this could be given from BarsArray. While this constructor allows controlling the Series<T>'s MaximumBarsLookBack, it is forced to **MaximumBarsLookBack.Infinite**

Parameters

ninjaScriptBase	The NinjaScript object used to create the Series
bars	The Bars object used to create the Series
maximumBarsLookBack	A MaximumBarsLookBack value used for memory performance

Methods and Properties

GetValueAt()	Returns the underlying input value at a specified bar index value.
IsValidDataPoint()	Determines if the specified input is set at a barsAgo value relative to the current bar.
Reset()	Resets the internal marker which is used for IsValidDataPoint() back to false.
Count	The total number of bars or data points.

Creating Series<T> Objects

When creating custom indicators, `Series<double>` objects are automatically created for you by calling the AddPlot() method and can be subsequently referenced by the Value and/or

Values property. However, you may have a requirement to create a `Series<T>` object to store values that are part of an overall indicator value calculation. This can be done within a custom indicator or strategy.

> **Note**: Custom Series<T> objects will hold the number of values specified by the MaximumBarsLookBack property when the custom series object is instantiated.

To create a `Series<T>` object:

1. Determine the data type of the `Series<T>` object you wish to create. This could be `double`, `bool`, `int`, `string` or any other object type you want.
2. Define a variable of type `Series<T>` that will hold a `Series<T>` object. This example will create "myDoubleSeries" as a `Series<double>`.
3. In the OnStateChange() method, in the State.DataLoaded create a new `Series<T>` object and assign it to the "myDoubleSeries" variable

```
private Series<double> myDoubleSeries; // Define a Series<T>
variable. In this instance we want it
                                       // as a double so we created
a Series<double> variable.

private Series<double> mySecondaryDoubleSeries; // Define a
Series<T> variable. In this instance we want it
                                                  // as a double so
we created a Series<double> variable.

// Create a Series object and assign it to the variable
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Add a secondary data series to sync our Secondary
Series<double>
          AddDataSeries(BarsPeriodType.Minute, 1);
     }
     else if (State == State.DataLoaded)
     {
          // "this" refers to the NinjaScript object itself. This
syncs the Series object to historical data bars
          // MaximumBarsLookBack determines how many values the
Series<double> will have access to
          myDoubleSeries = new Series<double>(this,
MaximumBarsLookBack.Infinite);

          // "BarsArray[1]" refers to the first data series added
to the script with AddDataSeries.
          mySecondaryDoubleSeries = new
Series<double>(BarsArray[1]);
     }
}
```

Setting Values

You can set the value for the current bar being evaluated by choosing a "barsAgo" value of "0" or, for historical bars, by choosing a "barsAgo" value that represents the number of bars ago that you want the value to be stored at.

Setting Series<T> values

```
protected override void OnBarUpdate()
{
     myDoubleSeries[0] = Close[0];
}
```

Note: The "barsAgo" value is only guaranteed to be in sync with the recent current bar during core data event methods, such as OnBarUpdate(), OnMarketUpdate(), and during strategy related order events such as OnOrderUpdate(), OnExecutionUpdate(), OnPositionUpdate(). For scenarios where you may need to set a value outside of a core data/order event, such as OnRender() or a custom event, you must first synchronize the "barsAgo" pointer via the TriggerCustomEvent() method.

Checking for Valid Values

It is possible that you may use a `Series<T>` object but decide not to set a value for a specific bar. However, you should *not* try to access a `Series<T>`value that has not been set. Internally, a dummy value does exists, but you want to check to see if it was a valid value that you set before trying to access it for use in your calculations. Please see IsValidDataPoint() more information.

Warning: Calling IsValidDataPoint() will only work a MaximumBarsLookBackInfinite series. Attempting to check IsValidDataPoint() MaximumBarsLookBack256 series throw an error. Please check the Log tab of the Control Center

Getting Values

You can access `Series<T>` object values using the syntax `Series<T>[int barsAgo]` where barsAgo represents the data value *n* (number of bars ago).

Accessing Series object values

```
protected override void OnBarUpdate()
{
    // Prints the current and last bar value
    Print("The values are " + myDoubleSeries[0] + " " +
myDoubleSeries[1]);
}
```

Alternatively, you can access a value at an absolute bar index using the GetValueAt() method.

Note: In most cases, you will access the historical price series using a core data event handler such as OnBarUpdate(). For more advance developers, you may find situations where you wish to access historical price series outside of the core data event methods, such as OnRender(), or your own custom event. In these advanced scenarios, you may run into situations where the "barsAgo" pointer is not in sync with the current bar, and may result in errors when trying to obtain this information. In those cases, please use the Bars.Get...() methods with the absolute bar index, e.g., GetValueAt().

Methods that Accept ISeries<T> as Arguments

All indicator methods accept `ISeries<double>` objects as arguments. Carrying from the prior examples, let's print out the 10 period simple moving average of range.

Using a custom Series object as indicator input

```
protected override void OnBarUpdate()
{
    // Calculate the range of the current bar and set the value
     myDoubleSeries[0] = (High[0] - Low[0]);

     // Print the current 10 period SMA of range
     Print("Value is " + SMA(myDoubleSeries, 10)[0]);
}
```

11.6.2.9.1.1 Reset()

Definition

Resets the internal marker which is used for IsValidDataPoint() back to false. Calling the Reset() method is unique and can be very powerful for custom indicator development. Series<T> objects will always contain a value which is assigned, however calling Reset() simply means you effectively ignore the value of the current bar for plotting purposes. For calculation purposes you will want to use IsValidDataPoint() to ensure you are not calculating off of any reset values assigned by the Reset() method.

Series Type	Value after Reset()
Series<bool>	false
Series<double>	0.00
Series<DateTime>	DateTime.MinValue
Series<float>	0

Series<int>	0
Series<long>	0
Series<string>	null

Method Return Value
This method does not return a value

Syntax
```
Reset()
Reset(int barsAgo)
```

Parameters
barsAgo	An int representing from the current bar the number of historical bars the method will check. If no barsAgo value is supplied, the current bar value will be reset instead (barsAgo 0)

Examples
```
protected override void OnBarUpdate()
{
   // set MyPlot to Low of current bar minus 1 tick
   MyPlot[0] = Low[0] - (1 * TickSize);

   //reset MyPlot every 10 bars
   if(CurrentBar % 10 == 0)
      MyPlot.Reset();

   // only calculate MyPlot value if it has not be reset
   if(MyPlot.IsValidDataPoint(0))
      Print(CurrentBar + " Value is: " + MyPlot[0]);
}
```

11.6.2.9.2 PriceSeries<double>

Definition
Represents historical data as an ISeries<double> interface which can be used for custom NinjaScript object calculations.

Note: In most cases, you will access the historical price series using a core event handler such as OnBarUpdate. For more advance developers, you may find situations where you wish to access historical price series outside of the core event methods, such as your own custom mouse click. In these advanced scenarios, you may run into situations where the barsAgo pointer is not in sync with the current bar, which may cause errors when trying to obtain this information. In those cases, please use the Bars.Get...() methods with the absolute bar index, e.g., Bars.GetClose(), Bars.GetOpen(), etc.

Single ISeries<double>

Close	A collection of historical bar close prices.
High	A collection of historical bar high prices.
Input	A collect of the the main historical input values.
Low	A collection of historical bar low prices.
Median	A collection of historical bar median prices.
Open	A collection of historical bar open prices.
Typical	A collection of historical bar typical prices.
Value	A collection of historical references to the first object (`Values[0]`) in the indicator
Weighted	A collection of historical bar weighted prices.

Multi-Time Frame ISeries<double>

Closes	Holds an array of `ISeries<double>` objects holding historical bar close prices.
Highs	Holds an array of `ISeries<double>` objects holding historical bar high prices.
Inputs	Holds an array of `ISeries<double>` objects holding main historical input values
Lows	Holds an array of `ISeries<double>` objects holding historical bar low prices.
Medians	Holds an array of `ISeries<double>`objects holding historical bar median prices.
Opens	Holds an array of `ISeries<double>` objects holding historical bar open prices.
Typicals	Holds an array of `ISeries<double>` objects holding historical bar typical prices.
Values	Holds an array of `ISeries<double>` objects holding hold the indicator's underlying calculated values.
Weighteds	Holds an array of `ISeries<double>` objects holding historical bar weighted prices.

11.6.2.9.2.1 Close

Definition

A collection of historical bar close prices.

Property Value

A `ISeries<double>` type object. Accessing this property via an index value `[int barsAgo]` returns a double value representing the price of the referenced bar.

> **Note**: When an indicator uses another indicator as input series, Close will represent the closing price of the input series' input series. For example, if MyCustomIndicator uses an ADX as input series, then referencing Close[0] in MyCustomIndicator will provide the Close price for the ADX's input series.

Syntax

```
Close
Close[int barsAgo]
```

Examples

```
// OnBarUpdate method
protected override void OnBarUpdate()
{
     // Evaluates if the current close is greater than the prior
bar close
     if (Close[0] > Close[1])
          Print("We had an up day");
}
```

11.6.2.9.2.2 Closes

Definition

Holds an array of ISeries<double> objects holding historical bar close prices. A ISeries<double> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the closing prices of all Bars objects in a multi-instrument or multi-time frame script.

Property Value

An array of `ISeries<double>` objects.

Syntax

```
Closes[int barSeriesIndex][int barsAgo]
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's close price to the 5-minute
bar's close price
     if (Closes[0][0] > Closes[1][0])
          Print("The primary bar's close price is greater");
}
```

11.6.2.9.2.3 High

Definition

A collection of historical bar high prices.

Property Value

An `ISeries<double>` type object. Accessing this property via an index value `[int barsAgo]` returns a double value representing the price of the referenced bar.

Syntax

`High`
`High[int barsAgo]`

Examples

```
// OnBarUpdate method
protected override void OnBarUpdate()
{
     // Make sure we have at least 20 bars
     if (CurrentBar < 20)
          return;

     // Evaluates for higher highs
     if (High[0] > High[1] && High[1] > High[2])
          Print("Two successive higher highs");

     // Gets the current value of a 20 period SMA of high prices
     double value = SMA(High, 20)[0];
     Print("The value is " + value.ToString());
}
```

11.6.2.9.2.4 Highs

Definition

Holds an array of ISeries<double> objects holding historical bar high prices. A ISeries<double> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the high prices of all Bars objects in a multi-instrument or multi-time frame script.

Property Value

An array of `ISeries<double>` objects.

Syntax

`Highs[int barSeriesIndex][int barsAgo]`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's high price to the 5-minute bar's
high price
     if (Highs[0][0] > Highs[1][0])
          Print("The primary bar's high price is greater");
}
```

11.6.2.9.2.5 Input

Definition

The main historical data input. If implemented in the NinjaScript object, it allows for more flexibility as non bars based series such as plot series could be passed in and drive the calculation outcomes - an example would be a custom moving average that should have the ability to operate on another moving average (i.e. the SMA) as input series.

Property Value

An `ISeries<double>` type object that implements the `Series<double>` interface. Accessing this property via an index value `[int barsAgo]` returns a double value representing the price of the referenced bar.

Syntax

```
Input
Input[int barsAgo]
```

Examples

```
// Prints the the current value of input
Print(Input[0].ToString());
```

```
// Prints the the current type of input passed to the object, so we
can detect if we're working on a price based series such as OHLCV
or a derivative such as an SMA indicator
if (Input is PriceSeries)
Print("Price Series Input");
if (Input is Indicator)
Print("Indicator Input");
```

```
// Prints the the current selected price type for the input series
else if (State == State.DataLoaded)
{
     PriceSeries priceSeries = Inputs[0] as PriceSeries;

     if (priceSeries != null)
         Print("PriceType selected: " + priceSeries.PriceType);
}
```

Tip: When working with multi-series indicators, Input is not guaranteed to reference the primary BarsInProgress. Please be mindful as to when you access `Input[0]` as you will only be able to do so after the contextual BarsInProgress has bars. To check to ensure BarsInProgress has some bars you can use CurrentBars to check.

11.6.2.9.2.6 Inputs

Definition

Holds an array of ISeries<double> objects holding the main data input. A ISeries<double> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the main input all Bars objects in a multi-instrument or multi-time frame script.

Property Value

An array of `ISeries<double>` objects.

Syntax

`Inputs[int barSeriesIndex][int barsAgo]`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's input price to the 5-minute
bar's input price
     if (Inputs[0][0] > Inputs[1][0])
          Print("The primary bar's input is greater");
}
```

11.6.2.9.2.7 Low

Definition

A collection of historical bar low prices.

Property Value

An `ISeries<double>` type object. Accessing this property via an index value `[int barsAgo]` returns a double value representing the price of the referenced bar.

Syntax

`Low`
`Low[int barsAgo]`

Examples

```
// Current bar low price
double barLowPrice = Low[0];

// Low price of 10 bars ago
double barLowPrice = Low[10];

// Current bar value of a 20 period exponential moving average of
low prices
double value = EMA(Low, 20)[0];
```

11.6.2.9.2.8 Lows

Definition

Holds an array of ISeries<double> objects holding historical bar low prices. An ISeries<double> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the low prices of all Bars objects in a multi-instrument or multi-time frame script.

Property Value

An array of `ISeries<double>` objects.

Syntax

`Lows[int barSeriesIndex][int barsAgo]`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's low price to the 5-minute bar's
low price
     if (Lows[0][0] > Lows[1][0])
          Print("The primary bar's low price is greater");
}
```

11.6.2.9.2.9 Median

Definition

A collection of historical bar median prices. Median price = (High + Low) / 2.

Property Value

An `ISeries<double>` type object. Accessing this property via an index value `[int barsAgo]` returns a double value representing the price of the referenced bar.

Syntax

```
Median
Median[int barsAgo]
```

Examples

```
// Current bar median price
double barMedianPrice = Median[0];

// Median price of 10 bars ago
double barMedianPrice = Median[10];

// Current bar value of a 20 period exponential moving average of
median prices
double value = EMA(Median, 20)[0];
```

11.6.2.9.2.10 Medians

Definition

Holds an array of ISeries<double> objects holding historical bar median prices. An ISeries<double>> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the median prices of all Bars objects in a multi-instrument or multi-time frame script.

Property Value

An array of `ISeries<double>` objects.

Syntax

`Medians[int barSeriesIndex][int barsAgo]`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's median price to the 5-minute
bar's median price
     if (Medians[0][0] > Medians[1][0])
          Print("The primary bar's median price is greater");
}
```

11.6.2.9.2.11 Open

Definition

A collection of historical bar opening prices.

Property Value

An `ISeries<double>` type object. Accessing this property via an index value `[int barsAgo]` returns a double value representing the price of the referenced bar.

Syntax

`Open`
`Open[int barsAgo]`

Examples

```
// Current bar opening price
double barOpenPrice = Open[0];

// Opening price of 10 bars ago
double barOpenPrice = Open[10];

// Current bar value of a 20 period simple moving average of
opening prices
double value = SMA(Open, 20)[0];
```

11.6.2.9.2.12 Opens

Definition

Holds an array of ISeries<double> objects holding historical bar open prices. An ISeries<double> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the open prices of all Bars objects in a multi-instrument or multi-time frame script.

Property Value

An array of `ISeries<double>` objects.

Syntax

`Opens[int barSeriesIndex][int barsAgo]`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's open price to the 5-minute bar's
open price
     if (Opens[0][0] > Opens[1][0])
          Print("The primary bar's open price is greater");
}
```

11.6.2.9.2.13 Typical

Definition

A collection of historical bar typical prices. Typical price = (High + Low + Close) / 3.

Property Value

An `ISeries<double>` type object. Accessing this property via an index value `[int barsAgo]` returns a double value representing the price of the referenced bar.

Syntax

```
Typical
Typical[int barsAgo]
```

Examples

```
// Current bar typical price
double barTypicalPrice = Typical[0];

// Typical price of 10 bars ago
double barTypicalPrice = Typical[10];

// Current bar value of a 20 period exponential moving average of
typical prices
double value = EMA(Typical, 20)[0];
```

11.6.2.9.2.14 Typicals

Definition

Holds an array of ISeries<double> objects holding historical bar typical prices. An ISeries<double> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the typical prices of all Bars objects in a multi-instrument or multi-time frame script.

Property Value

An array of `ISeries<double>` objects.

Syntax

`Typicals[int barSeriesIndex][int barsAgo]`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5 minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's typical price to the 5-minute
bar's typical price
     if (Typicals[0][0] > Typicals[1][0])
          Print("The primary bar's typical price is greater");
}
```

11.6.2.9.2.15 Value

Definition

A collection of historical references to the first ISeries object Values[0] in the indicator. This is the primary indicator value (synched to the primary series in case of a MultiSeries indicator)

Property Value

An `ISeries<double>` object.

Syntax

`Value`

Examples

```
// OnBarUpdate method of a custom indicator
protected override void OnBarUpdate()
{
     // Ensures we have enough bars loaded for our indicator
     if (CurrentBar < 1)
          return;

     // Evaluates the indicator primary value 1 bar ago and sets
the value of the indicator
     // for the current bar being evaluated
     if (Value[1] < High[0] - Low[0])
          Value[0] = High[0] - Low[0];
     else
          Value[0] = High[0] - Close[0];
}
```

11.6.2.9.2.16 Values

Definition

Holds an array of ISeries<double> objects holding hold the indicator's underlying calculated values. ISeries<double> values are added to this array when calling the AddPlot() method. In case of a MultiSeries indicator synched to the primary series.

Property Value

A collection of `ISeries<double>` objects.

Syntax

`Values[int index]`

Examples

```
// OnBarUpdate method of a custom indicator
protected override void OnBarUpdate()
{
     // Ensures we have enough bars loaded for our indicator
     if (CurrentBar < 1)
          return;

     // Evaluates the indicator's secondary value 1 bar ago and
sets the value of the indicator
     // for the current bar being evaluated
     if (Values[1][1] < High[0] - Low[0])
          Value[0] = High[0] - Low[0];
     else
          Value[0] = High[0] - Close[0];
}
```

11.6.2.9.2.17 Weighted

Definition

A collection of historical bar weighted prices. Weighted price = (High + Low + Close + Close) / 4.

Property Value

An `ISeries<double>` type object. Accessing this property via an index value `[int barsAgo]` returns a `double` value representing the price of the referenced bar.

Syntax

```
Weighted
Weighted[int barsAgo]
```

Examples

```
// Current bar weighted price
double barWeigthedPrice = Weighted[0];

// Weighted price of 10 bars ago
double barWeigthedPrice = Weighted[10];

// Current bar value of a 20 period exponential moving average of
weighted prices
double value = EMA(Weighted, 20)[0];
```

11.6.2.9.2.18 Weighteds

Definition

Holds an array of ISeries<double> objects holding historical bar weighted prices. An ISeries<double> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the weighted prices of all Bars objects in a multi-instrument or multi-time frame script.

Property Value

An array of `ISeries<double>` objects.

Syntax

```
Weighteds[int barSeriesIndex][int barsAgo]
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's weighted price to the 5-minute
bar's weighted price
     if (Weighteds[0][0] > Weighteds[1][0])
          Print("The primary bar's weighted price is greater");
}
```

11.6.2.9.3 TimeSeries<DateTime>

Definition

Represents historical time stamps as an ISeries<DateTime> interface which can be used for custom NinjaScript object calculations.

Note: In most cases, you will access the historical time series using a core event handler such as OnBarUpdate. For more advance developers, you may find situations where you

wish to access historical time series outside of the core event methods, such as your own custom mouse click. In these advanced scenarios, you may run into situations where the barsAgo pointer is not in sync with the current bar, which may cause errors when trying to obtain this information. In those cases, use the Bars.Get...() methods with the absolute bar index, e.g., Bars.GetTime(), etc.

Single ISeries<DateTime>

Time	A collection of historical bar time stamp values.

Multi-Time Frame ISeries<DateTime>

Times	Holds an array of `ISeries<DateTime>` objects holding historical bar times

11.6.2.9.3.1 Time

Definition

A collection of historical bar time stamp values.

Property Value

An `ISeries<DateTime>` object.

Syntax

`Time`

`Time[int barsAgo]` (returns a DateTime structure)

Examples

```
// Prints the current bar time stamp
Print(Time[0].ToString());

// Checks if current time is greater than the bar time stamp
if (DateTime.Now.Ticks > Time[0].Ticks)
     Print("Do something");
```

11.6.2.9.3.2 Times

Definition

Holds an array of ISeries<DateTime> objects holding historical bar times. A ISeries<DateTime> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the times of all Bars objects in a multi-instrument or multi-time frame script.

Property Value

An array of `ISeries<DateTime>` objects.

Syntax

`Times[int barSeriesIndex][int barsAgo]`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's time to the 5-minute bar's time
     if (Times[0][0].Ticks > Times[1][0].Ticks)
          Print("The current bar's time is greater");
}
```

11.6.2.9.4 VolumeSeries<double>

Definition

Represents historical volume data as ISeries<double> interface which can be used for custom NinjaScript object calculations

Note: In most cases, you will access the historical volume series using a core event handler such as OnBarUpdate. For more advance developers, you may find situations where you wish to access historical volume series outside of the core event methods,

such as your own custom mouse click. In these advanced scenarios, you may run into situations where the barsAgo pointer is not in sync with the current bar, which may cause errors when trying to obtain this information. In those cases, use the Bars.Get...() methods with the absolute bar index, e.g., Bars.GetVolume().

Single ISeries<double>

Volume	A collection of historical bar volume values.

Multi-Time Frame ISeries<double>

Volumes	Holds an array of `ISeries<double>` objects holding historical bar volume.

11.6.2.9.4.1 Volume

Definition
A collection of historical bar volume values.

Note: For working with Cryptocurrency instruments which report volume fractional, please use the VOL() indicator series, or store the volume for your script in a custom variable and convert alongside our VOL() indicator (`Instrument.MasterInstrument.InstrumentType == InstrumentType.CryptoCurrency ? Core.Globals.ToCryptocurrencyVolume((long)Volume[0]) : Volume[0]`).

Property Value
An `ISeries<double>` object. Accessing this property via an index `[int barsAgo]` returns a `double` value representing the volume of the referenced bar.

Syntax
```
Volume
Volume[int barsAgo]
```

Examples

```
// OnBarUpdate method
protected override void OnBarUpdate()
{
     // Is current volume greater than twice the prior bar's volume
     if (Volume[0] > Volume[1] * 2)
          Print("We have increased volume");

     // Is the current volume greater than the 20 period moving
average of volume
     if (Volume[0] > SMA(Volume, 20)[0])
          Print("Increasing volume");
}
```

11.6.2.9.4.2 Volumes

Definition

Holds an array of ISeries<double> objects holding historical bar volumes. An ISeries<double> object is added to this array when calling the AddDataSeries() method. Its purpose is to provide access to the volumes of all Bars objects in a multi-instrument or multi-time frame script.

Note: For working with Cryptocurrency instruments which report volume fractional, please use the VOL() indicator series, or store the volume for your script in a custom variable and convert alongside our VOL() indicator (`Instrument.MasterInstrument.InstrumentType == InstrumentType.CryptoCurrency ? Core.Globals.ToCryptocurrencyVolume((long)Volume[0]) : Volume[0]`).

Property Value

An array of `ISeries<double>` objects.

Syntax

`Volumes[int barSeriesIndex][int barsAgo]`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Adds a 5-minute Bars object to the strategy and is
automatically assigned
          // a Bars object index of 1 since the primary data the
strategy is run against
          // set by the UI takes the index of 0.
          AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     // Compares the primary bar's volume to the 5-minute bar's
volume
     if (Volumes[0][0] > Volumes[1][0])
          Print("The primary bar's volume is greater");
}
```

11.6.2.9.5 Count

Definition

Indicates the number total number of values in the ISeries<T> array. This value should always be in sync with the CurrentBars array for that series.

Method Return Value

A `int` representing the total size of the series

Syntax

`Count`

Examples

```
protected override void OnBarUpdate()
{
   Print("Input count: " + Input.Count);
}
```

11.6.2.9.6 GetValueAt()

Definition

Returns the underlying input value at a specified bar index value.

Method Return Value

A `double` value representing the value at a specified bar.

Syntax

```
GetValueAt(int barIndex)
ISeries<T>.GetValueAt(int barIndex)
```

Tip: If called directly from the instance of the NinjaScript object, the value which is returned corresponds to the input series the object is running. (e.g., Close, High, Low, SMA, etc.). If you're attempting to obtain another indicator value, you will need to pull this from the calculated indicator Value or Plot:

```
SMA(20).GetValueAt(123); // bar value
SMA(20).Values[0].GetValueAt(123); // indicator value
(Input as Indicator).Values[0].GetValueAt(123) // passed in indicator value
```

Parameters

barIndex	An `int` representing an absolute bar index value

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // make sure there are bars displayed on the chart and the chart
control is ready before running
   if (Bars == null || chartControl == null)
      return;

   // loop through all the visable bars on the chart
   for (int i = ChartBars.FromIndex - 1; i >= BarsRequiredToPlot;
i--)
   {
      double value = GetValueAt(i);
      Print(string.Format("The value at bar {0} is {1}", i,
value));
   }
}
```

11.6.2.9.7 IsValidDataPoint()

Definition

Indicates if the specified input is set at a barsAgo value relative to the current bar. Please also see the Reset() method for more information.

Notes:

- If called directly from the instance of the NinjaScript object, the value returned corresponds to the Input Series (e.g., Close, High, Low, SMA, etc.)
- When checking a Bar or PriceSeries, IsValidDataPoint() returns **true** as long as the barAgo value falls between 0 and the total count for that series. These are special series which always contain a value set at every slot index for multi-series scripting purposes (e.g., comparing two price series with various session templates, or one series has more ticks than the other)
- For a Value series or custom Series<T>, IsValidDataPoint() returns **true** or **false** depending on if you have set a value at that index location

Method Return Value

A `bool` value, when **true** indicates that specified data point is set; otherwise **false**.

Syntax

```
IsValidDataPoint(int barsAgo)
ISeries<T>.IsValidDataPoint(int barsAgo)
```

Warning: Calling IsValidDataPoint() will only work a MaximumBarsLookBackInfinite series. Attempting to check IsValidDataPoint() MaximumBarsLookBack256 series throw an error. Please check the Log tab of the Control Center. In addition since this method references BarsAgo data, and therefore cannot be used during OnRender (see note 5).- instead please use the IsValidDataPointAt during OnRender.

Parameters

barsAgo	An `int` representing from the current bar the number of historical bars the method will check.

Examples

```
protected override void OnBarUpdate()
{
   // only set plot value if hosted indicator is not reset
   if(SMA(20).IsValidDataPoint(0))
      MyPlot[0] = SMA(20)[0];
}
```

11.6.2.9.8 IsValidDataPointAt()

Definition

Indicates if the specified input is set at a specified bar index value. Please also see the Reset() method for more information.

Notes:

- If called directly from the instance of the NinjaScript object, the value returned corresponds to the Inputs Series (e.g., Close, High, Low, SMA, etc.)
- When checking a Bar or PriceSeries, IsValidDataPoint() returns **true** as long as the barIndex value falls between 0 and the total count for that series. These are special series which always contain a value set at every slot index for multi-series scripting purposes (e.g., comparing two price series with various session templates, or one series has more ticks than the other)
- For a Value series or custom Series<T>, IsValidDataPoint() returns **true** or **false** depending on if you have set a value at that index location

Method Return Value

A `bool` value, when **true** indicates that specified data point is set; otherwise **false**.

Warning: Calling IsValidDataPointAt() will only work a MaximumBarsLookBackInfinite series. Attempting to check IsValidDataPointAt() MaximumBarsLookBack256 series throw an error. Please check the Log tab of the Control Center

Syntax

```
IsValidDataPointAt(int barIndex)
ISeries<T>.IsValidDataPointAt(int barIndex)
```

Parameters

`barIndex`	An `int` representing an absolute bar index value

Examples

```
protected override void OnBarUpdate()
{
   // only set plot value if hosted indicator is not reset
   if(SMA(20).IsValidDataPointAt(CurrentBar))
      MyPlot[0] = SMA(20)[0];
}
```

11.6.2.9.9 MaximumBarsLookBack

Definition

Determines memory performance of custom Series<T> objects (such as Series<double>, Series<long>, etc.). When using **MaximumBarsLookBack.TwoHundredFiftySix**, only the last 256 values of the series object will be stored in memory and be accessible for reference. This results in significant memory savings when using multiple series objects. In the rare case should you need older values you can use **MaximumBarsLookBack.Infinite** to allow full access of the series.

Notes:

- ISeries<T> objects that hold bar data (such as Close, High, Volume, Time, etc) always use **MaximumBarsLookBack.Infinite** which ensures all data points are always accessible during the lifetime of your NinjaScript indicator or strategy.
- Series<double> objects that hold indicator plot values always use **MaximumBarsLookBack.Infinite** which ensures that charts always display the entire indicator's calculated values.

Property Value

A **MaximumBarsLookBack** enum value. Default value is **MaximumBarsLookBack.TwoHundredFiftySix**

Possible values are:

MaximumBarsLookBack.TwoHundredFiftySix	Only the last 256 values of the series object will be stored in memory and accessible for reference (improves memory performance)

MaximumBarsLookBack.Infinite	Allow full access of the series, but you will then not be able to utilize the benefits of memory optimization

Tip: A **MaximumBarsLookBack.TwoHundredFiftySix** series works as a circular ring buffer, which will "loop" when the series reaches full capacity. Specifically, once there are 256 entries in the series, new data added to the series overwrite the oldest data.

Syntax

MaximumBarsLookBack

Examples

Setting all custom series to use the default MaximumBarsLookBack

```
Series<double> myDoubleSeries = null;
Series<string> myStringSeries = null;

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Example Indicator";
      // Store all series values instead of only the last 256
values
      MaximumBarsLookBack = MaximumBarsLookBack.Infinite;
   }
   else if (State == State.DataLoaded)
   {
      // The custom Series<t> below are all constructed using only
the NinjaScriptBase object (i.e., "this")
      // therefore, the Series<T> MaximumBarsLookBack is taken from
the NinjaScript's configured MaximumBarsLookBack property
      myDoubleSeries = new Series<double>(this);
      myStringSeries = new Series<string>(this);
   }
}
```

Optimizing custom series to use unique MaximumBarsLookBack behavior

```
Series<double> myDoubleSeries = null;
Series<string> myStringSeries = null;

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Example Indicator";
   }
   else if (State == State.DataLoaded)
   {
      // The custom Series<t> below are constructed using
MaximumBarsLookBack parameter
      // therefore, each Series<t> will use their uniquely
specified MaximumBarsLookBack properites
      myDoubleSeries = new Series<double>(this,
MaximumBarsLookBack.Infinite);  // stores all values
      myStringSeries = new Series<string>(this,
MaximumBarsLookBack.TwoHundredFiftySix); // only the last 256
values (better performance)
   }
}
```

11.6.2.10 OnBarUpdate()

Definition

An event driven method which is called whenever a bar is updated. The frequency in which OnBarUpdate is called will be determined by the "Calculate" property. OnBarUpdate() is the method where all of your script's core bar based calculation logic should be contained.

Notes:

- For multi-timeframe and instrument scripts, the OnBarUpdate method is called for each Bars object of a strategy. You **MUST** filter for the exact bar update events using the "BarsInProgress" property you want your system logic to execute against.
- Hosted indicators will need to be accessed by the hosting script to ensure OnBarUpdate functionality. This can be done by: 1) Calling Update on the hosted indicator within the host script, 2) Including a plot in the hosted indicator and accessing the plot in the host script, 3) Including a plot in the hosted indicator and adding the indicator to the chart with AddChartIndicator (strategies only)

Related Methods and Properties

BarsPeriod	The primary Bars object time frame (period type and interval).
Calculate	Determines how often OnBarUpdate() is called for each bar.
Count	The total number of bars or data points.
CurrentBar	A number representing the current bar in a Bars object that the OnBarUpdate() method in an indicator or strategy is currently processing.
IsDataSeriesRequired	Determines if a Data Series is required for calculating this NinjaScript object.
IsFirstTickOfBar	Indicates if the incoming tick is the first tick of a new bar.
IsResetOnNewTradingDays	Determines if the specified bar series is using Break at EOD.
IsTickReplays	Indicates the specified bar series is using Tick Replay.
Update()	Forces the OnBarUpdate() method to be called so that indicator values are updated to the current bar.

Method Return Value

This method does not return a value.

Syntax

You must override this method with the following syntax:

```
protected override void OnBarUpdate()
{

}
```

Tip: The NinjaScript code wizards automatically generates the method syntax for you.

Parameters
This method does not take any parameters

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 1)
          return;

     // Compares the primary bar's low price to the 5-minute bar's
low price
     if (Low[0] > Lows[1])
          Print("The current bar's low price is greater");
}
```

11.6.2.10.1 BarsPeriod

Definition
The primary Bars object time frame (period type and interval).

Warning: This property should **NOT** be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**

Property Value
A Bars series object representing the time frame of the Bars.

Syntax

BarsPeriod.BarsPeriodType	The type of bars used for the period, as well as the enumeration value under which the any of the 14 default NinjaTrader types are registered. Possible values include: Bars: 0

PeriodType.Tick	
BarsPeriodType.Volume	1
BarsPerio	2

dType.Range	
BarsPeriodType.Second	3
BarsPeriodTyp	4

e.Minute	
BarsPeriodType.Day	5
BarsPeriodType.Week	6

BarsPeriodType.Month	7
BarsPeriodType.Year	8
BarsPe	9

riodType.HeikenAshi	
BarsPeriodType.Kagi	10
BarsPer	11

iodType.Renko	
BarsPeriodType.PointAndFigure	12
Bar	13

sPeriodType.LineBreak	
BarsPeriodType.Volumetric	14

	Tip: When creating custom BarsTypes, it is recommended to pick high, unique enumeration value to avoid conflict from other BarsTypes that may be used by a single installation. `BarsPeriod = new BarsPeriod { BarsPeriodType = (BarsPeriodType)123456, BarsPeriodTypeName = "MyCustomBars", Value = 1 };`
BarsPeriod.BaseBarsPeriodType	Only relevant for HeikenAshi, Kagi, LineBreak, PointAndFigure and Volumetric Bars objects. Same possible values as BarsPeriod.BarsPeriodType
BarsPeriod.BaseBarsPeriodValue	Only relevant for HeikenAshi, Kagi, LineBreak, PointAndFigure and Volumetric Bars objects. Determines an integer value representing the basePeriodTypeValue parameter

BarsPeriod.MarketDataType	The data type used to build the bars. Possible values:
	`MarketDataType.Ask`
	`MarketDataType.Bid`
	`MarketDataType.Last`

BarsPeriod.PointAndFigurePriceType	Only relevant for PointAndFigure Bars objects. Possible values:
	`PointAndFigurePriceType.Close`
	`PointAndFigurePriceType.HighsAndLows`

BarsPeriod.ReversalType	Only relevant for Kagi Bars objects. Possible values:
	`ReversalType.Percent`
	`ReversalType.Tick`

BarsPeriod.Value	Determines an integer value representing the period parameter. • When using Kagi Bars objects this represents the "reversal" parameter

	• When using LineBreak Bars objects this represents the "lineBreakCount" parameter • When using PointAndFigure Bars objects this represents the "boxSize" parameter • When using Renko Bars objects this represents the "brickSize" parameter
BarsPeriod.Value2	Only relevant for PointAndFigure Bars objects. Determines an integer value representing the "reversal" parameter.

Examples

Checking BarsPeriod values

```
// Calculate only if there is a 100 tick chart or greater
protected override void OnBarUpdate()
{
     if (BarsPeriod.BarsPeriodType == BarsPeriodType.Tick &&
BarsPeriod.Value >= 100)
     {
          // Indicator calculation logic here
     }
}
```

Creating a new BarsPeriod object

```
protected override void OnStateChange()
{
      if (State == State.Configure)
      {
            // add a 1440 minute apple bars object using the RTH
session template
            AddDataSeries("AAPL", new BarsPeriod { BarsPeriodType =
 BarsPeriodType.Minute, Value = 1440 }, "US Equities RTH");

      }

      else if (State == State.DataLoaded)
      {
            // Print out the loaded bars period
            Print(Instrument.FullName + " " + BarsPeriod); // MSFT
1 Minute
            Print(BarsArray[1].Instrument.FullName + " " +
BarsArray[1].BarsPeriod);  // AAPL 1440 Minute
      }
}
```

11.6.2.10.2 Calculate

Definition

Determines how often OnBarUpdate() is called for each bar. **OnBarClose** means once at the close of the bar. **OnEachTick** means on every single tick. **OnPriceChange** means once for each price change. If there were two ticks in a row with the same price, the second tick would not trigger OnBarUpdate(). This can improve performance if calculations are only needed when new values are possible.

Notes:

1. On a historical data set, only the OHLCVT of the bar is known and not each tick that made up the bar. As a result, State.Historical data processes **OnBarUpdate()** only on the close of each historical bar even if this property is set to **OnEachTick** or **OnPriceChange**. You can use TickReplay or a Multi-time frame script to obtain intrabar data.
2. When set to Calculate **OnPriceChange**, the **OnBarUpdate()** method is **ONLY** called when the price has changed intrabar **OR** when the bar has closed

Property Value

An `enum` value determining the how frequently OnBarUpdate() will be called. Default value is set to Calculate.OnBarClose

Warning:
- This property should **ONLY** bet set from the **OnStateChange()** method during **State.SetDefaults** or **State.Configure**
- If your script relies on volume updates **OnPriceChange** should **NOT** be used since it can potentially miss volume updates if they occur at the same price

Syntax

```
Calculate.OnBarClose
Calculate.OnEachTick
Calculate.OnPriceChange
```

Tips:
1. Calculating indicators or systems for each incoming tick can be CPU intensive. Only calculate indicators on each incoming tick if you have a requirement to calculate it intra-bar.
2. For an example of how to separate some logic to be Calculate = Calculate.OnBarClose and other logic to be .OnEachTick please see this reference sample.
3. Embedded scripts within a calling parent script should not use a different Calculate property since it is already utilizing the Calculate property of the parent script (i.e. the strategy your indicator is called from).
4. Since the parent NinjaScript therefore governs this setting, it should be set to the highest needed setting satisfying all its childs.

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Calculate on the close of each bar
          Calculate = Calculate.OnBarClose;
     }
}
```

11.6.2.10.3 Count

Definition

The total number of bars or data points.

Property Value

An `int` value representing the the total number of bars.

Syntax
`Count`

Examples

```
//If there are less than 365 bars on the chart, text indicates how
many bars are on the chart
if (Count < 365)
{
	Draw.TextFixed(this, "tag1", "There are " + Count + " bars
on the chart", TextPosition.BottomRight);
}
```

Tip: CurrentBar value is guaranteed to be <= Count - 1. This is because of the NinjaTrader multi-threaded architecture, the Count value can have additional bars as inflight ticks come in to the system.

11.6.2.10.4 CurrentBar

Definition
A number representing the current bar in a Bars object that the OnBarUpdate() method in an indicator or strategy is currently processing. For example, if a chart has 100 bars of data, the very first bar of the chart (left most bar) will be number 0 (zero) and each subsequent bar from left to right is incremented by 1.

Note: In multi series processing, the CurrentBars starting value will be -1 until all series have processed the first bar.

Property Value
An `int` value that represents the current bar.

Syntax
`CurrentBar`

Examples

```
// OnBarUpdate method
protected override void OnBarUpdate()
{
     // Evaluates to make sure we have at least 20 or more bars
     if (CurrentBar < 20)
          return;

     // Indicator logic calculation code...
}
```

11.6.2.10.5 IsDataSeriesRequired

Definition

Determines if a Data Series is required for calculating this NinjaScript object. When set to false, data series related properties will not be displayed on the UI when configuring.

> **Note**: When set to **false**, methods and properties which are dependent on Bars will **NOT** be used. This means you will not receive any calls to OnBarUpdate() or be able to access historical bar prices.

Property Value

This property returns **true** if the NinjaScript requires a Data Series; otherwise, **false**. Default value is true.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

`IsDataSeriesRequired`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IsDataSeriesRequired = false;
     }
}
```

11.6.2.10.6 IsFirstTickOfBar

Definition

Indicates if the incoming tick is the first tick of a new bar. This property is only of value in scripts that run tick by tick which is when the Calculate property is set to **Calculate.OnEachTick** or **Calculate.OnPriceChange**.

Warning: This property should **NOT** be accessed outside of the OnBarUpdate() method.

Note: If a bar type is set up to remove the last bar on a chart, **IsFirstTickOfBar** will automatically be set to **True**.

Property Value

This property returns **true** if the incoming tick is the first tick of a new bar; otherwise, **false**.

Syntax

`IsFirstTickOfBar`

Tip:
In NinjaTrader's event driven framework, bar closures are signaled by the tick that opens the next bar. The price of the last tick of a bar can be referenced by checking `Close[1]` on **IsFirstTickOfBar**. For volume and tick based bars, Bars.TickCount and `Volume[0]` can be referenced to see if the number of ticks / volume meet the criteria to build a new bar.

Examples

```
// On a tick by tick strategy the only way you know when a bar is
closed is when
// the IsFirsTickOfBar is true.
protected override void OnBarUpdate()
{
     // Only process entry signals on a bar by bar basis (not tick
by tick)
     if (IsFirstTickOfBar)
     {
          if (CCI(20)[1] < -250)
               EnterLong();
          return;
     }

     // Process exit signals tick by tick
     if (CCI(20)[0] > 250)
          ExitLong();
}
```

11.6.2.10.7 IsResetOnNewTradingDays

Definition

Determines if the specified bar series is using Break at EOD

Note: The property available on the UI will override any values set in code. Please see the help guide topic on using Break at EOD for more information

Property Value

A `bool[]` when **true**, indicates the specified BarsArray is setup to run Break at EOD; otherwise **false**. Default value is **false**

Syntax

```
IsResetOnNewTradingDays[int idx]
```

Warning: This property should **NOT** be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";
   }

   else if (State == State.Configure)
   {
      //Add AAPL 1 minute with RTH trading hours, set to break EOD
      AddDataSeries("AAPL", new BarsPeriod() { BarsPeriodType =
BarsPeriodType.Minute, Value = 1 }, 50, "US Equities RTH", true);
   }

}
protected override void OnBarUpdate()
{
  //Print out the current bars series name and break EOD setting on
start up
  //   IsResetOnNewTradingDays[0]  Primary
  //   IsResetOnNewTradingDays[1]  AAPL

  if (CurrentBar == 0)
    Print(BarsArray[BarsInProgress].ToChartString() + " " +
IsResetOnNewTradingDays[BarsInProgress]);

  //Output:
  //ES 03-15 (1 Minute) True
  //AAPL (1 Minute) False
}
```

11.6.2.10.8 IsTickReplays

Definition

Indicates the specified bar series is using Tick Replay. Please see the help guide topic on using Tick Replay for general information on this mode.

Note: For a primary series, the **Tick Replay** option must be configured from the UI before a NinjaScript object can take use of this property. The setting on the Chart's Data Series menu will always take precedence for an object series which already exists on the user's chart.

Warning: This property should **NOT** be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**

Property Value

A `bool[]` when **true**, indicates the specified BarsArray is setup to run Tick Replay; otherwise **false**. Default value is **false**

Syntax

```
IsTickReplays[int idx]
```

Examples

```
protected override void OnStateChange()
{
   if(State == State.SetDefaults)
   {
       Name = "Examples Indicator";
   }

   else if (State == State.Configure)
   {
      AddDataSeries("AAPL", BarsPeriodType.Minute, 1);
   }
   else if (State == Data.Loaded)
   {
      // IsTickReplays[0] = true;
      // Programmatically setting this option here for Primary [0]
does not have any effect
      // Primary series must be configured from UI

      // It is not possible to combine Tick Replay series and non
Tick Replay series in a single chart or script
      // The assignment below would not be necessary if the primary
series were set to True via the UI
      // IsTickReplays[1] = true;
   }
}

protected override void OnBarUpdate()
{
   //Print out the current bars series name and tick replays
setting on start up
   if (CurrentBar == 0)
      Print(BarsArray[BarsInProgress].ToChartString() + " " +
IsTickReplays[BarsInProgress]);
}
```

11.6.2.10.9 Update()

Definition

Forces the OnBarUpdate() method to be called for all data series so that indicator values are updated to the current bar index. If the values are already up to date, the Update() method will not be run.

Notes:

- This method is only relevant in specific use cases and should only used by advanced programmers

- The additional overload where a bar index and BarsInProgress are specified should only be used when an indicator needs to be updated to a bar index that is not the CurrentBar index. For example, updating an indicator's secondary 1 tick data series to indicator.BarsArray[1].Count - 1, which is not the CurrentBar index. This is required for the proper function of Order Flow Cumulative Delta and Order Flow VWAP

When indicators are embedded (called) within a NinjaScript strategy, they are optimized to calculate only when they are called upon in a historical backtest. Since the NinjaTrader indicator model is very flexible, it is possible to create public properties on a custom indicator that return values of internal user defined variables. If these properties require that the OnBarUpdate() method is called before returning a value, include a call to this Update() method in the property getter.

Syntax

```
Update()
Update(int idx, int bip)
```

Parameters

idx	The current bar index value to update to
bip	The BarsInProgress to update

Examples

```
private double tripleValue = 0;

protected override void OnBarUpdate()
{
    if (CurrentBar < 20)
        return;

    tripleValue = SMA(20)[0] * 3;
    Value[0] = SMA(20)[0];
}

public double TripleValue
{
     get
     {
        //call OnBarUpdate before returning tripleValue
        Update();
        return tripleValue;
     }
}
```

11.6.2.11 OnConnectionStatusUpdate()

Definition

An event driven method used which is called for every change in connection status.

Method Return Value

This method does not return a value.

Syntax

You must override the method in your indicator with the following syntax:

```
protected override void OnConnectionStatusUpdate(ConnectionStatusEventArgs
connectionStatusUpdate)
{

}
```

Method Parameters

connectionStatusUpdate	A ConnectionStatusEventArgs object representing the most recent update in connection.

Status	Represents the status of the key adapter functionality. If the adapter supports live orders it will set Status to Disconnected when its order system is not connected.
PriceStatus	Represents the status of the price feed.

Examples

```
//Prints the status of the order system
protected override void
OnConnectionStatusUpdate(ConnectionStatusEventArgs
connectionStatusUpdate)
{
   if(connectionStatusUpdate.Status == ConnectionStatus.Connected)
   {
      Print("Connected for orders at " + DateTime.Now);
   }

   else if(connectionStatusUpdate.Status ==
ConnectionStatus.ConnectionLost)
   {
      Print("Connection for orders lost at: " + DateTime.Now);
   }
}

//Prints the status of the price feed
protected override void
OnConnectionStatusUpdate(ConnectionStatusEventArgs
connectionStatusUpdate)
{
   if(connectionStatusUpdate.PriceStatus ==
ConnectionStatus.Connected)
   {
      Print("Connected to price feed at " + DateTime.Now);
   }

   else if(connectionStatusUpdate.PriceStatus ==
ConnectionStatus.ConnectionLost)
   {
      Print("Connection to price feed lost at: " + DateTime.Now);
   }
}
```

11.6.2.11.1 ConnectionStatusEventArgs

Definition

ConnectionStatusEventArgs contains Connection-related information to be passed as an argument to the OnConnectionStatusUpdate() event.

Note: For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

The properties listed below are accessible from an instance of ConnectionStatusEventArgs:

Connection	The Connection object for which OnConnectionStatusUpdate() was called
Error	An ErrorCode thrown by the Connection object in question
NativeError	A string representing an error thrown by the connectivity provider
PreviousStatus	A ConnectionStatus object representing the status of the connection before this call to OnConnectionStatusUpdate()
Status	A ConnectionStatus object representing the new status of the connection
PreviousPriceStatus	A ConnectionStatus object representing the status of the connection's price feed before this call to OnConnectionStatusUpdate()
PriceStatus	A ConnectionStatus object representing the new status of the connection's price feed

Examples

```
// This method is fired on connection status events
private void OnConnectionStatusUpdate(object sender,
ConnectionStatusEventArgs e)
{
    // For multi-threading reasons, work with a copy of the
ConnectionStatusEventArgs to prevent situations in which the
EventArgs may already be ahead of us while in the middle processing
it.
    // This accomplishes the same goal as locking a collection to
prevent in-flight changes from affecting outcomes
    ConnectionStatusEventArgs eCopy = e;

    /* Dispatcher.InvokeAsync() is needed for multi-threading
considerations. When processing events outside of the UI thread,
and we want to
    influence the UI .InvokeAsync() allows us to do so. It can also
help prevent the UI thread from locking up on long operations. */
    Dispatcher.InvokeAsync(() =>
    {
        outputBox.AppendText(string.Format("{1} Status: {2}",
                Environment.NewLine,
                eCopy.Connection.Options.Name,
                eCopy.Status));
    });
}
```

11.6.2.12 OnFundamentalData()

Definition

An event driven method which is called for every change in fundamental data for the underlying instrument.

> **Note**: This method is **NOT** called on historical data (backtest)

Method Return Value

This method does not return a value.

Syntax

You must override the method in your strategy or indicator with the following syntax.

```
protected override void OnFundamentalData(FundamentalDataEventArgs
fundamentalDataUpdate)
{
```

```
}
```

Tip: The NinjaScript code wizards can automatically generate the method syntax for you when creating a new script.

Parameters

fundamentalDataUpdate	FundamentalDataEventArgs representing the recent change in fundamental data

Examples

```
protected override void
OnFundamentalData(FundamentalDataEventArgs
fundamentalDataUpdate)
{
     // Print some data to the Output window
     if (fundamentalDataUpdate.FundamentalDataType ==
FundamentalDataType.AverageDailyVolume)
          Print("The current ADV is " +
fundamentalDataUpdate.LongValue);
}
```

Tips

1. With multi-time frame and instrument strategies, OnFundamentalData() will be called for all unique instruments in your strategy. Use the BarsInProgress to filter the OnFundamentalData() method for a specific instrument.
2. Do not leave an unused OnFundamentalData() method declared in your NinjaScript object. This will unnecessarily attach a data stream to your script which uses unnecessary CPU cycles.

11.6.2.12.1 FundamentalDataEventArgs

Definition

Represents a change in fundamental data and is passed as a parameter in the OnFundamentalData() method.

Methods and Parameters

DateTimeValue	A DateTime value representing the time
DoubleValue	A double value representing fundamental data
FundamentalDataType	Possible values: AverageDailyVolume Beta CalendarYearHigh CalendarYearHighDate CalendarYearLow CalendarYearLowDate CurrentRatio DividendAmount DividendPayDate DividendYield EarningsPerShare FiveYearsGrowthPercentage High52Weeks High52WeeksDate HistoricalVolatility Low52Weeks Low52WeeksDate MarketCap NextYearsEarningsPerShare PercentHeldByInstitutions PriceEarningsRatio RevenuePerShare SharesOutstanding ShortInterest ShortInterestRatio VWAP
IsReset	A bool value representing if an UI reset is needed after a manual disconnect. Note: This is only relevant for columns. Whenever this property is true, the UI needs to be reset.
LongValue	A long value representing fundamental data
ToString()	A string representation of the FundamentalDataEventArgs object

Examples

```
protected override void OnFundamentalData(FundamentalDataEventArgs
fundamentalDataUpdate)
{
     // Print some data to the Output window
     if (fundamentalDataUpdate.FundamentalDataType ==
FundamentalDataType.AverageDailyVolume)
          Print("Average Daily Volume = " +
fundamentalDataUpdate.LongValue);
     else if (fundamentalDataUpdate.FundamentalDataType ==
FundamentalDataType.PriceEarningsRatio)
          Print("P/E Ratio = " +
fundamentalDataUpdate.DoubleValue);
}
```

Tips

1. Not all connectivity providers support all FundamentalDataTypes.
2. EarningsPerShare on eSignal is a trailing twelve months value. On IQFeed it is the last quarter's value.
3. RevenuePerShare is a trailing twelve months value.

11.6.2.13 OnMarketData()

Definition

An event driven method which is called and guaranteed to be in the correct sequence for every change in level one market data for the underlying instrument. OnMarketData() can include but is not limited to the bid, ask, last price and volume.

Notes

1. This is a real-time data stream and can be CPU intensive if your program code is compute intensive (not optimal)
2. By default, this method is not called on historical data (backtest), however it can be called historically by using TickReplay
3. If used with TickReplay, please keep in mind Tick Replay **ONLY** replays the Last market data event, and only stores the best inside bid/ask price at the time of the last trade event. You can think of this as the equivalent of the bid/ask price at the time a trade was reported. As such, historical bid/ask market data events (i..e, bid/ask volume) **DO NOT** work with Tick Replay. To obtain those values, you need to use a historical bid/ask series separately from TickReplay through OnBarUpdate(). More information can be found under Developing for Tick Replay.
4. With multi-time frame and instrument strategies, a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the

same). The market data subscription behavior occurs both in real-time and during TickReplay historical
5. Do not leave an unused **OnMarketData()** method declared in your NinjaScript object. This will unnecessarily attach a data stream to your strategy which uses unnecessary CPU cycles.
6. Should you wish to run comparisons against prior values you will need to store and update local variables to track the relevant values.
7. The **OnMarketData()** method is expected to be called after OnBarUpdate()

Method Return Value

This method does not return a value.

Syntax

You must override the method in your strategy or indicator with the following syntax.

```
protected override void OnMarketData(MarketDataEventArgs marketDataUpdate)
{

}
```

Tip: The NinjaScript code wizards can automatically generate the method syntax for you when creating a new script.

Parameters

marketDataUpdate	MarketDataEventArgs representing the recent change in market data

Examples

```
protected override void OnMarketData(MarketDataEventArgs
marketDataUpdate)
{
     // Print some data to the Output window
     if (marketDataUpdate.MarketDataType == MarketDataType.Last)
           Print(string.Format("Last = {0} {1} ",
marketDataUpdate.Price, marketDataUpdate.Volume));
     else if (marketDataUpdate.MarketDataType ==
MarketDataType.Ask)
          Print(string.Format("Ask = {0} {1} ",
marketDataUpdate.Price, marketDataUpdate.Volume));
     else if (marketDataUpdate.MarketDataType ==
MarketDataType.Bid)
          Print(string.Format("Bid = {0} {1}",
marketDataUpdate.Price, marketDataUpdate.Volume));
}
```

11.6.2.13.1 MarketDataEventArgs

Definition

Represents a change in level one market data and is passed as a parameter in the OnMarketData() method.

Methods and Parameters

Ask	A `double` value representing the ask price
Bid	A `double` value representing the bid price
Instrument	A `Instrument` object representing the instrument of the market data
IsReset	A `bool` value representing if a UI reset is needed after a manual disconnect. **Note**: This is only relevant for columns. Whenever this property is true, the UI needs to be reset.
MarketDataType	Possible values are: MarketDataType.Ask MarketDataType.Bid MarketDataType.DailyHigh MarketDataType.DailyLow

	MarketDataType.DailyVolume MarketDataType.Last MarketDataType.LastClose (prior session close) MarketDataType.Opening MarketDataType.OpenInterest (supported by IQFeed, Kinetick) MarketDataType.Settlement
Price	A `double` value representing the price
Time	A `DateTime` structure representing the time
ToString()	A `string` representation of the MarketDataEventArgs object
Volume	A `long` value representing volume

Critical: If used with TickReplay, please keep in mind Tick Replay **ONLY** replays the Last market data event, and only stores the best inside bid/ask price at the time of the last trade event. You can think of this as the equivalent of the bid/ask price at the time a trade was reported. Please also see Developing for Tick Replay.

Tips

- Not all connectivity providers support all MarketDataTypes.
- For an example of how to use IsReset please see \MarketAnalyzerColumns\AskPrice.cs

Examples

```
protected override void OnMarketData(MarketDataEventArgs
marketDataUpdate)
{
     // Print some data to the Output window
     if (marketDataUpdate.MarketDataType == MarketDataType.Last)
          Print("Last = " + marketDataUpdate.Price + " " +
marketDataUpdate.Volume);
     else if (marketDataUpdate.MarketDataType ==
MarketDataType.Ask)
          Print("Ask = " + marketDataUpdate.Price + " " +
marketDataUpdate.Volume);
     else if (marketDataUpdate.MarketDataType ==
MarketDataType.Bid)
          Print("Bid = " + marketDataUpdate.Price + " " +
marketDataUpdate.Volume);
}
```

11.6.2.14 OnMarketDepth()

Definition

An event driven method which is called and guaranteed to be in the correct sequence for every change in level two market data (market depth) for the underlying instrument. The OnMarketDepth() method can be used to build your own level two book.

Notes

1. This is a real-time data stream and can be CPU intensive if your program code is compute intensive (not optimal)
2. This method is not called on historical data (backtest)

Method Return Value

This method does not return a value.

Syntax

You must override the method in your strategy or indicator with the following syntax:

```
protected override void OnMarketDepth(MarketDepthEventArgs marketDepthUpdate)
{

}
```

Tip: The NinjaScript code wizards can automatically generate the method syntax for you

when creating a new script.

Parameters

marketDepthUpdate	MarketDepthEventArgs representing the recent change in market data

Examples

```
protected override void OnMarketDepth(MarketDepthEventArgs
marketDepthUpdate)
{
     // Print some data to the Output window
     if (marketDepthUpdate.MarketDataType == MarketDataType.Ask &&
marketDepthUpdate.Operation == Operation.Update)
          Print(string.Format("The most recent ask change is {0}
{1}", marketDepthUpdate.Price, marketDepthUpdate.Volume));
}
```

Tips

1. With multi-time frame and instrument strategies, OnMarketDepth will be called for all unique instruments in your strategy. Use the BarsInProgress to filter the OnMarketDepth() method for a specific instrument. (BarsInProgress will return the first BarsInProgress series that matches the instrument for the event)
2. Do not leave an unused OnMarketDepth() method declared in your NinjaScript object. This will unnecessarily attach a data stream to your strategy which uses unnecessary CPU cycles.
3. Should you wish to run comparisons against prior values you will need to store and update local variables to track the relevant values.
4. With NinjaTrader being multi-threaded, you should not rely on any particular sequence of events like OnMarketDepth() always being called before OnMarketData() or vice versa.

11.6.2.14.1 MarketDepthEventArgs

Definition

Represents a change in level two market data also known as market depth and is passed as a parameter in the OnMarketDepth() method.

Methods and Parameters

Instrument	A `Instrument` object representing the instrument of the market data
IsReset	A `bool` value representing if a UI reset is needed after a manual disconnect. Note: This is only relevant for columns. Whenever this property is true, the UI needs to be reset.
MarketDataType	Possible values are: MarketDataType.Ask MarketDataType.Bid
MarketMaker	A `string` representing the market maker id
Operation	Represents the action you should take when building a level two book. Possible values are: Operation.Add Operation.Update Operation.Remove
Position	An `int` value representing the zero based position in the depth ladder.
Price	A `double` value representing the price
Time	A `DateTime` structure representing the time
ToString()	A `string` representation of the MarketDataEventArgs object
Volume	A `long` value representing volume

Examples

```
protected override void OnMarketDepth(MarketDepthEventArgs
marketDepthUpdate)
{
     // Print some data to the Output window
     if (marketDepthUpdate.MarketDataType == MarketDataType.Ask &&
marketDepthUpdate.Operation == Operation.Update)
          Print("The most recent ask change is " +
marketDepthUpdate.Price + " " + marketDepthUpdate.Volume);
}
```

Tip: For an example of how to use IsReset please see \MarketAnalyzerColumns\AskPrice.cs

11.6.2.15 OnStateChange()

Definition

An event driven method which is called whenever the script enters a new State. The **OnStateChange()** method can be used to configure script properties, create one-time behavior when going from historical to real-time, as well as manage clean up resources on termination.

Notes:

- Viewing any UI element which lists NinjaScript classes (such as the Indicators or Strategies window, a chart's Chart Style dropdown menu, etc.) will initialize all classes of that Type when it is opened, which causes each script to enter **State.SetDefaults**, even if it is not actively configured or running in any window. It is important to keep this in mind when adding logic within **State.SetDefaults** in **OnStateChange()**, as this logic will be processed each time the script is initialized. For example, opening the Indicators window will trigger **State.SetDefaults** for all indicators in the system, and closing the Indicators window will trigger **State.Terminated** for all Indicators. In addition, disconnecting or connecting to a data provider can cause State transitions for any currently active scripts. Further discussion of this aspect of the state change model can be found via Understanding the lifecycle of your NinjaScript objects.
- When an indicator is configured on a chart while a Compile is taking place in the NinjaScript Editor, it can appear that the script passes through **State.Terminated**. However, this is the result of a copy of the script being initialized at compile-time, NOT the result of the indicator on the chart being disabled and re-initialized.

Related Methods and Properties

SetState()	Method is used for changing the State of any running NinjaScript object.
State	Represents the current progression of the object as it advances from setup, processing data, to termination.

Method Return Value

This method does not return a value.

Syntax

See example below. The NinjaScript wizards automatically generate the method syntax for you.

Possible states are:

State Name	This state is called when	This state is where you should
State.SetDefaults	**SetDefaults** is always called when displaying objects in a UI list such as the Indicators dialogue window since temporary objects are created for the purpose of UI display	• Keep as lean as possible • Set default values (pushed to UI)
State.Configure	**Configure** is called after a user adds an object to the applied list of objects and presses the OK or Apply button. This state is called only once for the life of the object.	• Add additional data series via AddDataSeries() • Declare custom resources • Override and configure values set by the UI
State.Active	**Active** is called once after the object is configured and is ready to process data (DrawingTools could see multiple calls as internally an object for hit testing is cloned)	• Used for objects such as Share Service which do not process price series data • Indicate the object is ready to being processing information

State.DataLoaded	**DataLoaded** is called only once after all data series have been loaded.	• Use for logic that needs to access data related objects like Bars, Instruments, BarsPeriod, TradingHours or instantiating indicators • Notify that all data series have been loaded • Initialize any class level variables (including custom `Series<T>` objects)
State.Historical	**Historical** is called once the object begins to process historical data. This state is called once when running an object in real-time. This object is called multiple times when running a backtest optimization and the property IsInstantiatedOnEachOptimizationIteration is true (default behavior)	• Notify that the object is processing historical data
State.Transition	**Transition** is called once as the object has finished processing historical data but before it starts to process realtime data.	• Notify that the indicator or strategy is is transitioning to realtime data • Prepare realtime related resources
State.Realtime	**Realtime** is called once when the object begins to process realtime data.	• Notify that the indicator or strategy is processing realtime data • Execute realtime related logic
State.Terminated	**Terminated** is called once when the object terminates.	• Notify the object is shutting down • Use to clean up/dispose of resources

Active States vs Data Processing States

After **State.Configure,** each type of NinjaScript type has its own state management system which can be classified under two categories:

- **Active state:** State.Active
- **Data Processing states:** State.DataLoaded, State.Historical, State.Transition, State.Realtime

The table below lists each NinjaScript type and it's designed state management system:

NinjaScript Type	State Management System
AddOns*	Active state
BarTypes	Active state
ChartStyles	Active state
DrawingTools	Active state
Indicators	Data Processing states
ImportTypes	Active state
Market Analyzer Columns	Data Processing states
OptimizationFitnesses	Active state
Optimizers	Active state
PerformanceMetrics	Active state
ShareServices	Active state
Strategies	Data Processing states
SuperDOM Columns	Active state

Tips:

- Resources created in **State.Configure** and not disposed of immediately will be kept and utilized if the NinjaScript object resides in grids (e.g. Strategy tab on Control Center), even if it is not enabled. Try to create resources in **State.Historical** or **State.DataLoaded** instead, if possible.
- **State.Historical** is called multiple times when running a backtest optimization on a strategy and the property "IsInstantiatedOnEachOptimizationIteration" is **true** (default behavior).

- Embedded scripts within a calling parent script should not use a different Calculate property since it is already utilizing the Calculate property of the parent script (i.e. the strategy your indicator is called from).
- Since the parent NinjaScript therefore governs this setting, it should be set to the highest needed setting satisfying all its childs.
- When instantiating indicators in a Multi-Series script in OnStateChange, the input any hosted indicator is running on should be explicitly stated (since a specific BarsInProgress is not guaranteed)

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      // Calculate once at the end of every single bar
      Calculate = Calculate.OnBarClose;

      // Add two plots
      AddPlot(Brushes.Blue, "Upper"));
      AddPlot(Brushes.Orange, "Lower"));
   }

   else if (State == State.Configure)
   {
      // Adds a 5-minute Bars object to the strategy and is
automatically assigned
      // a Bars object index of 1 since the primary data the
strategy is run against
      // set by the UI takes the index of 0.
      AddDataSeries("AAPL", BarsPeriodType.Minute, 5);
   }
}
```

11.6.2.15.1 SetState()

Definition

This method is used for changing the State of any running NinjaScript object.

Notes:

- Attempting to set a **State** earlier than the current **State** will be ignored
- Calling **SetState()** multiple times will be ignored to prevent the object from erroneously setting states unexpectedly

- Setting **State** to **State.Terminated** is meant as a way to abort the strategy as it is running. Doing this in a Strategy Analyzer backtest will abort the backtest entirely, and no partial backtest results will be shown.
- After setting **State.Terminated**, you should return from the calling method to help ensure subsequent logic is not processed asynchronously to OnStateChange()

Method Return Value

This method does not return a value.

Syntax

```
SetState(State state)
```

Warning: This method should only be call after the State reaches State.DataLoaded

Parameters

state	The State to be set

Examples

```
protected override void OnBarUpdate()
{
   // Terminate strategy at 2PM
   if (ToTime(Time[0]) == 140000)
   {
      SetState(State.Terminated);
      return;
   }
}
```

11.6.2.15.2 State

Definition

Represents the current progression of the object as it advances from setup, processing data, to termination. These states can be used for setting up or declaring various resources and properties.

Note: More detailed explanation of various states along with examples can be found in the OnStateChange() method section of this help guide. You can also attempt to set a new **State** using the SetState() method.

Property Value

An `enum` value representing the current state of the object. Possible values are:

SetDefaults	Default values are set (pushed to UI).
Configure	User the presses the OK or Apply button.
Active	Object is configured and is ready to receive instructions
DataLoaded	All data series have been loaded
Historical	Begins to process historical data
Transition	Finished processing historical data
Realtime	Begins to process realtime data.
Terminated	Begins to shut down

Syntax

```
State
```

Examples

Understanding the sequence of States

```
protected override void OnStateChange()
{
   Print(DateTime.Now + ": Current State is State."+State);
}
```

Using State to only process real-time data

```
protected override void OnBarUpdate()
{
   // only process real-time OnBarUpdate events
   if (State == State.Historical)
      return;

   //rest of logic
}
```

11.6.2.16 SessionIterator

Definition

Allows you to traverse through various trading hours data elements which apply to a segment of bars.

Note: Should you wish to obtain trading hours information for historical bar values, you need to construct and store your own session iterator object based of the desired bars series array.

Parameters

bars	The Bars object used to create the SessionIterator

Warning: The properties in this class should **NOT** be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**

Methods and Properties

ActualSessionBegin	Obtains the sessions start day and start time converted to the PC's local time zone
ActualSessionEnd	Obtains the sessions end day and end time converted to the PC's local time zone
ActualTradingDayEndLocal	Returns the sessions End-Of-Day (EOD) in the local timezone

ActualTradingDayExchange	Obtains the date of a session representing the trading date of the exchange
CalculateTradingDay()	Calculates the current trading date of a specified date
GetNextSession()	Calculates the next available session relative to a specified date
GetTradingDay()	Returns the actual trading date based on the exchange
GetTradingDayBeginLocal()	Converts the trading day begin time from the exchange timezone to local time
GetTradingDayEndLocal()	Converts the trading day end time from the exchange timezone to local time
IsInSession()	Indicates if a specified date is within the bounds of the current session
IsNewSession()	Indicates if a specified time is greater than the actual session end of the current session
IsTradingDayDefined()	Indicates if a trading day is defined for a specific date

Tip: In order to calculate a session information for another **multi-instrument** or **multi-time frame** script, you can pass in the desired BarsArray array value as the **SessionIterator** bars object.

Examples

```
private SessionIterator sessionIterator;

protected override void OnStateChange()
{
   if (State == State.DataLoaded)
   {
      //stores the sessions once bars are ready, but before
OnBarUpdate is called
      sessionIterator = new SessionIterator(Bars);
   }
}

protected override void OnBarUpdate()
{
   // on new bars session, find the next trading session
   if (Bars.IsFirstBarOfSession)
   {
      Print("Calculating trading day for " + Time[0]);
      // use the current bar time to calculate the next session
      sessionIterator.GetNextSession(Time[0], true);

      // store the desired session information
      DateTime tradingDay    =
sessionIterator.ActualTradingDayExchange;
      DateTime beginTime       =
sessionIterator.ActualSessionBegin;
      DateTime endTime      = sessionIterator.ActualSessionEnd;

      Print(string.Format("The Current Trading Day {0} starts at
{1} and ends at {2}",
                          tradingDay.ToShortDateString(),
beginTime, endTime));
      // Output:
      // Calculating trading day from 9/30/2015 4:01:00 PM
      //The Current Trading Day 10/1/2015 starts at 9/30/2015
4:00:00 PM and ends at 10/1/2015 3:00:00 PM
   }
}
```

11.6.2.16.1 ActualSessionBegin

Definition

Obtains the sessions start date and start time converted to the user's configured Time Zone.

Note: In order to obtain historical **ActualSessionBegin** information, you must call GetNextSession() from a stored **SessionIterator** object.

Property Value
A DateTime structure that represents beginning of a trading session.

Syntax
<sessionIterator>.ActualSessionBegin

Example


```
SessionIterator sessionIterator;

protected override void OnStateChange()
{
   if (State == State.Historical)
   {
      sessionIterator = new SessionIterator(Bars);
   }
}

protected override void OnBarUpdate()
{
   // on new bars session, find the next trading session
   if (Bars.IsFirstBarOfSession)
   {
      // use the current bar time to calculate the next session
      sessionIterator.GetNextSession(Time[0], true);

      Print("The current session start time is " +
sessionIterator.ActualSessionBegin);
   }
}
```

11.6.2.16.2 ActualSessionEnd

Definition
Obtains the session's end date and end time converted to the user's configured Time Zone.

> **Note**: In order to obtain historical **ActualSessionEnd** information, you must call GetNextSession() from a stored **SessionIterator** object.

Property Value
A DateTime structure that represents end of a trading session.

Syntax
<sessionIterator>.ActualSessionEnd

Example

```
SessionIterator sessionIterator;

protected override void OnStateChange()
{
   if (State == State.Historical)
   {
      sessionIterator = new SessionIterator(Bars);
   }
}

protected override void OnBarUpdate()
{
   // on new bars session, find the next trading session
   if (Bars.IsFirstBarOfSession)
   {
      // use the current bar time to calculate the next session
      sessionIterator.GetNextSession(Time[0], true);

      Print("The current session end time is " +
sessionIterator.ActualSessionEnd);
   }
}
```

11.6.2.16.3 ActualTradingDayEndLocal

Definition
Returns the session's End-Of-Day (EOD) in the user's configured timezone.

> **Note**: In order to obtain historical **ActualTradingDayEndLocal** information, you must call GetNextSession() from a stored **SessionIterator** object.

Property Value
A `DateTime` structure that represents end of a trading day (EOD).

Syntax
<sessionIterator>.ActualTradingDayEndLocal

Example

```
SessionIterator sessionIterator;

protected override void OnStateChange()
{
   if (State == State.Historical)
   {
      sessionIterator = new SessionIterator(Bars);
   }
}

protected override void OnBarUpdate()
{
   // on new bars session, find the next trading session
   if (Bars.IsFirstBarOfSession)
   {
      // use the current bar time to calculate the next session
      sessionIterator.GetNextSession(Time[0], true);

      Print("The current session end of day is " +
sessionIterator.ActualTradingDayEndLocal);
   }
}
```

11.6.2.16.4 ActualTradingDayExchange

Definition

Obtains the date of a trading session defined by the exchange.

Notes:

1. In order to obtain historical **ActualTradingDayExchange** information, you must call GetNextSession() from a stored **SessionIterator** object.
2. The calculated value may differ from the current date as some trading sessions will begin before the actual calender date changes. For example, the "**CME US Index Futures ETH**" actual session started on 3/30/2015 at 5:00PM Central Time, however the **actual exchange trading day** would be considered 3/31/2015 12:00:00AM

Property Value

A `DateTime` structure that represents the trading day.

Syntax

`<sessionIterator>.ActualTradingDayExchange`

Example

```
SessionIterator sessionIterator;

protected override void OnStateChange()
{
   if (State == State.Historical)
   {
      sessionIterator = new SessionIterator(Bars);
   }
}

protected override void OnBarUpdate()
{
   // on new bars session, find the next trading session
   if (Bars.IsFirstBarOfSession)
   {
      // use the current bar time to calculate the next session
      sessionIterator.GetNextSession(Time[0], true);

      Print("The current exchange trading day is " +
sessionIterator.ActualTradingDayExchange);
   }
}
```

11.6.2.16.5 CalculateTradingDay()

Definition

Calculates the trading date of the time value passed in as the timeLocal argument. This method may need to be used before you can accurately determine various session properties such as ActualSessionBegin or ActualTradingDayEndLocal, etc. **CalculateTradingDay()** also checks the local date/time against the exchange's current date/time to ensure that the script is in sync with the exchange's current day.

Warning: This method is resource intensive and should **ONLY** be reserved for situations when calculations would be limited to a few specific use cases.

Property Value

This method does not return a value.

Parameters

timeLocal	The DateTime value used to calculate the trading day.

includesEndTimeStamp	A bool determining if a timestamp of <n>:00 should fall into the current session. (e.g., used for time based intraday series such as minute or second).

Syntax

<sessionIterator>.CalculateTradingDay(DateTime timeLocal, bool includesEndTimeStamp)

Example

```
protected override void OnDataPoint(Bars bars, double open, double
high, double low, double close, DateTime time, long volume, bool
isBar, double bid, double ask)
{
   // build the bars type session iterator from the bars object
provided
   if (SessionIterator == null)
      SessionIterator = new SessionIterator(bars);

   // calculate the trading day of the time value provided
   SessionIterator.CalculateTradingDay(time, false);

   // add a new bar using the sessions exchanges date
   AddBar(bars, open, high, low, close,
SessionIterator.ActualTradingDayExchange, volume);
}
```

11.6.2.16.6 GetNextSession()

Definition

Calculates the next available session relative to the "timeLocal" value used in the method's input.

Note: This method needs to be used before you can accurately determine various session properties such as ActualSessionBegin or ActualTradingDayEndLocal, etc.

Property Value

A bool value when **true** indicates the method was able to successfully calculate the next trading session; otherwise **false**.

Warning: This method is resource intensive and should **ONLY** be reserved for situations

when calculations would be limited to a few specific use cases. For example, calling this method for each bar in the OnBarUpdate() method would **NOT** be recommended.

Parameters

timeLocal	The DateTime value used to calculate the next trading day.
includesEndTimeStamp	A `bool` determining if a timestamp of <n>:00 should fall into the current session. (e.g., used for time based intraday series such as minute or second).

Syntax

`<sessionIterator>.GetNextSession(DateTime timeLocal, bool` includesEndTimeStamp`);`

Example

Getting Next Session of the Primary Bars Object

```
SessionIterator sessionIterator;

protected override void OnStateChange()
{
   if (State == State.Historical)
   {
      sessionIterator = new SessionIterator(Bars);
   }
}

protected override void OnBarUpdate()
{
   // on new bars session, find the next trading session
   if (Bars.IsFirstBarOfSession)
   {
      // use the current bar time to calculate the next session
      sessionIterator.GetNextSession(Time[0], true);
   }
}
```

Getting Next Session of a Secondary Time Series

```
SessionIterator rthSessionIterator;

protected override void OnStateChange()
{
   if (State == State.Configure)
   {
      // add a 1440 minute bar using the RTH hours
      AddDataSeries(Instrument.FullName, new BarsPeriod
{ BarsPeriodType = BarsPeriodType.Minute, Value = 1440 }, "CME US
Index Futures RTH");
   }

   else if (State == State.Historical)
   {
      // store a session iterator built from the secondary (RTH)
bars
      rthSessionIterator = new SessionIterator(BarsArray[1]);
   }
}

protected override void OnBarUpdate()
{
   // on the primary bars session, find the next trading session
for the RTH bars
   if (Bars.IsFirstBarOfSession)
   {
      // use the current bar time to calculate the next RTH session
      rthSessionIterator.GetNextSession(Time[0], true);
   }
}
```

11.6.2.16.7 GetTradingDay()

Definition

Returns the actual trading date based on the exchange, calculated from a DateTime object passed with with the local time. GetTradingDay() calls CalculateTradingDay() on a custom SessionIterator object created by passing in a Bars object as an argument.

Warning: This method can **ONLY** be called when a **SessionIterator** was created with a 'Bars' parameter.

Property Value

A `DateTime` object representing the ActualTradingDayExchange property.

Syntax

```
<SessionIterator>.GetTradingDay(DateTime timeLocal)
```

Parameters

timeLocal	The DateTime value used to calculate the next trading day.

Example

```
// Declare a new custom SessionIterator
SessionIterator mySessionIterator;

protected override void OnStateChange()
{
    if (State == State.Historical)
    {
        // Instantiate mySessionIterator once in State.Configure
        mySessionIterator = new SessionIterator(BarsArray[0]);
    }
}

protected override void OnBarUpdate()
{
    // Obtain the ActualTradingDayExchange value for
mySessionIterator, based on today's date
    Print(mySessionIterator.GetTradingDay(DateTime.Now).ToString())
;
}
```

11.6.2.16.8 GetTradingDayBeginLocal()

Definition

Converts the trading day begin time from the exchange timezone to local time, and returns a DateTime object in the local timezone. The ActualTradingDayExchange property can be passed into GetTradingDayBeginLocal() for a quick timezone conversion.

Property Value

A `DateTime` object representing the exchange-based trading day begin time converted to local time.

Syntax

```
<SessionIterator>.GetTradingDayBeginLocal(DateTime tradingDayExchange)
```

Parameters

tradingDayExchange	The DateTime value used to calculate the trading day.

Example

```
private SessionIterator sessionIterator;

protected override void OnStateChange()
{
   if (State == State.DataLoaded)
   {
      //stores the sessions once bars are ready, but before
OnBarUpdate is called
      sessionIterator = new SessionIterator(Bars);
   }
}

protected override void OnBarUpdate()
{
    // Only process strategy logic starting three hours after
trading begins at the exchange
    if (Core.Globals.Now >=
sessionIterator.GetTradingDayBeginLocal(sessionIterator.ActualTradi
ngDayExchange).AddHours(3))
    {
        // Strategy logic here
    }
}
```

11.6.2.16.9 GetTradingDayEndLocal()

Definition

Converts the trading day end time from the exchange timezone to local time, and returns a DateTime object in the local timezone. The ActualTradingDayExchange property can be passed into GetTradingDayEndLocal() for a quick timezone conversion.

Property Value

A `DateTime` object representing the exchange-based trading day end time converted to local time.

Syntax

`<SessionIterator>.GetTradingDayEndLocal(DateTime tradingDayExchange)`

Parameters

tradingDayExchange	The DateTime value used to calculate the trading day.

Example

```
private SessionIterator sessionIterator;

protected override void OnStateChange()
{
   if (State == State.DataLoaded)
   {
      //stores the sessions once bars are ready, but before
OnBarUpdate is called
      sessionIterator = new SessionIterator(Bars);
   }
}

protected override void OnBarUpdate()
{
    // Only process strategy logic up until three hours prior to
the end of the trading day at the exchange
    if (Core.Globals.Now <=
sessionIterator.GetTradingDayEndLocal(sessionIterator.ActualTrading
DayExchange).AddHours(-3))
    {
        // Strategy logic here
    }
}
```

11.6.2.16.10 IsInSession()

Definition

Indicates a specified date is within the bounds of the current session, according to the configured Trading Hours template.

> **Note**: Additionally this method will internally trigger a GetNextSession() call to calculate the next available session relative to the "timeLocal" value used in the method's input.

Property Value

A `bool` value when **true** indicates the specified time is within the current trading session; otherwise **false**.

Parameters

timeLocal	The DateTime value used to calculate the next trading day.
includesEndTimeStamp	A `bool` determining if a timestamp of <n>:00 should fall into the current session. (e.g., used for time based intraday series such as minute or second).
isIntraDay	A `bool` determining if IsInSession() considers the time of day (when **true)** or only the date (when **false**)

Syntax

`<SessionIterator>.IsInSession(DateTime timeLocal, bool` includesEndTimeStamp, `bool isIntraDay)`

Example

```
private SessionIterator sessionIterator;

protected override void OnStateChange()
{
    if (State == State.Historical)
    {
        //stores the sessions once bars are ready, but before
OnBarUpdate is called
        sessionIterator = new SessionIterator(Bars);
    }
}

protected override void OnBarUpdate()
{
    // Only place an order if the time three hours from now will
still be within the current session
    if (sessionIterator.IsInSession(DateTime.Now.AddHours(3), true,
 true) /* && additional conditions here */)
        EnterLongStopMarket(CurrentDayOHL().High[0] + TickSize);
}
```

11.6.2.16.11 IsNew Session()

Definition

Indicates a specified time is greater than the ActualSessionEnd property on the configured Trading Hours template.

Property Value

A `bool` value when **true** indicates the specified time is later than **ActualSessionEnd**; otherwise **false**.

Parameters

time	The DateTime value used to compare
includesEndTimeStamp	A `bool` determining if a timestamp of <n>:00 should fall into the current session. (e.g., used for time based intraday series such as minute or second).

Syntax

`<SessionIterator>.IsNewSession(DateTime time, bool` includesEndTimeStamp`)`

Example

```
bool takeTrades;

protected override void OnBarUpdate()
{
    // Switch a bool named takeTrades to false when IsNewSession()
returns true.
    if (Bars.SessionIterator.IsNewSession(DateTime.Now, true)) ;
    {
        Alert("EOS", Priority.Medium, String.Format("New session
beginning. Waiting until {0} to begin trading again"), " ", 5,
Brushes.Black, Brushes.White);
        takeTrades = false;
    }

    // Set the bool back to true on the first bar of the new
session
    if (Bars.IsFirstBarOfSession)
        takeTrades = true;
}
```

11.6.2.16.12 IsTradingDayDefined()

Definition
Indicates a trading day is defined for a specific date.

Property Value
A `bool` value when **true** indicates that the date passed in as an argument is defined as a full or partial trading day in the configured **Trading Hours** template; otherwise **false**. Also returns **false** if the specified date is marked as a full-day exchange holiday.

Parameters

date	The DateTime value representing the date to check

Syntax
```
<SessionIterator>.IsTradingDayDefined(DateTime time);
```

Example
```
DateTime thanksGivingDay = new DateTime(2017, 11, 23);

// Determine if the current instrument's exchange is open for
trading on Thanksgiving day in 2017
if(Bars.SessionIterator.IsTradingDayDefined(thanksGivingDay))
    Print(String.Format("{0} will be open for trading on
Thanksgiving day, {1}", Instrument.MasterInstrument.Name,
thanksGivingDay.Date));
```

11.6.2.17 SimpleFont

Definition
Defines a particular font configuration.

> **Note**: **SimpleFont** objects are used for various Drawing methods, and can be used when defining UI element for Add-ons.

Constructors

`SimpleFont()`	Creates a **SimpleFont** object

	using a family name of "Arial" and a size of "12"
`SimpleFont(string familyName, int size)`	Creates a **SimpleFont** object using the specified family name and size

Methods and Properties

Bold	A bool value determining if the the Font is bold style
Family	A FontFamily representing a family of Fonts
Italic	A `bool` value determining if the the Font is italic style
Size	A `double` value determining the size of font in WPF units (please see the tip below)
Typeface	A Typeface used to represent the variation of the font used
ApplyTo()	Applies a custom SimpleFont object's properties (family, size, and style) to a Windows Control
ToDirectWriteTextFormat()	Converts a SimpleFont object to a SharpDX compatible font which can be used for chart rendering.

Tip: The WPF unit used is the default px one, so device independent pixels. With a default system DPI setting of 96, the physical pixel on the screen would be identical in size, but can vary if a custom DPI is employed. Both should not be confused with the points based font sizing known from other familiar Windows applications like Word, the advantage here is that the non points based size measurement will increase / decrease in size if the system DPI is changed - a more detailed discussion is located here.

Examples

```
// create custom Courier New, make it big and bold
NinjaTrader.Gui.Tools.SimpleFont myFont = new
NinjaTrader.Gui.Tools.SimpleFont("Courier New", 12) { Size = 50,
Bold = true };

Draw.Text(this, "myTag", false, "Hi There!", 0, Low[0], 5,
Brushes.Blue, myFont, TextAlignment.Center, Brushes.Black, null,
1);
```

11.6.2.17.1 ApplyTo()

Definition

Applies a custom SimpleFont object's properties (family, size, and style) to a Windows Control

Method Return Value

This method does not return a value.

Syntax

```
<SimpleFont>.ApplyTo(DependencyObject target)
```

target	The DependencyObject to apply the SimpleFont object

Examples

```
// Define the custom button control object
System.Windows.Controls.Button myButton = new
System.Windows.Controls.Button
{
   Name = "myButton",
   Content = "Buy",
   Foreground = Brushes.White,
   Background = Brushes.Green,
};

// Create a custom SimpleFont object and then apply it to the
button
SimpleFont myFont = new SimpleFont("Consolas", 22);

myFont.ApplyTo(myButton);
```

11.6.2.17.2 ToDirectWriteTextFormat()

Definition

Converts a SimpleFont object to a SharpDX compatible font which can be used for chart rendering.

> **Note**: For more information please see the educational resource on Using SharpDX for Custom Chart Rendering

Method Return Value

A DirectWrite.TextFormat object

> **Warning**: The returned **DirectWrite.TextFormat** object should be disposed of immediately when finished drawing text.

Syntax

```
<SimpleFont>.ToDirectWriteTextFormat()
```

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // Set text to chart label simple font object
   SharpDX.DirectWrite.TextFormat textFormat =
chartControl.Properties.LabelFont.ToDirectWriteTextFormat();

   // use the textFormat in a RenderTarget.DrawText() or
DrawTextLayout() method

   // do not forget to dispose text format when finished
   textFormat.Dispose();
}
```

11.6.2.18 System Indicator Methods

The "Indicators" reference provides definitions, syntax, parameter definitions and examples for NinjaTrader system indicator methods.

› Valid Input Data for Indicator Methods
› Accumulation/Distribution (ADL)

- Stochastics
- Stochastics Fast
- Stochastics RSI (StochRSI)
- Summation (SUM)
- Swing
- Time Series Forecast (TSF)
- Trend Lines
- True Strength Index (TSI)
- Ultimate Oscillator
- Volume (VOL)
- Volume Moving Average (VOLMA)
- Volume Oscillator
- Volume Rate of Change (VROC)
- Volume Up Down
- Vortex
- Williams %R
- Wiseman Alligator
- Wiseman Awesom Oscillator
- Woodies CCI
- Woodies Pivots
- ZigZag

11.6.2.18.1 Valid Input Data for Indicator Methods

System indicator methods require valid input data to function property. Indicator methods can accept the following forms of input data:

Default Input

The default input (Inputs[BarsInProgress]) of the custom indicator, Market Analyzer row or strategy is used if input is not specified.

```
// Printing the current value of the 10 period SMA of closing
prices
// using the default input.
double value = SMA(10)[0];
Print("The current SMA value is " + value.ToString());
```

Price Series

Open, High, Low, Close and Volume can all be used as input for an indicator method.

```
// Passing in the a price series of High prices and printing out
the current value of the
// 14 period simple moving average
double value = SMA(High, 14)[0];
Print("The current SMA value is " + value.ToString());
```

Indicator

Indicators can be used as input for other indicators.

```
// Printing the current value of the 20 period simple moving
average of a 14 period RSI
// using a data series of closing prices
double value = SMA(RSI(Close, 14, 3), 20)[0];
Print("The current SMA value is " + value.ToString());
```

Series<double>

Series<double> can be used as input for indicators.

```
// Instantiating a new Series<double> object and passing it in as
input to calculate
// a simple moving average
Series<double> myDataSeries = new Series<double>(this);
double value = SMA(myDataSeries, 20)[0];
```

Bars Object

A Bars object (which holds a series that contains OHLC data) can be used as input for indicators.

```
// Passing in the second Bars object held in a multi-instrument and
timeframe strategy
// The default value used for the SMA calculation is the close
price
double value = SMA(BarsArray[1], 20)[0];
Print("The current SMA value is " + value.ToString());;
```

Tip: The input series of an indicator cannot be the hosting indicator itself, as this will cause recursive loops.

```
// Using the hosting indicator in this way will cause errors
with recursive loops
double value = SMA(this, 20)[0];
```

11.6.2.18.2 Accumulation/Distribution (ADL)

Description

There are many indicators available to measure volume and the flow of money for a particular stock, index or security. One of the most popular volume indicators over the years has been the Accumulation/Distribution Line. The basic premise behind volume indicators, including the Accumulation/Distribution Line, is that volume precedes price. Volume reflects the amount of shares traded in a particular stock, and is a direct reflection of the money flowing into and out of a stock. Many times before a stock advances, there will be period of increased volume just prior to the move. Most volume or money flow indicators are designed to identify early increases in positive or negative volume flow to gain an edge before the price moves. (Note: the terms "money flow" and "volume flow" are essentially interchangeable.)

Syntax

```
ADL()
ADL(ISeries<double> input)

Returns default value
ADL()[int barsAgo]
ADL(ISeries<double> input)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Example

```
// Evaluates if ADL is rising
bool isRising = IsRising(ADL());
Print("Is ADL rising? " + isRising);
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.3 Adaptive Price Zone (APZ)

Description

The Adaptive Price Zone indicator from the S&C, September 2006 article "Trading With An Adpative Price Zone" by Lee Leibfarth is a set of bands based on a short term double smooth exponential moving average. The bands form a channel that surrounds the average price and tracks price fluctuations quickly, especially in volatile markets. As price crosses above the zone it can signal an opportunity to sell in anticipation of a reversal. As price crosses below the zone it can signal an opportunity to buy in anticipation of a reversal.

Syntax

```
APZ(double bandPct, int period)
APZ(ISeries<double> input, double bandPct, int period)

Returns upper band value
APZ(double bandPct, int period).Upper[int barsAgo]
APZ(ISeries<double> input, double bandPct, int period).Upper[int barsAgo]

Returns lower band value
APZ(double bandPct, int period).Lower[int barsAgo]
APZ(ISeries<double> input, double bandPct, int period).Lower[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

bandPct	The number of standard deviations
input	Indicator source data (?)

period	Number of bars used in the calculation

Example

```
// Prints the current upper band value of a 20 period APZ
double upperValue = APZ(2, 20).Upper[0];
Print("The current APZ upper value is " + upperValue.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.4 Aroon

Description

Developed by Tushar Chande in 1995, Aroon is an indicator system that can be used to determine whether a stock is trending or not and how strong the trend is. "Aroon" means "Dawn's Early Light" in Sanskrit and Chande chose that name for this indicator since it is designed to reveal the beginning of a new trend.

The Aroon indicator system consists of two lines, 'Aroon(up)' and 'Aroon(down)'. It takes a single parameter which is the number of time periods to use in the calculation. Aroon(up) is the amount of time (on a percentage basis) that has elapsed between the start of the time period and the point at which the highest price during that time period occurred. If the stock closes at a new high for the given period, Aroon(up) will be +100. For each subsequent period that passes without another new high, Aroon(up) moves down by an amount equal to (1 / # of periods) x 100.

Syntax

```
Aroon(int period)
Aroon(ISeries<double> input, int period)

Returns up value
Aroon(int period).Up[int barsAgo]
Aroon(ISeries<double> input, int period).Up[int barsAgo]

Returns down value
Aroon(int period).Down[int barsAgo]
Aroon(ISeries<double> input, int period).Down[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current up/down values of a 20 period Aroon indicator
double upValue = Aroon(20).Up[0];
double downValue = Aroon(20).Down[0];
Print("The current Aroon up value is " + upValue);
Print("The current Aroon down value is " + downValue);
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.5 Aroon Oscillator

Description

A trend-following indicator that uses aspects of the Aroon indicator ("Aroon up" and "Aroon down") to gauge the strength of a current trend and the likelihood that it will continue. The Aroon oscillator is calculated by subtracting Aroon down from Aroon up. Readings above zero indicate that an uptrend is present, while readings below zero indicate that a downtrend is present.

... Courtesy of Investopedia

Syntax

```
AroonOscillator(int period)
AroonOscillator(ISeries<double> input, int period)

Returns default value
AroonOscillator(int period)[int barsAgo]
AroonOscillator(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current values of a 20 period AroonOscillator using
default price type
double upValue = AroonOscillator(20)[0];
Print("The current AroonOscillator value is " +
upValue.ToString());

// Prints the current values of a 20 period AroonOscillator using
high price type
double upValue = AroonOscillator(High, 20)[0];
Print("The current AroonOscillator value is " +
upValue.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.6 Average Directional Index (ADX)

Description

An indicator used in technical analysis as an objective value for the strength of trend. ADX is non-directional so it will quantify a trend's strength regardless of whether it is up or down. ADX is usually plotted in a chart window along with two lines known as the DMI (Directional Movement Indicators). ADX is derived from the relationship of the DMI lines.

... Courtesy of Investopedia

Syntax

```
ADX(int period)
ADX(ISeries<double> input, int period)
```

```
Returns default value
ADX(int period)[int barsAgo]
ADX(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period ADX
double value = ADX(20)[0];
Print("The current ADX value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.7 Average Directional Movement Rating (ADXR)

Description

The ADXR is equal to the current ADX plus the ADX from n bars ago divided by two.

Syntax

```
ADXR(int interval, int period)
ADXR(ISeries<double> input, int interval, int period)

Returns default value
ADXR(int interval, int period)[int barsAgo]
ADXR(ISeries<double> input, int interval, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
interval	The interval between the first ADX value and the current ADX value
period	Number of bars used in the calculation

Example

```
// Prints the current value of a 20 period ADXR using default price type
double value = ADXR(10, 20)[0];
Print("The current ADXR value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.8 Average True Range (ATR)

Description

A measure of volatility introduced by Welles Wilder in his book: New Concepts in Technical Trading Systems.

The True Range indicator is the greatest of the following:
-current high less the current low.
-the absolute value of the current high less the previous close.
-the absolute value of the current low less the previous close.

The Average True Range is a moving average (generally 14-days) of the True Ranges.

... Courtesy of Investopedia

The original Wilder formula for an exponential moving average with a smoothing constant (k = 1/ Period) is used to calculate the ATR.

Syntax

```
ATR(int period)
ATR(ISeries<double> input, int period)

Returns default value
ATR(int period)[int barsAgo]
ATR(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Prints the current value of a 20 period ATR using default price
type
double value = ATR(20)[0];
Print("The current ATR value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.9 Balance of Power (BOP)

Description

The balance of power (BOP) indicator measures the strength of the bulls vs. bears by assessing the ability of each to push price to an extreme level.

Syntax

```
BOP(int smooth)
BOP(ISeries<double> input, int smooth)
```

```
Returns default value
BOP(int smooth)[int barsAgo]
BOP(ISeries<double> input, int smooth)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
smooth	The smoothing period

Example

```
// Prints the current value of BOP using default price type and 3
period smoothing
double value = BOP(3)[0];
Print("The current BOP value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.10 Block Volume

Description

Block volume detects block trades and display how many occurred per bar. This can be displayed either as trades or volume. Historical tick data is required to plot historically.

Syntax

```
BlockVolume(int blockSize, CountType countType)
BlockVolume(ISeries<double> input, int blockSize, CountType countType)
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
blockSize	The minimum volume a trade must be to be considered a block trade
countType	The format to count the block trades. By number of block trades that occurred or total block trade volume

Examples

```
// A 1 tick data series must be added to OnStateChange() as this
indicator runs off of tick data
else if (State == State.Configure)
{
   AddDataSeries(Data.BarsPeriodType.Tick, 1);
}

// Prints the current value of an 80 block trade size counted in
volume for the Block Volume
if (BarsInProgress == 0)
{
double value = BlockVolume(80, CountType.Volume)[0];
Print("The current Block Volume value is " + value.ToString());
}
```

11.6.2.18.11 Bollinger Bands

Description

Developed by John Bollinger, Bollinger Bands are an indicator that allows users to compare volatility and relative price levels over a period time. The indicator consists of three bands designed to encompass the majority of a security's price action.

1. A simple moving average in the middle
2. An upper band (SMA plus 2 standard deviations)
3. A lower band (SMA minus 2 standard deviations)

Standard deviation is a statistical unit of measure that provides a good assessment of a price plot's volatility. Using the standard deviation ensures that the bands will react quickly to price movements and reflect periods of high and low volatility. Sharp price increases (or decreases), and hence volatility, will lead to a widening of the bands.

... Courtesy of StockCharts

Syntax

```
Bollinger(double numStdDev, int period)
Bollinger(ISeries<double> input, double numStdDev, int period)

Returns upper band value
Bollinger(double numStdDev, int period).Upper[int barsAgo]
Bollinger(ISeries<double> input, double numStdDev, int period).Upper[int barsAgo]

Returns lower band value
Bollinger(double numStdDev, int period).Lower[int barsAgo]
Bollinger(ISeries<double> input, double numStdDev, int period).Lower[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current upper band value of a 20 period Bollinger
using default price type
double upperValue = Bollinger(2, 20).Upper[0];
Print("The current Bollinger upper value is " +
upperValue.ToString());

// Prints the current upper band value of a 20 period Bollinger
using low price type
double upperValue = Bollinger(Low, 2, 20).Upper[0];
Print("The current Bollinger upper value is " +
upperValue.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.12 BuySell Pressure

Description

The BuySellPressure indicator displays both the current bar's buying and selling pressure as percentage values based on the categorization of trades as buy or sell trades. Trades are categorized in real-time as a buy (at the ask or above) or as a sell (at the bid or below).... Trades in between the market are ignored.

> **Note**: For historical calculations, Tick Replay must be enabled

Syntax

```
BuySellPressure()
BuySellPressure(ISeries<double> input)

Returns buy pressure value
BuySellPressure().BuyPressure[int barsAgo]
BuySellPressure(ISeries<double> input).BuyPressure[int barsAgo]

Returns sell pressure value
BuySellPressure().SellPressure[int barsAgo]
BuySellPressure(ISeries<double> input).SellPressure[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Indicators will inherit the Calculate mode from the
hosting script.
        // Since BuySellPressure requires the use of
Calculate.OnEachTick, we must ensure the hosting script has this
Calculate mode set
        Calculate = Calculate.OnEachTick;
    }
}

protected override void OnBarUpdate()
{
    // This checks that 70% or more of the volume hit the ask or
higher
    if (State == State.Historical ||
BuySellPressure().BuyPressure[0] > 70)
    {
        EnterLong();
    }
}
```

Tip: Since this indicator operates in a real-time environment, remember to check for State.Realtime, or enable Tick Replay on the associated Data Series. In the above example we check that 50% or more of the volume hit the ask or higher. Our statement checks if the data is being calculated on historical data first, if true, we enter long, if not true (live), the the statement then checks for the Buy Volume condition.

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.13 BuySell Volume

Description

The BuySellVolume indicator displays a real-time horizontal histogram of volume categorized as buy or sell trades. Trades are categorized in real-time as a buy (at the ask or above) or as a sell (at the bid or below) and then color coded .... Trades in between the market are ignored.

Note: For historical calculations, Tick Replay must be enabled

Syntax

```
BuySellVolume()
BuySellVolume(ISeries<double> input)

Returns buy volume
BuySellVolume().Buys[int barsAgo]
BuySellVolume(ISeries<double> input).Buys[int barsAgo]

Returns sell volume
BuySellVolume().Sells[int barsAgo]
BuySellVolume(ISeries<double> input).Sells[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Example

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Indicators will inherit the Calculate mode from the
hosting script.
        // Since BuySellVolume requires the use of
Calculate.OnEachTick, we must ensure the hosting script has this
Calculate mode set
        Calculate = Calculate.OnEachTick;
    }
}

protected override void OnBarUpdate()
{
    // This checks that 5,000 or more of the volume hit the bid or
lower
    if (State == State.Historical || BuySellVolume().Sells[0] >
5000)
    {
        EnterLong();
    }
}
```

Tip: Since this indicator operates in a real-time environment, remember to check for State.Realtime, or enable Tick Replay on the associated Data Series. In the above example we check that 5,000 or more of the volume hit the bid or lower. Our statement checks if the data is being calculated on historical data first, if true, we enter long, if not true (live), the the statement then checks for the Buy Volume condition.

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.14 Camarilla Pivots

Description

Camarilla pivots are a price analysis tool that generates potential support and resistance levels by multiplying the prior range then adding or subtracting it from the close.

Syntax

```
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width)
Pivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
```

priorDayHLC, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*)

Returns R1 value
Pivots(PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).R1[int *barsAgo*]
Pivots(ISeries<double> *input*, PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).R1[int *barsAgo*]

Returns R2 value
Pivots(PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).R2[int *barsAgo*]
Pivots(ISeries<double> *input*, PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).R2[int *barsAgo*]

Returns R3 value
Pivots(PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).R3[int *barsAgo*]
Pivots(ISeries<double> *input*, PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).R3[int *barsAgo*]

Returns R4 value
Pivots(PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).R3[int *barsAgo*]
Pivots(ISeries<double> *input*, PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).R4[int *barsAgo*]

Returns S1 value
Pivots(PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).S1[int *barsAgo*]
Pivots(ISeries<double> *input*, PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).S1[int *barsAgo*]

Returns S2 value
Pivots(PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).S2[int *barsAgo*]
Pivots(ISeries<double> *input*, PivotRange *pivotRangeType*, HLCCalculationMode *priorDayHLC*, double *userDefinedClose*, double *userDefinedHigh*, double *userDefinedLow*, int *width*).S2[int *barsAgo*]

```
Returns S3 value
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).S3[int
barsAgo]
Pivots(ISeries<double>input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).S3[int barsAgo]

Returns S4 value
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).S3[int
barsAgo]
Pivots(ISeries<double>input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).S4[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
pivotRangeType	Sets the range for the type of pivot calculated. Possible values are: PivotRange.Daily PivotRange.Weekly PivotRange.Monthly
priorDayHLC	Sets how the prior range High, Low, Close values are calculated. Possible values are: HLCCalculationMode.CalcFromIntradayData HLCCalculationMode.DailyBars HLCCalculationMode.UserDefinedValues
userDefinedClose	Sets the close for Pivots calculations when using HLCCalculationMode.UserDefinedValues.
userDefinedHigh	Sets the high for Pivots calculations when using HLCCalculationMode.UserDefinedValues.

userDefinedLow	Sets the low for Pivots calculations when using HLCCalculationMode.UserDefinedValues.
width	Sets how long the Pivots lines will be drawn

Examples

```
// Prints the current R1 pivotvalue
double valueR1 = CamarillaPivots(PivotRange.Daily,
HLCCalculationMode.CalcFromIntradayData, 0, 0, 0, 20).R1[0];
Print("The current Camarilla Pivots' R1 value is " +
valueR1.ToString());

// Prints the current S2 pivot value
double valueS2 = CamarillaPivots(PivotRange.Daily,
HLCCalculationMode.CalcFromIntradayData, 0, 0, 0, 20).S2[0];
Print("The current Camarilla Pivots' S2 pivot value is " +
valueS2.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

Tip: When using HLCCalculationMode.DailyBars it can be expected that a value of 0 is returned when the daily bars have not been loaded yet. Due to the asynchronous nature of this indicator calling daily bars you should only access the pivot values when the indicator has loaded all required Bars objects. To ensure you are accessing accurate values you can use .IsValidDataPoint() as a check:

```
// Evaluates that this is a valid pivot point value
if (CamarillaPivots(PivotRange.Daily,
HLCCalculationMode.DailyBars, 0, 0, 0,
20).Pp.IsValidDataPoint(0))
{
     // Prints the current pivot point value
     double valuePp = CamarillaPivots(PivotRange.Daily,
HLCCalculationMode.DailyBars, 0, 0, 0, 20).Pp[0];
     Print("The current Camarilla Pivots' pivot value is " +
valuePp.ToString());
}
```

11.6.2.18.15 CandleStickPattern

Description

Detects the specified candle stick pattern.

Syntax

```
CandleStickPattern(ChartPattern pattern, int trendStrength)
CandleStickPattern(ISeries<double> input, ChartPattern pattern, int trendStrength)

Returns a value indicating if the specified pattern was detected
CandleStickPattern(ChartPattern pattern, int trendStrength)[int barsAgo]
CandleStickPattern(ISeries<double> input, ChartPattern pattern, int trendStrength)[int
 barsAgo]
```

Return Value

A `double` value representing pattern found. Returns a value of 1 if the pattern is found; returns a value of 0 if no pattern was found.
Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
pattern	Possible values are: ChartPattern.BearishBeltHold ChartPattern.BearishEngulfing ChartPattern.BearishHarami

	ChartPattern.BearishHaramiCross ChartPattern.BullishBeltHold ChartPattern.BullishEngulfing ChartPattern.BullishHarami ChartPattern.BullishHaramiCross ChartPattern.DarkCloudCover ChartPattern.Doji ChartPattern.DownsideTasukiGap ChartPattern.EveningStar ChartPattern.FallingThreeMethods ChartPattern.Hammer ChartPattern.HangingMan ChartPattern.InvertedHammer ChartPattern.MorningStart ChartPattern.PiercingLine ChartPattern.RisingThreeMethods ChartPattern.ShootingStar ChartPattern.StickSandwich ChartPattern.ThreeBlackCrows ChartPattern.ThreeWhiteSoldiers ChartPattern.UpsideGapTwoCrows ChartPattern.UpsideTasukiGap
trendStrength	The number of required bars to the left and right of the swing point used to determine trend. A value of zero will exclude the requirement of a trend and only detect based on the candles themselves.

Example

```
// Go long if the current bar is a bullish engulfing pattern
if (CandlestickPattern(ChartPattern.BullishEngulfing, 4)[0] == 1)
    EnterLong();
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.16 Chaikin Money Flow

Description

The formula for Chaikin Money Flow is the cumulative total of the Accumulation/Distribution Values for 21 periods divided by the cumulative total of volume for 21 periods.

... Courtesy of StockCharts

Syntax

```
ChaikinMoneyFlow(int period)
ChaikinMoneyFlow(ISeries<double> input, int period)

Returns default value
ChaikinMoneyFlow(int period)[int barsAgo]
ChaikinMoneyFlow(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Prints the current value of a 20 period ChaikinMoneyFlow using
default price type
double value = ChaikinMoneyFlow(20)[0];
Print("The current ChaikinMoneyFlow value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.17 Chaikin Oscillator

Description

The Chaikin Oscillator is simply the Moving Average Convergence Divergence indicator (MACD) applied to the Accumulation/Distribution Line. The formula is the difference between the 3-day exponential moving average and the 10-day exponential moving average of the Accumulation/Distribution Line. Just as the MACD-Histogram is an indicator to predict moving average crossovers in MACD, the Chaikin Oscillator is an indicator to predict changes in the Accumulation/Distribution Line.

... Courtesy of StockCharts

Syntax

```
ChaikinOscillator(int fast, int slow)
ChaikinOscillator(ISeries<double> input, int fast, int slow)

Returns default value
ChaikinOscillator(int fast, int slow)[int barsAgo]
ChaikinOscillator(ISeries<double> input, int fast, int slow)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

fast	The number of bars to calculate the fast EMA
input	Indicator source data (?)
slow	The number of bars to calculate the slow EMA

Example

```
// Prints the current value of a ChaikinOscillator using default
price type
double value = ChaikinOscillator(3, 10)[0];
Print("The current ChaikinOscillator value is " +
value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.18 Chaikin Volatility

Description

The Chaikin Volatility Indicator is the difference between two moving averages of a volume weighted accumulation-distribution line. By comparing the spread between a security's high and low prices, it quantifies volatility as a widening of the range between the high and the low price.

Syntax

```
ChaikinVolatility(int mAPeriod, int rOCPeriod)
ChaikinVolatility(ISeries<double> input, int mAPeriod, int rOCPeriod)

Returns default value
ChaikinVolatility(int mAPeriod, int rOCPeriod)[int barsAgo]
ChaikinVolatility(ISeries<double> input, int mAPeriod, int rOCPeriod)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
mAPeriod	Number of bars used in the moving average calculation
rOCPeriod	Number of bars used in the rate of change calculation

Example

```
// Prints the current value of the 20 period Chaikin Volatility
double value = ChaikinVolatility(20, 20)[0];
Print("The current Chaikin Volatility value is " +
value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.19 Chande Momentum Oscillator (CMO)

Description

The Chande Momentum Oscillator was developed by Tushar S. Chande and is described in the 1994 book The New Technical Trader by Tushar S. Chande and Stanley Kroll. This indicator is a modified RSI. Where the RSI divides the upward movement by the net movement (up / (up + down)), the CMO divides the total movement by the net movement ((up - down) / (up + down)). Values under -50 indicate oversold conditions while values over 50 indicate overbought conditions.

Syntax

```
CMO(int period)
CMO(ISeries<double> input, int period)

Returns default value
CMO(int period)[int barsAgo]
CMO(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	The number of bars to include in the calculation

Examples

```
// Prints the current value of a 20 period CMO using default price
type
double value = CMO(20)[0];
Print("The current CMO value is " + value.ToString());

// Prints the current value of a 20 period CMO using high price
type
double value = CMO(High, 20)[0];
Print("The current CMO value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.20 Choppiness Index

Description

The Choppiness Index is designed to determine if the market is choppy (trading sideways) or not choppy (trading within a trend in either direction)

Syntax

```
ChoppinessIndex(int period)
ChoppinessIndex(ISeries<double> input, int period)
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 14 period Choppiness Index
double value = ChoppinessIndex(14)[0];
Print("The current Choppiness Index value is " + value.ToString());
```

11.6.2.18.21 Commitment Of Traders (COT)

Description

The COT indicator plots weekly data from the Commitment Of Traders report, indicating holdings of different participants in the U.S. futures market.

Notes:

1. Since the underlying COT reports are a weekly figure updated every Friday, it would not be meaningful to run this study outside Calculate.OnBarClose
2. Default values of the 5 hard-coded plots are : 1 - Futures Non Commercial Net, 2 - Futures Commercial Net, 3 - Futures Non Reportable Positions Net, 4 - Futures Open Interest, 5 - Futures Total Net
3. To access other reports and report fields, please see the 2nd example below. All fields available could be seen via Intelliprompt in the NinjaScript editor.
4. In the **CotReportField** enum, "Pmpu" represents : "Producer/merchant/processor/user" where **CotReportField.PmpuNet** would represent : "Producer/merchant/processor/user Net"
5. If a **CotReportField** enum is used that is not supported by the **ReportType**, OpenInterest will be seen.

Syntax

```
COT(int number)

Returns Cot1 value
COT(int number).Cot1[int barsAgo]

Returns Cot2 value
COT(int number).Cot2[int barsAgo]

Returns Cot3 value
COT(int number).Cot3[int barsAgo]

Returns Cot4 value
COT(int number).Cot4[int barsAgo]

Returns Cot5 value
COT(int number).Cot5[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

number	Sets the number of plots enabled

Examples

```
// Prints the current value of COT 4th plot (default Futures Open
Interest), the COT(4) would allow us to access the Cot1, Cot2, Cot3
and Cot4 plots, but not Cot5 (since not enabled)
double value = COT(4).Cot4[0];
Print("The current COT Futures Open Interest value is " +
value.ToString());
```

```
// Advanced example where two plots in total are enabled (COT(2)).
Next, the ReportType and Field are custom set per each plot.
else if (State == State.DataLoaded)
{
   cot1 = COT(2);
   cot1.CotReport1.ReportType = CotReportType.Combined;
   cot1.CotReport2.ReportType = CotReportType.Combined;
   cot1.CotReport1.Field = CotReportField.OpenInterest;
   cot1.CotReport2.Field = CotReportField.TotalNet;
}
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.22 Commodity Channel Index (CCI)

Description

Developed by Donald Lambert, the Commodity Channel Index (CCI) was designed to identify cyclical turns in commodities. The assumption behind the indicator is that commodities (or stocks or bonds) move in cycles, with highs and lows coming at periodic intervals.

... Courtesy of StockCharts

Syntax

```
CCI(int period)
CCI(ISeries<double> input, int period)

Returns default value
CCI(int period)[int barsAgo]
CCI(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period CCI using default price
type
double value = CCI(20)[0];
Print("The current CCI value is " + value.ToString());

// Prints the current value of a 20 period CCI using high price
type
double value = CCI(High, 20)[0];
Print("The current CCI value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.23 Correlation

Description

The correlation indicator will plot the correlation of the data series to a desired instrument. Values close to 1 indicate movement in the same direction. Values close to -1 indicate movement in opposite directions. Values near 0 indicate no correlation.

Syntax

```
Correlation(int period, string correlationSeries)
string correlationSeies(ISeries<double> input, int period, string correlationSeies)
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation
correlationSeries	The data series to compare to

Examples

```
// The correlation data series must be added to OnStateChange() as
this indicator runs off the correlation data series data
else if (State == State.Configure)
{
   AddDataSeries("SPY");
}

// Checks the bars in progress and prints the current correlation
to the SPY
if (BarsInProgress == 0)
{
   double value = Correlation(20, "SPY")[0];
   Print("The current correlation to the SPY is " +
value.ToString());
}
```

Note: If the correlation series does not plot during a time the input series plots, a value of zero would plot in the above example. You may consider ignroing zero values.

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.24 Current Day OHL

Description

The current day (session) open, high and low values.

> **Note**: Only use this indicator on intraday series.

Syntax

```
CurrentDayOHL()
CurrentDayOHL(ISeries<double> input)

Returns current session open value
CurrentDayOHL().CurrentOpen[int barsAgo]
CurrentDayOHL(ISeries<double> input).CurrentOpen[int barsAgo]

Returns current session high value
CurrentDayOHL().CurrentHigh[int barsAgo]
CurrentDayOHL(ISeries<double> input).CurrentHigh[int barsAgo]

Returns current session low value
CurrentDayOHL().CurrentLow[int barsAgo]
CurrentDayOHL(ISeries<double> input).CurrentLow[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Example

```
// Prints the current value of the session low
double value = CurrentDayOHL().CurrentLow[0];
Print("The current session low value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.25 Darvas

Description

A trading strategy that was developed in 1956 by former ballroom dancer Nicolas Darvas. Darvas' trading technique involved buying into stocks that were trading at new 52-week highs with correspondingly high volumes.

... Courtesy of Investopedia

Syntax

```
Darvas()
Darvas(ISeries<double> input)

Returns the upper value
Darvas().Upper[int barsAgo]
Darvas(ISeries<double> input).Upper[int barsAgo]

Returns the lower value
Darvas().Lower[int barsAgo]
Darvas(ISeries<double> input).Lower[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Example

```
// Prints the current upper Darvas value
double value = Darvas().Upper[0];
Print("The current upper Darvas value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.26 Directional Movement (DM)

Description

Same as the ADX indicator with the addition of the +DI and -DI values.

... Courtesy of Investopedia

Syntax

```
DM(int period)
DM(ISeries<double> input, int period)

Returns default ADX value
DM(int period)[int barsAgo]
DM(ISeries<double> input, int period)[int barsAgo]

Returns +DI value
DM(int period).DiPlus[int barsAgo]
DM(ISeries<double> input, int period).DiPlus[int barsAgo]

Returns -DI value
DM(int period).DiMinus[int barsAgo]
DM(ISeries<double> input, int period).DiMinus[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Prints the current value of a 20 period +DI using default price
type
double value = DM(20).DiPlus[0];
Print("The current +DI value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.27 Directional Movement Index (DMI)

Description

An indicator developed by J. Welles Wilder for identifying when a definable trend is present in an instrument. That is, the DMI tells whether an instrument is trending or not.

...Courtesy of FMLabs

Syntax

```
DMI(int period)
DMI(ISeries<double> input, int period)

Returns default value
DMI(int period)[int barsAgo]
DMI(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Prints the current value of a 20 period DMI using default price
type
double value = DMI(20)[0];
Print("The current DMI value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.28 Disparity Index

Description

The Disparity Index that measures the difference between the price and an exponential moving average. A value greater could suggest bullish momentum, while a value less than zero could suggest bearish momentum.

Syntax

```
DisparityIndex(int period)
DisparityIndex(ISeries<double> input, int period)
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 15 period Disparity Index
double value = DisparityIndex(15)[0];
Print("The current Disparity Index value is " + value.ToString());
```

11.6.2.18.29 Donchian Channel

Description

A moving average indicator developed by Richard Donchian. It plots the highest high and lowest low over a specific period.

Syntax

```
DonchianChannel(int period)
DonchianChannel(ISeries<double> input, int period)

Returns mean value (middle band) at a specified bar index
DonchianChannel(int period)[int barsAgo]
DonchianChannel(ISeries<double> input, int period)[int barsAgo]

Returns upper band value at a specified bar index
DonchianChannel(int period).Upper[int barsAgo]
DonchianChannel(ISeries<double> input, int period).Upper[int barsAgo]
```

```
Returns lower band value at a specified bar index
DonchianChannel(int period).Lower[int barsAgo]
DonchianChannel(ISeries<double> input, int period).Lower[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current upper value of a 20 period DonchianChannel
using default price type
double value = DonchianChannel(20).Upper[0];
Print("The current DonchianChannel upper value is " +
value.ToString());

// Note the above call with a barsAgo of 0 includes the current
Upper channel in the value. If we want to check for example for a
break of this value, storing the last bar's channel value would be
needed.
double value = DonchianChannel(20).Upper[1];

if (High[0] > value)
    Draw.ArrowUp(this, CurrentBar.ToString(), true, 0, Low[0] -
TickSize, Brushes.Blue);
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.30 Double Stochastics

Description

Double Stochastics is a variation of the Stochastics indicator developed by William Blau.

Syntax

```
DoubleStochastics(int period)
DoubleStochastics(ISeries<double> input, int period)

Returns default value
DoubleStochastics(int period)[int barsAgo]
DoubleStochastics(ISeries<double> input, int period)[int barsAgo]

Returns %K value
DoubleStochastics(int period).K[int barsAgo]
DoubleStochastics(ISeries<double> input, int period).K[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value
double value = DoubleStochastics(10)[0];
Print("The current Double Stochastics value is " +
value.ToString());

// Prints the current %K value
double value = DoubleStochastics(10).K[0];
Print("The current Double Stochastics %K value is " +
value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.31 Dynamic Momentum Index (DMIndex)

Description

An indicator used in technical analysis that determines overbought and oversold conditions of a particular asset. This indicator is very similar to the relative strength index (RSI). The main difference between the two is that the RSI uses a fixed number of time periods (usually 14), while the dynamic momentum index uses different time periods as volatility changes.

... Courtesy of Investopedia

Syntax

```
DMIndex(int smooth)
DMIndex(ISeries<double> input, int smooth)

Returns default value
DMIndex(int period)[int barsAgo]
DMIndex(ISeries<double> input, int smooth)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
smooth	The number of bars to include in the calculation

Example

```
// Prints the current value of DMIndex using default price type
double value = DMIndex(3)[0];
Print("The current DMIndex value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.32 Ease of Movement

Description

The Ease of Movement indicator was designed to illustrate the relationship between volume and price change. It shows how much volume is required to move prices.

High Ease of Movement values occur when prices are moving upward with light volume. Low values occur when prices are moving downward on light volume. If prices are not moving or if heavy volume is required to move prices then the indicator will read near zero. A buy signal is produced when it crosses above zero. A sell signal is produced when the indicator crosses below zero (prices are moving downward more easily).

Syntax

```
EaseOfMovement(int smoothing, int volumeDivisor)
EaseOfMovement(ISeries<double> input, int smoothing, int volumeDivisor)

Returns default value
EaseOfMovement(int smoothing, int volumeDivisor)[int barsAgo]
EaseOfMovement(ISeries<double> input, int smoothing, int volumeDivisor)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
smoothing	The number of bars used to smooth the signal
volumeDivisor	The value used to calculate the box ratio

Example

```
// Prints the current value of Ease of Movement using default price type
double value = EaseOfMovement(14, 10000)[0];
Print("The current Ease of Movement value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.33 Fibonacci Pivots

Description

Fibonacci pivots are a price analysis tool that generates potential support and resistance levels by multiplying the prior range against Fibonacci values then adding or subtracting it from the average of the prior high, low, and close.

Syntax

```
FibonacciPivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width)
FibonacciPivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width)

Returns pivot point value
FibonacciPivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).Pp[int barsAgo]
FibonacciPivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).PP[int barsAgo]

Returns R1 value
FibonacciPivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).R1[int barsAgo]
FibonacciPivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).R1[int barsAgo]

Returns R2 value
FibonacciPivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).R2[int barsAgo]
FibonacciPivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).R2[int barsAgo]

Returns R3 value
FibonacciPivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).R3[int barsAgo]
FibonacciPivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).R3[int barsAgo]
```

```
Returns S1 value
FibonacciPivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).S1[int
barsAgo]
FibonacciPivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).S1[int barsAgo]

Returns S2 value
FibonacciPivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).S2[int
barsAgo]
FibonacciPivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).S2[int barsAgo]

Returns S3 value
FibonacciPivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).S3[int
barsAgo]
FibonacciPivots(ISeries<double>input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).S3[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
pivotRangeType	Sets the range for the type of pivot calculated. Possible values are: PivotRange.Daily PivotRange.Weekly PivotRange.Monthly
priorDayHLC	Sets how the prior range High, Low, Close values are calculated. Possible values are: HLCCalculationMode.CalcFromIntradayData HLCCalculationMode.DailyBars HLCCalculationMode.UserDefinedValues

userDefinedClose	Sets the close for Pivots calculations when using HLCCalculationMode.UserDefinedValues.
userDefinedHigh	Sets the high for Pivots calculations when using HLCCalculationMode.UserDefinedValues.
userDefinedLow	Sets the low for Pivots calculations when using HLCCalculationMode.UserDefinedValues.
width	Sets how long the Pivots lines will be drawn

Examples

```
// Prints the current pivot point value
double valuePp =  FibonacciPivots(PivotRange.Daily,
HLCCalculationMode.CalcFromIntradayData, 0, 0, 0, 20).Pp[0];
Print("The current Fibonacci Pivots' pivot value is " +
valuePp.ToString());

// Prints the current S2 pivot value
double valueS2 = FibonacciPivots(PivotRange.Daily,
HLCCalculationMode.CalcFromIntradayData, 0, 0, 0, 20).S2[0];
Print("The current Fibonacci Pivots' S2 pivot value is " +
valueS2.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

Tip: When using HLCCalculationMode.DailyBars it can be expected that a value of 0 is returned when the daily bars have not been loaded yet. Due to the asynchronous nature of this indicator calling daily bars you should only access the pivot values when the indicator has loaded all required Bars objects. To ensure you are accessing accurate values you can use .IsValidDataPoint() as a check:

```
// Evaluates that this is a valid pivot point value
if (FibonacciPivots(PivotRange.Daily,
HLCCalculationMode.DailyBars, 0, 0, 0,
20).Pp.IsValidDataPoint(0))
{
     // Prints the current pivot point value
     double valuePp = FibonacciPivots(PivotRange.Daily,
HLCCalculationMode.DailyBars, 0, 0, 0, 20).Pp[0];
     Print("The current Pivots' pivot value is " +
valuePp.ToString());
}
```

11.6.2.18.34 Fisher Transform

Description

With distinct turning points and a rapid response time, the Fisher Transform uses the assumption that while prices do not have a normal or Gaussian probability density function (that familiar bell-shaped curve), you can create a nearly Gaussian probability density function by normalizing price (or an indicator such as RSI) and applying the Fisher Transform. Use the resulting peak swings to clearly identify price reversals.

Syntax

```
FisherTransform(int period)
FisherTransform(ISeries<double> input, int period)

Returns default value
FisherTransform(int period)[int barsAgo]
FisherTransform(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Prints the current value of a 10 period using default (median)
price type
double value = FisherTransform(10)[0];
Print("The current Fisher Transform value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.35 Forecast Oscillator (FOSC)

Description

The Forecast Oscillator calculates the percentage difference between the actual price and the Time Series Forecast (the endpoint of a linear regression line). When the price and the forecast are equal, the Oscillator is zero. When the price is greater than the forecast, the Oscillator is greater than zero. When the price is less than the forecast, the Oscillator is less than zero.

... Courtesy of FM Labs

Syntax

```
FOSC(int period)
FOSC(ISeries<double> input, int period)

Returns default value
FOSC(int period)[int barsAgo]
FOSC(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Evaluates if the current bar FOCS is above zero
if (FOSC(14)[0] > 0)
    Print("FOSC is above zero indicating prices may rise");
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.36 Keltner Channel

Description

Keltner Channel indicator is based on volatility using a pair of values placed as an "envelope" around the data field.

Syntax

```
KeltnerChannel(double offsetMultiplier, int period)
KeltnerChannel(ISeries<double> input, double offsetMultiplier, int period)

Returns midline value
KeltnerChannel(double offsetMultiplier, int period)[int barsAgo]
KeltnerChannel(ISeries<double> input, double offsetMultiplier, int period)[int
barsAgo]

Returns upper band value
KeltnerChannel(double offsetMultiplier, int period).Upper[int barsAgo]
KeltnerChannel(ISeries<double> input, double offsetMultiplier, int period).Upper[int
barsAgo]

Returns lower band value
KeltnerChannel(double offsetMultiplier, int period).Lower[int barsAgo]
KeltnerChannel(ISeries<double> input, double offsetMultiplier, int period).Lower[int
barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

period	Number of bars used in the calculation

Examples

```
// Prints the current upper value of a 20 period KeltnerChannel
using default price type
double value = KeltnerChannel(1.5, 20).Upper[0];
Print("The current KeltnerChannel upper value is " +
value.ToString());

// Prints the current lower value of a 20 period KeltnerChannel
using high price type
double value = KeltnerChannel(High, 1.5, 20).Lower[0];
Print("The current KeltnerChannel lower value is " +
value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.37 KeyReversalDown

Description

Returns a value of 1 when the current close is less than the prior close and the current high has penetrated the highest high of the last n bars.

Syntax

```
KeyReversalDown(int period)
KeyReversalDown(ISeries<double> input, int period)

Returns default value
KeyReversalDown(int period)[int barsAgo]
KeyReversalDown(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// If we get a reversal over the past 10 bars go short
if (KeyReversalDown(10)[0] == 1)
    EnterShort();
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.38 KeyReversalUp

Description

Returns a value of 1 when the current close is greater than the prior close and the current low has penetrated the lowest low of the last n bars.

Syntax

```
KeyReversalUp(int period)
KeyReversalUp(ISeries<double> input, int period)

Returns default value
KeyReversalUp(int period)[int barsAgo]
KeyReversalUp(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// If we get a reversal over the past 10 bars go long
if (KeyReversalUp(10)[0] == 1)
    EnterLong();
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.39 Linear Regression

Description

The Linear Regression Indicator plots the trend of a security's price over time. That trend is determined by calculating a Linear Regression Trendline using the least squares method. This ensures the minimum distance between the data points and a Linear Regression Trendline.

Syntax

```
LinReg(int period)
LinReg(ISeries<double> input, int period)

Returns default value
LinReg(int period)[int barsAgo]
LinReg(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period LinReg using default
price type
double value = LinReg(20)[0];
Print("The current LinReg value is " + value.ToString());

// Prints the current value of a 20 period LinReg using high price
type
double value = LinReg(High, 20)[0];
Print("The current LinReg value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.40 Linear Regression Intercept

Description

The Linear Regression Intercept provides the intercept value of the Linear Regression trendline.

Syntax

```
LinRegIntercept(int period)
LinRegIntercept(ISeries<double> input, int period)

Returns default value
LinRegIntercept(int period)[int barsAgo]
LinRegIntercept(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current intercept value of a 20 period LinReg using
default price type
double value = LinRegIntercept(20)[0];
Print("The current intercept value is " + value.ToString());

// Prints the current intercept value of a 20 period LinReg using
high price type
double value = LinRegIntercept(High, 20)[0];
Print("The current intercept value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.41 Linear Regression Slope

Description

The Linear Regression Slope provides the slope value of the Linear Regression trendline.

Syntax

```
LinRegSlope(int period)
LinRegSlope(ISeries<double> input, int period)

Returns default value
LinRegSlope(int period)[int barsAgo]
LinRegSlope(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current slope value of a 20 period LinReg using
default price type
double value = LinRegSlope(20)[0];
Print("The current slope value is " + value.ToString());

// Prints the current slope value of a 20 period LinReg using high
price type
double value = LinRegSlope(High, 20)[0];
Print("The current slope value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.42 MA Envelopes

Description

The Moving Average Envelope consists of moving averages calculated from the underling price, shifted up and down by a fixed percentage.

Syntax

```
MAEnvelopes(double envelopePercentage, int mAType, int period)
MAEnvelopes(ISeries<double> input, double envelopePercentage, int mAType, int period)

Returns upper band levels
MAEnvelopes(double envelopePercentage, int mAType, int period).Upper[int barsAgo]
MAEnvelopes(ISeries<double> input, double envelopePercentage, int mAType, int
period).Upper[int barsAgo]

Returns moving average value
MAEnvelopes(double envelopePercentage, int mAType, int period).Middle[int barsAgo]
MAEnvelopes(ISeries<double> input, double envelopePercentage, int mAType, int
period).Middle[int barsAgo]

Returns lower band levels
MAEnvelopes(double envelopePercentage, int mAType, int period).Lower[int barsAgo]
MAEnvelopes(ISeries<double> input, double envelopePercentage, int mAType, int
period).Lower[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

envelopePercentage	Percentage around MA that envelopes will be drawn
input	Indicator source data (?)
mAType	Moving average type: 1 = EMA 2 = HMA 3 = SMA 4 = TMA 5 = TEMA 6 = WMA
period	Number of bars used in the calculation

Examples

```
// Prints the current upper band value of a 20 period SMA envelope
using default price type
double upperValue = MAEnvelopes(0.2, 3, 20).Upper[0];
Print("The current SMA envelope upper value is " +
upperValue.ToString());

// Prints the current lower band value of a 20 period SMA envelope
using low price type
double lowerValue = MAEnvelopes(Low, 0.2, 3, 20).Lower[0];
Print("The current SMA envelope lower value is " +
lowerValue.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.43 Maximum (MAX)

Description

Returns the highest value over the specified period.

Syntax

```
MAX(int period)
MAX(ISeries<double> input, int period)

Returns default value
MAX(int period)[int barsAgo]
MAX(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Prints the highest high value over the last 20 periods
double value = MAX(High, 20)[0];
Print("The current MAX value is " + value.ToString());

// Note the above call with a barsAgo of 0 includes the current MAX
of the input high series in the value. If we want to check for
example for a break of this value, storing the last bar's MAX would
be needed.
double value = MAX(High, 20)[1];

if (High[0] > value)
    Draw.ArrowUp(this, CurrentBar.ToString(), true, 0, Low[0] -
TickSize, Brushes.Blue);
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.44 McClellan Oscillator

Description

McClellan Oscillator is the difference between two exponential moving averages of the NYSE advance decline spread. This indicator require ADV and DECL index data.

Syntax

```
McClellanOscillator(int fastPeriod, int slowPeriod)
McClellanOsillator(ISeries<double> input, int fastPeriod, int slowPeriod)
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
fastPeriod	Number of bars used in the fast moving average calculation
slowPeriod	Number of bars used in the slow moving average calculation

Examples

```
// An ADV and DECL data series must be added to OnStateChange()
else if (State == State.Configure)
{
	AddDataSeries("^ADV");
	AddDataSeries("^DECL");
}

// Prints the current value of the McClellan Oscillator with a 19
fast period moving average & 39 slow period
double value = McClellanOscillator(19, 39)[0];
Print("The current McClellan Oscillator value is " +
value.ToString());
```

11.6.2.18.45 Minimum (MIN)

Description

Returns the lowest value over the specified period.

Syntax

```
MIN(int period)
MIN(ISeries<double> input, int period)
```

```
Returns default value
MIN(int period)[int barsAgo]
MIN(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Prints the lowest low value over the last 20 periods
double value = MIN(Low, 20)[0];
Print("The current MIN value is " + value.ToString());

// Note the above call with a barsAgo of 0 includes the current MIN
of the input low series in the value. If we want to check for
example for a break of this value, storing the last bar's MIN would
be needed.
double value = MIN(Low, 20)[1];

if (Low[0] < value)
    Draw.ArrowDown(this, CurrentBar.ToString(), true, 0, High[0] +
TickSize, Brushes.Red);
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.46 Momentum

Description

By measuring the amount that a security's price has changed over a given time span, the Momentum indicator provides an indication of a market's velocity and to some degree, a measure of the extent to which a trend still holds true. It can also be helpful in spotting likely reversal points.

Syntax

```
Momentum(int period)
Momentum(ISeries<double> input, int period)

Returns default value
Momentum(int period)[int barsAgo]
Momentum(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period Momentum using default
price type
double value = Momentum(20)[0];
Print("The current Momentum value is " + value.ToString());

// Prints the current value of a 20 period Momentum using high
price type
double value = Momentum(High, 20)[0];
Print("The current Momentum value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.47 Money Flow Index (MFI)

Description

The Money Flow Index (MFI) is a momentum indicator that is similar to the Relative Strength Index (RSI) in both interpretation and calculation. However, MFI is a more rigid indicator in that it is volume-weighted, and is therefore a good measure of the strength of money flowing in and out of a security.

... Courtesy of StockCharts

Syntax

```
MFI(int period)
MFI(ISeries<double> input, int period)

Returns default value
MFI(int period)[int barsAgo]
MFI(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Prints the current value of a 20 period MFI using default price
type
double value = MFI(20)[0];
Print("The current MFI value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.48 Money Flow Oscillator

Description

The Money Flow Oscillator measures the amount of money flow volume over a specific period. A move into positive territory indicates buying pressure while a move into negative territory indicates selling pressure.

Syntax

```
MoneyFlowOscillator(int period)
MoneyFlowOscillator(ISeries<double> input, int period)
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 10 period Money Flow Oscillator
double value = MoneyFlowOscillator(10)[0];
Print("The current Money Flow Oscillator value is " +
value.ToString());
```

11.6.2.18.49 Moving Average - Double Exponential (DEMA)

Description

The Double Exponential Moving Average (DEMA) is a combination of a single exponential moving average and a double exponential moving average. The advantage is that gives a reduced amount of lag time than either of the two separate moving averages alone.

Syntax

```
DEMA(int period)
DEMA(ISeries<double> input, int period)

Returns default value
DEMA(int period)[int barsAgo]
DEMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period DEMA using default price type
double value = DEMA(20)[0];
Print("The current DEMA value is " + value.ToString());

// Prints the current value of a 20 period DEMA using high price type
double value = DEMA(High, 20)[0];
Print("The current DEMA value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.50 Moving Average - Exponential (EMA)

Description

The exponential moving average is but one type of a moving average. In a simple moving average, all price data has an equal weight in the computation of the average with the oldest value removed as each new value is added. In the exponential moving average equation the most recent market action is assigned greater importance as the average is calculated. The oldest pricing data in the exponential moving average is however never removed.

Syntax

```
EMA(int period)
EMA(ISeries<double> input, int period)

Returns default value
EMA(int period)[int barsAgo]
EMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period EMA using default price
type
double value = EMA(20)[0];
Print("The current EMA value is " + value.ToString());

// Prints the current value of a 20 period EMA using high price
type
double value = EMA(High, 20)[0];
Print("The current EMA value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.51 Moving Average - Hull (HMA)

Description

The HMA manages to keep up with rapid changes in price activity whilst having superior smoothing over an SMA of the same period. The HMA employs weighted moving averages and dampens the smoothing effect (and resulting lag) by using the square root of the period instead of the actual period itself. Developed by Alan Hull.

Syntax

```
HMA(int period)
HMA(ISeries<double> input, int period)

Returns default value
HMA(int period)[int barsAgo]
HMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period HMA using default price
type
double value = HMA(20)[0];
Print("The current HMA value is " + value.ToString());

// Prints the current value of a 20 period HMA using high price
type
double value = HMA(High, 20)[0];
Print("The current HMA value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.52 Moving Average - Kaufman's Adaptive (KAMA)

Description

Developed by Perry Kaufman, this indicator is an EMA using an Efficiency Ratio to modify the smoothing constant, which ranges from a minimum of Fast Length to a maximum of Slow Length.

Syntax

```
KAMA(int fast, int period, int slow)
KAMA(ISeries<double> input, int fast, int period, int slow)

Returns default value
KAMA(int fast, int period, int slow)[int barsAgo]
KAMA(ISeries<double> input, int fast, int period, int slow)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

fast	Fast length
input	Indicator source data (?)
period	Number of bars used in the calculation
slow	Slow length

Examples

```
// Prints the current value of a 20 period KAMA using default price
type
double value = KAMA(2, 20, 30)[0];
Print("The current KAMA value is " + value.ToString());

// Prints the current value of a 20 period KAMA using high price
type
double value = KAMA(High, 2, 20, 30)[0];
Print("The current KAMA value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.53 Moving Average - Mesa Adaptive (MAMA)

Description

The MESA Adaptive Moving Average (MAMA) adapts to price movement in an entirely new and unique way. The adaptation is based on the rate change of phase as measured by the Hilbert Transform Discriminator. The advantage of this method of adaptation is that it features a fast attack average and a slow decay average so that composite average rapidly ratchets behind price changes and holds the average value until the next ratchet occurs.

Syntax

```
MAMA(double fastLimit, double slowLimit)
MAMA(ISeries<double> input, double fastLimit, double slowLimit)

Returns MAMA value
MAMA(double fastLimit, double slowLimit)[int barsAgo]
MAMA(ISeries<double> input, double fastLimit, double slowLimit)[int barsAgo]
```

```
Returns Fama (Following Adaptive Moving Average) value
MAMA(double fastLimit, double slowLimit).Fama[int barsAgo]
MAMA(ISeries<double> input, double fastLimit, double slowLimit).Fama[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

fastLimit	Upper limit of the alpha value
input	Indicator source data (?)
slowLimit	Lower limit of the alpha value

Examples

```
// Prints the current value of a 20 period MAMA using default price
type
double value = MAMA(0.5, 0.05).Default[0];
Print("The current MAMA value is " + value.ToString());

// Prints the current value of a 20 period Fama using high price
type
double value = MAMA(High, 0.5, 0.05).Fama[0];
Print("The current Fama value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.54 Moving Average - Simple (SMA)

Description

The Simple Moving Average is calculated by summing the closing prices of the security for a period of time and then dividing this total by the number of time periods. Sometimes called an arithmetic moving average, the SMA is basically the average stock price over time.

Syntax

```
SMA(int period)
SMA(ISeries<double> input, int period)

Returns default value
SMA(int period)[int barsAgo]
SMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period SMA using default price type
double value = SMA(20)[0];
Print("The current SMA value is " + value.ToString());

// Prints the current value of a 20 period SMA using high price type
double value = SMA(High, 20)[0];
Print("The current SMA value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.55 Moving Average - T3 (T3)

Description

The T3 is a type of moving average, or smoothing function. It is based on the DEMA. The T3 takes the DEMA calculation and adds a vfactor which is between zero and 1. The resultant function is called the GD, or Generalized DEMA. A GD with vfactor of 1 is the same as the DEMA. A GD with a vfactor of zero is the same as an Exponential Moving Average. The T3 typically uses a vfactor of 0.7.

... Courtesy of FMLabs

Syntax

```
T3(int period, int tCount, double vFactor)
T3(ISeries<double> input, int period, int tCount, double vFactor)

Returns default value
T3(int period, int tCount, double vFactor)[int barsAgo]
T3(ISeries<double> input, int period, int tCount, double vFactor)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation
tCount	Number of smooth iterations
vFactor	A multiplier fudge factor

Examples

```
// Prints the current value of a 20 period T3 using default price
type
double value = T3(20, 3, 0.7)[0];
Print("The current T3 value is " + value.ToString());

// Prints the current value of a 20 period T3 using high price type
double value = T3(High, 20, 3, 0.7)[0];
Print("The current T3 value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.56 Moving Average - Triangular (TMA)

Description

The Triangular Moving Average is a form of Weighted Moving Average wherein the weights are assigned in a triangular pattern. For example, the weights for a 7 period Triangular Moving Average would be 1, 2, 3, 4, 3, 2, 1. This gives more weight to the middle of the time series and less weight to the oldest and newest data.

Syntax

```
TMA(int period)
TMA(ISeries<double> input, int period)

Returns default value
TMA(int period)[int barsAgo]
TMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period TMA using default price
type
double value = TMA(20)[0];
Print("The current TMA value is " + value.ToString());

// Prints the current value of a 20 period TMA using high price
type
double value = TMA(High, 20)[0];
Print("The current TMA value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.57 Moving Average - Triple Exponential (TEMA)

Description

The TEMA is a smoothing indicator. It was developed by Patrick Mulloy and is described in his article in the January, 1994 issue of Technical Analysis of Stocks and Commodities magazine.

Syntax

```
TEMA(int period)
TEMA(ISeries<double> input, int period)

Returns default value
TEMA(int period)[int barsAgo]
TEMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period TEMA using default price type
double value = TEMA(20)[0];
Print("The current TEMA value is " + value.ToString());

// Prints the current value of a 20 period TEMA using high price type
double value = TEMA(High, 20)[0];
Print("The current TEMA value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.58 Moving Average - Triple Exponential (TRIX)

Description

The triple exponential average (TRIX) indicator is an oscillator used to identify oversold and overbought markets, and it can also be used as a momentum indicator.

... Courtesy of Investopedia

Syntax

```
TRIX(int period, int signalPeriod)
TRIX(ISeries<double> input, int period, int signalPeriod)

Returns trix value
TRIX(int period, int signalPeriod)[int barsAgo]
TRIX(ISeries<double> input, int period, int signalPeriod)[int barsAgo]

Returns signal value
TRIX(int period, int signalPeriod).Signal[int barsAgo]
TRIX(ISeries<double> input, int period, int signalPeriod).Signal[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation
signalPeriod	Period for signal line

Examples

```
// Prints the current value of a 20 period TRIX using default price
type
double value = TRIX(20, 3).Default[0];
Print("The current TRIX value is " + value.ToString());

// Prints the current signal value of a 20 period TRIX using high
price type
double value = TRIX(High, 20, 3).Signal[0];
Print("The current TRIX signal value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.59 Moving Average - Variable (VMA)

Description

A Variable Moving Average is an exponential moving average that automatically adjusts its smoothing percentage based on market volatility. Giving more weight to the current data increases sensitivity thus making it a better signal indicator for short and long term markets.

Syntax

```
VMA(int period, int volatilityPeriod)
VMA(ISeries<double> input, int period, int volatilityPeriod)

Returns default value
VMA(int period, int volatilityPeriod)[int barsAgo]
VMA(ISeries<double> input, int period, int volatilityPeriod)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation
volatilityPeriod	The number of bars used to calculate the CMO based volatility index

Examples

```
// OnBarUpdate method of a strategy
protected override void OnBarUpdate()
{
    // Print out the VMA value of lows 3 bars ago for fun
    double value = VMA(Low, 9, 9)[3];
    Print("The value is " + value.ToString());

    // Go long if price closes above the current VMA value
    if (Close[0] > VMA(9, 9)[0])
        EnterLong();
}
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.60 Moving Average - Volume Weighted (VWMA)

Description

The Volume Weighted Moving Average is a weighted moving average that uses the volume as the weighting factor, so that higher volume days have more weight. It is a non-cumulative moving average, in that only data within the time period is used in the calculation.

Syntax

```
VWMA(int period)
VWMA(ISeries<double> input, int period)

Returns default value
VWMA(int period)[int barsAgo]
VWMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

period	Number of bars used in the calculation

Examples

```
// OnBarUpdate method
protected override void OnBarUpdate()
{
    // Evaluates for a VWMA cross over to the long side
    if (CrossAbove(VWMA(14), VWMA(40), 1))
        Print("We have a moving average cross over long");

    // Prints the current 14 period VWMA of high prices to the
output window
    double value = VWMA(High, 14)[0];
    Print("The current VWMA value of high prices is " +
value.ToString());
}
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.61 Moving Average - Weighted (WMA)

Description

The Weighted Moving Average gives the latest price more weight than prior prices. Each prior price in the period gets progressively less weight as they become older.

Syntax

```
WMA(int period)
WMA(ISeries<double> input, int period)

Returns default value
WMA(int period)[int barsAgo]
WMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period WMA using default price
type
double value = WMA(20)[0];
Print("The current WMA value is " + value.ToString());

// Prints the current value of a 20 period WMA using high price
type
double value = WMA(High, 20)[0];
Print("The current WMA value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.62 Moving Average - Zero Lag Exponential (ZLEMA)

Description

The Zero-Lag Exponential Moving Average is a variation on the Exponential Moving Average. The Zero-Lag keeps the benefit of the heavier weighting of recent values, but attempts to remove lag by subtracting older data to minimize the cumulative effect.

... Courtesy of FMLabs

Syntax

```
ZLEMA(int period)
ZLEMA(ISeries<double> input, int period)

Returns default value
ZLEMA(int period)[int barsAgo]
ZLEMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period ZLEMA using default
price type
double value = ZLEMA(20)[0];
Print("The current SMA value is " + value.ToString());

// Prints the current value of a 20 period ZLEMA using high price
type
double value = ZLEMA(High, 20)[0];
Print("The current ZLEMA value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.63 Moving Average Convergence-Divergence (MACD)

Description

MACD uses moving averages, which are lagging indicators, to include some trend-following characteristics. These lagging indicators are turned into a momentum oscillator by subtracting the longer moving average from the shorter moving average.

... Courtesy of StockCharts

Syntax

```
MACD(int fast, int slow, int smooth)
MACD(ISeries<double> input, int fast, int slow, int smooth)

Returns MACD value
MACD(int fast, int slow, int smooth)[int barsAgo]
MACD(ISeries<double> input, int fast, int slow, int smooth)[int barsAgo]
```

```
Returns average value
MACD(int fast, int slow, int smooth).Avg[int barsAgo]
MACD(ISeries<double> input, int fast, int slow, int smooth).Avg[int barsAgo]

Returns difference value
MACD(int fast, int slow, int smooth).Diff[int barsAgo]
MACD(ISeries<double> input, int fast, int slow, int smooth).Diff[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

fast	The number of bars to calculate the fast EMA
input	Indicator source data (?)
slow	The numbers of bars to calculate the slow EMA
smooth	The number of bars to calculate the EMA signal line

Examples

```
// Prints the current MACD value
double value = MACD(12, 26, 9)[0];
Print("The current MACD value is " + value.ToString());

// Prints the current MACD average value
double value = MACD(12, 26, 9).Avg[0];
Print("The current MACD average value is " + value.ToString());

// Prints the current MACD difference value
double value = MACD(12, 26, 9).Diff[0];
Print("The current MACD difference value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.64 Moving Average Ribbon

Description

The Moving Average Ribbon is a series of incrementing moving averages.

Syntax

```
MovingAverageribbon(RibbonMAType movingAverage, int basePeriod, int incrementalPeriod)
MovingAverageribbon(ISeries<double> input, RibbonMAType movingAverage, int basePeriod,
int incrementalPeriod)

Returns the MovingAverage1 value (Replace the 1 with the desired moving average you
want the value to return)
MovingAverageribbon(RibbonMAType movingAverage, int basePeriod, int
incrementalPeriod).MovingAverage1[int barsAgo]
MovingAverageribbon(ISeries<double> input, RibbonMAType movingAverage, int basePeriod,
int incrementalPeriod).MovingAverage1[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
RibbonMAType	Moving average to use for calculations
basePeriod	Number of bars used in the calculation for the fastest moving average
incrementalPeriod	Number of bars to increase for the calculation in each additional moving average

Examples

```
// Prints the current value of the 3rd moving average
double value = MovingAverageRibbon(RibbonMAType.Exponential, 10,
10).MovingAverage3[0];
Print("The current 3rd moving average's value is " +
value.ToString());
```

11.6.2.18.65 Net Change Display

Description

Displays net change on the chart.

Syntax

```
NetChangeDisplay(PerformanceUnit, NetChangePosition location)
NetChangeDisplay(ISeries<double> input, PerformanceUnit, NetChangePosition location)
```

Return Value

double

Parameters

input	Indicator source data (?)
PerformanceUnit	Format of the calculation of net change
NetChangePosition	Location to display net change on the chart

Examples

```
// Runs on realtime since there is no historical data for this
indicator
if (State == State.Historical)
    return;
else if (State >= State.Realtime)
{
// Prints the current tick value of the net change
    var ncd = NetChangeDisplay(PerformanceUnit.Ticks,
    NetChangePosition.BottomRight);
    Print("The current Net Change value is " + ncd.NetChange);
}
```

Note: This indicator only plots real-time. Historical values will print as 0.

11.6.2.18.66 n Bars Down

Description

Evaluates for n number of consecutive lower closes. Returns a value of 1 when the condition is true or 0 when false.

Syntax

```
NBarsDown(int barCount, bool barDown, bool lowerHigh, bool lowerLow)
NBarsDown(ISeries<double> input, int barCount, bool barDown, bool lowerHigh, bool
lowerLow)

Returns default value
NBarsDown(int barCount, bool barDown, bool lowerHigh, bool lowerLow)[int barsAgo]
NBarsDown(ISeries<double> input, bool barCount, int barDown, bool lowerHigh, bool
lowerLow)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
barCount	The number of required consecutive lower closes
barDown	Each bar's open must be less than the close; `true` or `false`
lowerHigh	Consecutive lower highs required; `true` or `false`
lowerLow	Consecutive lower lows required; `true` or `false`

Example

```
// OnBarUpdate method
protected override void OnBarUpdate()
{
    // Evaluates if we have 3 consecutive lower closes
    double value = NBarsDown(3, true, true, true)[0];

    if (value == 1)
        Print("We have three consecutive lower closes");
}
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.67 n Bars Up

Description

Evaluates for n number of consecutive higher closes. Returns a value of 1 when the condition is true or 0 when false.

Syntax

```
NBarsUp(int barCount, bool barUp, bool higherHigh, bool higherLow)
NBarsUp(ISeries<double> input, int barCount, bool barUp, bool higherHigh, bool higherLow)

Returns default value
NBarsUp(int barCount, bool barUp, bool higherHigh, bool higherLow)[int barsAgo]
NBarsUp(ISeries<double> input, int barCount, bool barUp, bool higherHigh, bool higherLow)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
barCount	The number of required consecutive higher closes
barUp	Each bar's close must be higher than the open; `true` or `false`
higherHigh	Consecutive higher highs required; `true` or `false`
higherLow	Consecutive higher lows required; `true` or `false`

Example

```
// OnBarUpdate method
protected override void OnBarUpdate()
{
    // Evaluates if we have 3 consecutive higher closes
    double value = NBarsUp(3, true, true, true)[0];

    if (value == 1)
        Print("We have three consecutive higher closes");
}
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.68 On Balance Volume (OBV)

Description

OBV is a simple indicator that adds a period's volume when the close is up and subtracts the period's volume when the close is down. A cumulative total of the volume additions and subtractions forms the OBV line. This line can then be compared with the price chart of the underlying security to look for divergences or confirmation.

... Courtesy of StockCharts

Syntax

```
OBV()
OBV(ISeries<double> input)

Returns default value
OBV()[int barsAgo]
OBV(ISeries<double> input)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Example

```
// Prints the current value of OBV
double value = OBV()[0];
Print("The current OBV value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.69 Order Flow Cumulative Delta

Description

An indicator that accumulates the volume of orders filled at bid and ask prices or up and down ticks throughout the session and compares them to determine buy/sell pressure.

Syntax

```
OrderFlowCumulativeDelta(CumulativeDeltaType deltaType, CumulativeDeltaPeriod period,
int sizeFilter)
OrderFlowCumulativeDelta(ISeries<double> input, CumulativeDeltaType deltaType,
CumulativeDeltaPeriod period, int sizeFilter)

Returns Open value
OrderFlowCumulativeDelta(CumulativeDeltaType deltaType, CumulativeDeltaPeriod period,
int sizeFilter).DeltaOpen[int barsAgo]
OrderFlowCumulativeDelta(ISeries<double> input, CumulativeDeltaType deltaType,
CumulativeDeltaPeriod period, int sizeFilter).DeltaOpen[int barsAgo]

Returns High value
OrderFlowCumulativeDelta(CumulativeDeltaType deltaType, CumulativeDeltaPeriod period,
int sizeFilter).DeltaHigh[int barsAgo]
OrderFlowCumulativeDelta(ISeries<double> input, CumulativeDeltaType deltaType,
CumulativeDeltaPeriod period, int sizeFilter).DeltaHigh[int barsAgo]

Returns Low value
OrderFlowCumulativeDelta(CumulativeDeltaType deltaType, CumulativeDeltaPeriod period,
int sizeFilter).DeltaLow[int barsAgo]
OrderFlowCumulativeDelta(ISeries<double> input, CumulativeDeltaType deltaType,
CumulativeDeltaPeriod period, int sizeFilter).DeltaLow[int barsAgo]

Returns Close value
OrderFlowCumulativeDelta(CumulativeDeltaType deltaType, CumulativeDeltaPeriod period,
int sizeFilter).DeltaClose[int barsAgo]
OrderFlowCumulativeDelta(ISeries<double> input, CumulativeDeltaType deltaType,
CumulativeDeltaPeriod period, int sizeFilter).DeltaClose[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
deltaType	The type of data to delta calculates on: BidAsk UpDownTick
period	The period in which the delta accumulates: Session Bar
sizeFilter	Input to exclude volume less than the selected value

Examples

Calling the OrderFlowCumulativeDelta() method directly

```
// A 1 tick data series must be added to the OnStateChange() as
this indicator runs off of tick data
else if (State == State.Configure)
{
   AddDataSeries(Data.BarsPeriodType.Tick, 1);
}

// OnBarUpdate() logic
if (BarsInProgress == 0)
{
// Print the close of the cumulative delta bar with a delta type of
Bid Ask and with a Session period
Print("Delta Close: " + OrderFlowCumulativeDelta(BarsArray[0],
CumulativeDeltaType.BidAsk, CumulativeDeltaPeriod.Session,
0).DeltaClose[0]);
}
else if (BarsInProgress == 1)
{
// We have to update the secondary series of the cached indicator
to make sure the values we get in BarsInProgress == 0 are in sync
OrderFlowCumulativeDelta(BarsArray[0], CumulativeDeltaType.BidAsk,
CumulativeDeltaPeriod.Session,
0).Update(OrderFlowCumulativeDelta(BarsArray[0],
CumulativeDeltaType.BidAsk, CumulativeDeltaPeriod.Session,
0).BarsArray[1].Count - 1, 1);
}
```

Calling the OrderFlowCumulativeDelta() method by reference

```
private OrderFlowCumulativeDelta cumulativeDelta;

// A 1 tick data series must be added to OnStateChange() as this
indicator runs off of tick data
else if (State == State.Configure)
{
	AddDataSeries(Data.BarsPeriodType.Tick, 1);
}
else if (State == State.DataLoaded)
{
	// Instantiate the indicator
	cumulativeDelta =
OrderFlowCumulativeDelta(CumulativeDeltaType.BidAsk,
CumulativeDeltaPeriod.Session, 0);
}

if (BarsInProgress == 0)
{
	// Print the close of the cumulative delta bar with a delta
type of Bid Ask and with a Session period
	Print("Delta Close: " + cumulativeDelta.DeltaClose[0]);
}
else if (BarsInProgress == 1)
{
	// We have to update the secondary series of the hosted
indicator to make sure the values we get in BarsInProgress == 0 are
in sync
	cumulativeDelta.Update(cumulativeDelta.BarsArray[1].Count -
1, 1);
}
```

11.6.2.18.70 Order Flow Volumetric Bars

Description

NinjaTrader Order Flow Volumetric bars provide a detailed 'x-ray' view into each price bar's aggressive buying and selling activity. This technique primarily attempts to answer the question which side was the most aggressive at each price level. This is done by calculating the delta (greek for difference) between buying and selling volume.

Many of the NinjaTrader Order Flow Volumetric Bar and Bar Statistics values could be accessed from your custom NinjaScript objects further leveraging the power of these analysis techniques.

Methods and Properties the VolumetricBarsType exposes

BarDelta	Gets a long value with the total bar's delta

`CumulativeDelta`	Gets a `long` value with the cumulative delta (**Note:** the accumulation is reset at the session break)
`DeltaSh`	The delta since last time price touched the high of the bar, usually negative
`DeltaSl`	The delta since last time price touched the low of the bar, usually positive.
`GetAskVolumeForPrice(double price)`	Gets the ask volume (`long` value) for the passed in price
`GetBidVolumeForPrice(double price)`	Gets the sell volume (`long` value) for the passed in price
`GetDeltaForPrice(double price)`	Gets the horizontal delta (`long` value) for the passed in price
`GetDeltaPercent()`	Gets a `double` value with the delta % of volume for the bar
`GetMaximumPositiveDelta()`	Gets the highest positive delta (`long` value) for the bar (if there is no positive delta in the bar, it will get the lowest negative delta)
`GetMaximumNegativeDelta()`	Gets the highest negative delta (`long` value) for the bar (if there is no negative delta in the bar, it will get the lowest positive delta)
`GetMaximumVolume(bool? askVolume, out double price)`	Gets the highest Ask, Bid or combined volume (`long` value) for the bar and returns the price at which it occurred. - pass in `true` for getting the highest Ask volume - pass in `false` for getting the highest Bid volume - pass in `null` for getting the highest combined volume

	For scenarios where Ticks per level is greater than 1, this method will return the lowest price - with Ticks per level known, the remaining prices in the result cell could be custom calculated if desired.
`GetTotalVolumeForPrice(double price)`	Gets the total volume (`long` value) for the passed in price
`MaxSeenDelta`	Gets the highest delta (`long` value) seen intrabar
`MinSeenDelta`	Gets the lowest delta (`long` value) seen intrabar
`TotalBuyingVolume`	Gets the total buying volume (`long` value) for the bar
`TotalSellingVolume`	Gets the total selling volume (`long` value) for the bar
`Trades`	Gets to total number of trades (`long` value) for the bar

Example

```
protected override void OnBarUpdate()
{
         if (Bars == null)
            return;

        // This sample assumes the Volumetric series is the primary
DataSeries on the chart, if you would want to add a Volumetric
series to a
        // script, you could call AddVolumetric() in
State.Configure and then for example use
        // NinjaTrader.NinjaScript.BarsTypes.VolumetricBarsType
barsType = BarsArray[1].BarsType as
        // NinjaTrader.NinjaScript.BarsTypes.VolumetricBarsType;

         NinjaTrader.NinjaScript.BarsTypes.VolumetricBarsType
barsType = Bars.BarsSeries.BarsType as
         NinjaTrader.NinjaScript.BarsTypes.VolumetricBarsType;

         if (barsType == null)
            return;

         try
         {
            double price;
            Print("=============================================
========================");
            Print("Bar: " + CurrentBar);
            Print("Trades: " +
barsType.Volumes[CurrentBar].Trades);
            Print("Total Volume: " +
barsType.Volumes[CurrentBar].TotalVolume);
            Print("Total Buying Volume: " +
barsType.Volumes[CurrentBar].TotalBuyingVolume);
            Print("Total Selling Volume: " +
barsType.Volumes[CurrentBar].TotalSellingVolume);
            Print("Delta for bar: " +
barsType.Volumes[CurrentBar].BarDelta);
            Print("Delta for bar (%): " +
barsType.Volumes[CurrentBar].GetDeltaPercent());
            Print("Delta for Close: " +
barsType.Volumes[CurrentBar].GetDeltaForPrice(Close[0]));
            Print("Ask for Close: " +
barsType.Volumes[CurrentBar].GetAskVolumeForPrice(Close[0]));
            Print("Bid for Close: " +
barsType.Volumes[CurrentBar].GetBidVolumeForPrice(Close[0]));
            Print("Volume for Close: " +
barsType.Volumes[CurrentBar].GetTotalVolumeForPrice(Close[0]));
            Print("Maximum Ask: " +
barsType.Volumes[CurrentBar].GetMaximumVolume(true, out price) + "
at price: " + price);
            Print("Maximum Bid: " +
barsType.Volumes[CurrentBar].GetMaximumVolume(false, out price) + "
at price: " + price);
```

Note: Please note in the example above a CurrentBar reference is used as index, and not a BarsAgo reference.

11.6.2.18.71 Order Flow VWAP

Description

Volume Weighted Average Price. A total of the dollars traded for every transaction (price multiplied by number of shares traded) and then divided by the total shares traded for the day. Also included are standard deviation bands.

Syntax

```
OrderFlowVWAP(VWAPResolution resolution, TradingHours tradingHoursInstance,
VWAPStandardDeviations numStandardDeviations, double sD1Multiplier, double
sD2Multiplier, double sD3Multiplier)
OrderFlowVWAP(ISeries<double> input, VWAPResolution resolution, TradingHours
tradingHoursInstance, VWAPStandardDeviations numStandardDeviations, double
sD1Multiplier, double sD2Multiplier, double sD3Multiplier)

Returns the VWAP value
OrderFlowVWAP(VWAPResolution resolution, TradingHours tradingHoursInstance,
VWAPStandardDeviations numStandardDeviations, double sD1Multiplier, double
sD2Multiplier, double sD3Multiplier).VWAP[int barsAgo]
OrderFlowVWAP(ISeries<double> input, VWAPResolution resolution, TradingHours
tradingHoursInstance, VWAPStandardDeviations numStandardDeviations, double
sD1Multiplier, double sD2Multiplier, double sD3Multiplier).VWAP[int barsAgo]

Returns the StdDev1Upper value
OrderFlowVWAP(VWAPResolution resolution, TradingHours tradingHoursInstance,
VWAPStandardDeviations numStandardDeviations, double sD1Multiplier, double
sD2Multiplier, double sD3Multiplier).StdDev1Upper[int barsAgo]
OrderFlowVWAP(ISeries<double> input, VWAPResolution resolution, TradingHours
tradingHoursInstance, VWAPStandardDeviations numStandardDeviations, double
sD1Multiplier, double sD2Multiplier, double sD3Multiplier).StdDev1Upper[int barsAgo]

Returns the StdDev1Lower value
OrderFlowVWAP(VWAPResolution resolution, TradingHours tradingHoursInstance,
VWAPStandardDeviations numStandardDeviations, double sD1Multiplier, double
sD2Multiplier, double sD3Multiplier).StdDev1Lower[int barsAgo]
OrderFlowVWAP(ISeries<double> input, VWAPResolution resolution, TradingHours
tradingHoursInstance, VWAPStandardDeviations numStandardDeviations, double
sD1Multiplier, double sD2Multiplier, double sD3Multiplier).StdDev1Lower[int barsAgo]

Returns the StdDev2Upper value
OrderFlowVWAP(VWAPResolution resolution, TradingHours tradingHoursInstance,
VWAPStandardDeviations numStandardDeviations, double sD1Multiplier, double
sD2Multiplier, double sD3Multiplier).StdDev2Upper[int barsAgo]
```

```
OrderFlowVWAP(ISeries<double> input, VWAPResolution resolution, TradingHours
tradingHoursInstance, VWAPStandardDeviations numStandardDeviations, double
sD1Multiplier, double sD2Multiplier, double sD3Multiplier).StdDev2Upper[int barsAgo]

Returns the StdDev2Lower value
OrderFlowVWAP(VWAPResolution resolution, TradingHours tradingHoursInstance,
VWAPStandardDeviations numStandardDeviations, double sD1Multiplier, double
sD2Multiplier, double sD3Multiplier).StdDev2Lower[int barsAgo]
OrderFlowVWAP(ISeries<double> input, VWAPResolution resolution, TradingHours
tradingHoursInstance, VWAPStandardDeviations numStandardDeviations, double
sD1Multiplier, double sD2Multiplier, double sD3Multiplier).StdDev2Lower[int barsAgo]

Returns the StdDev3Upper value
OrderFlowVWAP(VWAPResolution resolution, TradingHours tradingHoursInstance,
VWAPStandardDeviations numStandardDeviations, double sD1Multiplier, double
sD2Multiplier, double sD3Multiplier).StdDev3Upper[int barsAgo]
OrderFlowVWAP(ISeries<double> input, VWAPResolution resolution, TradingHours
tradingHoursInstance, VWAPStandardDeviations numStandardDeviations, double
sD1Multiplier, double sD2Multiplier, double sD3Multiplier).StdDev3Upper[int barsAgo]

Returns the StdDev3Lower value
OrderFlowVWAP(VWAPResolution resolution, TradingHours tradingHoursInstance,
VWAPStandardDeviations numStandardDeviations, double sD1Multiplier, double
sD2Multiplier, double sD3Multiplier).StdDev3Lower[int barsAgo]
OrderFlowVWAP(ISeries<double> input, VWAPResolution resolution, TradingHours
tradingHoursInstance, VWAPStandardDeviations numStandardDeviations, double
sD1Multiplier, double sD2Multiplier, double sD3Multiplier).StdDev3Lower[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
resolution	The data the indicator will run off of: Standard Tick
tradingHoursInstance	The trading hour template that will indicate when the VWAP resets
numStandardDeviations	The number of standard deviations of the VWAP

sD1Multiplier	The multiplier for the first standard deviation
sD2Multiplier	The multiplier for the second standard deviation
sD3Multiplier	The multiplier for the third standard deviation

Examples

```
// A 1 tick data series must be added to the OnStateChange() if
using a Tick Resolution (our second example call below in
OnBarUpdate())
else if (State == State.Configure)
{
	AddDataSeries(Data.BarsPeriodType.Tick, 1);
}

// OnBarUpdate() logic
if (BarsInProgress == 0)
{
// Prints the VWAP value using a standard resolution off of RTH
trading hours
double VWAPValue = OrderFlowVWAP(VWAPResolution.Standard,
TradingHours.String2TradingHours("CME US Index Futures RTH"),
VWAPStandardDeviations.Three, 1, 2, 3).VWAP[0];
Print("The current VWAP with a standard resolution on CME US Index
Futures RTH is " + VWAPValue.ToString());

// Prints the first upper standard deviation value using a tick
resolution off of trading hours of the Data Series
double VWAPStdDevUp1 = OrderFlowVWAP(VWAPResolution.Tick,
Bars.TradingHours, VWAPStandardDeviations.Three, 1, 2,
3).StdDev1Upper[0];
Print("The current VWAP with a tick resolution on " +
Bars.TradingHours.ToString() + " is " + VWAPStdDevUp1.ToString());
}
else if (BarsInProgress == 1)
{
// We have to update the secondary tick series of the cached
indicator using Tick Resolution to make sure the values we get in
BarsInProgress == 0 are in sync
OrderFlowVWAP(BarsArray[0], VWAPResolution.Tick,
BarsArray[0].TradingHours, VWAPStandardDeviations.Three, 1, 2,
3).Update(OrderFlowVWAP(BarsArray[0], VWAPResolution.Tick,
BarsArray[0].TradingHours, VWAPStandardDeviations.Three, 1, 2,
3).BarsArray[1].Count - 1, 1);
}
```

Notes:
1. Referencing multiple OrderFlowVWAP's with different ResetInterval's in a single NinjaScript Indicator / Strategy is not supported by default. Please contact platformsupport@ninjatrader.com for a workaround.

2. Referencing OrderFlowVWAP in a NinjaScript indicator or strategy which runs on either Calcuate.OnEachTick or .OnPriceChange, historical data is needed for accurate calculations.

11.6.2.18.72 Parabolic SAR

Description

The parabolic SAR is a technical indicator that is used by many traders to determine the direction of an asset's momentum and the point in time when this momentum has a higher-than-normal probability of switching directions.

... Courtesy of Investopedia

Syntax

```
ParabolicSAR(double acceleration, double accelerationMax, double accelerationStep)
ParabolicSAR(ISeries<double> input, double acceleration, double accelerationMax,
double accelerationStep)
```

```
Returns default value
ParabolicSAR(double acceleration, double accelerationMax, double accelerationStep)[int
 barsAgo]
ParabolicSAR(ISeries<double> input, double acceleration, double accelerationStep,
double accelerationMax)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

acceleration	Acceleration value
accelerationStep	Step value used to increment acceleration value
accelerationMax	Max acceleration value
input	Indicator source data (?)

Example

```
// Prints the current value of ParabolicSAR using default price
type
double value = ParabolicSAR(0.02, 0.2, 0.02)[0];
Print("The current ParabolicSAR value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.73 Percentage Price Oscillator (PPO)

Description

The Percentage Price Oscillator shows the percentage difference between two exponential moving averages.

Syntax

```
PPO(int fast, int slow, int smooth)
PPO(ISeries<double> input, int fast, int slow, int smooth)

Returns default value
PPO(int fast, int slow, int smooth)[int barsAgo]
PPO(ISeries<double> input, int fast, int slow, int smooth)[int barsAgo]

Returns smoothed value
PPO(int fast, int slow, int smooth).Smoothed[int barsAgo]
PPO(ISeries<double> input, int fast, int slow, int smooth).Smoothed[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

fast	The number of bars to calculate the fast EMA
input	Indicator source data (?)
slow	The number of bars to calculate the slow EMA

smooth	The number of bars to calculate the EMA signal line

Example

```
// Prints the current value of a 20 period Percentage Price
Oscillator
double value = PPO(12, 26, 9)[0];
Print("The current Percentage Price Oscillator value is " +
value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.74 Pivots

Description

The pivot point is used as a predictive indicator. If the following day's market price falls below the pivot point, it may be used as a new resistance level. Conversely, if the market price rises above the pivot point, it may act as the new support level.

... Courtesy of Investopedia

Syntax

```
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width)
Pivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width)

Returns pivot point value
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).Pp[int
barsAgo]
Pivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).PP[int barsAgo]

Returns R1 value
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).R1[int
barsAgo]
Pivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
```

```
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).R1[int barsAgo]

Returns R2 value
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).R2[int
barsAgo]
Pivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).R2[int barsAgo]

Returns R3 value
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).R3[int
barsAgo]
Pivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).R3[int barsAgo]

Returns S1 value
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).S1[int
barsAgo]
Pivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).S1[int barsAgo]

Returns S2 value
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).S2[int
barsAgo]
Pivots(ISeries<double> input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).S2[int barsAgo]

Returns S3 value
Pivots(PivotRange pivotRangeType, HLCCalculationMode priorDayHLC, double
userDefinedClose, double userDefinedHigh, double userDefinedLow, int width).S3[int
barsAgo]
Pivots(ISeries<double>input, PivotRange pivotRangeType, HLCCalculationMode
priorDayHLC, double userDefinedClose, double userDefinedHigh, double userDefinedLow,
int width).S3[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int *barsAgo*] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
pivotRangeType	Sets the range for the type of pivot calculated. Possible values are: PivotRange.Daily PivotRange.Weekly PivotRange.Monthly
priorDayHLC	Sets how the prior range High, Low, Close values are calculated. Possible values are: HLCCalculationMode.CalcFromIntradayData HLCCalculationMode.DailyBars HLCCalculationMode.UserDefinedValues
userDefinedClose	Sets the close for Pivots calculations when using HLCCalculationMode.UserDefinedValues.
userDefinedHigh	Sets the high for Pivots calculations when using HLCCalculationMode.UserDefinedValues.
userDefinedLow	Sets the low for Pivots calculations when using HLCCalculationMode.UserDefinedValues.
width	Sets how long the Pivots lines will be drawn

Examples

```
// Prints the current pivot point value
double value = Pivots(PivotRange.Daily,
HLCCalculationMode.CalcFromIntradayData, 0, 0, 0, 20).Pp[0];
Print("The current Pivots' pivot value is " + value.ToString());

// Prints the current S2 pivot value
double value = Pivots(PivotRange.Daily,
HLCCalculationMode.CalcFromIntradayData, 0, 0, 0, 20).S2[0];
Print("The current Pivots' S2 pivot value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

Tip: When using HLCCalculationMode.DailyBars it can be expected that a value of 0 is returned when the daily bars have not been loaded yet. Due to the asynchronous nature of this indicator calling daily bars you should only access the pivot values when the indicator has loaded all required Bars objects. To ensure you are accessing accurate values you can use .IsValidDataPoint() as a check:

```
// Evaluates that this is a valid pivot point value
if (Pivots(PivotRange.Daily, HLCCalculationMode.DailyBars, 0, 0,
 0, 20).Pp.IsValidDataPoint(0))
{
     // Prints the current pivot point value
     double value = Pivots(PivotRange.Daily,
HLCCalculationMode.DailyBars, 0, 0, 0, 20).Pp[0];
     Print("The current Pivots' pivot value is " +
value.ToString());
}
```

11.6.2.18.75 Polarized Fractal Efficiency (PFE)

Description

The Polarized Fractal Efficiency indicator uses fractal geometry to determine how efficiently the price is moving. When the PFE is zigzagging around zero, then the price is congested and not trending. When the PFE is smooth and above/below zero, then the price is in an up/down trend. The higher/lower the PFE value, the stronger the trend is.

... Courtesy of FMLabs

Syntax

```
PFE(int period, int smooth)
PFE(ISeries<double> input, int period, int smooth)

Returns default value
PFE(int period, int smooth)[int barsAgo]
PFE(ISeries<double> input, int period, int smooth)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation
smooth	The smoothing factor to be applied

Examples

```
// Prints the current value of a 20 period PFE using default price
type
double value = PFE(20, 2)[0];
Print("The current PFE value is " + value.ToString());

// Prints the current value of a 20 period PFE using high price
type
double value = PFE(High, 20, 2)[0];
Print("The current PFE value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.76 Price Oscillator

Description

The Price Oscillator is an indicator based on the difference between two moving averages, and is expressed as either a percentage or in absolute terms.

... Courtesy of StockCharts

Syntax

```
PriceOscillator(int fast, int slow, int smooth)
PriceOscillator(ISeries<double> input, int fast, int slow, int smooth)

Returns default value
PriceOscillator(int fast, int slow, int smooth)[int barsAgo]
PriceOscillator(ISeries<double> input, int fast, int slow, int smooth)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

fast	The number of bars to calculate the fast EMA
input	Indicator source data (?)
slow	The number of bars to calculate the slow EMA
smooth	The number of bars to calculate the EMA signal line

Example

```
// Prints the current value of a 20 period PriceOscillator using
default price type
double value = PriceOscillator(12, 26, 9)[0];
Print("The current PriceOscillator value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.77 Prior Day OHLC

Description

The prior day (session) open, high, low and close values.

> **Note**: Only use this indicator on intraday series.

Syntax

```
PriorDayOHLC()
PriorDayOHLC(ISeries<double> input)

Returns prior session open value
PriorDayOHLC().PriorOpen[int barsAgo]
PriorDayOHLC(ISeries<double> input).PriorOpen[int barsAgo]
```

```
Returns prior session high value
PriorDayOHLC().PriorHigh[int barsAgo]
PriorDayOHLC(ISeries<double> input).PriorHigh[int barsAgo]

Returns prior session low value
PriorDayOHLC().PriorLow[int barsAgo]
PriorDayOHLC(ISeries<double> input).PriorLow[int barsAgo]

Returns prior session close value
PriorDayOHLC().PriorClose[int barsAgo]
PriorDayOHLC(ISeries<double> input).PriorClose[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Example

```
// Prints the value of the prior session low
double value = PriorDayOHLC().PriorLow[0];
Print("The prior session low value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.78 Psychological Line

Description

The Psychological Line is the ratio of the number of rising bars over the specified number of bars.

Syntax

```
PsychologicalLine(int period)
PsychologicalLine(ISeries<double> input, int period)
```

Return Value

double; Accessing this method via an index value [int *barsAgo*] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 10 period Psychological Line
double value = PsychologicalLine(10)[0];
Print("The current Psychological Line value is " +
value.ToString());
```

11.6.2.18.79 Range

Description

Returns the range of a bar.

Syntax

```
Range()
Range(ISeries<double> input)

Returns default value
Range()[int barsAgo]
Range(ISeries<double> input)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int *barsAgo*] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the range of the current bar
double value = Range()[0];
Print("The current bar's range is " + value.ToString());

// Prints the 20 period simple moving average of range
double value = SMA(Range(), 20)[0];
Print("The 20 period average of range is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.80 Range Indicator (RIND)

Description

The Range indicator compares the intraday range (high - low) to the inter-day (close - previous close) range. When the inter-day range is less than the intraday range, the Range Indicator will be a high value. This signals an end to the current trend. When the Range Indicator is at a low level, a new trend is about to start.

The Range Indicator was developed by Jack Weinberg and was introduced in his article in the June, 1995 issue of Technical Analysis of Stocks & Commodities magazine.

Syntax

```
RIND(int periodQ, int smooth)
RIND(ISeries<double> input, int periodQ, int smooth)

Returns default value
RIND(int periodQ, int smooth)[int barsAgo]
RIND(ISeries<double> input, int periodQ, int smooth)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

periodQ	The number of bars to include in the calculation for the short term stochastic range lookback
smooth	The number of bars to include for the EMA smoothing of the indicator

Example

```
// Prints out a historical RIND value
double value = RIND(3, 10)[5];
Print("RIND value of 5 bars ago is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.81 Rate of Change (ROC)

Description

The Rate of Change (ROC) indicator is a very simple yet effective momentum oscillator that measures the percent change in price from one period to the next. The ROC calculation compares the current price with the price n periods ago.

... Courtesy of StockCharts

Syntax

```
ROC(int period)
ROC(ISeries<double> input, int period)

Returns default value
ROC(int period)[int barsAgo]
ROC(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period ROC using default price
type
double value = ROC(20)[0];
Print("The current ROC value is " + value.ToString());

// Prints the current value of a 20 period ROC using high price
type
double value = ROC(High, 20)[0];
Print("The current ROC value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.82 Regression Channel

Description

A Regression Channel is created by drawing parallel lines above and below the Linear Regression line.

Parallel and equidistant lines are drawn n standard deviations (width parameter) above and below a Linear Regression trendline. The distance between the channel lines and the regression line is the greatest distance that any one closing price is from the regression line. Regression Channels contain price movement, the top channel line provides resistance and the bottom channel line provides support. A reversal in trend may be indicated when prices remain outside the channel for a longer period of time.

A Linear Regression trendline shows where equilibrium exists but Linear Regression Channels show the range prices can be expected to deviate from a trendline.

Syntax

```
RegressionChannel(int period, double width)
RegressionChannel(ISeries<double> input, int period, double width)

Returns default midline value
RegressionChannel(int period, double width)[int barsAgo]
```

```
RegressionChannel(ISeries<double> input, int period, double width)[int barsAgo]

Returns upper channel value
RegressionChannel(int period, double width).Upper[int barsAgo]
RegressionChannel(ISeries<double> input, int period, double width).Upper[int barsAgo]

Returns lower channel value
RegressionChannel(int period, double width).Lower[int barsAgo]
RegressionChannel(ISeries<double> input, int period, double width).Lower[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation
width	Number of std deviations to calculate the channel lines

Tip: You should not access historical values of this indicator since the values can change from bar to bar. The values from n bars ago does not reflect what the values of the current bar really are. It is suggested that you only access the current bar value for this indicator.

Example

```
// Prints the current value of a 20 period channel using default
price type
double value = RegressionChannel(20, 2).Upper[0];
Print("The current upper channel value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.83 Relative Spread Strength (RSS)

Description

Developed by Ian Copsey, Relative Spread Strength is a variation to the Relative Strength Index.

Syntax

```
RSS(int eMA1, int eMA2, int length)
RSS(ISeries<double> input, int eMA1, int eMA2, int length)

Returns default value
RSS(int eMA1, int eMA2, int length)[int barsAgo]
RSS(ISeries<double> input, int eMA1, int eMA2, int length)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

eMA1	First EMA's period
eMA2	Second EMA's period
input	Indicator source data (?)
length	Number of bars used in the calculation

Examples

```
// Prints the current value of the RSS using default price type
double value = RSS(10, 40, 5)[0];
Print("The current RSS value is " + value.ToString());

// Prints the current value of the RSS using high price type
double value = RSS(High, 10, 40, 5)[0];
Print("The current RSS value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.84 Relative Strength Index (RSI)

Description

Developed by J. Welles Wilder and introduced in his 1978 book, New Concepts in Technical Trading Systems, the Relative Strength Index (RSI) is an extremely useful and popular momentum oscillator. The RSI compares the magnitude of a stock's recent gains to the magnitude of its recent losses and turns that information into a number that ranges from 0 to 100.

... Courtesy of StockCharts

The original Wilder formula for an exponential moving average with a smoothing constant (k = 1/ Period) is used to calculate the RSI.

Syntax

```
RSI(int period, int smooth)
RSI(ISeries<double> input, int period, int smooth)

Returns default value
RSI(int period, int smooth)[int barsAgo]
RSI(ISeries<double> input, int period, int smooth)[int barsAgo]

Returns avg value
RSI(int period, int smooth).Avg[int barsAgo]
RSI(ISeries<double> input, int period, int smooth).Avg[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation
smooth	Smoothing period

Examples

```
// Prints the current value of a 20 period RSI using default price
type
double value = RSI(20, 3)[0];
Print("The current RSI value is " + value.ToString());

// Prints the current value of a 20 period RSI using high price
type
double value = RSI(High, 20, 3)[0];
Print("The current RSI value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.85 Relative Vigor Index

Description

The Relative Vigor Index measures the strength of a trend by comparing an instruments closing price to its price range. It's based on the fact that prices tend to close higher than they open in up trends, and closer lower than they open in downtrends.

Syntax

```
RelativeVigorIndex(int period)
RelativeVigorIndex(ISeries<double> input, int period)
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 10 period Relative Vigor Index
double value = RelativeVigorIndex(10)[0];
Print("The current Relative Vigor Index value is " +
value.ToString());
```

11.6.2.18.86 Relative Volatility Index (RVI)

Description

Developed by Donald Dorsey, the Relative Volatility Index is the RSI using the standard deviation over the indicator period in place of the daily price change. The RVI measures the direction of volatility on a scale from 0 to 100. Readings below 50 indicate that the direction of volatility is to the downside and that you should be looking to sell, readings above 50 indicate that the direction of volatilty is to the upside and that you should be looking to buy.

Syntax

```
RVI(int period)
RVI(ISeries<double> input, int period)

Returns default value
RVI(int period)[int barsAgo]
RVI(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// OnBarUpdate method
protected override void OnBarUpdate()
{
    // Check for buy condition
    if (RVI(14)[0] > 50 && CrossAbove(SMA(9), SMA(14), 1))
    {
        EnterLong();
    }
}
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.87 R-squared

Description

The r-squared indicator calculates how well the price approximates a linear regression line. The indicator gets its name from the calculation, which is, the square of the correlation coefficient (referred to in mathematics by the Greek letter rho, or r). The range of the r-squared is from zero to one.

... Courtesy of FMLabs

Syntax

```
RSquared(int period)
RSquared(ISeries<double> input, int period)

Returns default value
RSquared(int period)[int barsAgo]
RSquared(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period R-squared using default
price type
double value = RSquared(20)[0];
Print("The current R-squared value is " + value.ToString());

// Prints the current value of a 20 period R-squared using high
price type
double value = RSquared(High, 20)[0];
Print("The current R-squared value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.88 Standard Deviation (StdDev)

Description

In probability theory and statistics, standard deviation is a measure of the variability or dispersion of a population, a data set, or a probability distribution. A low standard deviation indicates that the data points tend to be very close to the same value (the mean), while high standard deviation indicates that the data are “spread out” over a large range of values.

... Courtesy of Wikipedia

Syntax

```
StdDev(int period)
StdDev(ISeries<double> input, int period)

Returns default value
StdDev(int period)[int barsAgo]
StdDev(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period StdDev using default
price type
double value = StdDev(20)[0];
Print("The current StdDev value is " + value.ToString());

// Prints the current value of a 20 period StdDev using high price
type
double value = StdDev(High, 20)[0];
Print("The current StdDev value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.89 Standard Error (StdError)

Description

The standard error of a method of measurement or estimation is the standard deviation of the sampling distribution associated with the estimation method. The term may also be used to refer to an estimate of that standard deviation, derived from a particular sample used to compute the estimate.

... Courtesy of Wikipedia

Syntax

```
StdError(int period)
StdError(ISeries<double> input, int period)

Returns default value which is the mid line (also known as linear regression)
StdError(int period)[int barsAgo]
StdError(ISeries<double> input, int period)[int barsAgo]

Returns upper value
StdError(int period).Upper[int barsAgo]
StdError(ISeries<double> input, int period).Upper[int barsAgo]

Returns lower value
StdError(int period).Lower[int barsAgo]
StdError(ISeries<double> input, int period).Lower[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current upper value of a 20 period StdError using
default price type
double value = StdError(20).Upper[0];
Print("The current upper Standard Error value is " +
value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.90 Stochastics

Description

Developed by George C. Lane in the late 1950s, the Stochastic Oscillator is a momentum indicator that shows the location of the current close relative to the high/low range over a set number of periods. Closing levels that are consistently near the top of the range indicate accumulation (buying pressure) and those near the bottom of the range indicate distribution (selling pressure).

... Courtesy of StockCharts

Syntax

```
Stochastics(int periodD, int periodK, int smooth)
Stochastics(ISeries<double> input, int periodD, int periodK, int smooth)

Returns %D value
Stochastics(int periodD, int periodK, int smooth).D[int barsAgo]
Stochastics(ISeries<double> input, int periodD, int periodK, int smooth).D[int
barsAgo]

Returns %K value
Stochastics(int periodD, int periodK, int smooth).K[int barsAgo]
```

```
Stochastics(ISeries<double> input, int periodD, int periodK, int smooth).K[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
periodD	The period for the moving average of periodD
periodK	The period for the moving average of periodK
smooth	The smoothing value to be used

Examples

```
// Prints the current %D value
double value = Stochastics(7, 14, 3).D[0];
Print("The current Stochastics %D value is " + value.ToString());

// Prints the current %K value
double value = Stochastics(7, 14, 3).K[0];
Print("The current Stochastics %K value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.91 Stochastics Fast

Description

Developed by George C. Lane in the late 1950s, the Stochastic Oscillator is a momentum indicator that shows the location of the current close relative to the high/low range over a set number of periods. Closing levels that are consistently near the top of the range indicate accumulation (buying pressure) and those near the bottom of the range indicate distribution (selling pressure).

... Courtesy of StockCharts

Syntax

```
StochasticsFast(int periodD, int periodK)
StochasticsFast(ISeries<double> input, int periodD, int periodK)

Returns %D value
StochasticsFast(int periodD, int periodK).D[int barsAgo]
StochasticsFast(ISeries<double> input, int periodD, int periodK).D[int barsAgo]

Returns %K value
StochasticsFast(int periodD, int periodK).K[int barsAgo]
StochasticsFast(ISeries<double> input, int periodD, int periodK).K[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
periodD	The period for the moving average of periodD
periodK	The period for the moving average of periodK

Examples

```
// Prints the current %D value
double value = StochasticsFast(3, 14).D[0];
Print("The current StochasticsFast %D value is " +
value.ToString());

// Prints the current %K value
double value = StochasticsFast(3, 14).K[0];
Print("The current StochasticsFast %K value is " +
value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.92 Stochastics RSI (StochRSI)

Description

This is an indicator on indicator implementation. It is simply a Stochastics indicator applied on RSI.

Syntax

```
StochRSI(int period)
StochRSI(ISeries<double> input, int period)

Returns default value
StochRSI(int period)[int barsAgo]
StochRSI(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Evaluates if the current bar StochRSI value is greater than the
value one bar ago
if (StochRSI(14)[0] > StochRSI(14)[1])
    Print("Stochastics RSI is rising");
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.93 Summation (SUM)

Description

Returns the sum of the values taken over a specified period.

Syntax

```
SUM(int period)
SUM(ISeries<double> input, int period)

Returns default value
SUM(int period)[int barsAgo]
SUM(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period SUM using default price
type
double value = SUM(20)[0];
Print("The current SUM value is " + value.ToString());

// Prints the current value of a 20 period SUM using high price
type
double value = SUM(High, 20)[0];
Print("The current SUM value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.94 Swing

Description

The Swing indicator will plot lines that represent the swing points based on the strength (number of bars to the left and right of the swing point) parameter provided, it's mostly a visual tool and not meant to be predictive in nature. Only after the strength number of bars has passed since the extreme point, the swing return value could be definitely set, thus the indicator updates it's calculations as new incoming data warrants so.

You can access methods within this indicator to determine the number of bars ago a swing point occurred or the current swing value.

Tip: To workaround the situation, where the indicator has to recalculate - you could only access the SwingHigh / Low values the number of swing strength bars ago - those values are calculated in their final state.

Syntax - Bars Ago

```
High Bar
Swing(int strength).SwingHighBar(int barsAgo, int instance, int lookBackPeriod)
Swing(ISeries<double> input, int strength).SwingHighBar(int barsAgo, int instance, int
 lookBackPeriod)

Low Bar
Swing(int strength).SwingLowBar(int barsAgo, int instance, int lookBackPeriod)
Swing(ISeries<double> input, int strength).SwingLowBar(int barsAgo, int instance, int
lookBackPeriod)
```

Return Value

An `int` value representing the number of bars ago. Returns a value of -1 if a swing point is not found within the look back period.

Syntax - Value

```
High Value
Swing(int strength).SwingHigh[int barsAgo]
Swing(ISeries<double> input, int strength).SwingHigh[int barsAgo]

Low Value
Swing(int strength).SwingLow[int barsAgo]
Swing(ISeries<double> input, int strength).SwingLow[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

*** A return value of 0 (zero) will be returned if the CurrentBar number is less than the "strength" value, or a swing pivot has not yet been found.**

Parameters

barsAgo	The number of bars ago that serves as the starting bar from which to work backwards

input	Indicator source data (?)
instance	The occurrence to check for (1 is the most recent, 2 is the 2nd most recent, etc...)
lookBackPeriod	Number of bars to look back to check for the test condition, which is evaluated on the current bar and the bars in the look back period.
strength	The number of required bars to the left and right of the swing point

Example

```
// Prints the high price of the most recent swing high
Print("The high of the swing bar is " + High[Math.Max(0,
Swing(5).SwingHighBar(0, 1, 10))]);
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.95 Time Series Forecast (TSF)

Description

The Time Series Forecast function displays the statistical trend of a security's price over a specified time period based on linear regression analysis. Instead of a straight linear regression trendline, the Time Series Forecast plots the last point of multiple linear regression trendlines. This is why this indicator may sometimes referred to as the "moving linear regression" indicator or the "regression oscillator."

Syntax

```
TSF(int forecast, int period)
TSF(ISeries<double> input, int forecast, int period)

Returns default value
TSF(int forecast, int period)[int barsAgo]
TSF(ISeries<double> input, int forecast, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

forecast	Forecast period
input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Prints the current value of a 20 period TSF using default price
type
double value = TSF(3, 20)[0];
Print("The current TSF value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.96 Trend Lines

Description

When a high swing is followed by a lower high swing, a trend line high is automatically plotted. When a low swing is followed by a higher low swing, a trend line low is automatically plotted.

Syntax

```
TrendLines(int strength, int numberOfTrendLines, int oldTrendsOpacity, bool
alertOnBreak)
TrendLines(ISeries<double> input, int strength, int numberOfTrendLines, int
oldTrendsOpacity, bool alertOnBreak)
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

strength	The number of required bars to the left and right of the swing point
numberOfTrendLines	The number of recent trend lines to plot
oldTrendOpacity	The opacity to apply to old trend lines
alertOnBreak	Sets if there should be an alert when the price breaks the current trend line

Examples

```
// Prints the current value of a 5 strength Trend Lines
double value = TrendLines(5, 4, 25, true)[0];
Print("The current Trend Lines value is " + value.ToString());
```

11.6.2.18.97 True Strength Index (TSI)

Description

The True Strength Index (TSI) is a momentum-based indicator, developed by William Blau. Designed to determine both trend and overbought/oversold conditions, the TSI is applicable to intraday time frames as well as long term trading.

Syntax

```
TSI(int fast, int slow)
TSI(ISeries<double> input, int fast, int slow)

Returns default value
TSI(int fast, int slow)[int barsAgo]
TSI(ISeries<double> input, int fast, int slow)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

fast	Period of the fast smoothing factor

input	Indicator source data (?)
slow	Period of the slow smoothing factor

Examples

```
// Prints the current value of a 20 period TSI using default price
type
double value = TSI(20, 10)[0];
Print("The current TSI value is " + value.ToString());

// Prints the current value of a 20 period TSI using high price
type
double value = TSI(High, 20, 10)[0];
Print("The current TSI value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.98 Ultimate Oscillator

Description

Developed by Larry Williams and introduced in his article in the April, 1985 issue of Technical Analysis of Stocks and Commodities magazine, this indicator is the weighted sum of three oscillators of different time periods. The there time periods represent short, intermediate and long term market cycles. Typical periods are 7, 14 and 28. The values of the Ultimate Oscillator range from zero to 100. Values over 70 indicate overbought conditions, and values under 30 indicate oversold conditions.

Syntax

```
UltimateOscillator(int fast, int intermediate, int slow)
UltimateOscillator(ISeries<double> input, int fast, int intermediate, int slow)

Returns default value
UltimateOscillator(int fast, int intermediate, int slow)[int barsAgo]
UltimateOscillator(ISeries<double> input, int fast, int intermediate, int slow)[int
barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

fast	The number of bars to include in the short term period
input	Indicator source data (?)
intermediate	The number of bars to include in the intermediate term period
slow	The number of bars to include in the long term period

Example

```
// Prints the current value of a typical Ultimate Oscillator using
default price type
double value = UltimateOscillator(7, 14, 28)[0];
Print("The current Ultimate Oscillator value is " +
value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.99 Volume (VOL)

Description

Volume is simply the number of shares (or contracts) traded during a specified time frame (e.g., hour, day, week, month, etc). The analysis of volume is a basic yet very important element of technical analysis. Volume provides clues as to the intensity of a given price move.

... Courtesy of Market In Out

Syntax

```
VOL()
VOL(ISeries<double> input)
```

```
Returns default value
VOL()[int barsAgo]
VOL(ISeries<double> input)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Example

```
// Prints the current value VOL
double value = VOL()[0];
Print("The current VOL value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.100 Volume Moving Average (VOLMA)

Description

The Volume Moving Average indicator is an indicator on indicator implementation. It calculates and returns the value of an exponential moving average of volume.

Syntax

```
VOLMA(int period)
VOLMA(ISeries<double> input, int period)

Returns default value
VOLMA(int period)[int barsAgo]
VOLMA(ISeries<double> input, int period)[int barsAgo]
```

Return Value

double; Accessing this method via an index value [int barsAgo] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Example

```
// Evaluates if the current volume is greater than the 20 period
EMA of volume
if (Volume[0] > VOLMA(20)[0])
    Print("Volume has risen above its 20 period average");
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.101 Volume Oscillator

Description

The Volume Oscillator uses the difference between two moving averages of volume to determine if the trend is increasing or decreasing. A value above zero indicates that the shorter term volume moving average has risen above the longer term volume moving average. This indicates that the shorter term trend is higher than the longer term trend. Rising prices with with increased short term volume is bullish as is falling prices with decreased volume. Falling prices with increased volume or rising prices with decreased volume indicate market weakness.

Syntax

```
VolumeOscillator(int fast, int slow)
VolumeOscillator(ISeries<double> input, int fast, int slow)

Returns default value
VolumeOscillator(int fast, int slow)[int barsAgo]
VolumeOscillator(ISeries<double> input, int fast, int slow)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

fast	The number of bars to include in the short term moving average
input	Indicator source data (?)
slow	The number of bars to include in the long term moving average

Example

```
// Prints the current value of a Volume Oscillator
double value = VolumeOscillator(12, 26)[0];
Print("The current Volume Oscillator value is " +
value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.102 Volume Rate of Change (VROC)

Description

Volume Rate of Change is identical to Price Rate Of Change (ROC) indicator except that it uses volume instead of price.

Syntax

```
VROC(int period, int smooth)
VROC(ISeries<double> input, int period, int smooth)

Returns default value
VROC(int period, int smooth)[int barsAgo]
VROC(ISeries<double> input, int period, int smooth)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation
smooth	The number of bars for smoothing the signal

Example

```
// Prints the current value of VROC
double value = VROC(13, 3)[0];
Print("The current VROC value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.103 Volume Up Down

Description

Variation of the VOL (Volume) indicator that colors the volume histogram different color depending if the current bar is up or down bar.

Syntax

```
VolumeUpDown()
VolumeUpDown(ISeries<double> input)

Returns default value
VolumeUpDown()[int barsAgo]
VolumeUpDown(ISeries<double> input)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Example

```
// Prints the current value VolumeUpDown
double value = VolumeUpDown()[0];
Print("The current Volume value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.104 Vortex

Description

The Vortex indicator is an oscillator used to identify trends. A bullish signal triggers when the VIPlus line crosses above the VIMinus line. A bearish signal triggers when the VIMinus line crosses above the VIPlus line.

Syntax

```
Vortex(int period)
Vortex(ISeries<double> input, int period)

Returns VIPlus value
Vortex(int period).VIPlus[int barsAgo]
Vortex(ISeries<double> input, int period).VIPlus[int barsAgo]

Returns VIMinus value
Vortex(int period).VIMinus[int barsAgo]
Vortex(ISeries<double> input, int period).VIMinus[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current VIPlus value of a 14 period Vortex
double valueP = Vortex(14).VIPlus[0];
Print("The current Vortex VIPlus value is " + valueP.ToString());

// Prints the current VIMinus value of a 14 period Vortex
double valueM = Vortex(14).VIMinus[0];
Print("The current Vortex VIMinusvalue is " + valueM.ToString());
```

11.6.2.18.105 Williams %R

Description

Developed by Larry Williams, Williams %R is a momentum indicator that works much like the Stochastic Oscillator. It is especially popular for measuring overbought and oversold levels. The scale ranges from 0 to -100 with readings from 0 to -20 considered overbought, and readings from -80 to -100 considered oversold.

... Courtesy of StockCharts

Syntax

```
WilliamsR(int period)
WilliamsR(ISeries<double> input, int period)

Returns default value
WilliamsR(int period)[int barsAgo]
WilliamsR(ISeries<double> input, int period)[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
period	Number of bars used in the calculation

Examples

```
// Prints the current value of a 20 period WilliamsR using default
price type
double value = WilliamsR(20)[0];
Print("The current WilliamsR value is " + value.ToString());

// Prints the current value of a 20 period WilliamsR using high
price type
double value = WilliamsR(High, 20)[0];
Print("The current WilliamsR value is " + value.ToString());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.18.106 Wiseman Alligator

Description

The Wiseman Alligator is an indicator that consists of 3 moving averages with offsets applied to identify trend absence, formation, and direction. This indicator was provided by Profitunity: http://www.profitunity.com

Syntax

```
Wiseman Alligator(int jawPeriod, int teethPeriod, int lipsPeriod, int jawOffset, int
teethOffset, int lipsOffset)
Wiseman Alligator(ISeries<double> input, int jawPeriod, int teethPeriod, int
lipsPeriod, int jawOffset, int teethOffset, int lipsOffset)

Returns Teeth
Wiseman Alligator(int jawPeriod, int teethPeriod, int lipsPeriod, int jawOffset, int
teethOffset, int lipsOffset).Teeth[int barsAgo]
Wiseman Alligator(ISeries<double> input, int jawPeriod, int teethPeriod, int
lipsPeriod, int jawOffset, int teethOffset, int lipsOffset).Teeth[int barsAgo]

Returns Lips
Wiseman Alligator(int jawPeriod, int teethPeriod, int lipsPeriod, int jawOffset, int
teethOffset, int lipsOffset).Teeth[int barsAgo]
Wiseman Alligator(ISeries<double> input, int jawPeriod, int teethPeriod, int
lipsPeriod, int jawOffset, int teethOffset, int lipsOffset).Lips[int barsAgo]

Returns Jaw
Wiseman Alligator(int jawPeriod, int teethPeriod, int lipsPeriod, int jawOffset, int
teethOffset, int lipsOffset).Teeth[int barsAgo]
Wiseman Alligator(ISeries<double> input, int jawPeriod, int teethPeriod, int
lipsPeriod, int jawOffset, int teethOffset, int lipsOffset).Jaw[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
jawPeriod	Number of bars used in the jaw moving average calculation
teethPeriod	Number of bars used in the teeth moving average calculation
lipsPeriod	Number of bars used in the lips moving average calculation
jawOffset	The offset for the jaw moving average
teethOffset	The offset for the teeth moving average
lipsOffset	The offset for the lips moving average

Examples

```
// Prints the current value of the teeth for the Wiseman Alligator
double value = WisemanAlligator(13, 8, 5, 8, 5, 3).Teeth[0];
Print("The current Wiseman Alligator teeth value is " +
value.ToString());
```

11.6.2.18.107 Wiseman Awesome Oscillator

Description

The Wiseman Awesome Oscillator is a momentum indicator to identify trends and reversals. This indicator was provided by Profitunity: http://www.profitunity.com

Syntax

```
WisemanAwesomeOscillator()
WisemanAwesomeOscillator(ISeries<double> input)
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Examples

```
// Prints the current value of the Wiseman Awesome Oscillator
double value = WisemanAwesomeOscillator()[0];
Print("The current Wiseman Awesome Oscillator value is " +
value.ToString());
```

11.6.2.18.108 Woodies CCI

Description

NinjaTrader provides the Woodies CCI indicator. It's implemented as specified by Woodie.

Syntax

```
WoodiesCCI()
WoodiesCCI(ISeries<double> input)

Returns default value
WoodiesCCI()[int barsAgo]
WoodiesCCI(ISeries<double> input)[int barsAgo]

Returns turbo value
WoodiesCCI().Turbo[int barsAgo]
WoodiesCCI(ISeries<double> input).Turbo[int barsAgo]

Returns histogram bar color
WoodiesCCI().ZoneBars[int barsAgo]
WoodiesCCI(ISeries<double> input).ZoneBars[int barsAgo]

Return values representing the chopzone plot color are as follows:
0 = Negative (default color is red)
1 = Positive (default color is blue)
2 = Neutral (default color is gray)
3 = Last neutral bar (default color is yellow)

Returns chopzone value
WoodiesCCI().ChopZone[int barsAgo]
WoodiesCCI(ISeries<double> input).ChopZone[int barsAgo]

Return values representing the chopzone plot color are as follows:

-4 = DarkRed
-3 = LightRed
```

```
-2 = DarkOrange
-1 = LightOrange
0 = Yellow
1 = Lime
2 = LightGreen
3 = DarkGreen
4 = Cyan

Returns sidewinder value
WoodiesCCI().Sidewinder[int barsAgo]
WoodiesCCI(ISeries<double> input).Sidewinder[int barsAgo]

Return values representing the sidewinder plot value are as follows:

-1 = Warning
0 = Neutral
1 = Trending
```

Return Value

double; Accessing this method via an index value [int *barsAgo*] returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)

Example

```
// Prints the current value of a 14 period WoodiesCCI using default
price type
double value = WoodiesCCI(2, 5, 14, 34, 25, 6, 60, 100, 2)[0];
Print("The current WoodiesCCI value is " + value.ToString());

// Prints the current turbo value of a 14 / 6 period WoodiesCCI
using default price type
double value2 = WoodiesCCI(2, 5, 14, 34, 25, 6, 60, 100,
2).Turbo[0];
Print("The current WoodiesCCI turbo value is " +
value2.ToString());
```

11.6.2.18.109 Woodies Pivots

Description

Woodies CCI Club pivots indicator.

Syntax

```
WoodiesPivots(HLCCalculationModeWoodie priorDayHLC, int width)
WoodiesPivots(ISeries<double> input, HLCCalculationModeWoodie priorDayHLC, int width)

Returns pivot point value
WoodiesPivots(HLCCalculationModeWoodie priorDayHLC, int width).PP[int barsAgo]
WoodiesPivotsISeries<double> input, HLCCalculationModeWoodie priorDayHLC, int
width).PP[int barsAgo]

Returns R1 value
WoodiesPivots(HLCCalculationModeWoodie priorDayHLC, int width).R1[int barsAgo]
WoodiesPivots(ISeries<double> input, HLCCalculationModeWoodie priorDayHLC, int
width).R1[int barsAgo]

Returns R2 value
WoodiesPivots(HLCCalculationModeWoodie priorDayHLC, int width).R2[int barsAgo]
WoodiesPivots(ISeries<double> input, HLCCalculationModeWoodie priorDayHLC, int
width).R2[int barsAgo]

Returns S1 value
WoodiesPivots(HLCCalculationModeWoodie priorDayHLC, int width).S1[int barsAgo]
WoodiesPivots(ISeries<double> input, HLCCalculationModeWoodie priorDayHLC, int
width).S1[int barsAgo]

Returns S2 value
WoodiesPivots(HLCCalculationModeWoodie priorDayHLC, int width).S2[int barsAgo]
WoodiesPivots(ISeries<double> input, HLCCalculationModeWoodie priorDayHLC, int
width).S2[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

Parameters

input	Indicator source data (?)
priorDayHLC	Sets how the prior range High, Low, Close values are calculated. Possible values are: HLCCalculationModeWoodie.CalcFromIntradayD

	ata HLCCalculationModeWoodie.DailyBars HLCCalculationModeWoodie.UserDefinedValues
width	An `int` determining the width of the pivot values plotted

Example

```
// Prints the current pivot point value
double ppValue =
WoodiesPivots(HLCCalculationModeWoodie.CalcFromIntradayData,
20).PP[0];
Print("The current Woodies Pivots' pivot value is " + ppValue);

// Prints the current S2 pivot value
double s2Value =
WoodiesPivots(HLCCalculationModeWoodie.CalcFromIntradayData,
20).S2[0];
Print("The current Woodies Pivots' S2 pivot value is " + s2Value);
```

Tip: When using HLCCalculationMode.DailyBars it can be expected that a value of 0 is returned when the daily bars have not been loaded yet. Due to the asynchronous nature of this indicator calling daily bars you should only access the pivot values when the indicator has loaded all required Bars objects. To ensure you are accessing accurate values you can use .IsValidDataPoint()as a check:

```
// Evaluates that this is a valid Woodies Pivots value
if (WoodiesPivots(HLCCalculationModeWoodie.DailyBars,
20).PP.IsValidDataPoint(0))
{
     // Prints the current pivot point value
     double value =
WoodiesPivots(HLCCalculationModeWoodie.DailyBars, 20).PP[0];
     Print("The current Woodies Pivots' pivot value is " +
value.ToString());
}
```

11.6.2.18.110 ZigZag

Description

The ZigZag indicator highlights trends based on user defined threshold values and helps filtering the noise in price charts, it's not a classical indicator but more a reactive filter showing extreme price points. In processing it's calculations it can update it's current direction and price extreme point based on newly incoming data, the current developing leg should be thought of temporary until a new leg in opposite direction has been set.

You can access methods within this indicator to determine the number of bars ago a zigzag high or low point occurred or the current zigzag value, it is only meaningful to work with in Calculate.OnBarClose mode for the Calculate property.

Syntax - Bars Ago

```
High Bar
ZigZag(DeviationType deviationType, double deviationValue, bool
useHighLow).HighBar(int barsAgo, int instance, int lookBackPeriod)
ZigZag(ISeries<double> input, DeviationType deviationType, double deviationValue, bool
 useHighLow).HighBar(int barsAgo, int instance, int lookBackPeriod)

Low Bar
ZigZag(DeviationType deviationType, double deviationValue, bool useHighLow).LowBar(int
 barsAgo, int instance, int lookBackPeriod)
ZigZag(ISeries<double> input, DeviationType deviationType, double deviationValue, bool
 useHighLow).LowBar(int barsAgo, int instance, int lookBackPeriod)
```

Return Value

An `int` value representing the number of bars ago. Returns a value of -1 if a swing point is not found within the look back period.

Syntax - Value

```
High Value
ZigZag(DeviationType deviationType, double deviationValue, bool
useHighLow).ZigZagHigh[int barsAgo]
ZigZag(ISeries<double> input, DeviationType deviationType, double
deviationValue, bool useHighLow).ZigZagHigh[int barsAgo]

Low Value
ZigZag(DeviationType deviationType, double deviationValue, bool
useHighLow).ZigZagLow[int barsAgo]
ZigZag(ISeries<double> input, DeviationType deviationType, double
deviationValue, bool useHighLow).ZigZagLow[int barsAgo]
```

Return Value

`double`; Accessing this method via an index value `[int barsAgo]` returns the indicator value of the referenced bar.

*** A return value of 0 (zero) indicates that a zigzag high or low has not yet formed.**

Parameters

barsAgo	The number of bars ago that serves as the starting bar and works backwards
deviationType	Possible values are: DeviationType.Points DeviationType.Percent
deviationValue	The deviation value
input	Indicator source data (?)
instance	The occurrence to check for (1 is the most recent, 2 is the 2nd most recent etc...)
lookBackPeriod	Number of bars to look back to check for the test condition. Test is evaluated on the current bar and the bars in the look back period.
useHighLow	When true, both High and Low price series are used. When false, the default input is used for both highs and lows.

Example

```
// Prints the high price of the most recent zig zag high
Print("The high of the zigzag bar is " + High[Math.Max(0,
ZigZag(DeviationType.Points, 0.5, false).HighBar(0, 1, 100))]);
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

11.6.2.19 TradingHours

Definition

Represents the Trading Hours information returned from the current bars series. The Trading Hours object contains several methods and properties for working with various trading sessions.

> **Warning**: The properties in this class should **NOT** be accessed within the OnStateChange() method before the **State** has reached **State.DataLoaded**

Methods and Properties

Get()	Returns the Trading Hours object for the specified Trading Hours template name
GetPreviousTradingDayEnd()	Returns the end date and time of the previous trading session relative to the time passed in the methods parameters.
Holidays	A collection of full holidays which are configured for a Trading Hours template
Name	Indicates the name of the trading hours template applied to the Bars series object.
PartialHolidays	A collection of partial holidays which are configured for a Trading Hours template
Sessions	A collection of session definitions of the trading hours template.
TimeZoneInfo	Indicates a time zone that is configured by a Trading Hour template

11.6.2.19.1 Get

Definition

Returns the TradingHours object for the specified Trading Hours template name, such as "CME US Index Futures RTH"

Method Return Value

A TradingHours object representing the specified Trading Hours template name.

Syntax

```
Get(string name)
```

Parameters

name	The name of the desired TradingHours object to return

Examples

```
// Loop through and print all regular holidays in the found
TradingHours object
foreach(KeyValuePair<DateTime, string> holiday in
TradingHours.Get("CME US Index Futures RTH").Holidays)
{
   Print(String.Format("Date: {0} Description: {1}", holiday.Key,
holiday.Value));
}
```

11.6.2.19.2 GetPreviousTradingDayEnd()

Definition

Returns the end date and time of the previous trading session regarding the time passed in the methods parameters.

Method Return Value

A `DateTime` structure representing the previous trading days end date and time

Syntax

```
GetPreviousTradingDayEnd(DateTime timeLocal)
```

Warning: This method is resource intensive and should **ONLY** be reserved for situations when calculations would be limited to a few specific use cases. For example, calling this method for each bar in the OnBarUpdate() method would **NOT** be recommended.

Parameters

timeLocal	An `DateTime` structure which is used to calculate the current trading day

Examples

```
protected override void OnBarUpdate()
{
   if (Bars.IsFirstBarOfSession)
   {
      DateTime previousEndDate =
TradingHours.GetPreviousTradingDayEnd(Time[0]);

      Print(string.Format("The current bars date is {0} - the
previous trading session ended on {1}", Time[0], previousEndDate));
   }
   //Output:  The current bars date is 2/18/2015 12:35:00 PM - the
previous trading session ended on 2/17/2015 3:15:00 PM
}
```

11.6.2.19.3 Holidays

Definition

A collection of full holidays configured for a Trading Hours template. Holidays are days which fall outside of the regular trading schedule.

Note: For more information please see the "Understanding trading holidays" section of the Using the Trading Hours window.

Property Value

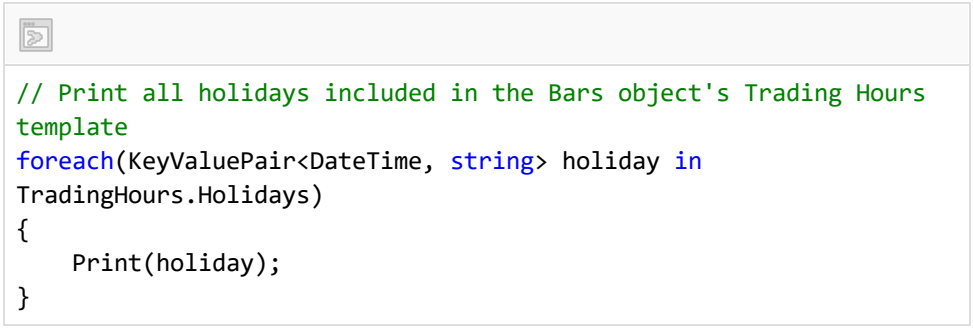
A Dictionary holding a collection of holiday Dates and Descriptions of each holiday.

Date	A `DateTime` representing the date of the trading hours holiday
Description	A `string` which is used to describe the holiday (e.g., Christmas)

Syntax

```
TradingHours.Holidays
```

Examples



```
// Print all holidays included in the Bars object's Trading Hours
template
foreach(KeyValuePair<DateTime, string> holiday in
TradingHours.Holidays)
{
    Print(holiday);
}
```

11.6.2.19.4 Name

Definition

Indicates the name of the trading hours template applied to the Bars series object.

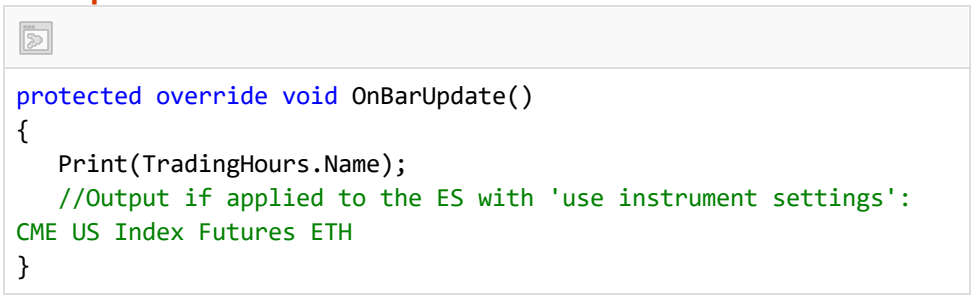
Property Value

A `string` representing the name of the trading hours template.

Syntax

`Bars.TradingHours.Name`

Examples



```
protected override void OnBarUpdate()
{
   Print(TradingHours.Name);
   //Output if applied to the ES with 'use instrument settings':
CME US Index Futures ETH
}
```

11.6.2.19.5 PartialHolidays

Definition

A collection of partial holidays which are configured for a Trading Hours template. Holidays are days which fall outside of the normal trading schedule, on which data will be excluded. For more information please see the "Understanding trading holidays" section of the Using the Trading Hours window.

Property Value

A Dictionary holding a collection of holiday Dates and PartialHoliday objects for each partial holiday.

Date	A `DateTime` representing the trading date of the **Trading**

	Hours holiday
PartialHoliday	An object containing a `DateTime` representing the date of the early close or late begin, a description of the partial holiday, and two `bool` properties, IsEarlyClose and IsLateBegin

Syntax
`TradingHours.PartialHolidays`

Examples

```
// Print all partial holidays included in the Bars object's Trading Hours template
foreach(KeyValuePair<DateTime, PartialHoliday> holiday in TradingHours.PartialHolidays)
{
    Print(holiday);
}
```

11.6.2.19.6 Sessions

Definition
A collection of session definitions of the configured Trading Hours template.

Available Properties

BeginDay	A `DayOfWeek` value representing the begin day
BeginTime	An `int` value representing the begin time
EndDay	A `DayOfWeek` value representing the end day
EndTime	An `int` value representing the end time

TradingDay	A DayOfWeek value representing the trading day this session belongs to

Syntax
```
Bars.TradingHours.Sessions[int idx]
```

> **Tip**: Each index value will represent a new defined session for the **Trading Hours** template. For example, accessing `Bars.TradingHours.Sessions[0]` would provide you with information for the first trading session configured in the **Trading Hours** template:
>
> ```
> Bars.TradingHours.Sessions[0].TradingDay = DayOfWeek.Monday,
> Bars.TradingHours.Sessions[1].TradingDay = DayOfWeek.Tuesday,
> Bars.TradingHours.Sessions[2].TradingDay = DayOfWeek.Wednesday, etc.
> ```

Examples
```
// Print details for all sessions in the Trading Hours template
for (int i = 0; i < TradingHours.Sessions.Count; i++)
{
    Print(String.Format("Session {0}: {1} at {2} to {3} at {4}", i,
 TradingHours.Sessions[i].BeginDay,
TradingHours.Sessions[i].BeginTime,
      TradingHours.Sessions[i].EndDay,
TradingHours.Sessions[i].EndTime));
}
```

11.6.2.19.7 TimeZoneInfo

Definition
Indicates a time zone that is configured by a **Trading Hours** template

Property Value
A TimeZoneInfo object the represents the time zone for a configured **Trading Hours** template.

Syntax
```
Bars.TradingHours.TimeZoneInfo
```

Examples

```
// Print the timezone before printing all sessions
Print(String.Format("All sessions are in {0}",
Bars.TradingHours.TimeZoneInfo));

// Print details for all sessions in the Trading Hours template
for (int i = 0; i < TradingHours.Sessions.Count; i++)
{
    Print(String.Format("Session {0}: {1} at {2} to {3} at {4}", i,
 TradingHours.Sessions[i].BeginDay,
TradingHours.Sessions[i].BeginTime,
      TradingHours.Sessions[i].EndDay,
TradingHours.Sessions[i].EndTime));
}
```

11.6.2.20 Clone()

Definition

Used to override the default NinjaScript Clone() method which is called any time an instance of a NinjaScript object is created. By default, the NinjaScript Clone() method will copy all the Property Info and Browsable Attributes to the new instance when the object is created (e.g., when an optimization is ran a new instance of the strategy will be created for each iteration). However it is possible to override this behavior if desired for custom development. There is no requirement to override the Clone behavior and this method will use the default constructor if not overridden.

> **Note**: This method is reserved for advanced developers who would like to change the default behavior when a NinjaScript object is created

Method Return Value

A virtual `object` representing the NinjaScript type.

Syntax

```
public override object Clone()
```

Parameters

This method does not take any parameters

Examples

```
public override object Clone()
{
   // custom logic to handle before the base clone

   return base.Clone();

   // custom logic to hand after the base clone
}
```

11.6.2.21 Description

Definition

Text which is used on the UI's information box to be displayed to a user when configuration a NinjaScript object.

Method Return Value

A `string` value representing text used to describe the object.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

`Description`

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";
      Description = @"An indicator used to demonstrate various
NinjaScript methods and properties";
   }
}
```

11.6.2.22 DisplayName

Definition

Determines the text display on the chart panel. This is also listed in the UI as the "Label" which can be manually changed (if not overridden). The default behavior of this property will include the NinjaScript type Name along with its input and data series parameters. However this behavior can be overridden if desired.

Note: For modifying the string which is used in the list of available indicators, please see the Name property.

Property Value

A virtual string. This property is read-only.

Syntax

DisplayName

You may choose to override this property using the following syntax:

```
public override string DisplayName
{
    get { }
}
```

Examples

Printing the default DisplayName which displays on the chart label

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Example Indicator";
   }
}


protected override void OnBarUpdate()
{
   Print(DisplayName);   //Output:  Example Indicator(ES 03-15 (1 Minute))
}
```

Overriding the DisplayName to customize the chart label

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Example Indicator";
   }
}

public override string DisplayName
{
    get { return "My Custom Display " + Name; }
}

protected override void OnBarUpdate()
{
   Print(DisplayName);   //Output:  My Custom Display Example
Indicator
}
```

11.6.2.23 IsVisible

Definition

Determines if the current NinjaScript object should be visible on the chart. When an object's IsVisible property is set to false, the object will NOT be displayed on the chart and will not be calculated to save resources.

Note: Strategies intentionally contain no **IsVisible** property.

Warning: This property should **NOT** be set on indicators which add a panel or own their own panel. Panel addition/removal is determined when an indicator is added/removed to a chart and is not modified through the IsVisible property.

Property Value

A `bool` value when **true** will be displayed on the chart; otherwise **false**; default value is **true**.

Syntax

`IsVisible`

Examples

```
protected override void OnBarUpdate()
{
   // Loops through the DrawObjects collection via a threadsafe list copy
   foreach (DrawingTool draw in DrawObjects.ToList())
   {
      // Detect all manual drawn line objects and change their visibility
      if (draw is DrawingTools.Line && draw.IsUserDrawn)
      {
         draw.IsVisible = false;
      }
   }
}
```

11.6.2.24 Name

Definition
Determines the listed name of the NinjaScript object.

Property Value
A `string` value.

Syntax
```
Name
```

Examples
```
protected override void OnStateChange()
{

   if (State == State.SetDefaults)
   {
      Name                    = "Examples indicator";
      Description             = @"An example of an indicator used for
documentation purposes";
   }
}
```

11.6.2.25 TriggerCustomEvent()

Definition
Provides a way to use your own custom events (such as a Timer object) so that internal NinjaScript indexes and pointers are correctly set prior to processing user code triggered by your custom event. When calling this event, NinjaTrader will synchronize all internal pointers and then call your custom event handler where your user code is located.

Note: The TriggerCustomEvent() method does **NOT** execute before **State.DataLoaded** or during or after **State.Terminated**. In effect, attempting to trigger custom events may be unavailable in some circumstances (e.g., while an indicator is terminating, or viewing the Strategy Analyzer chart display after backtest has completed, etc.)

Method Return Value

This method does not have a return value.

Syntax

```
TriggerCustomEvent(Action<object> customEvent, object state)
TriggerCustomEvent(Action<object> customEvent, int barsSeriesIndex, object state)
```

Parameters

barsIndex	Index of the bar series you want to synchronize to
customEvent	Delegate of your custom event method
state	Any object you want passed into your custom event method

Tips:

- There may be scenarios in which you need to set a Series<T> value outside of one of the core data event methods. In these cases, you can use **TriggerCustomEvent()** to reliably synchronize the barAgo indexer to the recent. current bar being updated. Please see the example below.
- Usually the correct approach is to use the WPF Dispatcher timer, however in cases where you need the timer to update a WinForms window it opened - please use the WinForms timer.

Examples

Using TriggerCustomEvent() in simple timer event

```
protected override void OnBarUpdate()
{
  // OnBarUpdate() only runs as bars are processed, which is not
guaranteed to occur at a specific interval
  // e.g., even on a 5 second bar series, there may be time periods
where there are no updates due to low trading activity
  // or could be buffered due to running Calculate.OnBarClose.
Instead of trying to obtain the Close[0] value
  // at some interval here, we are going to do it in our custom
TimerEventProcessor
}

// This is the method to run when the timer is raised.
private void TimerEventProcessor(Object myObject, EventArgs
myEventArgs)
{
  // Do not process your code here but instead call the
TriggerCustomEvent() method
  // and process your code in the custom handler method e.g., our
custom PrintThePrice()
  // Doing so ensures all internal indexers are up-to-date
  if (CurrentBar > 0)
  {
    TriggerCustomEvent(PrintThePrice, Close[0]);
  }
}

// Print the latest closing price with the current time
private void PrintThePrice(object price)
{
  Print("The Last Bar's Closing Value as of " +
NinjaTrader.Core.Globals.Now + " was " + price);
}

// Declare the WPF dispatcher timer
private System.Timers.Timer myTimer;

protected override void OnStateChange()
{
  if (State == State.SetDefaults)
  {
    Name = "SampleTriggerCustomEventTimer";
  }
  else if (State == State.DataLoaded)
  {
     // Instantiates the timer and sets the interval to 5 seconds.
     myTimer = new System.Timers.Timer(5000);
     // Adds the event handler for the method that will
     // process the timer event to the timer.
     myTimer.Elapsed += TimerEventProcessor;
     // Starts the timer
     myTimer.Enabled = true;
  }
```

Using TriggerCustomEvent to update a previously set custom Series<T> value

```
// using the virtual on render method for demonstration
// but concept could apply to any custom event that does not rely
on bars data
// e.g., from a custom mouse event or other 3rd party dependency
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // we want to reset a custom series values 20 barsAgo during
some condition
   if (conditionWhichRequiredUpdate)
   {
      // First, synchronize the index value used in via
Series[barsAgo]
      // to the NinjaScriptBase.CurrentBar being currently being
processed in OnBarUpdate
      TriggerCustomEvent(o =>
      {
         // For debugging, check the previous value
         Print("Before value which was set in OnBarUpdate(): " +
SomeVolumeData[20]);

         SomeVolumeData[20] = 5;  // set to our new custom value

         // For debugging, check the updated value
         Print("After value which was updated later in OnRender():
" + SomeVolumeData[20]);
      }, null);

      // reset our flag until we need to update a value again
      conditionWhichRequiredUpdate = false;
   }
   //Output:
   //Before value which was set in OnBarUpdate(): 1165
   //After value which was updated later in OnRender(): 5
}

private Series<double> SomeVolumeData; // custom Series for
tracking volume which will be modified through its lifetime
private bool conditionWhichRequiredUpdate = true;

protected override void OnStateChange()
{
   protected override void OnStateChange()
   {
      if (State == State.SetDefaults)
      {
         Name = "SampleUpdateCustomSeries";
      }

      else if (State == State.Historical)
      {
         SomeVolumeData = new Series<double>(this);
```

11.6.3 Add On

Custom Add Ons can be used to extend NinjaTrader's functionality. The methods and properties covered in this section are unique to custom Add On development.

For more information on the Add On development process please see this article.

NinjaTrader Controls	This section contains controls that are native NinjaTrader controls.
Account	The Account class can be used to subscribe to account related events as well as accessing account related information.
BarsRequest	BarsRequest can be used to request Bars data and subscribe to real-time Bars data events.
Connection	The Connection class can be used to monitor connection related events as well as accessing connection related information.
IInstrumentProvider Interface	When creating your NTTabPage, if you wish to use the instrument link, be sure to implement the IInstrumentProvider interface.
IIntervalProvider Interface	When creating your NTTabPage, if you wish to use the interval link, be sure to implement the IIntervalProvider interface.
INTTabFactory Interface	If you wish to have tab page functionality like adding, removing, moving, duplicating tabs you must create a class which implements the INTTabFactory interface.
IWorkspacePersistence Interface	When creating your NTWindow, be sure to implement the IWorkspacePersistence interface as well for the ability to save and restore your window with NinjaTrader workspaces.
NTTabPage Class	This is where the actual content for tabs inside the custom add on NTWindow can be defined.

Alert and Debug Concepts	In most scenarios you can use the NinjaScript provided methods for triggering alerts and debugging functionality. However, when building your own custom objects, you may find yourself wanting to use this functionality outside the NinjaScript scope.
AtmStrategy	AtmStrategy contains properties and methods used to manage ATM Strategies.
ControlCenter	ControlCenter is a XAML-defined class containing the layout and properties of the Control Center window.
FundamentalData	FundamentalData is used to access fundamental snapshot data and for subscribing to fundamental data events.
MarketData	MarketData can be used to access snapshot market data and for subscribing to market data events.
MarketDepth	MarketDepth can be used to access snapshot market depth and for subscribing to market depth events.
NewsItems	NewsItems can be used to store news articles.
NewsSubscription	NewsSubscription can be used for subscribing to News events.
NTMenuItem	NTMenuItem is used to create new menu entries.
NTWindow	The **NTWindow** class defines parent windows for custom window creation. Instances of NTWindow act as containers for instances of NTTabPage, in which UI elements and their related logic are contained.
NumericTextBox	NumericTextBox provides functionality for numeric text boxes to capture user input.

OnWindowCreated()	This method is called whenever a new NTWindow is created.
OnWindowDestroyed()	This method is called whenever a new NTWindow is destroyed.
OnWindowRestored()	This method is used to recall any custom XElement data from the workspace by referencing a window.
OnWindowSaved()	This method is used to save any custom XElement data associated with your window.
StartAtmStrategy()	StartAtmStrategy can be used to submit entry orders with ATM strategies.
StrategyBase	StrategyBase contains properties and methods for managing a Strategy object, and is the base class from which AtmStrategy derives.
PropagateInstrumentChange()	In an NTWindow, PropagateInstrumentChange() sends an Instrument to other windows with the same Instrument Linking color configured.
PropagateIntervalChange()	In an NTWindow, PropagateIntervalChange() sends an interval to other windows with the same Interval Linking color configured.
TabControl	The TabControl class provides functionality for working with NTTabPage objects within an NTWindow.
TabControlManager	The TabControlManager class can be used to set or check several properties of a TabControl object.

11.6.3.1 NinjaTrader Controls

The following section contains controls that are native NinjaTrader controls. To fully integrate your Add On within NinjaTrader it is recommended to use these controls as opposed to building your own when possible.

Note: For cleaning up these resources, please see the NTTabPage.Cleanup() method

AccountSelector	AccountSelector can be used as an UI element users can interact with for selecting accounts.
AtmStrategySelector	AtmStrategySelector is an UI element users can interact with for selecting ATM Strategies.
InstrumentSelector	InstrumentSelector is a UI element users can interact with for selecting instruments. This can be used with instrument linking between windows.
IntervalSelector	IntervalSelector is as a UI element users can interact with for selecting intervals. This can be used with interval linking between windows.
TifSelector	TifSelector can be used as an UI element users can interact with for selecting TIF.
QuantityUpDown	QuantityUpDown can be used as an UI element users can interact with for selecting quantity.

11.6.3.1.1 AccountSelector

Definition

AccountSelector can be used as an UI element users can interact with for selecting accounts.

Events and Properties

Cleanup()	Disposes of the AccountSelector (**Note**: calling the NTTabPage base.Cleanup() is sufficient to clean up this control)
SelectedAccount	Returns an Account representing the selected account
SelectionChanged	Event handler for when the selected account has changed

Examples

C#

```
/* Example of subscribing/unsubscribing to market data from an Add
On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private AccountSelector accountSelector

     public MyAddOnTab()
     {
          // Note: pageContent (not demonstrated in this example)
is the page content of the XAML
          // Find account selector
          accountSelector =
LogicalTreeHelper.FindLogicalNode(pageContent, "accountSelector")
as AccountSelector;

          // When the account selector's selection changes,
unsubscribe and resubscribe
          accountSelector.SelectionChanged += (o, args) =>
          {
               if (accountSelector.SelectedAccount != null)
               {
                    // Unsubscribe to any prior account
subscriptions

     accountSelector.SelectedAccount.AccountItemUpdate -=
OnAccountItemUpdate;
                    accountSelector.SelectedAccount.ExecutionUpdate
-= OnExecutionUpdate;
                    accountSelector.SelectedAccount.OrderUpdate -=
OnOrderUpdate;
                    accountSelector.SelectedAccount.PositionUpdate
-= OnPositionUpdate;

                    // Subscribe to new account subscriptions

     accountSelector.SelectedAccount.AccountItemUpdate   +=
OnAccountItemUpdate;
                    accountSelector.SelectedAccount.ExecutionUpdate
    += OnExecutionUpdate;
                    accountSelector.SelectedAccount.OrderUpdate
     += OnOrderUpdate;
                    accountSelector.SelectedAccount.PositionUpdate
     += OnPositionUpdate;
               }
          };
     }

     // Called by TabControl when tab is being removed or window is
closed
     public override void Cleanup()
     {
          // Clean up our resources
```

XAML

```
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:Tools="clr-
namespace:NinjaTrader.Gui.Tools;assembly=NinjaTrader.Gui"
   xmlns:AccountPerformance="clr-
namespace:NinjaTrader.Gui.AccountPerformance;assembly=NinjaTrader.G
ui"
   xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui"
   xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=NinjaTra
der.Gui">

<Grid>
     <Tools:AccountSelector x:Name="accountSelector"
HorizontalAlignment="Left" VerticalAlignment="Top"/>
</Grid>
```

11.6.3.1.2 AtmStrategySelector

Definition

AtmStrategySelector is an UI element users can interact with for selecting ATM Strategies.

Events and Properties

Cleanup()	Disposes of the AtmStrategySelector (**Note**: calling the NTTabPage base.Cleanup() is sufficient to clean up this control)
CustomProperties Changed	Event handler for when properties have changed on the ATM strategy
Id	A string identifying the ATM Strategy selector
SelectedAtmStrategy	Returns an AtmStrategy representing the selected ATM strategy
SelectionChanged	Event handler for when the selected ATM strategy has changed

Examples

This example demonstrates how to use the ATM strategy selector and properly link its behavior with the quantity up/down and TIF selectors.

Examples

C#

```
private QuantityUpDown                  qudSelector;
private TifSelector                     tifSelector;
private AtmStrategy.AtmStrategySelector atmStrategySelector;

private DependencyObject LoadXAML()
{
     // Note: pageContent (not demonstrated in this example) is the
page content of the XAML
     // Find the Quantity Up-Down selector
     qudSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"qudSelector") as QuantityUpDown;

     // Find the TIF selector
     tifSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"tifSelector") as TifSelector;

     // Be sure to bind our account selector to our TIF selector to
ensure proper functionality
     tifSelector.SetBinding(TifSelector.AccountProperty, new
Binding { Source = accountSelector,
          Path = new PropertyPath("SelectedAccount") });

     // When our TIF selector's selection changes
     tifSelector.SelectionChanged += (o, args) =>
     {
          // Change the selected TIF in the ATM strategy too
          if (atmStrategySelector.SelectedAtmStrategy != null)
               atmStrategySelector.SelectedAtmStrategy.TimeInForce
= tifSelector.SelectedTif;
     };

     // Find ATM Strategy selector and attach event handler
     atmStrategySelector =
LogicalTreeHelper.FindLogicalNode(pageContent,
"atmStrategySelector") as AtmStrategy.AtmStrategySelector;
     atmStrategySelector.Id = Guid.NewGuid().ToString("N");
     if (atmStrategySelector != null)
          atmStrategySelector.CustomPropertiesChanged +=
OnAtmCustomPropertiesChanged;

     // Be sure to bind our account selector to our ATM strategy
selector to ensure proper functionality
     atmStrategySelector.SetBinding(AtmStrategy.AtmStrategySelector
.AccountProperty,
          new Binding { Source = accountSelector, Path = new
PropertyPath("SelectedAccount") });

     // When our ATM selector's selection changes
     atmStrategySelector.SelectionChanged += (o, args) =>
     {
          if (atmStrategySelector.SelectedItem == null)
               return;
          if (args.AddedItems.Count > 0)
```

XAML

```
<AtmStrategy:AtmStrategySelector x:Name="atmStrategySelector"
LinkedQuantity="{Binding ElementName=qudSelector, Path=Value,
Mode=OneWay}" Grid.Row="12" Grid.Column="2">
      <AtmStrategy:AtmStrategySelector.Margin>
            <Thickness Left="{StaticResource MarginButtonLeft}"
Top="{StaticResource MarginControl}" Right="{StaticResource
MarginBase}" Bottom="0" />
      </AtmStrategy:AtmStrategySelector.Margin>
</AtmStrategy:AtmStrategySelector>
```

11.6.3.1.3 InstrumentSelector

Definition

InstrumentSelector is a UI element users can interact with for selecting instruments. This can be used with instrument linking between windows.

Events and Properties

Cleanup()	Disposes of the InstrumentSelector (**Note**: calling the NTTabPage base.Cleanup() is sufficient to clean up this control)
Instrument	An Instrument representing the selected instrument
InstrumentChanged	Event handler for when the instrument changes on the instrument selector

Examples

This example demonstrates how to use the instrument selector and properly link its behavior to windows linking.

C#

```
private InstrumentSelector instrumentSelector;

private DependencyObject LoadXAML()
{
     // Note: pageContent (not demonstrated in this example) is the
page content of the XAML

     // Find the Instrument selector
```

C#

```
     instrumentSelector =
LogicalTreeHelper.FindLogicalNode(pageContent,
"instrumentSelector") as InstrumentSelector;
     if (instrumentSelector != null)
          instrumentSelector.InstrumentChanged +=
OnInstrumentChanged;
}

// This method is fired when our instrument selector changes
instruments
private void OnInstrumentChanged(object sender, EventArgs e)
{
     Instrument = sender as Cbi.Instrument;
}

// IInstrumentProvider member. Required if you want to use the
instrument link mechanism in this Add On window
public Cbi.Instrument Instrument
{
     get { return instrument }
     set
     {
          instrument = value;
          if (instrumentSelector != null)
               instrumentSelector.Instrument = value;

          // Send instrument to other windows linked to the same
color
          PropagateInstrumentChange(value);
     }
}

// NOTE: Don't forget to clean up resources and unsubscribe to
events
// Called by TabControl when tab is being removed or window is
closed
public override void Cleanup()
{
     // Clean up our resources
     if (instrumentSelector != null)
     {
          instrumentSelector.InstrumentChanged -=
OnInstrumentChanged;
     }
     base.Cleanup();
}
```

XAML

```
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:Tools="clr-
namespace:NinjaTrader.Gui.Tools;assembly=NinjaTrader.Gui"
       xmlns:AccountPerformance="clr-
namespace:NinjaTrader.Gui.AccountPerformance;assembly=NinjaTrader.G
ui"
       xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui"
       xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=NinjaTra
der.Gui">

<Grid>
     <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition Width="*"/>
     </Grid.ColumnDefinitions>

     <Tools:InstrumentSelector x:Name="instrumentSelector"
Grid.Column="0" LastUsedGroup="MyAddOn"/>
</Grid>
```

11.6.3.1.4 IntervalSelector

Definition

IntervalSelector is as a UI element users can interact with for selecting intervals. This can be used with interval linking between windows.

Events and Properties

Cleanup()	Disposes of the IntervalSelector (**Note**: calling the NTTabPage base.Cleanup() is sufficient to clean up this control)
Interval	A BarsPeriod representing the interval currently selected
IntervalChanged	Event handler for when the interval changed

Examples

This example demonstrates how to use the interval selector and properly link its behavior to windows linking.

C#

```
private IntervalSelector   intervalSelector;

private DependencyObject LoadXAML()
{
     // Note: pageContent (not demonstrated in this example) is the
page content of the XAML

     // Find the Interval selector
     intervalSelector =
LogicalTreeHelper.FindLogicalNode(pageContent, "intervalSelector")
as IntervalSelector;
     if (intervalSelector != null)
          intervalSelector.IntervalChanged += OnIntervalChanged;

}

// This method is fired when our interval selector changes
intervals
private void OnIntervalChanged(object sender, BarsPeriodEventArgs
e)
{
     if (e.BarsPeriod == null)
          return;
}

/* IIntervalProvider member. Required if you want to use the
interval linker mechanism on this window.
No functionality has been linked to the interval linker in this
sample. */
public BarsPeriod BarsPeriod { get; set; }

// NOTE: Don't forget to clean up resources and unsubscribe to
events
// Called by TabControl when tab is being removed or window is
closed
public override void Cleanup()
{
     // Clean up our resources
     if (intervalSelector != null)
          intervalSelector.IntervalChanged -= OnIntervalChanged;

     base.Cleanup();
}
```

XAML

```
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:Tools="clr-
namespace:NinjaTrader.Gui.Tools;assembly=NinjaTrader.Gui"
       xmlns:AccountPerformance="clr-
namespace:NinjaTrader.Gui.AccountPerformance;assembly=NinjaTrader.G
ui"
       xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui"
       xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=NinjaTra
der.Gui">

<Grid>
     <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition Width="*"/>
     </Grid.ColumnDefinitions>

     <Tools:IntervalSelector x:Name="intervalSelector"
Grid.Column="0" HorizontalAlignment="Left"/>
</Grid>
```

11.6.3.1.5 TifSelector

Definition

TifSelector can be used as an UI element users can interact with for selecting TIF.

Events and Properties

Cleanup()	Disposes of the TifSelector (**Note**: calling the NTTabPage base.Cleanup() is sufficient to clean up this control)
SelectedTif	A TimeInForce representing the selected TIF Possible values: TimeInForce.Day TimeInForce.Gtc TimeInForce.Gtd TimeInForce.Ioc TimeInForce.Opg

SelectionChanged	Event handler for when the selected ATM strategy has changed

Examples

This example demonstrates how to use the TIF selector and properly link its behavior with the quantity up/down and TIF selectors.

C#

```
private QuantityUpDown                  qudSelector;
private TifSelector                     tifSelector;
private AtmStrategy.AtmStrategySelector atmStrategySelector;

private DependencyObject LoadXAML()
{
     // Note: pageContent (not demonstrated in this example) is the
page content of the XAML
     // Find the Quantity Up-Down selector
     qudSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"qudSelector") as QuantityUpDown;

     // Find the TIF selector
     tifSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"tifSelector") as TifSelector;

     // Be sure to bind our account selector to our TIF selector to
ensure proper functionality
     tifSelector.SetBinding(TifSelector.AccountProperty, new
Binding { Source = accountSelector,
          Path = new PropertyPath("SelectedAccount") });

     // When our TIF selector's selection changes
     tifSelector.SelectionChanged += (o, args) =>
     {
          // Change the selected TIF in the ATM strategy too
          if (atmStrategySelector.SelectedAtmStrategy != null)
               atmStrategySelector.SelectedAtmStrategy.TimeInForce
= tifSelector.SelectedTif;
     };

     // Find ATM Strategy selector and attach event handler
     atmStrategySelector =
LogicalTreeHelper.FindLogicalNode(pageContent,
"atmStrategySelector") as AtmStrategy.AtmStrategySelector;
     atmStrategySelector.Id = Guid.NewGuid().ToString("N");
     if (atmStrategySelector != null)
          atmStrategySelector.CustomPropertiesChanged +=
OnAtmCustomPropertiesChanged;

     // Be sure to bind our account selector to our ATM strategy
selector to ensure proper functionality
     atmStrategySelector.SetBinding(AtmStrategy.AtmStrategySelector
.AccountProperty,
          new Binding { Source = accountSelector, Path = new
PropertyPath("SelectedAccount") });

     // When our ATM selector's selection changes
     atmStrategySelector.SelectionChanged += (o, args) =>
     {
          if (atmStrategySelector.SelectedItem == null)
               return;
          if (args.AddedItems.Count > 0)
```

XAML

```
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:Tools="clr-
namespace:NinjaTrader.Gui.Tools;assembly=NinjaTrader.Gui"
       xmlns:AccountPerformance="clr-
namespace:NinjaTrader.Gui.AccountPerformance;assembly=NinjaTrader.G
ui"
       xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui"
       xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=NinjaTra
der.Gui">

<Grid>
     <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition Width="*"/>
     </Grid.ColumnDefinitions>

     <Tools:QuantityUpDown x:Name="qudSelector" Value="1"
Grid.Column="0"/>
     <Tools:TifSelector x:Name="tifSelector" Grid.Column="1"/>
     <AtmStrategy:AtmStrategySelector x:Name="atmStrategySelector"
LinkedQuantity="{Binding Value,
     ElementName=qudSelector, Mode=OneWay}" Grid.Column="2"/>
</Grid>
```

11.6.3.1.6 QuantityUpDown

Definition

QuantityUpDown can be used as an UI element users can interact with for selecting quantity.

Events and Properties

Value	An int representing the quantity

Examples

This example demonstrates how to use the quantity up/down selector and properly link its behavior with the ATM strategy and TIF selectors.

C#

```
private QuantityUpDown                  qudSelector;
private TifSelector                     tifSelector;
private AtmStrategy.AtmStrategySelector atmStrategySelector;

private DependencyObject LoadXAML()
{
     // Note: pageContent (not demonstrated in this example) is the
page content of the XAML
     // Find the Quantity Up-Down selector
     qudSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"qudSelector") as QuantityUpDown;

     // Find the TIF selector
     tifSelector = LogicalTreeHelper.FindLogicalNode(pageContent,
"tifSelector") as TifSelector;

     // Be sure to bind our account selector to our TIF selector to
ensure proper functionality
     tifSelector.SetBinding(TifSelector.AccountProperty, new
Binding { Source = accountSelector,
          Path = new PropertyPath("SelectedAccount") });

     // When our TIF selector's selection changes
     tifSelector.SelectionChanged += (o, args) =>
     {
          // Change the selected TIF in the ATM strategy too
          if (atmStrategySelector.SelectedAtmStrategy != null)
               atmStrategySelector.SelectedAtmStrategy.TimeInForce
= tifSelector.SelectedTif;
     };

     // Find ATM Strategy selector and attach event handler
     atmStrategySelector =
LogicalTreeHelper.FindLogicalNode(pageContent,
"atmStrategySelector") as AtmStrategy.AtmStrategySelector;
     atmStrategySelector.Id = Guid.NewGuid().ToString("N");
     if (atmStrategySelector != null)
          atmStrategySelector.CustomPropertiesChanged +=
OnAtmCustomPropertiesChanged;

     // Be sure to bind our account selector to our ATM strategy
selector to ensure proper functionality
     atmStrategySelector.SetBinding(AtmStrategy.AtmStrategySelector
.AccountProperty,
          new Binding { Source = accountSelector, Path = new
PropertyPath("SelectedAccount") });
```

C#

```
     // When our ATM selector's selection changes
     atmStrategySelector.SelectionChanged += (o, args) =>
     {
          if (atmStrategySelector.SelectedItem == null)
               return;
          if (args.AddedItems.Count > 0)
          {
               // Change the selected TIF in our TIF selector too
               AtmStrategy selectedAtmStrategy = args.AddedItems[0]
as AtmStrategy;
               if (selectedAtmStrategy != null)
                    tifSelector.SelectedTif =
selectedAtmStrategy.TimeInForce;
             }
       };

}

private void OnAtmCustomPropertiesChanged(object sender,
NinjaScript.AtmStrategy.CustomPropertiesChangedEventArgs args)
{
     // Adjust our TIF and Quantity selectors to the new ATM
strategy values
     tifSelector.SelectedTif = args.NewTif;
     qudSelector.Value = args.NewQuantity;
}

// NOTE: Don't forget to clean up resources and unsubscribe to
events
// Called by TabControl when tab is being removed or window is
closed
public override void Cleanup()
{
     // Clean up our resources
     base.Cleanup();
}
```

XAML

```
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:Tools="clr-
namespace:NinjaTrader.Gui.Tools;assembly=NinjaTrader.Gui"
       xmlns:AccountPerformance="clr-
namespace:NinjaTrader.Gui.AccountPerformance;assembly=NinjaTrader.G
ui"
       xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui"
       xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=NinjaTra
der.Gui">

<Grid>
     <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition Width="*"/>
     </Grid.ColumnDefinitions>

     <Tools:QuantityUpDown x:Name="qudSelector" Value="1"
Grid.Column="0"/>
     <Tools:TifSelector x:Name="tifSelector" Grid.Column="1">
     <AtmStrategy:AtmStrategySelector x:Name="atmStrategySelector"
LinkedQuantity="{Binding Value,
     ElementName=qudSelector, Mode=OneWay}" Grid.Column="2"/>
</Grid>
```

11.6.3.2 Account

Definition

The Account class can be used to subscribe to account related events as well as accessing account related information.

Static Account Class Properties

All	A collection of Account objects
AccountStatusUpdate	Event handler for account status updates
SimulationAccountReset	Event handler for resets on sim accounts

	NOTE: Also happens when rewinding/fast forwarding Playback connections)

Methods and Properties From Account instances

AccountItem	Represents various account variables used to reflect values the status of the account
AccountItemUpdate	Event handler for changes to account values
Cancel()	Cancels specified order(s) on the account
CancelAllOrders()	Cancels all orders of an instrument on the account
Change()	Changes specified order(s) on the account
Connection	A Connection representing the connection this account is associated with
CreateOrder()	Creates orders for the account that need to be submitted via Submit()
Denomination	A `Currency` representing the denomination currency of this connection
Executions	A collection of executions on this account
ExecutionUpdate	Event handler for when new executions come in, an existing execution is amended, or an execution is removed
Flatten()	Flattens the account on specified instrument(s)
Get()	Returns the value of an AccountItem
Name	A string representing the name of this account
Orders	A collection of orders on this account

OrderUpdate	Event handler for changes to orders
Positions	A collection of positions on this account
PositionUpdate	Event handler for changes to positions
Strategies	A collection of strategies on this account
Submit()	Submits specified order(s)

Example

```
private Account myAccount;

protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Find our Sim101 account
          lock (Account.All)
               myAccount = Account.All.FirstOrDefault(a => a.Name
== "Sim101");

          // Subscribe to static events. Remember to unsubscribe
with -= when you are done
          Account.AccountStatusUpdate += OnAccountStatusUpdate;

          if (myAccount != null)
          {
               // Print some information about our account using
the AccountItem indexer
               Print(string.Format("Account Name: {0} Connection
Name: {1} Cash Value {2}",
                    myAccount.Name,
                    myAccount.Connection.Options.Name,
                    myAccount.Get(AccountItem.CashValue,
Currency.UsDollar)
                    ));

               // Print the prices of the executions on our account
               lock (myAccount.Executions)
                    foreach (Execution execution in
myAccount.Executions)
                         Print("Price: " + execution.Price);

               // Subscribe to events. Remember to unsubscribe with
-= when you are done
               myAccount.AccountItemUpdate += OnAccountItemUpdate;
               myAccount.ExecutionUpdate += OnExecutionUpdate;
          }
     }
     else if (State == State.Terminated)
     {
          // Unsubscribe to events
          myAccount.AccountItemUpdate -= OnAccountItemUpdate;
          myAccount.ExecutionUpdate -= OnExecutionUpdate;
          Account.AccountStatusUpdate -= OnAccountStatusUpdate;
     }
}

private void OnAccountStatusUpdate(object sender,
AccountStatusEventArgs e)
{
     // Do something with the account status update
}
```

11.6.3.2.1 AccountItem

Definition

Represents various account variables used to reflect values the status of the account. Each account connected in NinjaTrader will have it's own unique AccountItem values.

Tip: For strategies, see also OnAccountItemUpdate(). For other objects, you can also subscribe to the AccountItemUpdate stream.

Syntax

```
AccountItem
```

Parameters

AccountItem.BuyingPower
AccountItem.CashValue
AccountItem.Commission
AccountItem.ExcessIntradayMargin
AccountItem.ExcessInitialMargin
AccountItem.ExcessMaintenanceMargin
AccountItem.ExcessPositionMargin
AccountItem.Fee
AccountItem.GrossRealizedProfitLoss
AccountItem.InitialMargin
AccountItem.IntradayMargin
AccountItem.LongOptionValue
AccountItem.LookAheadMaintenanceMargin
AccountItem.LongStockValue

AccountItem.MaintenanceMargin
AccountItem.NetLiquidation
AccountItem.NetLiquidationByCurrency
AccountItem.PositionMargin
AccountItem.RealizedProfitLoss
AccountItem.ShortOptionValue
AccountItem.ShortStockValue
AccountItem.SodCashValue
AccountItem.SodLiquidatingValue
AccountItem.UnrealizedProfitLoss
AccountItem.TotalCashBalance

11.6.3.2.2 AccountItemUpdate

Definition

AccountItemUpdate is used for subscribing to account item update events.

> **Note**: Remember to unsubscribe if you are no longer using the subscription.

Syntax

```
AccountItemUpdate
```

Example

```
/* Example of subscribing/unsubscribing to account item update
events from an Add On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private Account account;
     public MyAddOnTab()
     {
          // Find our Sim101 account
          lock (Account.All)
               account = Account.All.FirstOrDefault(a => a.Name ==
"Sim101");

          // Subscribe to account item updates
          if (account != null)
               account.AccountItemUpdate += OnAccountItemUpdate;
     }

     // This method is fired on any change of an account value
     private void OnAccountItemUpdate(object sender,
AccountItemEventArgs e)
     {
          // Output the account item
          NinjaTrader.Code.Output.Process(string.Format("Account:
{0} AccountItem: {1} Value: {2}",
               e.Account.Name, e.AccountItem, e.Value),
PrintTo.OutputTab1);
     }

     // Called by TabControl when tab is being removed or window is
closed
     public override void Cleanup()
     {
          // Make sure to unsubscribe to the account item
subscription
          if (account != null)
               account.AccountItemUpdate -= OnAccountItemUpdate;
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.2.3 AccountStatusUpdate

Definition

AccountStatusUpdate can be used for subscribing to account status events from all accounts.

Note: Remember to unsubscribe if you are no longer using the subscription.

Syntax

AccountStatusUpdate

Examples

```
/* Example of subscribing/unsubscribing to account status update
events from an Add On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     public MyAddOnTab()
     {
          // Subscribe to account status updates
          Account.AccountStatusUpdate += OnAccountStatusUpdate;
     }

     // This method is fired on any status change of any account
     private void OnAccountStatusUpdate(object sender,
AccountStatusEventArgs e)
     {
          // Output the account name and status
          NinjaTrader.Code.Output.Process(string.Format("Account:
{0} Status: {1}",
               e.Account.Name, e.Status), PrintTo.OutputTab1);
     }

     // Called by TabControl when tab is being removed or window is
closed
     public override void Cleanup()
     {
          // Make sure to unsubscribe to the account status
subscription
          Account.AccountStatusUpdate -= OnAccountStatusUpdate;
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.2.4 All

Definition

A collection of Account objects

Property Value
A Collection of Account objects

Syntax
Accounts.All

Examples

```
protected override void OnStateChange()
{
    if (State == State.DataLoaded)
    {
       foreach (Account sampleAccount in Account.All)
          Print(String.Format("The account {0} has a {1} unit FX
lotsize set", sampleAccount.Name, sampleAccount.ForexLotSize));
    }
}
```

11.6.3.2.5 Cancel()

Definition
Cancels specified Order object(s).

Syntax
Cancel(IEnumerable<Order> *orders*)

Parameters

orders	Order(s) to cancel

Examples

```
private Account myAccount;
Order stopOrder = null;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Initialize myAccount
    }
}

private void OnExecutionUpdate(object sender, ExecutionEventArgs e)
{
    // Cancel the stop order if an execution results in a long
position
    if(e.MarketPosition == MarketPosition.Long)
        myAccount.Cancel(new[] { stopOrder });
}
```

11.6.3.2.6 CancelAllOrders()

Definition

Cancels all Orders of an instrument.

Syntax

```
CancelAllOrders(Instrument instrument)
```

Parameters

instrument	Instrument of the orders to be cancelled

Example

```
private Account myAccount;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Initialize myAccount
    }
}

private void OnExecutionUpdate(object sender, ExecutionEventArgs e)
{
    // Cancel all orders if an execution is triggered after 9pm
    if (e.Time > new DateTime(now.Year, now.Month, now.Day, 21, 0,
0))
        myAccount.CancelAllOrders(e.Execution.Instrument);
}
```

11.6.3.2.7 Change()

Definition

Changes specified Order object(s).

Syntax

```
Change(IEnumerable<Order> orders)
```

Parameters

orders	Order(s) to change

Example

```
Order stopOrder;
stopOrder.StopPriceChanged = stopOrder.StopPrice - 4 *
stopOrder.Instrument.MasterInstrument.TickSize;

private void OnExecutionUpdate(object sender, ExecutionEventArgs e)
{
    // Change the stop order if an execution results in a long
position
    if(e.MarketPosition == MarketPosition.Long)
        myAccount.Change(new[] { stopOrder });
}
```

11.6.3.2.8 Connection

Definition
Indicates the data connection used for the specified account.

Property Value
An instance of the `Connection` class containing information about the connection used for a specified account

Syntax
`<Account>.Connection`

Examples

```
private Account myAccount;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        myAccount = Account.All.FirstOrDefault(a => a.Name ==
"Sim101");
    }
}

private void OnAccountStatusUpdate(object sender,
AccountStatusEventArgs e)
{
    Print(String.Format("{0} connection updated",
myAccount.Connection.Options.Name));
}
```

11.6.3.2.9 ConnectOptions

Definition
ConnectOptions is an abstract class used to configure options for a specific configured Connection. An instance of ConnectOptions can be passed into the Connection.Connect() method to initiate a connection, as seen in the example below.

> **Note**: For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

Properties accessible from an instance of ConnectOptions include:

BrandName	A string representing the provider name
CanEnableHds	A bool determining the connection can use NinjaTrader Historical Data Servers. Related properties include HasHdsAlwaysEnabled and IsHdsEnabled
CanManageOrders	A bool determining orders can be managed on the Connection. Related properties include IsDataProviderOnly
Mode	A NinjaTrader.Cbi.Mode object representing the current mode of the connection (Mode.Live or Mode.Simulation)
Name	The user-configured name of the Connection
Provider	The provider configured in the Connection

Examples

```
// Connecting to a configured connection
private Connection Connect(string connectionName)
{
    // Get the configured account connection by using the string
passed into this custom Connect() method
    // We will lock the ConnectOptions collection to avoid in-
flight changes causing any issues
    ConnectOptions connectOptions = null;
    lock (Core.Globals.ConnectOptions)
        connectOptions =
Core.Globals.ConnectOptions.FirstOrDefault(o => o.Name ==
connectionName);

    // If connection is not already connected, connect to it
    lock (Connection.Connections)
        if (Connection.Connections.FirstOrDefault(c =>
c.Options.Name == connectionName) == null)
        {
            Connection connect =
Connection.Connect(connectOptions);

            // Only return connection if successfully connected
            if (connect.Status == ConnectionStatus.Connected)
                return connect;
            else
                return null;
        }
}
```

11.6.3.2.10 CreateOrder()

Definition

Creates an Order to be submitted via Submit().

Syntax

`CreateOrder(Instrument instrument, OrderAction action, OrderType orderType, OrderEntry orderEntry, TimeInForce timeInForce, int quantity, double limitPrice, double stopPrice, string oco, string name, DateTime gtd, CustomOrder customOrder)`

Parameters

instrument	Order instrument
orderAction	Possible values: OrderAction.Buy

	OrderAction.BuyToCover OrderAction.Sell OrderAction.SellShort
orderType	Possible values: OrderType.Limit OrderType.Market OrderType.MIT OrderType.StopMarket OrderType.StopLimit
orderEntry	Possible values: OrderEntry.Automated OrderEntry.Manual Allows setting the tag for orders submitted manually or via automated trading logic (CME tag 1028).
timeInForce	Possible values: TimeInForce.Day TimeInForce.Gtc TimeInForce.Gtd TimeInForce.Ioc TimeInForce.Opg
quantity	Order quantity
limitPrice	Order limit price. Use "0" should this parameter be irrelevant for the OrderType being submitted.
stopPrice	Order stop price. Use "0" should this parameter be irrelevant for the OrderType being submitted.
oco	A string representing the OCO ID used to link OCO orders together
name	A string representing the name of the order. Max 50 characters.

	Note: If using ATM Strategy StartAtmStrategy(), this value **MUST** be "Entry"
gtd	A `DateTime` value to be used with TimeInForce.Gtd - for all other cases you can pass in `Core.Globals.MaxDate`
customOrder	Custom order if it is being used

Examples

```
Order stopOrder;
stopOrder = myAccount.CreateOrder(myInstrument, OrderAction.Sell,
OrderType.StopMarket, OrderEntry.Automated, TimeInForce.Day, 1, 0,
1400, "myOCO", "stopOrder", Core.Globals.MaxDate, null);

myAccount.Submit(new[] { stopOrder });
```

11.6.3.2.11 Denomination

Definition

Indicates the currency set on an account

Property Value

A Currency object containing information about the currency denomination specified in the referenced account

Syntax

`<Account>.Connection`

Examples

```
private Account myAccount;

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      // Initialize myAccount here

      // Print myAccount's currency denomination
      NinjaTrader.Code.Output.Process("myAccount currency is set to
" + myAccount.Denomination, PrintTo.OutputTab1);
   }
}
```

11.6.3.2.12 Executions

Definition

A collection of `Execution` objects generated for the specified account. These are the current sessions executions and should match executions reported in the Executions tab of the NinjaTrader Account Data window.

Property Value

An Collection of `Execution` objects

Syntax

`<Account>.Executions`

> **Note**: At this time there is not a supported method to retrieve historical executions from the local database.

Examples

```
private Account myAccount;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Initialize myAccount
    }
}

private void OnExecutionUpdate(object sender, ExecutionEventArgs e)
{
    foreach (Execution execution in myAccount.Executions)
    {
        Print(String.Format("Execution triggered for Order {0}",
execution.Order));
    }
}
```

11.6.3.2.13 ExecutionUpdate

Definition

ExecutionUpdate is used for subscribing to execution update events.

> **Note**: Remember to unsubscribe if you are no longer using the subscription.

Syntax

`ExecutionUpdate`

Examples

```
/* Example of subscribing/unsubscribing to execution update events
from an Add On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private Account account;
     public MyAddOnTab()
     {
          // Find our Sim101 account
          lock (Account.All)
               account = Account.All.FirstOrDefault(a => a.Name ==
"Sim101");

          // Subscribe to execution updates
          if (account != null)
               account.ExecutionUpdate += OnExecutionUpdate;
     }

     /* This method is fired as new executions come in, an existing
execution is amended
     (e.g. by the broker's back office), or an execution is removed
(e.g. by the broker's back office) */
     private void OnExecutionUpdate(object sender,
ExecutionEventArgs e)
     {
          // Output the execution
          NinjaTrader.Code.Output.Process(string.Format("Instrument
: {0} Quantity: {1} Price: {2}",
               e.Execution.Instrument.FullName, e.Quantity,
e.Price), PrintTo.OutputTab1);
     }

     // Called by TabControl when tab is being removed or window is
closed
     public override void Cleanup()
     {
          // Make sure to unsubscribe to the execution subscription
          if (account != null)
               account.ExecutionUpdate -= OnExecutionUpdate;
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.2.14 Flatten()

Definition

Flattens the account on an instrument.

Syntax

```
Flatten(ICollection<Instrument> instruments)
```

Parameters

instruments	A collection of Instruments for orders to be cancelled and positions closed

Examples

Flatten a single instrument

```
Account.Flatten(new [] { Instrument.GetInstrument("ES 12-15") });
```

Flatten a list of instruments

```
// Please note that your 'Using declarations' section needs to
have
//
// using System.Collections.ObjectModel;
//
// added in order for this example to compile correctly

// instantiate a list of instruments
Collection<Cbi.Instrument> instrumentsToClose = new
Collection<Instrument>();

// add instruments to the collection
instrumentsToClose.Add(Instrument.GetInstrument("AAPL"));
instrumentsToClose.Add(Instrument.GetInstrument("MSFT"));

// pass the instrument collection to the Flatten() method to be
flattened
Account.Flatten(instrumentsToClose);
```

11.6.3.2.15 Get()

Definition

Returns the value of an AccountItem, such as BuyingPower, CashValue, etc.

Method Return Value

A `double` representing the value of the requested AccountItem

Syntax
```
Get(AccountItem itemType, Cbi.Currency currency)
```

Parameters

itemType	The desired AccountItem to return
Currency	The account currency the value should be denoted (required parameter, but has no effect on returned value)

Examples

```
// Evaluates to see if the account has more than $25000
if (Account.Get(AccountItem.CashValue, Currency.UsDollar) > 25000)
{
   // Do something;
}
```

11.6.3.2.16 Name

Definition
Indicates the name of the specified account

Property Value
An `string` representing the name of the account

Syntax
```
<Account>.Name
```

Example

```
private Account myAccount;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Initialize myAccount
    }
}

private void OnAccountStatusUpdate(object sender,
AccountStatusEventArgs e)
{
    // Print the name of each account updated
    Print(String.Format("{0} account updated", myAccount.Name));
}
```

11.6.3.2.17 Orders

Definition

A collection of `Order` objects generated for the specified account

Property Value

An Collection of `Order` objects

> **Note**: Please keep in mind that orders placed when in State.Historical are not submitted live to an account.

Syntax

`<Account>.Orders`

Examples

```
private Account myAccount;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Initialize myAccount
    }
}

private void OnAccountItemUpdate(object sender,
AccountItemEventArgs e)
{
    // Print the name and order action of each order processed on
the account
    foreach (Order order in myAccount.Orders)
    {
        Print(String.Format("Order placed: {0} - {1}", order.Name,
order.OrderAction));
    }
}
```

11.6.3.2.18 OrderUpdate

Definition

OrderUpdate can be used for subscribing to order update events.

Note: Remember to unsubscribe if you are no longer using the subscription.

Syntax

OrderUpdate

Examples

```
/* Example of subscribing/unsubscribing to order update events from
an Add On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private Account account;
     private Order myEntryOrder;
     private Order profitTarget;
     private Order stopLoss;

     public MyAddOnTab()
```

```
    {
        // Find our Sim101 account
        lock (Account.All)
            account = Account.All.FirstOrDefault(a => a.Name ==
"Sim101");

        // Subscribe to order updates
        if (account != null)
            account.OrderUpdate += OnOrderUpdate;
    }

    // This method is fired as the status of an order changes
    private void OnOrderUpdate(object sender, OrderEventArgs e)
    {
        // Submit stop/target bracket orders
        if (myEntryOrder != null && myEntryOrder == e.Order)
        {
            if (e.OrderState == OrderState.Filled)
            {
                string oco = Guid.NewGuid().ToString("N");

                profitTarget =
account.CreateOrder(e.Order.Instrument, OrderAction.Sell,
OrderType.Limit, OrderEntry.Manual, TimeInForce.Day,
                    e.Quantity, e.AverageFillPrice + 10 *
e.Order.Instrument.MasterInstrument.TickSize, 0, oco, "Profit
Target", Core.Globals.MaxDate, null);
                stopLoss     =
account.CreateOrder(e.Order.Instrument, OrderAction.Sell,
OrderType.StopMarket, OrderEntry.Manual, TimeInForce.Day,
                    e.Quantity, 0, e.AverageFillPrice - 10 *
e.Order.Instrument.MasterInstrument.TickSize, oco, "Stop Loss",
Core.Globals.MaxDate, null);
                account.Submit(new[] { profitTarget,
stopLoss });
            }
        }
    }

    // Called by TabControl when tab is being removed or window is
closed
    public override void Cleanup()
    {
        // Make sure to unsubscribe to the orders subscription
        if (account != null)
            account.OrderUpdate -= OnOrderUpdate;
    }
```

```
    // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.2.19 Positions

Definition

A collection of `Position` objects generated for the specified account

Property Value

An Collection of `Position` objects

Syntax

```
Account.Positions
<Account>.Positions
```

Examples

```
private Account myAccount;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Find our Sim101 account
        lock (Account.All)
            myAccount = Account.All.FirstOrDefault(a => a.Name ==
"Sim101");
    }

    if (State == State.DataLoaded)
    {
        lock (myAccount.Positions)
        {
            Print("Positions in State.DataLoaded:");

            foreach (Position position in myAccount.Positions)
            {
                Print(String.Format("Position: {0} at {1}",
position.MarketPosition, position.AveragePrice));
            }
        }
    }
}
```

11.6.3.2.20 PositionUpdate

Definition

PositionUpdate can be used for subscribing to position update events.

Note: Remember to unsubscribe if you are no longer using the subscription.

Syntax

```
PositionUpdate
```

Examples

```
/* Example of subscribing/unsubscribing to position update events
from an Add On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private Account account;
     public MyAddOnTab()
     {
          // Find our Sim101 account
          lock (Account.All)
               account = Account.All.FirstOrDefault(a => a.Name ==
"Sim101");

          // Subscribe to position updates
          if (account != null)
               account.PositionUpdate += OnPositionUpdate;
     }

     // This method is fired as a position changes
     private void OnPositionUpdate(object sender, PositionEventArgs
 e)
     {
          // Output the new position
          NinjaTrader.Code.Output.Process(string.Format("Instrument
: {0} MarketPosition: {1} AveragePrice: {2} Quantity: {3}",
               e.Position.Instrument.FullName, e.MarketPosition,
e.AveragePrice, e.Quantity), PrintTo.OutputTab1);
     }

     // Called by TabControl when tab is being removed or window is
closed
     public override void Cleanup()
     {
          // Make sure to unsubscribe to the positions subscription
          if (account != null)
               account.PositionUpdate -= OnPositionUpdate;
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

Tip: The core MarketPosition e.Position is considered flat when Operation.Remove is seen, thus any related tracking in your logic you want to trigger or update should be aware.

An Operation.Update would be seen if there was no flat state in between, i.e. on a reverse of the position.

11.6.3.2.21 SimulationAccountReset

Definition

SimulationAccountReset can be used for subscribing to simulation account reset events. These resets occur whenever the user manually resets an account as well as when the user rewinds/fast forwards the Playback connection. When the reset occurs due to changes to the Playback connection it is important to recreate bar requests.

Note: Remember to unsubscribe if you are no longer using the subscription.

Syntax

```
SimulationAccountRest
```

Examples

```
/* Example of subscribing/unsubscribing to sim account reset events
from an Add On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     public MyAddOnTab()
     {
          // Subscribe to sim account resets
          Account.SimulationAccountReset +=
OnSimulationAccountReset;
     }

     /* This method is fired on sim account reset events. It is
important to recreate bar requests
     after a reset on the Playback connection */
     private void OnSimulationAccountReset(object sender, EventArgs
e)
     {
          Account simAccount = (sender as Account);

          // If the account was reset due to a rewind/fast forward
of the Playback connection
          if (simAccount != null && simAccount.Provider ==
Provider.Playback)
          {
               // Redo our bars requests here
          }
     }

     // Called by TabControl when tab is being removed or window is
closed
     public override void Cleanup()
     {
          // Make sure to unsubscribe to the simulation account
reset subscription
          Account.SimulationAccountReset -=
OnSimulationAccountReset;
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.2.22 Strategies

Definition

A collection of `StrategyBase` objects generated for the specified account

Property Value
An Collection of StrategyBase objects

Syntax
<Account>.Strategies

Examples

```
private Account myAccount;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Initialize myAccount
    }
}

private void OnAccountStatusUpdate(object sender,
AccountStatusEventArgs e)
{
    foreach (StrategyBase strategy in myAccount.Strategies)
    {
        Print(String.Format("Account status updated. {0} strategy
applied with position {1}", strategy.Name, strategy.Position));
    }
}
```

11.6.3.2.23 Submit()

Definition
Submits specified Order object(s).

Syntax
Submit(IEnumerable<Order> *orders*)

Parameters

orders	Order(s) to submit

Examples

```
Order stopOrder = null;
stopOrder = myAccount.CreateOrder(myInstrument, OrderAction.Sell,
OrderType.StopMarket, TimeInForce.Day, 1, 0, 1400, "myOCO",
"stopOrder", null);

myAccount.Submit(new[] { stopOrder });
```

11.6.3.3 BarsRequest

Definition

BarsRequest can be used to request Bars data and subscribe to real-time Bars data events.

Notes:

1. When using the DateTime fromLocal and toLocal parameters, the dates are converted to local daily timestamps (12:00 AM) and return a BarsRequest representing full trading days. If you need to request less than one full trading day, please use the barsBack parameter
2. A BarsRequest should be called only once and subscribe to the .Update event. Remember to unsubscribe from the .Update Event handler if you are no longer using the subscription.
3. A BarsRequest provides underlying market data for an instrument, but is not synchronized with an indicator or strategies primary data series. You will need to implement your own BarsUpdateEvent logic.
4. BarsRequest data **CANNOT** be used as input for a NinjaTrader indicator
5. Performing a BarsRequest in Playback will always yield bars up to the current playback time / slider position.
6. The documented BarsRequest behavior would be the same for all NinjaScript types.

Syntax

```
BarsRequest(Cbi.Instrument instrument, int barsBack)
BarsRequest(Cbi.Instrument instrument, DateTime fromLocal, DateTime toLocal)
```

Parameters

Instrument	The Instrument to request
barsBack	An `int` value determining the number of bars to request from the current time

fromLocal	A DateTime value determining the starting date to request
toLocal	A DateTime value determining the ending date to request

Methods and Properties

Bars	The Bars object returned from the request
BarsBack	An `int` representing the number of bars back used in the request
BarsPeriod	The BarsPeriod for the bars request
FromLocal	A DateTime representing the starting date used in the request
IsDividendAdjusted	A `bool` representing if the bars request will be dividend adjusted
IsResetOnNewTradingDay	A `bool` representing if the bars request will Break at EOD
IsSplitAdjusted	A `bool` representing if the bars request will be split adjusted
Instrument	The Instrument of the bars request
LookupPolicy	The lookup policies for the bars request. Possible Values are: • Provider - Queries the provider. The repository is updated on provider's reply • Repository - Looks up the local repository only
MergePolicy	The merge policy for the bars request.
Request()	Requests the bars as parametrized

TradingHours	The trading hours for the bars request
ToLocal	A DateTime representing the end date used in the request
Update	A BarsUpdateEvent handler for subscribing/unsubscribing to bar update events

Examples

```
/* Example of subscribing/unsubscribing to bars data events from an
Add On as well as making bars requests.
The concept can be carried over to any NinjaScript object you may
be working on. */
public class MyAddOnTab : NTTabPage
{
  private int daysBack = 5;
  private bool barsRequestSubscribed = false;
  private BarsRequest barsRequest;

  public MyAddOnTab()
  {
    // create a new bars request.  This will determine the
insturment and range for the bars to be requested
    barsRequest = new
BarsRequest(Cbi.Instrument.GetInstrument("AAPL"),
DateTime.Now.AddDays(-daysBack), DateTime.Now);

    // Parametrize your request.
    barsRequest.BarsPeriod  = new BarsPeriod { BarsPeriodType =
BarsPeriodType.Minute, Value = 1 };
    barsRequest.TradingHours = TradingHours.Get("Default 24 x 7");

    // Attach event handler for real-time events if you want to
process real-time data
    barsRequest.Update      += OnBarUpdate;

    // Request the bars
    barsRequest.Request(new Action<BarsRequest, ErrorCode,
string>((bars, errorCode, errorMessage) =>
    {
      if (errorCode != ErrorCode.NoError)
      {
        // Handle any errors in requesting bars here
        NinjaTrader.Code.Output.Process(string.Format("Error on
requesting bars: {0}, {1}",
                                        errorCode, errorMessage),
PrintTo.OutputTab1);
        return;
      }

      // Output the bars we requested. Note: The last returned bar
may be a currently in-progress bar
      for (int i = 0; i < bars.Bars.Count; i++)
      {
        // Output the bars
        NinjaTrader.Code.Output.Process(string.Format("Time: {0}
Open: {1} High: {2} Low: {3} Close: {4} Volume: {5}",
                                        bars.Bars.GetTime(i),
                                        bars.Bars.GetOpen(i),
                                        bars.Bars.GetHigh(i),
                                        bars.Bars.GetLow(i),
                                        bars.Bars.GetClose(i),
```

11.6.3.3.1 Request()

Definition

Performs the bars request for a BarsRequest object

Syntax

```
BarsRequest.Request(Action<BarsRequest, ErrorCode, string> callback)
```

Properties

BarsRequest	A BarsRequest representing the bars
ErrorCode	An ErrorCode representing error status
string	A string representing error message

Example

```
// Request the bars
barsRequest.Request(new Action<BarsRequest, ErrorCode,
string>((bars, errorCode, errorMessage) =>
{
     if (errorCode != ErrorCode.NoError)
     {
          // Handle any errors in requesting bars here
          NinjaTrader.Code.Output.Process(string.Format("Error on
requesting bars: {0}, {1}",
                                          errorCode, errorMessage),
 PrintTo.OutputTab1);
          return;
     }

     // Do something with the returned bars here.
     for (int i = 0; i < bars.Bars.Count; i++)
     {
          // Output the bars
          NinjaTrader.Code.Output.Process(string.Format("Time: {1}
Open: {2} High: {3} Low: {4} Close: {5} Volume: {6}",
                                          bars.Bars.GetTime(i),
                                          bars.Bars.GetOpen(i),
                                          bars.Bars.GetHigh(i),
                                          bars.Bars.GetLow(i),
                                          bars.Bars.GetClose(i),
                                          bars.Bars.GetVolume(i)),
PrintTo.OutputTab1);
     }
}));
```

11.6.3.3.2 MergePolicy

Definition

Determines the merge policy of the bars request.

Notes:

- This property is **ONLY** applicable to Futures contracts
- General information regrading **merge policies** can be found from the Market Data Configuration section
- For an Instruments configured **merge policy**, please see the MasterInstrument.MergePolicy property

Property Value

Represents the **MergePolicy** used for the bars request.

Possible values are:

DoNotMerge	No merge policy is applied
MergeBackAdjusted	Merge policy is applied between contracts along with rollover offsets
MergeNonBackAdjusted	Merge policy is applied between contracts without offsets
UseGlobalSettings	Uses the value configured from **Tools** -> **Options** -> **Market Data**
UseDefault	Uses the default values configured for the MasterInstrument

Syntax

`MergePolicy`

Example

```
// request the last 365 1 day bars
BarsRequest useGlobalRequest = new
BarsRequest(Instrument.GetInstrument("ES 09-16"), 365);
useGlobalRequest.BarsPeriod = new BarsPeriod { BarsPeriodType =
BarsPeriodType.Day, Value = 1 };

// use the merge policy the user has configured as their global
setting
useGlobalRequest.MergePolicy = MergePolicy.UseGlobalSettings;
useGlobalRequest.Request(new Action<BarsRequest, ErrorCode,
string>((barsRequest, errorCode, errorMessage) =>{

   Print("bars returned=" + barsRequest.Bars.Count);

}));

// dispose of the bars request if we are done with it
useGlobalRequest.Dispose();
```

11.6.3.4 Connection

Definition

The Connection class can be used to monitor connection related events as well as accessing connection related information.

Static Connection Class Events and Properties

CancelAllOrders()	Cancels all orders
Connect()	Connects to a connection
ConnectionStatusUpdate	Event handler for connection status updates

Events and Properties from Connection instances

Accounts	List of accounts from the connection
Disconnect()	Disconnects from the connection
Options	The connection's configuration options
PriceStatus	A ConnectionStatus representing the status of the price feed. Possible values are: ConnectionStatus.Connected ConnectionStatus.Connecting ConnectionStatus.ConnectionLost ConnectionStatus.Disconnecting ConnectionStatus.Disconnected
Status	A ConnectionStatus representing the status of the order feed. Possible values are: ConnectionStatus.Connected ConnectionStatus.Connecting ConnectionStatus.ConnectionLost ConnectionStatus.Disconnecting ConnectionStatus.Disconnected

Example

```
// Example of accessing information on all connected connections
public class MyAddOnTab : NTTabPage
{
   public MyAddOnTab()
   {
      // Print information about all connected connections
      lock (Connection.Connections)
        foreach(Connection c in Connection.Connections)
            NinjaTrader.Code.Output.Process(string.Format("Connecti
on: {0} Provider: {1}", c.Options.Name, c.Options.Provider),
PrintTo.OutputTab1);

      // Other required NTTabPage members left out for
demonstration purposes. Be sure to add them in your own code.
   }
}
```

11.6.3.4.1 CancelAllOrders()

Definition

Cancels all orders for the specified instrument on the connection.

Syntax

`<Connection>.CancelAllOrders(Instrument instrument)`

instrument	An Instrument object used to identify the instrument for which to cancel orders

Example

```
private Account myAccount;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Initialize myAccount
    }
}

private void OnExecutionUpdate(object sender, ExecutionEventArgs e)
{
    // Cancel all orders if an execution is triggered after 9pm
    if (e.Time > new DateTime(now.Year, now.Month, now.Day, 21, 0,
0))
        myAccount.CancelAllOrders(e.Execution.Instrument);
}
```

11.6.3.4.2 Connect()

Definition

Connects to a connection.

Syntax

`Connection.Connect(ConnectOptions options)`

Parameters

options	The connection option of what you want to connect to

Example

```
/* Example of subscribing/unsubscribing to execution update events
from an Add On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private Connection connection;
     public MyAddOnTab()
     {
          // Connect to Kinetick EOD
          if (connection == null)
               connection = Connect("Kinetick - End Of Day
```

```
(Free)");
     }

     private Connection Connect(string connectionName)
     {
          // Output the execution
          try
          {
               // Get the configured account connection
               ConnectOptions connectOptions = null;
               lock (Core.Globals.ConnectOptions)
                    connectOptions =
Core.Globals.ConnectOptions.FirstOrDefault(o => o.Name ==
connectionName);

               if (connectOptions == null)
               {
                    NinjaTrader.Code.Output.Process("Could not
connect. No connection found.", PrintTo.OutputTab1);
                    return null;
               }

               // If connection is not already connected, connect.
               lock (Connection.Connections)
                    if (Connection.Connections.FirstOrDefault(c =>
c.Options.Name == connectionName) == null)
                    {
                         Connection connect =
Connection.Connect(connectOptions);

                         // Only return connection if successfully
connected
                         if (connect.Status ==
ConnectionStatus.Connected)
                              return connect;
                         else
                              return null;
                    }

               return null;
          }
          catch (Exception error)
          {
               NinjaTrader.Code.Output.Process("Connect exception:
" + error.ToString(), PrintTo.OutputTab1);
               return null;
          }
```

```
    }

    // Called by TabControl when tab is being removed or window is
closed
    public override void Cleanup()
    {
        // Disconnect from our connection
        if (connection != null)
            connection.Disconnect();
    }

    // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.4.3 ConnectionStatusUpdate

Definition

ConnectionStatusUpdate can be used for subscribing to connection status update events.

Note: Remember to unsubscribe if you are no longer using the subscription.

Syntax

`ConnectionStatusUpdate`

Example

```
/* Example of subscribing/unsubscribing to connection update events
from an Add On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private Connection connection;
     public MyAddOnTab()
     {
          // Subscribe to connection updates
          Connection.ConnectionStatusUpdate +=
OnConnectionStatusUpdate;
     }

     // This method is fired on connection status update events
     private void OnConnectionStatusUpdate(object sender,
ConnectionStatusEventArgs e)
     {
          /* For multi-threading reasons, work with a copy of the
ConnectionStatusEventArgs to prevent situations
          where the ConnectionStatusEventArgs may already be ahead
of us while in the middle processing it. */
          ConnectionStatusEventArgs eCopy = e;

          // If the Kinetick EOD connection disconnects, do
something
          if (eCopy.Connection.Options.Name == "Kinetick - End Of
Day (Free)")
          {
               if (eCopy.Status == ConnectionStatus.Disconnected)
                    // Do something
          }
     }

     // Called by TabControl when tab is being removed or window is
closed
     public override void Cleanup()
     {
          // Make sure to unsubscribe to the connection status
subscription
          Connection.ConnectionStatusUpdate -=
OnConnectionStatusUpdate;
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.4.4 Disconnect()

Definition
Disconnects from the data connection.

Syntax
```
<Connection>.Disconnect()
```

Example


```
private void OnExecutionUpdate(object sender, ExecutionEventArgs e)
{
    // If an execution triggers after 9pm, disconnect from the
account's data source
    if (e.Time > new DateTime(now.Year, now.Month, now.Day, 21, 0,
0))
        myAccount.Connection.Disconnect();
}
```

11.6.3.4.5 Options

Definition
The connection's configuration options

Properties

ConnectOnStartup	A `bool` representing if this connection auto connects on startup
Name	A `string` representing the connection's name
Provider	A `Provider` representing the connection's provider

Example

```
// Example of accessing information on all connected connections
public class MyAddOnTab : NTTabPage
{
     public MyAddOnTab()
     {
          // Print information about all connected connections
          lock (Connection.Connections)
               Connection.Connections.ToList().ForEach(c =>
NinjaTrader.Code.Output.Process(string.Format("Connection: {0}
                    Provider: {1}", c.Options.Name,
c.Options.Provider), PrintTo.OutputTab1);
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.4.6 PriceStatus

Definition

Indicates the current status of the price feed of the primary data connection

Syntax

<Connection>.PriceStatus

Example

```
private int priceLost;
private int mainLost;

private void OnAccountItemUpdate(object sender,
AccountItemEventArgs e)
{
    // Count the number of times OnAccountItemUpdate() is called
with a lost price connection
    if (myAccount.Connection.PriceStatus ==
ConnectionStatus.ConnectionLost)
        priceLost += 1;

    // Count the number of times OnAccountItemUpdate() is called
with a lost primary connection
    if (myAccount.Connection.Status ==
ConnectionStatus.ConnectionLost)
        mainLost += 1;

    // Print the number of times each connection was lost during
OnAccountItemUpdate()
    if (mainLost > 0 || priceLost > 0)
        Print(String.Format("Main connection lost {0} times. Price
feed lost {1} times.", mainLost, priceLost));
}
```

11.6.3.4.7 Status

Definition

Indicates the current status of the primary data connection.

Properties

<Connection>.Status

Example

```
private int priceLost;
private int mainLost;

private void OnAccountItemUpdate(object sender,
AccountItemEventArgs e)
{
    // Count the number of times OnAccountItemUpdate() is called
with a lost price connection
    if (myAccount.Connection.PriceStatus ==
ConnectionStatus.ConnectionLost)
        priceLost += 1;

    // Count the number of times OnAccountItemUpdate() is called
with a lost primary connection
    if (myAccount.Connection.Status ==
ConnectionStatus.ConnectionLost)
        mainLost += 1;

    // Print the number of times each connection was lost during
OnAccountItemUpdate()
    if (mainLost > 0 || priceLost > 0)
        Print(String.Format("Main connection lost {0} times. Price
feed lost {1} times.", mainLost, priceLost));
}
```

11.6.3.4.8 ReloadAllHistoricalData()

Definition

To be used only in the OnConnectionStatusUpdate() event. Forces the data repository to be reloaded for any bars series running in the hosting script after. Data will be reloaded for any charts currently running which match the hosting scripts bars series (minute, tick, day). This method will also check and reload the max number of days or bars to load used in every chart running which matches the bars series contained in the script. Reloading historical data refreshes the UI which will force the NinjaScript object to re-transition to real-time. This method was designed for reloading historical data after an OnConnectionStatusUpdate event.

Critical: This method should **NOT** be called from any of the event methods which access data or any of the OnStateChange() states as it may be called recursively while the hosting object transitions through states. The designed use case for this method is reloading historical data after a connection update therefore we suggest **ONLY** using this method in the OnConnectionStatusUpdate method. Please see the examples below for an demonstration of the intended use case.

Method Return Value
This method does not return a value

Syntax
`ReloadAllHistoricalData()`

Parameters
This method does not take any parameters

Examples

```
//monitor our connection status so our NinjaScript object would
know to reload historical data
//create a bool which tracks when historical data would need to be
reloaded after a connection loss
private bool IsReloadAllHistoricalDataNeeded = false;
protected override void
OnConnectionStatusUpdate(ConnectionStatusEventArgs
connectionStatusUpdate)
{
   //if the connection status update detects a lost connection
   if(connectionStatusUpdate.Status ==
ConnectionStatus.ConnectionLost)
   {
      Print("Connection Lost, setting IsReloadAllHistorical Data to
true");
      // switch the reload data bool to true
      IsReloadAllHistoricalDataNeeded = true;


   }
   // only if we needed to reload historical data && only after
when we have reconnected
   else if (IsReloadAllHistoricalDataNeeded &&
connectionStatusUpdate.Status == ConnectionStatus.Connected )
   {
      Print("Connection is reconnected, reloading all historical
data");
      //then reload data and set our bool back to false.
      ReloadAllHistoricalData();
      IsReloadAllHistoricalDataNeeded = false;
   }

}
```

11.6.3.4.9 PlaybackConnection

Definition
Defines the method/property

> **Note**: This important note

Method Return Value
A `bool` value when **true**; otherwise **false**.

Syntax
```
MethodName(int input)
```

Parameters

input	An `int` which represents the method input

Examples

```
1
```

11.6.3.5 IInstrumentProvider Interface

When creating your NTTabPage, if you wish to use the instrument link, be sure to implement the IInstrumentProvider interface.

Examples

```
public class MyWindowTabPage : NTTabPage, IInstrumentProvider
{
     private Instrument instrument;

     public MyWindowTabPage()
     {
          /* Define the content for our NTTabPage. We can load
loose XAML to define controls and layouts
          if we so choose here as well.

          Note: XAML with event handlers defined inside WILL FAIL
when attempted to load.
```

```
          Note: XAML with "inline code" WILL FAIL when attempted to
load */
     }

     // IInstrumentProvider member
     public Instrument Instrument
     {
          get { return instrument; }
          set
          {
               if (instrument != null)
               {
                    // Unsubscribe to subscriptions to previously
selected instrument
               }

               if (value != null)
               {
                    // Create subscriptions for the newly selected
instrument
               }

               instrument = value;

               // Send instrument to other windows linked to the
same color
               PropagateInstrumentChange(value);

               // Update the tab header name
               RefreshHeader();
          }
     }

     // Be sure to include all the required NTTabPage members as
well
}
```

11.6.3.5.1 Instrument

In order for instrument linking to work properly in your Add On, Instrument must be created.

Examples

```
// IInstrumentProvider member
public Instrument Instrument
{
     get { return instrument; }
     set
     {
          if (instrument != null)
          {
               // Unsubscribe to subscriptions to previously
selected instrument
          }

          if (value != null)
          {
               // Create subscriptions for the newly selected
instrument
          }

          instrument = value;

          // Send instrument to other windows linked to the same
color
          PropagateInstrumentChange(value);

          // Update the tab header name
          RefreshHeader();
     }
}
```

11.6.3.6 IIntervalProvider Interface

When creating your NTTabPage, if you wish to use the interval link, be sure to implement the IIntervalProvider interface.

Examples

```
public class MyWindowTabPage : NTTabPage, IIntervalProvider
{
     public MyWindowTabPage()
     {
          /* Define the content for our NTTabPage. We can load
loose XAML to define controls and layouts
          if we so choose here as well.

          Note: XAML with event handlers defined inside WILL FAIL
when attempted to load.
          Note: XAML with "inline code" WILL FAIL when attempted to
load */
     }

     // IIntervalProvider member
     public BarsPeriod BarsPeriod { get; set; }

     // Be sure to include all the required NTTabPage members as
well
}
```

11.6.3.6.1 BarsPeriod

In order for interval linking to work properly in your Add On, BarsPeriod must be created.

Examples

```
// IIntervalProvider member
public BarsPeriod BarsPeriod { get; set; }
```

11.6.3.7 INTTabFactory Interface

If you wish to have tab page functionality like adding, removing, moving, duplicating tabs you must create a class which implements the INTTabFactory interface.

This interface contains two methods which must be hidden:

```
NTWindow CreateParentWindow();
NTTabPage CreateTabPage(string typeName, bool isNewWindow = false);
```

Examples

```
public class MyWindowFactory : INTTabFactory
{
     // INTTabFactory member. Creates the parent window that
contains tabs
     public NTWindow CreateParentWindow()
     {
          return new MyWindow();
     }

     // INTTabFactory member. Creates new tab pages whenever the
user presses the + button
     public NTTabPage CreateTabPage(string typeName)
     {
          return new MyWindowTabPage();
     }
}
```

11.6.3.7.1 CreateParentWindow ()

This determines which NTWindow is created as the parent window for our Add On.

Examples

```
// INTTabFactory member. Creates the parent window that contains
tabs
public NTWindow CreateParentWindow()
{
     return new MyWindow();
}
```

11.6.3.7.2 CreateTabPage()

This determines which NTTabPage is created whenever a new tab is needed in our parent window for our Add On.

Examples

```
// INTTabFactory member. Creates new tab pages whenever the user
presses the + button
public NTTabPage CreateTabPage(string typeName, bool isNewWindow =
false)
{
   return new MyWindowTabPage();
}
```

11.6.3.8 IWorkspacePersistence Interface

When creating your NTWindow, be sure to implement the IWorkspacePersistence interface as well for the ability to save and restore your window with NinjaTrader workspaces.

> **Note**: AddOn Classes which derive from **NTWindow** or implements **IWorkspacePersistance CANNOT** be a nested type of another class and **MUST** have a default constructor

This interface contains two methods and one property which must be hidden by the implementing class:

Restore()	Restores the window from workspaces.
Save()	Saves the window to workspaces.
WorkspaceOptions	Sets required workspace options.

Examples

```
public class MyWindow : NTWindow, IWorkspacePersistence
{
    // default constructor
    public MyWindow()
     {
          // Define our NTWindow. If we want to use NT style tabs,
we would define that here.

          // WorkspaceOptions property must be set
          Loaded += (o, e) =>
          {
               if (WorkspaceOptions == null)
                    WorkspaceOptions = new
WorkspaceOptions("MyWindow-" + Guid.NewGuid().ToString("N"), this);
          };
     }

     // IWorkspacePersistence member. Required for restoring window
from workspaces
     public void Restore(XDocument document, XElement)
     {
          if (MainTabControl != null)
               MainTabControl.RestoreFromXElement(element);
     }

     // IWorkspacePersistence member. Required for saving window to
workspaces
     public void Save(XDocument document, XElement element)
     {
          if (MainTabControl != null)
               MainTabControl.SaveToXElement(element);
     }

     // IWorkspacePersistence member
     public WorkspaceOptions WorkspaceOptions { get; set; }
}
```

11.6.3.8.1 Restore()

Restores the window from workspaces.

Examples

```
// IWorkspacePersistence member. Required for restoring window from
workspaces
public void Restore(XDocument document, XElement)
{
     if (MainTabControl != null)
          MainTabControl.RestoreFromXElement(element);
}
```

11.6.3.8.2 Save()

Saves the window to workspaces.

Examples

```
// IWorkspacePersistence member. Required for saving window to
workspaces
public void Save(XDocument document, XElement element)
{
     if (MainTabControl != null)
          MainTabControl.SaveToXElement(element);
}
```

11.6.3.8.3 WorkspaceOptions

Definition

Sets required workspace options.

Notes:

- The **WorkspaceOptions** class includes logic for opening, closing, saving, and restoring workspaces, checking windows are off screen, and setting basic properties such as the workspace name and current status.
- A **WorkspaceOptions** property must simply be declared within your **NTWindow**, as in the example below. All of its contained logic is taken care of automatically.

Tip: For a complete, working example of this class in use, please download the AddOn Framework NinjaScript Basic Example to your desktop.

Examples

```
// IWorkspacePersistence member
public WorkspaceOptions WorkspaceOptions { get; set; }
```

11.6.3.9 NTTabPage Class

This is where the actual content for tabs inside the custom add on NTWindow can be defined.

> **Note**: A class derived from NTTabPage has to be created if instrument link or interval link functionality is desired. IInstrumentProvider and IIntervalProvider interfaces should be implemented as well to ensure proper linking.

Cleanup()	Unregisters LinkControls and calls Cleanup() on ICleanable controls on the NTTabPage
GetHeaderPart()	Indicates the tab header name.
Restore()	Restores any elements in our NTTabPage from the workspace.
Save()	Saves elements in our NTTabPage to the workspace.

Examples

```
public class MyWindowTabPage : NTTabPage,
NinjaTrader.Gui.Tools.IInstrumentProvider, IIntervalProvider
{
     private Instrument instrument;

     public MyWindowTabPage()
     {
          /* Define the content for our NTTabPage. We can load
loose XAML to define controls and layouts
          if we so choose here as well.

          Note: XAML with event handlers defined inside WILL FAIL
when attempted to load.
          Note: XAML with "inline code" WILL FAIL when attempted to
```

```
load */
     }

     // Called by TabControl when a tab is being removed or window
is closed
     public override void Cleanup()
     {
          /* Unsubscribe and clean up resources used by the tab
that just closed. You may have
          resources you don't want to clean up just yet because the
window is still being used */
     }

     // NTTabPage member. Required for determining the tab header
name
     protected override string GetHeaderPart(string variable)
     {
          // Determine the text for the tab header name
       return variable;
     }

     // NTTabPage member. Required for restoring elements from
workspaces
     protected override void Restore(System.Xml.Linq.XElement
element)
     {
          if (element == null)
               return;

          // Restore any elements you may have saved. e.g. selected
accounts or instruments
     }

     // NTTabPage member. Required for saving elements to
workspaces
     protected override void Save(System.Xml.Linq.XElement element)
     {
          if (element == null)
               return;

          // Save any elements you may want persisted. e.g.
selected accounts or instruments
     }

     // IInstrumentProvider member
     public Instrument Instrument
     {
```

```
            get { return instrument; }
            set
            {
                  if (instrument != null)
                  {
                        // Unsubscribe to subscriptions to previously
selected instrument
                  }

                  if (value != null)
                  {
                        // Create subscriptions for the newly selected
instrument
                  }

                  instrument = value;

                  // Update the tab header name
                  RefreshHeader();
            }
      }

      // IIntervalProvider member
      public BarsPeriod BarsPeriod { get; set; }
}
```

11.6.3.9.1 Cleanup()

Definition

Unregisters LinkControls (IInstrumentProvider IIntervalProvider) and calls Cleanup() on ICleanable controls on the NTTabPage. Override this to, e.g., unsubscribe from events or perform any other cleanup operations when the tab is closed.

Note: When overriding **Cleanup()**, it is strongly recommended when you call `base.Cleanup()` which ensures any link controls are also unregistered. The base implementation will also handle cleaning up any controls which implement ICleanable: AccountSelector, AtmStrategySelector, InstrumentSelector, IntervalSelector, TifSelector

Method Return Value

This method does not return a value

Syntax

```
public override void Cleanup()
{
```

```
}
```

Parameters

This method does not accept any parameters

Examples

```
public override void Cleanup()
{
   // unregister from any custom events
   Connection.ConnectionStatusUpdate                    -=
OnConnectionStatusUpdate;

   // a call to base.Cleanup() will loop through the visual tree
looking for all ICleanable children
   // i.e., AccountSelector, AtmStrategySelector,
InstrumentSelector, IntervalSelector, TifSelector,
   // as well as unregister any link control events

   base.Cleanup();
}
```

11.6.3.9.2 GetHeaderPart()

Definition

Indicates the tab header name.

Examples

```
// NTTabPage member. Required for determining the tab header name
protected override string GetHeaderPart(string variable)
{
     // Determine the text for the tab header name
     switch (variable)
     {
         case "@INSTRUMENT": return Instrument == null ?
Resource.GuiNewTab : Instrument.MasterInstrument.Name;
         case "@INSTRUMENT_FULL": return Instrument == null ?
Resource.GuiNewTab : Instrument.FullName;
     }
     return variable;
}
```

11.6.3.9.3 Restore()

Restores any elements in our NTTabPage from the workspace. (e.g. Selected accounts or instruments)

Examples

```
// NTTabPage member. Required for restoring elements from
workspaces
public void Restore(XElement element)
{
     if (element == null)
          return;

     // Restore any elements you may have saved. e.g. selected
accounts or instruments
}
```

11.6.3.9.4 Save()

Saves elements in our NTTabPage to the workspace (e.g. Selected accounts or instruments)

Examples

```
// NTTabPage member. Required for saving elements to workspaces
public void Save(XElement element)
{
     if (element == null)
          return;

     // Save any elements you may want persisted. e.g. selected
accounts or instruments
}
```

11.6.3.10 Alert and Debug Concepts

In most scenarios you can use the NinjaScript provided methods for triggering alerts and debugging functionality. However, when building your own custom objects, you may find yourself wanting to use this functionality outside the NinjaScript scope (e.g. when building a NTTabPage for Add Ons).

Using the NinjaScript Output

Instead of Print(), use Output.Process() to write a message.
Instead of ClearOutputWindow(), use Output.Reset() to clear the output window.

Example

```
// Instead of Print()
NinjaTrader.Code.Output.Process("my message", PrintTo.OutputTab1);

// Instead of ClearOutputWindow()
NinjaTrader.Code.Output.Reset()
```

Using Alerts

Instead of Alert(), use NinjaTrader.NinjaScript.Alert.AlertCallback() for sending an alert.
Instead of ResetAlert(), use NinjaTrader.NinjaScript.Alert.RearmAlert()

Example

```
// Instead of Alert()
NinjaTrader.NinjaScript.Alert.AlertCallback(NinjaTrader.Cbi.Instrum
ent.GetInstrument("MSFT"), this, "someId",
NinjaTrader.Core.Globals.Now, Priority.High, "message", null,
Brushes.Blue, Brushes.White, 0);

// Instead of ResetAlert()
NinjaTrader.NinjaScript.Alert.ResetAlert("someId");
```

Miscellaneous

Instead of Log(), use NinjaScript.Log() to send a message to the NinjaTrader logs.
Instead of PlaySound(), use Globals.PlaySound() to play a sound.
Instead of SendMail(), use Globals.SendMail() to send a mail.

Tip: Both the Globals.PlaySound() and .SendMail() above could be used in a regular NinjaScript objects as well, however this is not recommended practice since those would not ignore the calls outside State.Realtime which could yield unexpected results.

Examples

```
// Instead of Log()
NinjaScript.Log("My log message", LogLevel.Error);

// Instead of PlaySound()
NinjaTrader.Core.Globals.PlaySound(@"C:\mySound.wav");

// Instead of SendMail()
NinjaTrader.Core.Globals.SendMail("customers@email.com",
"cc_these_people@email.com", "Subject", "Mail body", null);
```

Error Codes in Log Files

The ErrorCode enumeration can be found in NinjaTrader logs from time to time when an error occurs, and these can provide further clues into the cause of unexpected behavior during your debugging. These error codes are not necessarily related to your code, but they can provide an indication of an issue to address outside of the scope of your code, saving you time in trying to find the source of errors in your code. Below is a list of ErrorCode enum values and their meanings:

NoError	No errors were thrown
LogOnFailed	Failed to log on due to invalid credentials
OrderRejected	Broker rejected the current order
UnableToCancelOrder	Order cannot be canceled now, but may be successfully canceled later
UnableToChangeOrder	Either the exchange or broker does not support order updates for the instrument in question, or the order has not yet been submitted
UserAbort	The operation was aborted by the user
Panic	An unspecified error was thrown

11.6.3.10.1 AlertCallback()

Definition

Creates an alert event to be raised specified by a string "id" and a corresponding .wav file will be played matching the "soundLocation" parameter. Once an alert has triggered, its message is reflected in the "Alerts Log" window based on the background and foreground brushes provided in the callback.

Notes:

1. If the **AlertCallback()** method is called again with the same string "id" parameter *before* the provided "rearmSeconds" duration has passed, the alert event will be reset based on the new "rearmSeconds" parameter provided. Doing so could consequently cause an alert to be reset inadvertently, in which case you should pass a "rearmSeconds" parameter of "0" to ensure the specified alert event is always raised.
2. The **AlertCallback()** method is the same core function used by the simpler Alert() method which can alternatively be used with NinjaScript indicators and strategies. The AlertCallback() was exposed for use with Add-ons or other more advance use cases.
3. Providing a "rearmSeconds" parameter greater than "0" will add the matching alert id to a rearmed state, which only allows the alert to be reissued after the specified time interval in seconds has lapsed. You can reset an alert's rearm parameter by using the ResetAlertRearmById().

Method Return Value

This method does not return a value.

Syntax

```
NinjaTrader.NinjaScript.Alert.AlertCallback(Instrument instrument, object source,
string id, DateTime time, Priority priority, string message, string soundLocation,
Brush backBrush, Brush foreBrush, int rearmSeconds)
```

Warning: An "id" parameter **MUST** be provided otherwise a null argument exception will be generated

Parameters

instrument	An Instrument object associated with the alert.
source	A generic object type which created the alert (e.g. "this")
id	A `string` representing a unique id for the alert

time	The DateTime representing the time associated with the alert
priority	Sets the precedence of the alert in relation to other alerts. Any one of the following values: Priority.High Priority.Low Priority.Medium
message	A `string` representing the Alert message
soundLocation	A `string` representing the absolute file path of the .wav file to play.
backBrush	Sets the background color of the Alerts window row for this alert when triggered (reference)
foreBrush	Sets the foreground color of the Alerts window row for this alert when triggered (reference)
rearmSeconds	An `int` which sets the number of seconds an alert will rearm. **Note**: If the same alert (identified by the id parameter) is called within a time window of the time of last alert + rearmSeconds, the alert will be ignored.

Tips: You can obtain the default NinjaTrader installation directory to access the sounds folder by using `NinjaTrader.Core.Globals.InstallDir` property. Please see the example below for usage.

Examples

```
NinjaTrader.NinjaScript.Alert.AlertCallback(NinjaTrader.Cbi.Instrum
ent.GetInstrument("MSFT"), this, "someId",
NinjaTrader.Core.Globals.Now, Priority.High, "message",
NinjaTrader.Core.Globals.InstallDir+@"\sounds\Alert1.wav", new
SolidColorBrush(Colors.Blue), new SolidColorBrush(Colors.White),
0);
```

11.6.3.10.2 RearmAlert()

Definition

Rearms an existing alert event by the string "id" parameter created via the AlertCallback() method. A NinjaScript generated alert by may need to be rearmed after the alert is triggered depending on the Alert()'s rearmSeconds parameter.

Note: The NinjaScriptBase has a non-static method implemented with the same name. Please see the RearmAlert() method for Indicator or Strategies.

Method Return Value

This method does not return a value.

Syntax

```
NinjaTrader.NinjaScript.Alert.RearmAlert(string id)
```

Parameters

id	A unique `string` id representing an alert id to reset

Examples

```
if (resetCondition)
{
    NinjaTrader.NinjaScript.Alert.ResetAlertRearmById("someId");
    resetCondition = false;
}
```

11.6.3.11 AtmStrategy

AtmStrategy contains properties and methods used to manage ATM Strategies. When working with an AtmStrategySelector, selected objects can be case to AtmStrategy to obtain or change their properties.

Notes:
1. For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview
2. For more information on working with the ATM strategies programmatically in general, please see the Using ATM Strategies section.

Example

```
// Using AtmStrategy to handle user selections in an ATM Strategy
Selector
myAtmStrategySelector.SelectionChanged += (o, args) =>
{
    if (myAtmStrategySelector.SelectedItem == null)
        return;
    if (args.AddedItems.Count > 0)
    {
        // Change the selected TIF in a TIF selector based on what
is selected in the ATM Strategy Selector
        NinjaTrader.NinjaScript.AtmStrategy selectedAtmStrategy =
args.AddedItems[0] as NinjaTrader.NinjaScript.AtmStrategy;
        if (selectedAtmStrategy != null)
        {
            myTifSelector.SelectedTif =
selectedAtmStrategy.TimeInForce;
        }
    }
};
```

11.6.3.12 ControlCenter

Definition

ControlCenter is a XAML-defined class containing the layout and properties of the Control Center window. When altering the Control Center window (for example, to add a menu item into the "New" menu to launch an NTWindow as part of an AddOn, as seen in the example below), a generic reference to a Window object can be cast to ControlCenter specifically.

Note: For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

Example

```
private NTMenuItem ControlCenterNewMenu;

protected override void OnWindowCreated(Window window)
{
    // We want to place the menu item for the AddOn in the Control
Center's "New" menu
    // First obtain a reference to the Control Center window
    ControlCenter cc = window as ControlCenter;
    if (cc == null)
        return;

    /* Determine we want to place the AddOn in the Control Center's
"New" menu
    Other menus can be accessed via the control's "Automation ID".
For example: toolsMenuItem, workspacesMenuItem,
connectionsMenuItem, helpMenuItem. */
    ControlCenterNewMenu = cc.FindFirst("ControlCenterMenuItemNew")
 as NTMenuItem;
}
```

11.6.3.13 FundamentalData

Definition

FundamentalData is used to access fundamental snapshot data and for subscribing to fundamental data events.

> **Note**: Remember to unsubscribe if you are no longer using the subscription.

Properties

AverageDailyVolume	A `double` representing the average daily volume
Beta	A `double` representing the beta
CalendarYearHigh	A `double` representing the high price of the calendar year
CalendarYearHighDate	A `DateTime` representing the date of the calendar year's high price

CalendarYearLow	A `double` representing the low price of the calendar year
CalendarYearLow Date	A `DateTime` representing the date of the calendar year's low price
CurrentRatio	A `double` representing the current ratio
DividendAmount	A `double` representing the dividend amount
DividendPayDate	A `DateTime` representing the date dividends are paid
DividendYield	A `double` representing the dividend yield
EarningsPerShare	A `double` representing the earnings per share
FiveYearsGrowthPercentage	A `double` representing the 5yr growth percent
High52Weeks	A `double` representing the 52 week high
High52WeeksDate	A `DateTime` representing the date of the 52 week high price
HistoricalVolatility	A `double` representing the historical volatility
InsiderOwned	A `double` representing the insider owned amount
Instrument	An Instrument representing the instrument
Low52Weeks	A `double` representing the 52 week low
Low52WeeksDate	A `DateTime` representing the date of the 52 week low price
MarketCap	A `double` representing the market capitalization
NextYearsEarningsPerShare	A `double` representing next year's earnings per share

PercentHeldByInstitutions	A `double` representing the percent held by institutions
PriceEarningsRatio	A `double` representing the P/E ratio
RevenuePerShare	A `double` representing the revenue per share
SharesOutstanding	A `long` representing the shares outstanding
ShortInterest	A `double` representing the short interest
ShortInterestRatio	A `double` representing the short interest ratio
VWAP	A `double` representing the VWAP
Update	Event handler for subscribing/unsubscribing to market depth events

Syntax

```
FundamentalData
```

Example

```
/* Example of subscribing/unsubscribing to fundamental data from an
Add On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
	private Instrument instrument;

	public MyAddOnTab()
	{
		instrument = Instrument.GetInstrument("AAPL");

		if (instrument == null)
			return;

		// Subscribe to fundamental data. Snapshot data is
provided right on subscription
		if (!instrument.Dispatcher.HasShutdownStarted)
			instrument.Dispatcher.InvokeAsync(() =>
instrument.FundamentalData.Update += OnFundamentalData);

		// Printing snapshot fundamental data for average
daily volume

NinjaTrader.Code.Output.Process(instrument.FundamentalData.AverageD
ailyVolume, PrintTo.OutputTab1);
	}

	// This method is fired on fundamental data events
	private void OnFundamentalData(object sender,
FundamentalDataEventArgs e)
	{
	   // Do something with fundamental data events
	}

	// Called by TabControl when tab is being removed or window
is closed
	public override void Cleanup()
	{
		// Make sure to unsubscribe to the fundamental data
subscription
		if (instrument != null)
			instrument.FundamentalData.Update -=
OnFundamentalData;
	}

	// Other required NTTabPage members left out for
demonstration purposes. Be sure to add them in your own code.
}
```

11.6.3.14 MarketData

Definition

MarketData can be used to access snapshot market data and for subscribing to market data events.

> **Notes**:
> 1. Remember to unsubscribe if you are no longer using the subscription.
> 2. You should only unsubscribe to a market data event if you are actually subscribed.

Properties

Ask	A MarketDataEventArgs representing the ask price
Bid	A MarketDataEventArgs representing the bid price
DailyHigh	A MarketDataEventArgs representing the daily high
DailyLow	A MarketDataEventArgs representing the daily low
DailyVolume	A MarketDataEventArgs representing the daily volume
Instrument	An Instrument representing the instrument
Last	A MarketDataEventArgs representing the last price
LastClose	A MarketDataEventArgs representing the last close
Opening	A MarketDataEventArgs representing the opening price
OpenInterest	A MarketDataEventArgs representing the open interest

Settlement	A MarketDataEventArgs representing the settlement price
Update	Event handler for subscribing/unsubscribing to market depth events **Note**: Attempting to unsubscribe to this event before there is a subscription will generate errors.

Syntax

`MarketData`

Example

```
/* Example of subscribing/unsubscribing to market data from an Add
On. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
	private Instrument instrument;

	public MyAddOnTab()
	{
		instrument = Instrument.GetInstrument("AAPL");
		if (instrument == null)
			return;

		// Subscribe to market data. Snapshot data is
provided right on subscription
		// Note: "instrument" is a placeholder in this
example, you will need to replace
		// with a valid Instrument object through various
methods or properties available depending
		// on the NinjaScript type you are working with
(e.g., Bars.Instrument or Instrument.GetInstrument()
		if (!instrument.Dispatcher.HasShutdownStarted)
			instrument.Dispatcher.InvokeAsync(() =>
instrument.MarketData.Update += OnMarketData);

		// Printing snapshot market data for the last price
and time

NinjaTrader.Code.Output.Process(instrument.MarketData.Last.Price.To
String() + " " + instrument.MarketData.Last.Time.ToString(),
			PrintTo.OutputTab1);
	}

	// This method is fired on market data events
	private void OnMarketData(object sender, MarketDataEventArgs
e)
	{
		// Do something with market data events
	}

	// Called by TabControl when tab is being removed or window
is closed
	public override void Cleanup()
	{
		// Make sure to unsubscribe to the market data
subscription
		if (instrument != null)
			instrument.MarketData.Update -= OnMarketData;
	}

	// Other required NTTabPage members left out for
demonstration purposes. Be sure to add them in your own code.
```

11.6.3.15 MarketDepth

Definition

MarketDepth can be used to access snapshot market depth and for subscribing to market depth events.

> **Notes**:
> 1. Remember to unsubscribe if you are no longer using the subscription.
> 2. You should only unsubscribe to a market depth event if you are actually subscribed.
> 3. You must unsubscribe from the same thread where the subscription is made. It is therefore recommended to user an Instrument's Dispatcher to ensure this is handled properly.

Properties

Asks	List of ask prices
Bids	List of bid prices
Instrument	Instrument representing the instrument of the market depth event
Update	Event handler for subscribing/unsubscribing to market depth events

Syntax

```
MarketDepth
```

Example

```
/* Example of subscribing/unsubscribing to market depth from an Add
On. */
public class MyAddOnTab : NTTabPage
{
     private Instrument instrument;

     public MyAddOnTab()
     {
          instrument = Instrument.GetInstrument("AMD");

          if (instrument == null)
              return;

          // Follow this pattern to subscribe to MarketDepth events
so they may be unsubscribed from the same instrument thread
          if (!instrument.Dispatcher.HasShutdownStarted)
              instrument.Dispatcher.InvokeAsync(() =>
instrument.MarketDepth.Update += OnMarketDepth);

          // Print the Ask's price ladder
          for (int i = 0; i < instrument.MarketDepth.Asks.Count; i+
+)
          {
              NinjaTrader.Code.Output.Process(string.Format("Positi
on: {0} Price: {1} Volume: {2}", i,
                  instrument.MarketDepth.Asks[i].Price,
instrument.MarketDepth.Asks[i].Volume), PrintTo.OutputTab1);
          }
     }

     // This method is fired on market depth events and after the
snapshot data is updated.
     private void OnMarketDepth(object sender, MarketDepthEventArgs
e)
     {
          return;
     }

     // Called by TabControl when tab is being removed or window is
closed
     public override void Cleanup()
     {
          // Follow this pattern to subscribe to MarketDepth events
so they may be unsubscribed from the same intrument thread
          if (instrument != null &&
!instrument.Dispatcher.HasShutdownStarted)
              instrument.Dispatcher.InvokeAsync(() =>
instrument.MarketDepth.Update -= OnMarketDepth);
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.16 NewsItems

Definition

NewsItems can be used to store news articles.

Properties

Items	Collection of NewsEventArgs representing news articles
NewsToMaintain	An `int` representing the number of articles to maintain
Update()	For storing news articles

Syntax

`NewsItems`

Example

```
/* Example of storing and accessing news items from an Add On. The
concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private NewsSubscription newsSubscription;
     private NewsItems        newsItems;

     public MyAddOnTab()
     {
          // Subscribe to news
          newsSubscription         = new NewsSubscription();
          newsSubscription.Update += OnNews;
          newsItems                = new NewsItems(10);

          // Print news
          PrintNews(newsItems);
     }

     // This method is fired as new News events come in. Old News
events are not provided when you subscribe.
     private void OnNews(object sender, NewsEventArgs e)
     {
          // Store the news items
```

```
            newsItems.Update(e);
        }

        // Loop through the stored news articles and output them
        private void PrintNews(NewsItems news)
        {
            for (int x = 0; x < news.Items.Count; x++)
            {
                NinjaTrader.Code.Output.Process(string.Format("ID:
{0} News Provider: {1} Headline: {2}",
                    news.Items[x].Id,
                    news.Items[x].NewsProvider,
                    news.Items[x].Headline), PrintTo.OutputTab1);
            }
        }

        // Called by TabControl when tab is being removed or window is
closed
        public override void Cleanup()
        {
            // Make sure to unsubscribe to the News subscription
            if (newsSubscription != null)
                newsSubscription.Update -= OnNews;
        }

        // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.17 NewsSubscription

Definition

NewsSubscription can be used for subscribing to News events.

> **Note**: Remember to unsubscribe if you are no longer using the subscription.

Properties

Update	Event handler for subscribing/unsubscribing to market depth events

Syntax

NewsSubscription

Example

```
/* Example of subscribing/unsubscribing to news from an Add On. The
concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private NewsSubscription newsSubscription;
     private NewsItems        newsItems;

     public MyAddOnTab()
     {
          // Subscribe to news
          newsSubscription         = new NewsSubscription();
          newsSubscription.Update += OnNews;
          newsItems                = new NewsItems(10);
     }

     // This method is fired as new News events come in. Old News
events are not provided when you subscribe.
     private void OnNews(object sender, NewsEventArgs e)
     {
          // Print the headline of the news
          NinjaTrader.Code.Output.Process(string.Format("ID: {0}
News Provider: {1} Headline: {2}",
               e.Id,
               e.NewsProvider,
               e.Headline), PrintTo.OutputTab1);

          // Maintain the news items
          newsItems.Update(e);
     }

     // Called by TabControl when tab is being removed or window is
closed
     public override void Cleanup()
     {
          // Make sure to unsubscribe to the News subscription
          if (newsSubscription != null)
               newsSubscription.Update -= OnNews;
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code.
}
```

11.6.3.18 NTMenuItem

Definition

NTMenuItem is used to create new menu entries. For example, an instance of this class can be placed in an existing Control Center menu to launch an NTWindow as part of an AddOn, as seen in the example code below.

> **Note**: For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

Examples

```
private NTMenuItem myNewMenuItem;
private NTMenuItem existingControlCenterNewMenu;

protected override void OnWindowCreated(Window window)
{
    // We want to place the menu item for the AddOn in the Control
Center's "New" menu
    // First obtain a reference to the Control Center window
    ControlCenter cc = window as ControlCenter;
    if (cc == null)
        return;

    /* Determine we want to place the AddOn in the Control Center's
"New" menu
    Other menus can be accessed via the control's "Automation ID".
For example: toolsMenuItem, workspacesMenuItem,
connectionsMenuItem, helpMenuItem. */
    existingControlCenterNewMenu =
cc.FindFirst("ControlCenterMenuItemNew") as NTMenuItem;
    if (existingControlCenterNewMenu == null)
        return;

    // Instantiate myNewMenuItem
    // 'Header' sets the name of our AddOn seen in the menu
structure. 'Style' sets the font style.
    myNewMenuItem = new NTMenuItem { Header = "AddOn Framework",
Style = Application.Current.TryFindResource("MainMenuItem") as
Style };

    // Add our AddOn menu item into the "New" menu
    existingControlCenterNewMenu.Items.Add(myNewMenuItem);

    // Subscribe to the event for when the user presses the menu
item
    myNewMenuItem.Click += OnMenuItemClick;
}
```

11.6.3.19 NTMessageBoxSimple.Show()

Definition

Creates a message box window.

Note: For more information on using MessageBox windows, please see .NET MessageBox Class Documentation

Method Return Value

MessageBoxResult; an enum representing the button press used to close the MessageBox window

Syntax

```
NTMessageBoxSimple.Show(Window input, string messageTxt, string caption,
MessageBoxButton buttonSet, MessageBoxImage icon)
```

Parameters

parent	A Window (DependencyObject) which represents the owning window
messageTxt	The message body of the MessageBox window
caption	The header of the MessageBox window
buttonSet	A MesageBoxButton enum determining the buttons used for the MessageBox window
icon	A MesageBoxImage enum determining the icon used for the MessageBox window

Examples

```
// Create a MessageBox window from a Chart
ChartControl.Dispatcher.InvokeAsync(new Action(() => {
   NinjaTrader.Gui.Tools.NTMessageBoxSimple.Show(Window.GetWindow(C
hartControl.OwnerChart as DependencyObject), "Message Body",
"Message Header", MessageBoxButton.OK, MessageBoxImage.None);
}));
```

```
// Create a MessageBox window from a button press in an AddOn
private void OnMenuItemClick(object sender, RoutedEventArgs e)
{
    NinjaTrader.Gui.Tools.NTMessageBoxSimple.Show(Window.GetWindow(
e.Source as DependencyObject), "Message Body", "Message Header",
MessageBoxButton.OK, MessageBoxImage.None);
}
```

11.6.3.20 NTWindow

Definition

The **NTWindow** class defines parent windows for custom window creation. Instances of NTWindow act as containers for instances of NTTabPage, in which UI elements and their related logic are contained.

Notes:

- The IWorkspacePersistance interface should be implemented if you want your window to be saved and restored with NinjaTrader workspaces.
- AddOn Classes which derive from **NTWindow** or implements **IWorkspacePersistance** **CANNOT** be a nested type of another class and **MUST** have a default constructor

Example

The example below shows how to instantiate an NTWindow while:

- Implementing IWorkspacePersistence to ensure the window is saved/restored in workspaces
- Setting the window caption and dimensions
- Instantiating a TabControl to support tabs within the window
- Setting workspace options

Tip: For a complete, working example of this class in use, download framework example located on our AddOn Development Overview

```
public class AddOnFrameworkWindow : NTWindow, IWorkspacePersistence
{
    // default constructor
    public AddOnFrameworkWindow()
    {
```

```
        // set Caption property (not Title), since Title is managed
internally to properly combine selected Tab Header and Caption for
display in the Windows taskbar
        // This is the name displayed in the top-left of the window
        Caption = "AddOn Framework";

        // Set the default dimensions of the window
        Width   = 1085;
        Height  = 900;

        // TabControl should be created for window content if tab
features are wanted
        TabControl tc = new TabControl();

        // Attached properties defined in the TabControlManager
class should be set to achieve adding, removing, and moving tabs
        TabControlManager.SetIsMovable(tc, true);
        TabControlManager.SetCanAddTabs(tc, true);
        TabControlManager.SetCanRemoveTabs(tc, true);

        // if ability to add new tabs is desired, TabControl has to
have attached property "Factory" set.
        TabControlManager.SetFactory(tc, new
AddOnFrameworkWindowFactory());
        Content = tc;

        /* In order to have link buttons functionality, tab control
items must be derived from Tools.NTTabPage
        They can be added using extension method
AddNTTabPage(NTTabPage page) */
        tc.AddNTTabPage(new AddOnFrameworkTab());

        // WorkspaceOptions property must be set
        Loaded += (o, e) =>
        {
            if (WorkspaceOptions == null)
                WorkspaceOptions = new
WorkspaceOptions("AddOnFramework-" + Guid.NewGuid().ToString("N"),
this);
        };
    }

    // IWorkspacePersistence member. Required for restoring window
from workspace
    public void Restore(XDocument document, XElement element)
    {
        if (MainTabControl != null)
```

```
            MainTabControl.RestoreFromXElement(element);
    }

    // IWorkspacePersistence member. Required for saving window to
workspace
    public void Save(XDocument document, XElement element)
    {
        if (MainTabControl != null)
            MainTabControl.SaveToXElement(element);
    }

    // IWorkspacePersistence member
    public WorkspaceOptions WorkspaceOptions { get; set; }
}
```

11.6.3.21 NumericTextBox

NumericTextBox provides functionality for numeric text boxes to capture user input. This UI element can be defined in XAML for an AddOn if desired, with functionality and logic related to the text box defined in C#, as in the examples below.

> **Note**: For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

NumericTextBox inherits from System.Windows.Controls.Textbox, and the following additional properties can be accessed for an instance the class:

Minimum	Determines the minimum value which can be entered
Maximum	Determines the maximum value which can be entered
ValueType	Determines the System.Type which can be accepted

Examples

XAML Definition of the UI Element

```
<!-- Create a grid in which to place the NumericTextBox -->
<Grid>
    <!-- Define a NumericTextBox -->
    <t:NumericTextBox x:Name="daysBackSelector" Text="5"
ValueType="{x:Type system:Int32}" Width="50" Grid.Column="2">
        <!-- Set the margins for the box -->
        <t:NumericTextBox.Margin>
            <Thickness Left="{StaticResource MarginButtonLeft}"
Top="{StaticResource PaddingColumn}" Right="{StaticResource
MarginBase}"/>
        </t:NumericTextBox.Margin>
    </t:NumericTextBox>
</Grid>
```

C# Code Handling Logic

```
private NumericTextBox daysBack;

private DependencyObject LoadXAML()
{
        // Find days back selector
        daysBack = LogicalTreeHelper.FindLogicalNode(pageContent,
"daysBackSelector") as NumericTextBox;
}
```

11.6.3.22 OnWindowCreated()

Definition

This method is called whenever a new NTWindow is created. It will be called in the thread of that window. This is where you would install your AddOn to an existing window, or if creating your own custom window, add a Menu item to the NinjaTrader Control Center.

> **Note**: This method will also be called on a **recompile** of the NinjaTrader.Custom project (e.g., when you compile an indicator, strategy, or add-on)

Method Return Value

This method does not return a value

Syntax

```
OnWindowCreated(Window window)
```

Parameters

window	A Window object which is being added to the workspace

Examples

```
public class MyWindowAddOn : AddOnBase
{
     private NTMenuItem myMenuItem;
     private NTMenuItem existingMenuItem;

     protected override void OnStateChange()
     {
          if (State == State.SetDefaults)
          {
               Description = "Our custom MyWindow add on";
               Name        = "MyWindow";
          }
     }

     // Will be called as a new NTWindow is created. It will be
called in the thread of that window
     protected override void OnWindowCreated(Window window)
     {
          // We want to place our add on in the Control Center's
menus
          ControlCenter cc = window as ControlCenter;
          if (cc == null)
               return;

          /* Determine we want to place our add on in the Control
Center's "New" menu
          Other menus can be accessed via the control's Automation
ID. For example: toolsMenuItem,
          workspacesMenuItem, connectionsMenuItem, helpMenuItem. */
          existingMenuItem =
cc.FindFirst("ControlCenterMenuItemNew") as NTMenuItem;
          if (existingMenuItem == null)
               return;

          // 'Header' sets the name of our add on seen in the menu
structure
          myMenuItem = new NTMenuItem { Header = "My Menu Item",
               Style =
Application.Current.TryFindResource("MainMenuItem") as Style };

          // Place our add on into the "New" menu
          existingMenuItem.Items.Add(myMenuItem);

          // Subscribe to the event for when the user presses our
add on's menu item
          myMenuItem.Click += OnMenuItemClick;
     }

     // Open our add on's window when the menu item is clicked on
     private void OnMenuItemClick(object sender, RoutedEventArgs e)
     {
          // Show the NTWindow "MyWindow"
          Core.Globals.RandomDispatcher.InvokeAsync(new Action(()=>
```

11.6.3.23 OnWindowDestroyed()

Definition

This method is called whenever a new NTWindow is destroyed. It will be called in the thread of that window. A window is destroyed either by the user closing the window, closing a workspace, or on a shut down of NinjaTrader.

> **Note**: This method will also be called on a **recompile** of the NinjaTrader.Custom project (e.g., when you compile an indicator, strategy, or add-on)

Method Return Value

This method does not return a value

Syntax

```
OnWindowDestroyed(Window window)
```

Parameters

window	A Window object which is being removed from the workspace

Examples

```
public class MyWindowAddOn : AddOnBase
{
     private NTMenuItem myMenuItem;
     private NTMenuItem existingMenuItem;

     protected override void OnStateChange()
     {
          if (State == State.SetDefaults)
          {
               Description = "Our custom MyWindow add on";
               Name        = "MyWindow";
          }
     }

     // Will be called as a new NTWindow is destroyed. It will be
called in the thread of that window
     protected override void OnWindowDestroyed(Window window)
     {
          if (myMenuItem != null && window is ControlCenter)
          {
               if (existingMenuItem != null &&
existingMenuItem.Items.Contains(myMenuItem))
                    existingMenuItem.Items.Remove(myMenuItem);

               myMenuItem.Click -= OnMenuItemClick;
               myMenuItem = null;
          }
     }
}
```

11.6.3.24 OnWindowRestored()

Definition

Called when the window is restored from a workspace, which is called after OnWindowCreated(). This method is used to recall any custom XElement data from the workspace by referencing a window. Please also see OnWindowSaved() for information on how to store custom XElement data when a window is saved.

Method Return Value

This method does not return a value

Syntax

```
OnWindowRestored(Window window, XElement element)
```

Parameters

window	A Window object which is being restored from a workspace
element	The XElement object representing the workspace being restored

Examples

```
protected override void OnWindowRestored(Window window, XElement
element)
{
    Print("OnWindowRestored for " + window.GetHashCode());

    // locate the worksapces "SampleAddOn" elemenet which was
created and saved earlier using the OnWindowSaved() method
    XElement sampleAddOnElement = element.Element("SampleAddOn");

    // do not do anything if that element does not exist
    if (sampleAddOnElement == null)
     return;

     // loop through all the contents of the "SampleAddOn" element
    foreach (XElement content in sampleAddOnElement.Elements())
    {
        // find the "ButtonState" content, restore it's value and
set that as our tracked buttonState
        if (content.Name == "ButtonState")
        {
            bool buttonState = false;
            bool.TryParse(content.Value, out buttonState);
             continue;
        }
        //Parse additional elements here
    }

    //Don't forget to call the base OnWindowRestored method after
you're done.
    base.OnWindowRestored(window, element);
}
```

11.6.3.25 OnWindowSaved()

Definition
Called when the window is saved to a workspace, which is called before OnWindowDestroyed(). This method is used to save any custom XElement data associated with your window.

Method Return Value
This method does not return a value

Syntax
```
OnWindowSaved(Window window, XElement element)
```

Parameters

window	A Window object which is being saved to the workspace
element	A XElement object representing the workspace being saved

Examples

```
protected override void OnWindowSaved(Window window, XElement
element)
{
    Print("OnWindowSaved for " + window.GetHashCode());

    // create a new XElement to save the last state of a custom
button to the workspace
    XElement xml = new XElement("SampleAddOn", new
XElement("ButtonState", true));

    // e.g.,
    // <SampleAddOn>
    //  <ButtonState>true</ButtonState>
    // </SampleAddOn>

    // add the new element to the workspace which can be restored
later
    element.Add(xml);

    //Don't forget to call the base OnWindowSaved method after
you've finished your operation.
    base.OnWindowSaved(window, element);
}
```

11.6.3.26 StartAtmStrategy()

Definition

StartAtmStrategy can be used to submit entry orders with ATM strategies.

Syntax

NinjaTrader.NinjaScript.AtmStrategy.StartAtmStrategy(AtmStrategy *atmStrategyTemplate*, Order *entryOrder*)
NinjaTrader.NinjaScript.AtmStrategy.StartAtmStrategy(string *atmStrategyTemplateName*, Order *entryOrder*)

Properties

atmStrategyTemplate	An AtmStrategy representing the ATM strategy you wish to use
atmStrategyTemplateName	A string representing the name of the ATM strategy you wish to use
entryOrder	An Order representing the entry order

Critical: The "name" argument on the CreateOrder() method **MUST** be named "Entry" for the ATM Strategy to be started successfully.

Example

```
/* Example of starting an ATM strategy from an Add On window. The
concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private Account account;
     private Order entryOrder;

     public MyAddOnTab()
     {
          // Find our Sim101 account
          lock (Account.All)
               account = Account.All.FirstOrDefault(a => a.Name ==
"Sim101");

          if (account != null)
          {
               entryOrder =
account.CreateOrder(Cbi.Instrument.GetInstrument("AAPL"),
OrderAction.Buy, OrderType.Market,
                    TimeInForce.Day, 1, 0, 0, string.Empty,
"Entry", null);

               // Submits our entry order with the ATM strategy
named "myAtmStrategyName"
               NinjaTrader.NinjaScript.AtmStrategy.StartAtmStrategy
("myAtmStrategyName", entryOrder);
          }
     }

     // Other required NTTabPage members left out for demonstration
purposes. Be sure to add them in your own code if building an Add
On window.
}
```

11.6.3.27 StrategyBase

StrategyBase contains properties and methods for managing a Strategy object, and is the base class from which AtmStrategy derives.

Note: For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

Example

```
// A button called acctStratButton in an NTTabPage displays all ATM and NinjaScript strategies configured on a selected Account when clicked
private void OnButtonClick(object sender, RoutedEventArgs e)
{
    Button button = sender as Button;

    if (button != null && ReferenceEquals(button, acctStratButton))
    {
        // When the button is pressed, iterate through all ATM and NinjaScript strategies
        // This comprises all which are active, recovered upon last connect, or deactived since last connect
        // First, lock the Strategies collection to avoid in-flight changes to the collection affecting our output
        lock (accountSelector.SelectedAccount.Strategies)
            // Iterate through the Strategies collection in the selected Account
            foreach (StrategyBase strategy in accountSelector.SelectedAccount.Strategies)
                outputBox.AppendText(string.Format("{0}Name: {1}{0}ATM Template Name: {2}{0}Instrument: {3}{0}State: {4}{0}Category: {5}{0}",
                    Environment.NewLine,
                    strategy.Name,
                    strategy.Template,
                    strategy.Instruments[0].FullName,
                    strategy.State,
                    strategy.Category));
    }
}
```

11.6.3.28 PropagateInstrumentChange()

Definition

In an NTWindow, PropagateInstrumentChange() sends an Instrument to other windows with the same Instrument Linking color configured.

Notes:

- A public Instrument property must be defined in order to use PropagateInstrumentChange(), as in the example below
- For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

Example

```
// IInstrumentProvider member. Required if you want to use the
instrument link mechanism on an NTWindow.
public Cbi.Instrument Instrument
{
    get { return instrument; }
    set
    {
        // Process logic related to switching instruments, such as:
        // Unsubscribe to subscriptions to old instruments...
        // Subscribe for the new instrument...
        // Change the value displayed in an Instrument Selector in
the NTWindow...
        // Update the tab header name on AddOnFramework to be the
same name as the new instrument...
        // etc...

        // Send instrument to other windows linked to the same
color
        PropagateInstrumentChange(value);
    }
}
```

11.6.3.29 PropagateIntervalChange()

Definition

In an NTWindow, PropagateIntervalChange() sends an interval to other windows with the same Interval Linking color configured.

Notes:

1. A public Instrument property must be defined in order to use PropagateInstrumentChange(), as in the example below
2. For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

Example

```
// This custom method will be fired when an interval selector in a
custom NTTabPage changes intervals
private void OnIntervalChanged(object sender, BarsPeriodEventArgs
args)
{
    if (args.BarsPeriod == null)
        return;

    PropagateIntervalChange(args.BarsPeriod);
}
```

11.6.3.30 TabControl

Definition

The TabControl class provides functionality for working with NTTabPage objects within an NTWindow. TabControl should be instantiated within the constructor for an NTWindow instance, in order to configure the window to be able to host and work with tabs.

Note: For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

Example

In the example below, we define an instance of NTWindow, then use TabControl to accomplish various setup tasks:

- Provide the NTWindow with the ability to add, remove, and move tabs
- Attach a Factory to the TabControl to handle logic for creating new tabs
- Set up the TabControl with the ability to utilize window linking

```
public class MyWindow : NTWindow, IWorkspacePersistence
{
    public MyWindow()
    {
        // TabControl should be created for window content if tab
features are wanted
        TabControl tc = new TabControl();

        // Attached properties defined in the TabControlManager
class should be set to add, remove, or move tabs
        TabControlManager.SetIsMovable(tc, true);
        TabControlManager.SetCanAddTabs(tc, true);
        TabControlManager.SetCanRemoveTabs(tc, true);

        // if the ability to add new tabs is desired, TabControl
must have attached property "Factory" set.
        TabControlManager.SetFactory(tc, new MyWindowFactory());
        Content = tc;

        /* In order to have link buttons functionality, tab control
items must be derived from Tools.NTTabPage
        They can be added using extention method
AddNTTabPage(NTTabPage page) */
        tc.AddNTTabPage(new MyTab());
    }
}

/* Class which implements Tools.INTTabFactory must be created and
set as an attached property for TabControl
in order to use tab page add/remove/move/duplicate functionality */
public class MyWindowFactory : INTTabFactory
{
    // INTTabFactory member. Required to create parent window
    public NTWindow CreateParentWindow()
    {
        return new MyWindow();
    }

    // INTTabFactory member. Required to create tabs
    public NTTabPage CreateTabPage(string typeName, bool isTrue)
    {
        return new MyTab();
    }
}
```

11.6.3.31 TabControlManager

Definition

The TabControlManager class can be used to set or check several properties of a TabControl object. Rather than instantiating a TabControlManager object, you can use the public static methods of the class to set specific properties for a specified TabControl, as in the example code below.

> **Note**: For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

Setters

SetCanAddTabs(DependencyObject obj, bool value)	Sets a TabControl can add new tabs
SetCanDuplicateTabs(DependencyObject obj, bool value)	Sets a TabControl can duplicate tabs in new tabs or new windows
SetCanRemoveTabs(DependencyObject obj, bool value)	Sets a TabControl can remove tabs
SetFactory(DependencyObject obj, bool value)	Sets the NTTabFactory for the TabControl
SetIsSimulation(DependencyObject obj, bool value)	Sets the current NTTabPage to the simulation color if true. This method needs to be used logically from within your NTTabPage.
SetIsMovable(DependencyObject obj, bool value)	Sets a TabControl allows changing the order of tabs in a window

Getters

GetCanAddTabs(DependencyObject obj)	Indicates a TabControl can add new tabs

GetCanDuplicateTabs(DependencyObject obj)	Indicates a TabControl can duplicate tabs in new tabs or new windows
GetCanRemoveTabs(DependencyObject obj)	Indicates a TabControl can remove tabs
GetFactory(DependencyObject obj)	Obtains the NTTabFactory used by a TabControl
GetIsSimulation(DependencyObject obj)	Indicates the Simulation Color selected in the Options menu is visible in the tab background when a simulation account is selected in the tab
GetIsMovable(DependencyObject obj)	Indicates a TabControl allows changing the order of tabs in a window

Example

```
public AddOnFrameworkWindow()
{
    // TabControl should be created for window content if tab
features are wanted
    TabControl tc = new TabControl();

    // Attached properties defined in TabControlManager class
should be set to achieve tab moving, adding/removing tabs
    TabControlManager.SetIsMovable(tc, true);
    TabControlManager.SetCanAddTabs(tc, true);
    TabControlManager.SetCanRemoveTabs(tc, true);

    // if ability to add new tabs is desired, TabControl has to
have attached property "Factory" set.
    TabControlManager.SetFactory(tc, new
AddOnFrameworkWindowFactory());
    Content = tc;

    /* In order to have link buttons functionality, tab control
items must be derived from Tools.NTTabPage
    They can be added using extention method AddNTTabPage(NTTabPage
page) */
    tc.AddNTTabPage(new AddOnFrameworkTab());

}
```

11.6.4 Bars Type

Creating custom Bars Types allows for incredible flexibility in the way you want to present data in a chart. The methods and properties covered in this section are unique to custom Bars Type development.

Methods and Properties

AddBar()	Adds new data points for the Bars Type.
ApplyDefaultBase PeriodValue	Sets the default base values used for the BarsPeriod selected by the user (e.g., the default PeriodValue, DaysToLoad, etc.) for your custom Bar Type.
ApplyDefaultValue	Sets the default BarsPeriod values used for a custom Bar Type.

BuiltFrom	Determines the base dataset used to build the **BarsType** (i.e., Tick, Minute, Day).
GetInitialLookBackDays()	Determines how many days of data load when a user makes a "bars back" data request.
GetPercentComplete()	Determines the value your BarsType would return for Bars.PercentComplete
Icon	The shape which displays next to the Bars Type menu item.
IsRemoveLastBarSupported	Determines if the bars type can use the RemoveLastBar() method when **true**, otherwise an exception will be thrown.
IsTimebased	Used to indicate the **BarsType** is built from time-based bars (day, minute, second).
OnDataPoint()	**OnDataPoint()** method is where you should adjust data points (bar values) of your series through AddBar() and UpdateBar().
RemoveLastBar()	Removes the last data point for the Bars Type.
SessionIterator	Provides trading session information to the bars type. Must be built using the bars object.
UpdateBar()	Updates a data point in our Bars Type.

11.6.4.1 AddBar()

Definition

Adds new data points for the Bars Type.

Syntax

```
AddBar(Bars bars, double open, double high, double low, double close, DateTime time,
long volume)
AddBar(Bars bars, double open, double high, double low, double close, DateTime time,
long volume, double bid, double ask)
```

Parameters

bars	The Bars object of your bars type

open	A `double` value representing the open price
high	A `double` value representing the high price
low	A `double` value representing the low price
close	A `double` value representing the close price
time	A `DateTime` value representing the time
volume	A `long` value representing the volume
bid	A `double` value representing the bid price
ask	A `double` value representing the ask price

Examples

```
AddBar(bars, open, high, low, close, time, (long)
Math.Min(volumeTmp, bars.BarsPeriod.Value));
```

11.6.4.2 ApplyDefaultBasePeriodValue

Definition

Sets the default base values used for the BarsPeriod selected by the user (e.g., the default PeriodValue, DaysToLoad, etc.) for your custom Bar Type.

Method Return Value

This method does not return a value.

Parameters

period	The BarsPeriod chosen by the user when utilizing this Bars type

Syntax

You must override the method in your Bars Type with the following syntax:

```
public override void ApplyDefaultBasePeriodValue(BarsPeriod period)
{
```

```
}
```

Examples

```
public override void ApplyDefaultBasePeriodValue(BarsPeriod period)
{
   //sets the default Minute bars period value to 1, and days to
load to 5
    if (period.BaseBarsPeriodType == BarsPeriodType.Minute)
    {
       period.BaseBarsPeriodValue = 1;
       DaysToLoad = 5;
    }
   //sets the default Tick bars period value to 150, and days to
load to 3
    else if (period.BaseBarsPeriodType == BarsPeriodType.Tick)
    {
        period.BaseBarsPeriodValue = 150;
        DaysToLoad = 3;
    }

}
```

11.6.4.3 ApplyDefaultValue

Definition

Sets the default BarsPeriod values used for a custom Bar Type.

Method Return Value

This method does not return a value.

Parameters

period	The BarsPeriod chosen by the user when utilizing this Bars type

Syntax

You must override the method in your Bars Type with the following syntax:

```
public override void ApplyDefaultValue(BarsPeriod period)
{
```

```
}
```

Examples

```
public override void ApplyDefaultValue(BarsPeriod period)
{
	period.BarsPeriodTypeName = "MyBarType";
	period.Value = 1;
}
```

11.6.4.4 BuiltFrom

Definition

Determines the base dataset used to build the **BarsType** (i.e., Tick, Minute, Day). The **BuiltFrom** property will control the frequency in which OnDataPoint() processes historical data.

Property Value

A BarsPeriodType enum. Values that will be recognized include:

- BarsPeriodType.Tick
- BarsPeriodType.Minute
- BarsPeriodType.Day

Warning: Using other bars period types (e.g., Range, Volume, or other custom bars types) is **NOT** supported. The **BarsPeriodType** values mentioned above represent all of the fundamental data points needed to build a bar.

Syntax

BuiltFrom

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name      = "MyCustomBarsType";
      BarsPeriod    = new BarsPeriod { BarsPeriodType =
(BarsPeriodType) 15, BarsPeriodTypeName = "MyCustomBarsType(15)",
Value = 1 };
      BuiltFrom    = BarsPeriodType.Minute;  // update
OnDataPoint() every minute on historical data
      DaysToLoad    = 5;
   }

   else if (State == State.Configure)
   {
   }
}
```

11.6.4.5 DefaultChartStyle

Definition

Allows to set a default ChartStyle for usage with a NinjaTrader bars type

Property Value

A `ChartStyleType enum` value representing the ChartStyle to be set as default. System defaults include:

- ChartStyleType.Box,
- ChartStyleType.CandleStick,
- ChartStyleType.LineOnClose,
- ChartStyleType.OHLC,
- ChartStyleType.PointAndFigure,
- ChartStyleType.KagiLine,
- ChartStyleType.OpenClose,
- ChartStyleType.Mountain

Syntax

`DefaultChartStyle`

Examples

```
protected override void OnStateChange()
{
        if (State == State.SetDefaults)
        {
            Name                        = "SampleBarsType";
            BarsPeriod                  = new BarsPeriod
{ BarsPeriodType = (BarsPeriodType) 15, BarsPeriodTypeName =
"SampleBarsType(15)", Value = 1 };
            BuiltFrom                   = BarsPeriodType.Minute;
            DaysToLoad                  = 5;
            DefaultChartStyle           =
Gui.Chart.ChartStyleType.CandleStick;
            IsIntraday                  = true;
         }
}
```

11.6.4.6 GetInitialLookBackDays()

Definition

Determines how many days of data load when a user makes a "bars back" data request.

Method Return Value

This method returns an int value.

Method Parameters

barsPeriod	The bars period chosen by the user when utilizing this Bars type
tradingHours	The trading hours chosen by the user when utilizing this Bars type
barsBack	The bars back chosen by the user when utilizing this Bars type

Syntax

You must override the method in your Bars Type with the following syntax.

```
public override int GetInitialLookBackDays(BarsPeriod barsPeriod, TradingHours
tradingHours, int barsBack)
{

}
```

Examples

```
public override int GetInitialLookBackDays(BarsPeriod barsPeriod,
TradingHours tradingHours, int barsBack)
{
     // Returns the minimum number of days needed to successfully
load the number
     // of bars back requested for a monthly Bars type
     return (int) barsPeriod.Value * barsBack * 31;
}
```

Tip: Try to request an amount of data that is just right for what is needed. Requesting too large a data set will result in unnecessary data being loaded. Requesting too small a data set will result in multiple requests being done.

11.6.4.7 GetPercentComplete()

Definition

Determines the value your BarsType would return for Bars.PercentComplete

Method Return Value

This method returns a double value.

Method Parameters

bars	The bars object chosen by the user when utilizing this Bars type
now	The DateTime value to measure

Syntax

You must override the method in your Bars Type with the following syntax.

```
public override double GetPercentComplete(Bars bars, DateTime now)
{

}
```

Examples

```
public override double GetPercentComplete(Bars bars, DateTime now)
{
     // Calculate the percent complete for our monthly bars
     if (now.Date <= bars.LastBarTime.Date)
     {
          int month = now.Month;
          int daysInMonth = (month == 2) ?
(DateTime.IsLeapYear(now.Year) ? 29 : 28) :
               (month == 1 || month == 3 || month == 5 || month ==
7 || month == 8 || month == 10 || month == 12 ? 31 : 30);
          return (daysInMonth -
(barsSeries.LastBarTime.Date.AddDays(1).Subtract(now).TotalDays /
barsSeries.BarsPeriod.Value)) /
               daysInMonth; // an estimate
     }
     return 1;
}
```

11.6.4.8 Icon

Definition

The shape which displays next to the Bars Type menu item. Since this is a standard object, any type of icon can be used (unicode characters, custom image file resource, geometry path, etc).

For more information on using images to create icons, see the Using Images with Custom Icons page.

> **Note**: When using UniCode characters, first ensure that the desired characters exist in the icon pack for the font family used in NinjaTrader.

Property Value

A generic virtual `object` representing the drawing tools menu icon. This property is read-only.

Syntax

You must override this property using the following syntax:

```
public override object Icon
```

Examples

```
public override object Icon
{
   get
   {
      //use a unicode character as our string which will render an
arrow
      string uniCodeArrow = "\u279A";
      return uniCodeArrow;
   }
}
```

11.6.4.9 IsRemoveLastBarSupported

Definition

Determines if the bars type can use the RemoveLastBar() method when **true**, otherwise an exception will be thrown. **Bar Types** which use remove last bar concepts **CANNOT** be used with Tick Replay, and as a result **Tick Replay** will be disabled on the UI when **IsRemoveLastBarSupported** is set to true.

> **Note**: This property is read-only, but may be overridden in a custom bar type.

Syntax

`IsRemoveLastBarSupported`

Property value

A `bool` determining if the BarsType can remove the last; default value is **false.**

Examples

```
// allows RemoveLastBar() to be called
public override bool IsRemoveLastBarSupported { get { return
true; } }
```

11.6.4.10 IsTimeBased

Definition

Used to indicate the **BarsType** is built from time-based bars (day, minute, second). Setting this property on a custom bar type is useful for correct calculations from many core data and session logic, and can also be used by 3rd party NinjaScript objects to determine how to interact with the bars.

Property Value

A `bool` which when **true** tells other objects the bars are built from time; default set to **false.**

Syntax

```
Bars.IsTimeBased
```

Examples

Setting the IsTimeBased defaults in a custom BarsType

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name      = "Custom BarsType";
      IsTimeBased   = true; // indicates to the core the these bars
are built using time.
   }
}
```

Reading IsTimeBased from a custom NinjaScript object

```
protected override void OnBarUpdate()
{
   // include milliseconds time stamps for tick based bars
   string timeFormat = "HH:mm:ss:fff";

   if (Bars.BarsType.IsTimeBased)
   {
      // on time based bars, only format up to "seconds"
      timeFormat = "HH:mm:ss";
   }
   // format string based on the appropriate time format
   Print(Time[0].ToString(timeFormat));
}
```

11.6.4.11 OnDataPoint()

Definition

Called for each record in the corresponding base dataset used to build the **BarType** (i.e., for every tick, minute, or day). The **OnDataPoint()** method is where you should adjust data points (bar values) of your series through AddBar() and UpdateBar(). See also the BuiltFrom property.

Notes:

1. Historical data processing receives a single update for every base bar determined by the **BuiltFrom** property

2. When using TickReplay, historical updates will call for every tick handled by the core regardless of the **BuiltFrom** property defined
3. Once transitioned to real-time, updates will call on every tick processed by the core
4. The bid/ask parameters will **ONLY** be available historically when using Tick Replay, unless you are using a 1-tick series
5. isBar could be true in case the BarsSeries was internally copied to another BarsSeries and is only needed for IsTimeBased = true BarsTypes (e.g. Second/Minute/Day...).

Method Return Value

This method does not return a value.

Method Parameters

bars	The Bars object of your bars type
open	A `double` value representing the open price
high	A `double` value representing the high price
low	A `double` value representing the low price
close	A `double` value representing the close price
time	A `DateTime` value representing the time
volume	A `long` value representing the volume
isBar	A `bool` value representing if OnDataPoint should treat the timestamp as an already built bar instead of an intrabar timestamp.
bid	A `double` value representing the bid price
ask	A `double` value representing the ask price

Syntax

You must override the method in your Bars Type with the following syntax.

```
protected override void OnDataPoint(Bars bars, double open, double high, double low,
double close,
```

```
    DateTime time, long volume, bool isBar, double bid, double ask)
{

}
```

Examples

```
protected override void OnDataPoint(Bars bars, double open, double
high, double low,
     double close, DateTime time, long volume, bool isBar, double
bid, double ask)
{
     int minIndex;

     // Create the first data point of our series
     if (bars.Count == 0)
     {
          minIndex = 0;
          AddBar(bars, open, high, low, close, TimeToBarTime(time,
(int) bars.BarsPeriod.Value), volume);
     }
     // Update our data point with the latest information
     else if ((time.Month <= bars.LastBarTime.Month && time.Year ==
 bars.LastBarTime.Year) || time.Year < bars.LastBarTime.Year)
     {
          if (high != bars.GetHigh(bars.Count - 1) || low !=
bars.GetLow(bars.Count - 1) ||
               close != bars.GetClose(bars.Count - 1) || volume >
0)
          {
               minIndex = bars.Count - 1;
               UpdateBar(bars, high, low, close, bars.LastBarTime,
volume);
          }
          else
               minIndex = -1;
     }
     // Add new data points
     else
     {
          minIndex = bars.Count;
          AddBar(bars, open, high, low, close, time, (long)
Math.Min(volumeTmp, bars.BarsPeriod.Value));
     }
     FirstBarAmended = minIndex;
}
```

11.6.4.12 RemoveLastBar()

Definition
Removes the last data point for the Bars Type. There may be cases where your custom bar type may need to amend the last values added on a bar that has already closed. Calling **RemoveLastBar()** will remove the last points for that bar type and allow you to call **AddBar()** with the updated values.

Notes:
- In order to use this method, the IsRemoveLastBarSupported method must be **true.**
- RemoveLastBar() **CANNOT** be used with TickReplay

Syntax
```
RemoveLastBar(Bars bars)
```

Parameters

bars	The Bars object of your bars type

Examples

```
RemoveLastBar(bars);
```

11.6.4.13 SetPropertyName

Definition
Sets a default property name to a custom string to be displayed on the UI.

Method Return Value
This method does not return a value.

Syntax
```
SetPropertyName(string propertyName, string displayName)
```

Method Parameters

propertyName	A `string` representing the property to be renamed. Possible values include: • UpBrush • DownBrush • BarWidth

	• Stroke • Stroke2 • Value • Value2 • BaseBarsPeriodType • BaseBarsPeriodValue • PointAndFigurePriceType • ReversalType
displayName	A `string` representing the desired property name

Example

```
protected override void OnStateChange()
{
    if (State == State.Configure)
    {
        Properties.Remove(Properties.Find("Stroke", true));
        Properties.Remove(Properties.Find("Stroke2", true));

        SetPropertyName("UpBrush", "AdvanceBar");
        SetPropertyName("DownBrush", "DeclineBar");
    }
}
```

Note: If you do not wish to use specific properties accessible via `SetPropertyName()`, you will need to remove them from the list via `Properties.Remove`, as shown in the example above.

11.6.4.14 SessionIterator

Definition

Provides trading session information to the bars type. Must be built using the bars object.

Property Value

A SessionIterator object which is used to to calculate trading day/session information.

Syntax

`SessionIterator`

Examples

```
protected override void OnDataPoint(Bars bars, double open, double
high, double low, double close, DateTime time, long volume, bool
isBar, double bid, double ask)
{
   // build a session iterator from the bars object being updated
   if (SessionIterator == null)
      SessionIterator = new SessionIterator(bars);

   // check if we are in a new trading session based on the trading
hours selected by the user
   bool isNewSession = SessionIterator.IsNewSession(time, isBar);

   // calculate the new trading day
   if (isNewSession)
      SessionIterator.CalculateTradingDay(time, isBar);

   Print(SessionIterator.ActualTradingDayExchange);

}
```

11.6.4.15 UpdateBar()

Definition

Updates a data point in our Bars Type.

Syntax

UpdateBar(Bars *bars*, double *high*, double *low*, double *close*, DateTime *time*, long *volumeAdded*)

Parameters

bars	The Bars object of your bars type
high	A double value representing the high price
low	A double value representing the low price
close	A double value representing the close price
time	A DateTime value representing the time
volume	A long value representing the volume

Examples

```
UpdateBar(bars, high, low, close, time, volume);
```

11.6.5 Chart Style

Custom Chart Styles can be used on charts to present bars information in a different visual representation. The methods and properties covered in this section are unique to custom Chart Style development. Following is an index of properties and methods documented for Chart Styles.

Methods and Properties

BarWidth	The painted width of a ChartStyle bar
BarWidthUI	The Bar width value which displays on the UI
ChartStyleType	Defines a unique identifier value used to register a custom ChartStyle
DownBrush	A Brush object used to determine the color to paint the down bars for the ChartStyle
DownBrushDX	A SharpDX.Brush object used to paint the down bars for the ChartStyle
GetBarPaintWidth()	Returns the painted width of the chart bar
IsTransparent	Indicates the bars in the ChartStyle are transparent
OnRender()	An event driven method used to render content to a ChartStyle
SetPropertyName()	Sets a default property name to a custom string to be displayed on the UI
TransformBrush()	Scales a non-solid color brush used for rendering the chart style to properly display in NinjaTrader

UpBrush	A Brush object used to determine the color to paint the up bars for the ChartStyle
UpBrush DX	A SharpDX.Brush object used to paint the up bars for the ChartStyle

11.6.5.1 BarWidth

Definition
The painted width of a ChartStyle bar. This value will updated as the ChartControl is resized.

Property Value
A `double` value representing the current width the chart bars

Syntax
`BarWidth`

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name              = "Example ChartStyle";
      ChartStyleType    = (ChartStyleType) 52;
      BarWidth          = 1;
   }
}
```

11.6.5.2 BarWidthUI

Definition
The Bar width value which displays on the UI. This value will be rounded from the internal BarWidth property which is updated as the ChartControl is resized

Property Value
A `int` value representing the width of the chart bars which can be set by a user.

Syntax
`BarWidthUI`

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale, ChartBars chartBars)
{

   int  barWidth  = GetBarPaintWidth(BarWidthUI);

}
```

11.6.5.3 ChartStyleType

Definition

Defines a unique identifier value used to register a custom ChartStyle. There are 11 default ChartStyles which come with NinjaTrader which are reserved per the table on this page under the Parameters section of this page.

Note: The ChartStyle property can allow a large number of ChartStyles to be registered on a single user's installation (up to 2,147,483,647). However it's important to note that it is still possible for two installed ChartStyles on a user's computer to conflict should they be register to the same enumerator value. In this case, NinjaTrader will ignore the conflicting ChartStyle type and information pertaining to this conflict will be displayed on the Log tab of the NinjaTrader Control Center.

Added 1/31/2018 : We advise users to use values larger then 1023 when selecting an enum. As NinjaTrader from time to time may add a new enum value in that range which may cause conflicts.

Property Value

A `enum` value representing the ChartStyle to be registered.

Tip: It is recommended to pick high, unique enumeration value to avoid conflict from other ChartStyles that may be used by a single installation.

Syntax

You must cast `ChartStyleType` from an `int` using the following syntax:

```
(ChartStyleType) 80;
```

Parameters

Reserved enumeration values are listed below:

0	Box

1	CandleStick
2	LineOnClose
3	OHLC
4	PointAndFigure
5	KagiLine
6	OpenClose
7	Mountain
8	Volumetric
9	HollowCandleStick
10	Equivolume

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name           = "Example ChartStyle";
      ChartStyleType = (ChartStyleType) 80;
      BarWidth       = 1;
   }
}
```

11.6.5.4 DownBrush

Definition

A Brush object used to determine the color to paint the down bars for the ChartStyle.

Note: This Windows Presentation Forms (WPF) implementation of the Brush class is not directly used to paint bars on the chart. Instead it is converted to a SharpDX Brush in the DownBrushDX property. This property is used to capture user input for changing brush colors.

Property Value
A WPF `Brush` object used to paint the down bars

Syntax
`DownBrush`

Example

```
protected override void OnStateChange()
{
    if (State == State.Configure)
    {
        // Set a new name for the DownBrush property
        SetPropertyName("DownBrush", "DecliningBrush");
    }
}
```

11.6.5.5 DownBrushDX

Definition
A SharpDX Brush object used to paint the down bars for the ChartStyle.

Property Value
A SharpDX `Brush` object used to paint the down bars

Syntax
`DownBrushDX`

Example

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale, ChartBars chartBars)
{
    for (int idx = chartBars.FromIndex; idx <= chartBars.ToIndex;
idx++)
        {
            double      closeValue              =
bars.GetClose(idx);
            double      openValue               =
bars.GetOpen(idx);

            // Set the brush of the current candle to UpBrushDX or
DownBrushDX, depending on the
            // bar direction
            Brush brush = closeValue >= openValue ? UpBrushDX :
DownBrushDX;
        }
}
```

11.6.5.6 GetBarPaintWidth()

Definition

Returns the painted width of the chart bar. The GetBarPintWidth() method will return a minimum value of 1.

> **Note**: This is an abstract method which is **required** to compile a ChartStyle object. If you do not plan on recalculating a barWidth, simply return the default barWidth parameter which is passed in this method. Please see the Examples section of this page for more information.

Method Return Value

An `int` value

Syntax

You must over ride this method using the following syntax:

```
public override int GetBarPaintWidth(int barWidth)
{

}
```

Method Parameters

barWidth	An `int` value representing the

	current width of the bar to calculate

Examples

Returning the default barWidth

```
public override int GetBarPaintWidth(int barWidth)
{
     return barWidth
}
```

Calculating and returning a new barWidth from the original barWidth

```
public override int GetBarPaintWidth(int barWidth)
{
   // calculate a new bar width
   return 1 + 2 * (barWidth - 1) + 2 * (int)
Math.Round(Stroke.Width);
}
```

11.6.5.7 Icon

Definition

The shape which displays next to the Chart Style menu item. Since this is a standard object, any type of icon can be used (unicode characters, custom image file resource, geometry path, etc).

For more information on using images to create icons, see the Using Images with Custom Icons page.

> Note: When using UniCode characters, first ensure that the desired characters exist in the icon pack for the font family used in NinjaTrader.

Property Value

A generic virtual `object` representing the drawing tools menu icon. This property is read-only.

Syntax

You must override this property using the following syntax:

```
public override object Icon
```

Examples

```
public override object Icon
{
   get
   {
      //use a unicode character as our string which will render an
arrow
      string uniCodeArrow = "\u279A";
      return uniCodeArrow;
   }
}
```

11.6.5.8 IsTransparent

Definition

Indicates the bars in the ChartStyle are transparent.

Property Value

A `bool` which, when `true`, indicates that the UpBrush, DownBrush, and Stroke.Brush are all set to transparent. Returns `false` if any of the three are not transparent.

Syntax

`IsTransparent`

Example

```
protected override void OnStateChange()
{
    if (State == State.Configure)
    {
        //Print a message if the UpBrush, DownBrush, and
Stroke.Brush are all transparent
        if (IsTransparent)
            Print("All bars are currently set to transparent");
    }
}
```

11.6.5.9 OnRender()

Definition
An event driven method used to render content to a ChartStyle. The OnRender() method is called every time the chart values are updated. These updates are driven by incoming data to the chart bars or by a user manually interacting with the chart control or chart scale.

Method Return Value
This method does not return a value.

Syntax
You must override the method in your ChartStyle with the following syntax:

```
protected override void OnRender(ChartControl chartControl, ChartScale chartScale,
ChartBars chartBars)
{

}
```

Method Parameters

chartControl	A ChartControl representing the x-axis
chartScale	A ChartScale representing the y-axis
chartBars	A ChartBars representing the Bars series for the chart

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale, ChartBars chartBars)
{
     // Rendering logic for our chart style
}
```

11.6.5.10 SetPropertyName()

Definition
Sets a default property name to a custom string to be displayed on the UI.

Method Return Value
This method does not return a value.

Syntax

```
SetPropertyName(string propertyName, string displayName)
```

Method Parameters

propertyName	A `string` representing the property to be renamed. Possible values include: • UpBrush • DownBrush • BarWidth • Stroke • Stroke2
displayName	A `string` representing the desired property name

Example

```
protected override void OnStateChange()
{
    if (State == State.Configure)
    {
        Properties.Remove(Properties.Find("Stroke", true));
        Properties.Remove(Properties.Find("Stroke2", true));

        SetPropertyName("UpBrush", "AdvanceBar");
        SetPropertyName("DownBrush", "DeclineBar");
    }
}
```

Note: If you do not wish to use specific properties accessible via `SetPropertyName()`, you will need to remove them from the list via `Properties.Remove`, as shown in the example above.

11.6.5.11 TransformBrush()

Definition

Scales a non-solid color brush used for rendering the chart style to properly display in NinjaTrader.

Note: This method has no impact on solid color brushes. You would only need to pass in either a linear or radial gradient brush.

Method Return Value
This method does not return a value.

Syntax
```
TransformBrush(SharpDX.Direct2D1.Brush brush, RectangleF rect)
```

Method Parameters

brush	A SharpDX.Direct2D1.Brush object representing the brush used to render
rect	A RectangleF structure representing the rectangle to be rendered

Examples
```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale, ChartBars chartBars)
{
     TransformBrush(brush, rect);
}
```

11.6.5.12 UpBrush

Definition
A Brush object used to determine the color to paint the up bars for the ChartStyle.

Note: This Windows Presentation Forms (WPF) implementation of the Brush class is not directly used to paint bars on the chart. Instead it is converted to a SharpDX Brush in the UpBrushDX property. This property is used to capture user input for changing brush colors.

Property Value
A WPF `Brush` object used to paint the up bars

Syntax

UpBrush

Examples

```
protected override void OnStateChange()
{
    if (State == State.Configure)
    {
        // Set a new name for the UpBrush property
        SetPropertyName("UpBrush", "AdvancingBrush");
    }
}
```

11.6.5.13 UpBrushDX

Definition

A SharpDX Brush object used to paint the up bars for the ChartStyle.

Property Value

A SharpDX `Brush` object used to paint the up bars

Syntax

UpBrushDX

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale, ChartBars chartBars)
{
    for (int idx = chartBars.FromIndex; idx <= chartBars.ToIndex;
idx++)
        {
            double      closeValue              =
bars.GetClose(idx);
            double      openValue               =
bars.GetOpen(idx);

            // Set the brush of the current candle to UpBrushDX or
DownBrushDX, depending on the
            // bar direction
            Brush brush = closeValue >= openValue ? UpBrushDX :
DownBrushDX;
        }
}
```

11.6.6 Drawing Tool

Custom Drawing Tools can be used to render custom shapes to a point on the chart to represent various information. The methods and properties covered in this section are unique to custom Drawing Tools development. Following is an index of the documented properties and methods related to drawing tools.

Methods and Properties

AddPastedOffset()	A virtual method which is called every time a Drawing Tool is copied and pasted to a chart
Anchors	Creates a collection of Chart Anchors which will represent various points of the drawing tool
AttachedTo	An object which holds information regarding where the drawing tool is attached
ChartAnchor	Defines objects used by Drawing Tools which represent a point on the chart where the Drawing Tool is located
ConvertToVerticalPixels	Used to convert the cursor position (pixels) to device pixels represented on the Y axis of the chart
CreateAnchor()	Used to create a new chart anchor at a specified mouse point
DisplayOnChartsMenus	Determines if the drawing tool should be listed in the chart's drawing tool menus
Dispose()	Releases any device resources used for the drawing tool
DrawingState	Represents the current state of the drawing tool in order to perform various actions, such as building, editing, or moving
DrawnBy	Represents the NinjaScript object by which the drawing tool was created
GetAttachedToChartBars()	Returns information which relate to the underlying bars series in which the drawing tool is attached

GetClosestAnchor()	Returns the closest chart anchor within a specified maximum distance from the mouse cursor
GetCursor()	An event driven method which is called when a chart object is selected
GetSelectionPoints()	Returns the chart object's data points where the user can interact
Icon	The shape which displays next to the drawing tool menu item
IgnoresSnapping	Determines if the drawing tool chart anchor's will use the chart's **Snap Mode** mouse coordinates
IgnoresUserInput	Determines if the drawing tool can be clicked on by the user
IsAttachedToNinjaScript	Indicates if the drawing tool is currently attached to a NinjaScript object (such an indicator or a strategy)
IsGlobalDrawingTool	Indicates if the drawing tool is currently set as a Global Drawing object
IsLocked	Determines if the drawing tool should be be locked in place
IsUserDrawn	Indicates if the drawing tool was manually drawn by a user
OnBarsChanged()	An event driven method which is called any time the underlying bar series have changed for the chart where the drawing tool resides
OnMouseDown()	An event driven method which is called any time the mouse pointer over the chart control has the mouse button pressed
OnMouseMove()	An event driven method which is called any time the mouse pointer is over the chart control and a mouse is moving

OnMouseUp()	An event driven method is called any time the mouse pointer is over the chart control and a mouse button is being released
SupportsAlerts	Indicates if the drawing tool can be used for manually configured alerts through the UI
ZOrderType	Determintes the order in which the drawing tool will be rendered

11.6.6.1 AddPastedOffset()

Definition

A virtual method which is called every time a DrawingTool is copied and pasted to a chart. The default behavior will offset the chart anchors price value down by 1, percent. However, this behavior can be overridden for your custom drawing tool if desired.

Method Return Value

This method does not return a value

Syntax

You must override this method using the following syntax:

```
public override void AddPastedOffset(ChartPanel panel, ChartScale chartScale)
{

}
```

Method Parameters

panel	A ChartPanel representing the the panel for the chart
chartScale	A ChartScale representing the Y-axis

Examples

```
public override void AddPastedOffset(ChartPanel chartPanel,
ChartScale chartScale)
{
   foreach (ChartAnchor anchor in Anchors)
   {
      //bump each anchor 1 minute to the right
      DateTime tmpTime = anchor.Time;
      anchor.Time = tmpTime.AddMinutes(1);
   }
}
```

11.6.6.2 Anchors

Definition

Returns a custom collection of ChartAnchors which will represent various points of the drawing tool.

> **Note**: You must declare this property with the chart anchors used in the drawing tool which you plan on using for iteration. Doing so will expose a simple enumerator which will allow you to to iterate over the chart anchors in which have been defined in this interface.

Property Value

A virtual IEnumerable interface consisting of ChartAnchors

Syntax

You must override this property using the following syntax:

```
public override IEnumerable<ChartAnchor> Anchors
{

}
```

Examples

```
//defines the chart anchors used for the drawing tool
public ChartAnchor      StartAnchor    { get; set; }
public ChartAnchor      MiddleAnchor   { get; set; }
public ChartAnchor      EndAnchor      { get; set; }

//create a collection of chart anchors used for a simple iteration
public override IEnumerable<ChartAnchor> Anchors
{
   get
   {
      return new[] { StartAnchor, MiddleAnchor, EndAnchor };
   }
}

//setup our chart anchor instances and assign a display name to
each
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {

      Name                  = "My Drawing Tool";

      StartAnchor           = new ChartAnchor();
      MiddleAnchor          = new ChartAnchor();
      EndAnchor             = new ChartAnchor();

      StartAnchor.DisplayName    = "My Start Anchor";
      MiddleAnchor.DisplayName   = "My Middle Anchor";
      EndAnchor.DisplayName      = "My End Anchor";

   }
}

//for each render pass, print out the display name of the chart
anchors
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   foreach (ChartAnchor anchor in Anchors)
   {
      Print(anchor.DisplayName);
   }
}
```

11.6.6.3 AttachedTo

Definition

An object which holds information regarding where the drawing tool is attached.

Available Properties

AttachedToType	An `enum` representing the type of object the drawing to is attached. Possible values are: • Bars - The chart bars of the parent chart • GlobalInstrument - The bars of an instrument crossed all charts • Indicator - A NinjaScript indicator • Strategy - A NinjaScript strategy
ChartObject	A ChartObject interface such an indicator, strategy, chart bars
DisplayName	A `string` value indicating the name of the object the drawing tool is attached
Instrument	The Instrument that the drawing tool is attached

Syntax

`AttachedTo`

Examples

```
if (AttachedTo.AttachedToType == AttachedToType.Indicator)
   // do something
```

11.6.6.4 ChartAnchor

Definition

Defines objects used by Drawing Tools which represent a point on the chart where the Drawing Tool is located.

Syntax

```
class ChartAnchor
```

Constructors

`new ChartAnchor()`	Initializes a new instance of the ChartAnchor object
`new ChartAnchor(DateTime time, double price, ChartControl chartControl)`	Initializes a new instance of the ChartAnchor object using time, price, and relative chart control
`new ChartAnchor(DateTime time, double yValue, int currentBar, ChartControl chartControl)`	Initializes a new instance of the ChartAnchor object using time, y-axis coordinates, current bar, and relative chart control

Methods and Properties

CopyDataValues()	Copies the ChartAnchor time and price values from on anchor to another
DisplayName	A string value which sets the name prefix used for all properties for a chart anchor
DrawingTool	The drawing tool which owns a chart anchor
DrawnOnBar	Gets the current bar value that the chart anchor is drawn by a NinjaScript object.
GetPoint()	Returns a chart anchor's data points.
IsBrowsable	A bool value determining the anchor is visible on the UI.
IsEditing	A bool value determining the anchor is currently being edited
IsNinjaScriptDrawn	Indicates if the chart anchor was drawn by a NinjaScript object

IsXPropertiesVisible	A bool value determining the X properties are visible on the UI
IsYPropertyVisible	A bool value determining the Y data value is visible on the UI
MoveAnchor()	Moves a Chart Anchor's x and y values from start point by a delta point amount.
MoveAnchorX()	Moves an anchor x values from start point by a delta point amount
MoveAnchorY()	Moves an anchor y values from start point by a delta point amount
Price	Determines price value the chart anchor is drawn.
SlotIndex	Indicates the nearest bar slot where anchor is drawn.
Time	Determines date/time value the chart anchor is drawn.
UpdateFromPoint()	Updates an anchor's x and y values from a given point (in device pixels)
UpdateXFromPoint()	Updates an anchor's X values from a given point (in device pixels)
UpdateYFromPoint()	Updates an anchor's Y value from a given point (in device pixels)

Examples

```
public ChartAnchor MyAnchor { get; set; }   // declares the
"MyAnchor" ChartAnchor object

public override IEnumerable<ChartAnchor> Anchors { get { return
new[] { MyAnchor }; } } //adds the "MyAnchor" ChartAnchor object to
a collection of anchors used to interact with your anchors

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Description = @"Drawing tool example";
      Name = "SampleDrawingTool";

      MyAnchor = new ChartAnchor();  //creates a new instances of
the ChartAnchor object
      MyAnchor.IsEditing   = true;
      MyAnchor.DrawingTool = this;
      MyAnchor.IsBrowsable = false;
   }
}

public override void OnMouseUp(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   if (DrawingState == DrawingState.Editing)
   {
      if (MyAnchor.IsEditing)
      {
         //if anchor is editing, update anchor point
         dataPoint.CopyDataValues(MyAnchor);
      }
   }
}
```

11.6.6.4.1 CopyDataValues()

Definition

Copies the ChartAnchor time and price values from on anchor to another. This includes the BarsAgo, SlotIndex, Time, Price, and DrawnOnBar values. This method is useful for updating a chart anchor to a recent data point when the user interacts with the drawing chart anchor.

Method Return Value

This method does not return a value.

Syntax
<chartAnchor>.CopyDataValues(ChartAnchor toAnchor)

Method Parameters

toAnchor	The ChartAnchor to copy

Examples

```
public override void OnMouseMove(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   // if the user is moving the draw object, copy the most recent
dataPoint to MyAnchor
   if (DrawingState == DrawingState.Moving)
      dataPoint.CopyDataValues(Anchor);
}
```

11.6.6.4.2 DisplayName

Definition
Sets the display name prefix used for all properties for a chart anchor.

Property Value
A `string` value that is used to identify the name for a corresponding anchor. Default value is **null**.

Syntax
<ChartAnchor>.DisplayName

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        MyAnchor = new ChartAnchor();
        MyAnchor.DisplayName = "MyChartAnchor";
    }
}
```

11.6.6.4.3 DrawingTool

Definition
The DrawingTool object which owns a chart anchor.

Property Value
A `IDrawingTool` object representing the owner of the chart anchor

Syntax
`<ChartAnchor>.DrawingTool`

Examples

```
protected override void OnStateChange()
{
	if (State == State.SetDefaults)
	{
		Name = "SampleDrawingTool";
		MyAnchor = new ChartAnchor();
		MyAnchor.DrawingTool =
		this
		; //NinjaTrader.NinjaScript.DrawingTools.SampleDrawingTool
	}
	else if (State == State.Configure)
	{

	}
}
```

11.6.6.4.4 DrawnOnBar

Definition
Gets the current bar value that the chart anchor is drawn by a NinjaScript object. Please see the Drawing section for more information.

> **Note**: This value will **NOT** work on manually drawn objects. This property is reserved for chart anchors which were drawn by another NinjaScript object (e.g, using a Draw method in an indicator). For manually drawn objects, please see the SlotIndex property

Property Value
A `int` value that value which the current bar the chart anchor is drawn. This property is read-only.

Syntax
<ChartAnchor>.DrawnOnBar

Examples

```
//Places text if high is 2419 and prints what bar the text was
drawn on
if (High[0] == 2419)
{
	Text myText = Draw.Text(this, @"Text " + CurrentBar, @"High
is 2419" , 0, High[0]);
	Print("Text is on bar " + myText.Anchor.DrawnOnBar);
}
```

11.6.6.4.5 GetPoint()

Definition
Returns a chart anchor's data point in device pixels

Method Return Value
A Point structure; a point value in device pixels for a chart's given panel & scale

Syntax
<chartAnchor>.GetPoint(ChartControl chartControl, ChartPanel chartPanel, ChartScale,
[bool pixelAlign])

Method Parameters

chartControl	A ChartControl representing the x-axis
chartPanel	A ChartPanel representing the a panel of the chart
chartScale	A ChartScale representing the y-axis
pixelAlign	An optional bool determining if the data point should be rounded to closest .5 pixel point

Examples

```
//gets the chart anchors data points
Point anchorPoint = MyAnchor.GetPoint(chartControl, chartPanel,
chartScale);
```

11.6.6.4.6 IsBrowsable

Definition

Determines if the anchor are visible on the UI. When set to true, the anchors Y and X values can be viewed from the Drawing Objects properties.

Property Value

A `bool` value which when true will display the anchor data values from the drawing object properties; otherwise **false**. Default value is **true.**

Syntax

`<ChartAnchor>.IsBrowsable`

Examples

```
protected override void OnStateChange()
{
	if (State == State.SetDefaults)
	{
		MyAnchor = new ChartAnchor();
		MyAnchor.IsBrowsable = true;
	}
	else if (State == State.Configure)
	{

	}
}
```

11.6.6.4.7 IsEditing

Definition

Determines if the anchor can be edited.

Property Value

A `bool` value which when true determines if the chart anchor is currently in a state it can be edited. Default is **false**.

Syntax

`<ChartAnchor>.IsEditing`

Examples

```
public override void OnMouseDown(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, Point point)
{
    if(DrawingState == DrawingState.Building)
    {
       // if drawing tool is currently editing, update to current
    mouse point
        if(MyAnchor.IsEditing)
        {
             MyAnchor.UpdateFromPoint(point, chartControl,
            chartScale);

             //set the anchor to disable editing when done updating
             MyAnchor.IsEditing = false;

        }
    }
}
```

11.6.6.4.8 IsNinjaScriptDrawn

Definition

Indicates if the chart anchor was drawn by a NinjaScript object (such as an indicator or strategy).

Property Value

A `bool` value which returns **true** of the object was drawn by other NinjaScript object; otherwise **false**. This property is read-only.

Syntax

`<ChartAnchor>.IsNinjaScriptDrawn`

Examples

```
//unlocks the NinjaScript drawn object and allows the user to
modify the anchor, while the NinjaScript object still 'owns' the
object
protected override void OnBarUpdate()
{
     foreach(IDrawingTool dt in DrawObjects)
         {
            DrawingTools.Line sampleLine = dt as DrawingTools.Line;

            if (sampleLine != null &&
sampleLine.StartAnchor.IsNinjaScriptDrawn)
            {
               sampleLine.IsLocked = false;
               Print(sampleLine.StartAnchor.ToString());
            }
         }
}
```

11.6.6.4.9 IsXPropertiesVisibile

Definition

Indicates the anchor's X properties are visible on the UI. When set to true, the X values can be viewed from the Drawing Objects properties.

Property Value

A `bool` value which when true will display the anchor's X (time) data values from the drawing object properties; otherwise **false**. Default value is **true.**

Syntax

`<ChartAnchor>.IsXPropertiesVisibile`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          MyAnchor = new ChartAnchor();
          MyAnchor.IsXPropertiesVisible = true;
     }
     else if (State == State.Configure)
     {

     }
}
```

11.6.6.4.10 IsYPropertyVisibile

Definition

Indicates the anchor's Y properties are visible on the UI. When set to true, the Y values can be viewed from the Drawing Objects properties.

Property Value

A `bool` value which when true will display the anchor's Y (price) data values from the drawing object properties; otherwise **false**. Default value is **true.**

Syntax

```
<ChartAnchor>.IsYPropertyVisibile
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          MyAnchor = new ChartAnchor();
          MyAnchor.IsYPropertyVisibile = true;
     }
     else if (State == State.Configure)
     {

     }
}
```

11.6.6.4.11 MoveAnchor()

Definition

Moves a Chart Anchor's x and y values from start point by a delta point amount.

Method Return Value

This method does not return a value.

Syntax

```
<ChartAnchor>.MoveAnchor(ChartAnchor startDataPoint, ChartAnchor deltaDataPoint,
ChartControl chartControl, ChartPanel chartPanel, ChartScale chartScale, DrawingTool
drawingTool)
```

Method Parameters

startPoint	The chart anchor's original starting location value represented by a point structure

startDataPoint	A chart anchor's original starting location value represented by a chart anchor
deltaPoint	The chart anchor's new location value to be updated represented by a point structure
deltaDataPoint	The chart anchor's new location value to be udpated represened by a chart anchor
chartControl	A ChartControl representing the x-axis
chartScale	A ChartScale representing the y-axis
chartPanel	A ChartPanel representing the the panel for the chart
drawingTool	The drawing tool which owns the chart anchor to be moved (usually `this`).

Examples

```
//move the chart anchors x and y values
MyAnchor.MoveAnchor(lastPoint, newPoint, chartControl, chartPanel,
chartScale, this);
```

11.6.6.4.12 MoveAnchorX()

Definition

Moves an anchor's x value from start point by a delta point amount.

Method Return Value

This method does not return a value.

Syntax

```
<ChartAnchor>.MoveAnchorX(Point startPoint, Point deltaPoint, ChartControl
chartControl, ChartPanel chartPanel, ChartScale chartScale)
```

Method Parameters

startPoint	The chart anchor's original starting point value

deltaPoint	The chart anchor's new point value to be updated
chartControl	A ChartControl representing the x-axis
chartScale	A ChartScale representing the y-axis

Examples

```
//move only the chart anchors x (bar/time) value
MyAnchor.MoveAnchorX(lastPoint, newPoint, chartControl,
chartScale);
```

11.6.6.4.13 MoveAnchorY()

Definition

Moves an anchor's y value from start point by a delta point amount.

Method Return Value

This method does not return a value.

Syntax

```
<ChartAnchor>.MoveAnchorY(Point startPoint, Point deltaPoint, ChartControl
chartControl, ChartScale chartScale)
```

Method Parameters

startPoint	The chart anchor's original starting point value
deltaPoint	The chart anchor's new point value to be updated
chartControl	A ChartControl representing the x-axis
chartScale	A ChartScale representing the y-axis

Examples

```
//move only the chart anchors Y (price) value
MyAnchor.MoveAnchorY(lastPoint, newPoint, chartControl, chartPanel,
chartScale);
```

11.6.6.4.14 Price

Definition
Determines price value the chart anchor is drawn.

Property Value
An `double` value representing a price value

Syntax
`<ChartAnchor>.Price`

Examples
```
public override void OnMouseDown(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, Point point)
{
   Print(MyAnchor.Price);  // prints the Y axis data point of the
chart anchor
   // 1999.25
}
```

11.6.6.4.15 SlotIndex

Definition
Indicates the nearest bar slot value where anchor is drawn on a chart. In a single series chart there will always be equal number of slots and bars, however for multi-series charts there may be additonal slots compared to the bar series your drawing tool resides.

Property Value
An `double` value representing the current bar.

> **Note**: The bar index value is represented as a `double` as it is possible (and likely) that a given chart anchor is drawn between bars (i.e., if a user draws the tool with snap mode disabled)

Syntax
`ChartAnchor.SlotIndex`

Examples

```
public override void OnMouseDown(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
    Print(MyAnchor.SlotIndex);  // prints the nearest current bar
value
    //4502.02734375
}
```

11.6.6.4.16 Time

Definition

Determines date/time value the chart anchor is drawn.

Property Value

An DateTime value representing a time value

Syntax

<ChartAnchor>.Time

Examples

```
public override void OnMouseDown(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, Point point)
{
    Print(MyAnchor.Time);  // prints the X axis datetime of the
    chart anchor
    // 8/26/2014 6:55:00 PM
}
```

11.6.6.4.17 UpdateFromPoint()

Definition

Updates an anchor's x and y values from a given point (in device pixels).

Method Return Value

This method does not return a value.

Syntax

<ChartAnchor>.UpdateFromPoint(Point point, ChartControl chartControl, ChartScale chartScale)

Method Parameters

point	The chart anchor's point value to be updated
chartControl	A ChartControl representing the x-axis
chartScale	A ChartScale representing the y-axis

Examples

```
//set the chart anchors x and y point value
MyAnchor.UpdateFromPoint(point, chartControl, chartScale);
```

11.6.6.4.18 UpdateXFromPoint()

Definition

Updates an anchor's X value from a given point (in device pixels).

Method Return Value

This method does not return a value.

Syntax

<ChartAnchor>.UpdateXFromPoint(Point *point*, ChartControl *chartControl*, ChartScale *chartScale*)

Method Parameters

point	The chart anchor's point value to be updated
chartControl	A ChartControl representing the x-axis
chartScale	A ChartScale representing the y-axis

Examples

```
//set the chart anchors x point value
MyAnchor.UpdateXFromPoint(point, chartControl, chartScale);
```

11.6.6.4.19 UpdateYFromPoint()

Definition

Updates an anchor's Y value from a given point (in device pixels).

Method Return Value
This method does not return a value.

Syntax
```
<ChartAnchor>.UpdateYFromPoint(Point point, ChartScale chartScale)
```

Method Parameters

point	The chart anchor's point value to be updated
chartScale	A ChartScale representing the y-axis

Examples

```
//set the chart anchors x point value
MyAnchor.UpdateYFromPoint(point, chartScale);
```

11.6.6.5 ConvertToVerticalPixels()

Definition
Used to convert the cursor position (pixels) to device pixels represented on the Y axis of the chart. This method would only be needed if the value you are given is provided in WPF pixel point (such as the data point used in OnMouseDown), but you would need the value in the chart's rendered pixels. This is useful when handling drawing tools and charts which would have multiple chart panels.

Method Return Value
An `int` value representing the converted value in device pixels

Syntax
```
ConvertToVerticalPixels(ChartControl chartControl, ChartPanel chartPanel, double wpfY)
```

Method Parameters

chartControl	A ChartControl representing the x-axis
chartPanel	A ChartPanel representing the the panel for the chart
wpfY	A `double` value which needs to be converted

Examples

```
public override void OnMouseDown(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   //get chart anchors data point when mouse is clicked
   Point myPoint = dataPoint.GetPoint(chartControl, chartPanel,
chartScale);

   Print("before convert: " + myPoint.Y);  //before convert: 630.5

   //convert the data point to device pixels
   double yPixel = ConvertToVerticalPixels(chartControl,
chartPanel, myPoint.Y);

   Print("after convert: " + yPixel); //after convert: 1108
}
```

11.6.6.6 CreateAnchor()

Definition

Used to create a new chart anchor at a specified mouse point.

Method Return Value

A new ChartAnchor at a specified point in device pixels.

Syntax

```
CreateAnchor(Point point, ChartControl chartControl, ChartScale chartScale)
```

Method Parameters

point	A Point in device pixels representing the current mouse cursor position
chartControl	A ChartControl representing the x-axis
chartScale	A ChartScale representing the y-axis

Examples

```
public override void OnMouseDown(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   // get the point where the mouse was clicked
   Point myPoint = dataPoint.GetPoint(chartControl, chartPanel,
chartScale);

   // create an anchor at that point
   ChartAnchor MyAnchor = CreateAnchor(myPoint, chartControl,
chartScale);

   Print(MyAnchor.Time); // 3/16/2015 8:18:48 AM
}
```

11.6.6.7 DisplayOnChartsMenus

Definition

Determines if the drawing tool displays in the chart's drawing tool menus.

Property Value

A `bool` value, when **true** the drawing tool will be created on the chart's drawing tool menu; otherwise **false**. Default value is **true**.

Syntax

`DisplayOnChartsMenus`

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name                  = @"My Drawing Tool";
      DisplayOnChartsMenus  = true;
   }
}
```

11.6.6.8 Dispose()

Definition

Releases any device resources used for the drawing tool.

Method Return Value

This method does not return a value

Syntax
Dispose()

Method Parameters
This method does not accept any parameters

Examples


```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name                    = @"My Drawing Tool";
   }

   else if (State == State.Terminated)
      Dispose();
}
```

11.6.6.9 DrawingState

Definition
Represents the current state of the drawing tool to perform various actions, such as building, editing, or moving.

Property Values
An enum representing the current state of the drawing tool. Possible values are:

DrawingState.Building	The initial state when a drawing tool is first being drawn, allowing for the anchors to be set for the drawing.
DrawingState.Editing	Allows for changing the values of any of the drawing tools anchors
DrawingState.Normal	The drawing tool is normal on the chart and is not in a state to allow for changes.
DrawingState.Moving	The entire drawing tool to be moved by a user.

Syntax
DrawingState

Examples

```
public override void OnMouseDown(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, Point point)
{
   switch(DrawingState)
   {
      case DrawingState.Normal:
         DrawingState = DrawingState.Editing;  // set state to
allow editing
         break;
      case DrawingState.Editing:
         // do your edits here
         break;
      case DrawingState.Moving:
         return; // don't allow move whe editing
   }

}
```

11.6.6.10 DrawnBy

Definition
Represents the NinjaScript object which created the drawing object

Property Value
The NinjaScript object which created the drawing tool; this value will be `null` if drawn by a user.

Syntax
DrawnBy

Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // if the drawing tool was not created by a user,
   // print the name of the object that it was created
   if(!IsUserDrawn)
    Print(DrawnBy.Name);
}
```

11.6.6.11 GetAttachedToChartBars()

Definition
Returns information which relate to the underlying bars series in which the drawing tool is attached. If the drawing tool is attached to an indicator rather than a bar series, the indicator's bars series used for input will be returned.

> **Note**: For drawing tools made global, this method will not be returning meaningful values - since those are not attached to a specific bars series

Method Return Value
A ChartBars object

Syntax
```
GetAttachedToChartBars()
```

Method Parameters
This method does not accept any parameters

Examples
```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   // get the attached chart bars
   ChartBars myBars = GetAttachedToChartBars();

   Print(myBars.Bars.ToChartString());  // NQ 03-15 (1 Minute)
}
```

11.6.6.12 GetClosestAnchor()

Definition
Returns the closest chart anchor within a specified maximum distance from the mouse cursor.

Method Return Value
This method returns an existing ChartAnchor

Syntax
```
GetClosestAnchor(ChartControl chartControl, ChartPanel chartPanel, ChartScale
chartScale, double maxDist, Point point)
```

Method Parameters

chartControl	A ChartControl representing the x-axis
chartPanel	A ChartPanel representing the the panel for the chart
chartScale	A ChartScale representing the y-axis
maxDist	A `double` value representing the cursor's sensitivity used to detect the nearest anchor
point	A Point in device pixels representing the current mouse cursor position

Examples

```
public override Cursor GetCursor(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, Point point)
{
   // get the closest anchor to where the user has clicked
   ChartAnchor   closest = GetClosestAnchor(chartControl,
chartPanel, chartScale, 10, point);

   if (closest != null)
   {
      // set cursor to indicate that it can be moved
      return Cursors.SizeNWSE;
   }
   // otherwise set cursor back to arrow
   else return Cursors.Arrow;
}
```

11.6.6.13 GetCursor()

Definition

An event driven method which is called when a chart object is selected. This method can be used to change the cursor image used in various states.

Method Return Value

This method returns a Cursor used to paint the mouse pointer.

Syntax

You must override the method in your Drawing Tool with the following syntax:

```
public override Cursor GetCursor(ChartControl chartControl, ChartPanel chartPanel,
ChartScale chartScale, Point point)
{

}
```

Method Parameters

chartControl	A ChartControl representing the x-axis
chartPanel	A ChartPanel representing the the panel for the chart
chartScale	A ChartScale representing the y-axis
point	A Point in device pixels representing the current mouse cursor position

Examples

```
public override Cursor GetCursor(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, Point point)
{
   switch (DrawingState)
   {
      //when drawing, display the cursor as a pen
      case DrawingState.Building:   return Cursors.Pen;

      // when moving, display a four-headed sizing cursor
      case DrawingState.Moving:   return Cursors.SizeAll;

      default: return Cursors.Pen;
   }
}
```

11.6.6.14 GetSelectionPoints()

Definition

Returns the chart object's data points where the user can interact. These points are used to visually indicate that the chart object is selected and allow the user to manipulate the chart object. This method is only called when IsSelected is set to true.

Method Return Value

A collection of Points representing the x- and y-coordinates of the chart object.

Syntax

You must override the method using the following syntax:

```
public override Point[] GetSelectionPoints(ChartControl chartControl, ChartScale
chartScale)
{

}
```

Method Parameters

chartControl	A ChartControl representing the x-axis
chartScale	A ChartScale representing the y-axis

Examples

```
public override Point[] GetSelectionPoints(ChartControl
chartControl, ChartScale chartScale)
{

       ChartPanel chartPanel =
chartControl.ChartPanels[chartScale.PanelIndex];

       // get the anchor point to be displayed on the drawing tool
       Point anchorPoint = Anchor.GetPoint(chartControl,
chartPanel, chartScale, false);
              return new[] { anchorPoint } ;
}
```

11.6.6.15 Icon

Definition

The shape which displays next to the Drawing Tool menu item. Since this is a standard object, any type of icon can be used (unicode characters, custom image file resource, geometry path, etc). For more information on using images to create icons, see the Using Images with Custom Icons page.

> Note: When using UniCode characters, first ensure that the desired characters exist in the icon pack for the font family used in NinjaTrader.

Property Value

A generic virtual `object` representing the drawing tools menu icon. This property is read-only.

Syntax

You must override this property using the following syntax:

```
public override object Icon
```

Examples

```
public override object Icon
{
   get
   {
      //use a unicode character as our string which will render an
arrow
      string uniCodeArrow = "\u279A";
      return uniCodeArrow;
   }
}
```

11.6.6.16 IgnoresSnapping

Definition

Determines if the drawing tool chart anchor's will use the chart's **Snap Mode** mouse coordinates.

Property Value

A `bool` value which when **true** the drawing tool ignores snapping; otherwise **false**. Default is set to **false**.

Syntax

```
IgnoresSnapping
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IgnoresSnapping = true; // Set this to true to receive
non-snapped mouse coordinates
     }
     else if (State == State.Configure)
     {

     }
}
```

11.6.6.17 IgnoresUserInput

Definition

Determines if the drawing tool can be clicked on by the user.

Property Value

A `bool` value which wen **true** if the drawing tool cannot not be interacted with by a user; otherwise **false**. Default is set to **false**.

Syntax

`IgnoresUserInput`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {

          IgnoresUserInput = true; // Set this to true to make the
       drawing object non-interactive
     }
     else if (State == State.Configure)
     {

     }
}
```

11.6.6.18 IsAttachedToNinjaScript

Definition

Indicates if the drawing tool is currently attached to a NinjaScript object (such an indicator or a strategy).

Property Value

A `bool` value which when **true** if the drawing tool is attached to a NinjaScript object; otherwise **false**. This property is read-only.

Syntax

`IsAttachedToNinjaScript`

Examples

```
public override void OnMouseMove(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   // do not interact if drawn by an indicator or strategy
   if (IsAttachedToNinjaScript)
      return;
}
```

11.6.6.19 IsGlobalDrawingTool

Definition

Indicates if the drawing tool is currently set as a Global Drawing object. Global draw objects display on any chart which matches the parent chart's underlying instrument.

Property Value

A `bool` value which returns **true** if the drawing tool is currently attached as a global drawing object; otherwise **false**.

Syntax

`IsGlobalDrawingTool`

Examples

```
public override void OnMouseMove(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   // do not interact if attached to global chart
   if (IsGlobalDrawingTool)
      return;
}
```

11.6.6.20 IsLocked

Definition

Determines if the drawing tool should be be locked in place. This property can be set either manually through the UI or explicitly through code.

Property Value

A `bool` value which when **true** if the drawing tool is locked; otherwise **false**. Default is set to **false**.

> **Note**: For Drawing tools which are drawn by an indicator or strategy, this property will default to **true**.

Syntax
IsLocked

Examples

```
public override void OnMouseMove(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, Point point)
{
   if (IsLocked)  //if the object is locked, do not attempt to move
      return;
}
```

11.6.6.21 IsUserDrawn

Definition
Indicates if the drawing tool was manually drawn by a user as opposed to programmatically drawn by a NinjaScript object (such as an indicator or strategy).

Property Value
A `bool` value which when **true** if the draw object was manually drawn ; otherwise **false.** This property is read-only

Syntax
IsUserDrawn

Examples

```
if (IsUserDrawn)
{
  // do something only if the object was drawn manually
}
```

11.6.6.22 OnBarsChanged()

Definition
An event driven method which is called any time the underlying bar series have changed for the chart where the drawing tool resides. For example if a user has changed the primary instrument or the time frame of the bars used on the chart.

Method Return Value

This method does not return a value

Syntax
You must override this method using the following syntax:

```
public override void OnBarsChanged()
{

}
```

Method Parameters
This method does not accept any parameters

Examples

```
public override void OnBarsChanged()
{
    //bars have change, do something
}
```

11.6.6.23 OnMouseDown()

Definition
An event driven method which is called any time the mouse pointer over the chart control has the mouse button pressed.

Method Return Value
This method does not return a value.

> **Note**: For a combined single click operation, i.e. mouse down click, move and release the dataPoint reported will always be the initial starting one.

Syntax
You must override the method in your Drawing Tool with the following syntax.

```
public override void OnMouseDown(ChartControl chartControl, ChartPanel chartPanel,
ChartScale chartScale, ChartAnchor dataPoint)
{

}
```

Method Parameters

chartControl	A ChartControl representing the x-axis

chartPanel	A ChartPanel representing the the panel for the chart
chartScale	A ChartScale representing the y-axis
dataPoint	A ChartAnchor representing a point where the user clicked

Examples

```
public override void OnMouseDown(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   switch (DrawingState)
   {
      case DrawingState.Building:
         dataPoint.CopyDataValues(Anchor);
         Anchor.IsEditing   = false;
         DrawingState      = DrawingState.Normal;
         IsSelected         = false;
         break;
      case DrawingState.Normal:
         // make sure they clicked near us. use GetCursor incase
something has more than one point
         Point point = dataPoint.GetPoint(chartControl, chartPanel,
 chartScale);
         if (GetCursor(chartControl, chartPanel, chartScale,
point) != null)
            DrawingState = DrawingState.Moving;
         else
            IsSelected = false;
         break;
   }
}
```

11.6.6.24 OnMouseMove()

Definition

An event driven method which is called any time the mouse pointer is over the chart control and a mouse is moving.

Method Return Value

This method does not return a value.

Note: For a combined single click operation, i.e. mouse down click, move and release the dataPoint reported will always be the initial starting one.

Syntax

You must override the method in your Drawing Tool with the following syntax.

```
public override void OnMouseMove(ChartControl chartControl, ChartPanel chartPanel,
ChartScale chartScale, ChartAnchor dataPoint)
{

}
```

Method Parameters

chartControl	A ChartControl representing the x-axis
chartPanel	A ChartPanel representing the the panel for the chart
chartScale	A ChartScale representing the y-axis
dataPoint	A ChartAnchor representing a point where the user is moving the mouse

Examples

```
private   ChartAnchor                    lastMouseMoveAnchor
= new ChartAnchor();
private ChartAnchor MyAnchor;
public override void OnMouseMove(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   // add any logic for when the mouse is moved here
   if (DrawingState == DrawingState.Moving)
   {
      //move the chart anchor when the drawing tool is in a moving
state

      MyAnchor.MoveAnchor(lastMouseMoveAnchor, dataPoint,
chartControl, chartPanel, chartScale, this);
      // dont forget to update delta point to last used!
      dataPoint.CopyDataValues(lastMouseMoveAnchor);
   }
}
```

11.6.6.25 OnMouseUp()

Definition

An event driven method is called any time the mouse pointer is over the chart control and a mouse button is being released.

Method Return Value

This method does not return a value

> **Note**: For a combined single click operation, i.e. mouse down click, move and release the dataPoint reported will always be the initial starting one.

Syntax

You must override the method with the following syntax.

```
public override void OnMouseUp(ChartControl chartControl, ChartPanel chartPanel,
ChartScale chartScale, ChartAnchor dataPoint)
{

}
```

Method Parameters

chartControl	A ChartControl representing the x-axis

chartPanel	A ChartPanel representing the the panel for the chart
chartScale	A ChartScale representing the y-axis
dataPoint	A ChartAnchor representing a point where the user is releasing the mouse

Examples

```
public override void OnMouseUp(ChartControl chartControl,
ChartPanel chartPanel, ChartScale chartScale, ChartAnchor
dataPoint)
{
   //when the user releases the mouse, ensure the drawing state is
set to normal
   if (DrawingState == DrawingState.Editing || DrawingState ==
DrawingState.Moving)
      DrawingState = DrawingState.Normal;
}
```

11.6.6.26 SupportsAlerts

Definition

Determines if the drawing tool can be used for manually configured alerts through the UI.

Property Value

A bool which when **true** determines that user can setup an alert based off this drawing tool; otherwise **false**.

> **Note**: This property is **false** by default and **MUST** be overridden upon initialization to allow for manually configured alerts. You cannot set this during run-time.

Syntax

```
SupportsAlerts
```

You may choose to override this property using the following syntax:

```
public override bool SupportsAlerts
```

Examples

```
public override bool SupportsAlerts { get { return true; } }
```

11.6.6.27 ZOrderType

Definition

Determines the order in which the drawing tool will be rendered. This will help control the ZOrder index between chart objects

Property Value

An enum determining the drawing tool's ZOrder type. Possible values are:

DrawingToolZOrder.Normal	Default behavior, drawing tools are rendered as they appear in the ZOrder index
DrawingToolZOrder.AlwaysDrawnFirst	Ensures the drawing tool is always the first to be rendered
DrawingToolZOrder.AlwaysDrawnLast	Ensures the drawing tool is always the last object to be rendered

Syntax

ZOrderType

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name               = @"My Drawing Tool";

      // always draw this last
      ZOrderType            = DrawingToolZOrder.AlwaysDrawnLast;
   }
   else if (State == State.Configure)
   {
   }
}
```

11.6.7 Import Type

Custom data Import Types can be developed to allow for the importing of historical data from any format. Two important event handler methods are documented in this section:

Methods and Properties

OnNextInstrument()	Called at the beginning of the import process
OnNextDataPoint()	Called for each line of data contained in the file being imported

11.6.7.1 OnNextInstrument()

Definifition

The OnNextInstrument() method is called at the beginning of the import process for each file that is being imported. This method is only called after it has determined the file contains a valid instrument.

Method Return Value

This method does not return a value.

Syntax

See example below. The NinjaScript code wizard automatically generates the method syntax for you.

Example

```
private int currentInstrumentIdx = -1;

public string[] FileNames
{ get; set; }

protected override void OnNextInstrument()
{
    if (FileNames == null)
        return;

        // Try to read from file into the FileNames array created
above
        // Log an error and continue if the data is unreadable
        try
        {
            reader = new
StreamReader(FileNames[currentInstrumentIdx]);
        }
        catch (Exception exp)
        {
            NinjaScript.Log(FileNames[currentInstrumentIdx],
exp.Message, LogLevel.Error);
            continue;
        }
}
```

11.6.7.2 OnNextDataPoint()

Definition

The OnNextDataPoint() method is called for each line of data contained in the file being imported. This method is only called if the import type determines that the file has a valid data point, and will continue to be called until it reaches the end of the file, or until the data point is no longer valid.

Method Return Value

This method does not return a value.

Syntax

See example below. The NinjaScript code wizards automatically generate the method syntax for you.

Example

```
private StreamReader reader;

protected override void OnNextDataPoint()
{
    if (reader == null)
        return;

    // Continually read data using the StreamReader defined above
    while (true)
    {
        DataPointString = reader.ReadLine();
        // Additional data formatting here
    }
}
```

11.6.8 Indicator

The methods and properties covered in this section are unique to custom indicator development. Indicator configuration properties globally define various behaviors of indicators. All properties have default values and can be overridden by setting them in the OnStateChange() method of the indicator.

Tip: See also the "Common" section for more method and properties which are shared by NinjaScript types

Methods and Properties

AddLine()	Adds line objects on a chart.
AddPlot()	Adds plot objects that define how an indicator or strategy data series render on a chart.
BarsRequiredToPlot	The number of bars on a chart required before the script plots.
DisplayInDataBox	Determines if plot(s) display in the chart data box.
DrawHorizontalGridLines	Plots horizontal grid lines on the indicator panel.

DrawOnPricePanel	Determines the chart panel the draw objects renders.
DrawVerticalGridLines	Plots vertical grid lines on the indicator panel.
IndicatorBaseConverter	A custom TypeConverter class handling the designed behavior of an indicator's property descriptor collection.
IsChartOnly	If true, any indicator will be only available for charting usage - indicators with this property enabled would for example not be expected to show if called in a SuperDOM or MarketAnalyzer window.
IsSuspendedWhileInactive	Prevents real-time market data events from being raised while the indicator's hosting feature is in a state that would be considered suspended and not in immediate use by a user.
PaintPriceMarkers	If true, any indicator plot values display price markers in the y-axis.
ShowTransparentPlotsInDataBox	Determines if plot(s) values which are set to a Transparent brush display in the chart data box.

11.6.8.1 AddLine()

Definition

Adds line objects on a chart.

> **Note:** Lines are **ONLY** visible from the UI property grid when AddLine() is called from **State.SetDefaults**. If your indicator or strategy dynamically adds lines during **State.Configure**, you will **NOT** have an opportunity to select the line or to set the line configuration via the UI. Alternatively, you may use custom public Brush, Stroke or value properties which are accessible in the **State.SetDefaults** and pass those values to AddLine() during **State.Configure**. Calling AddLine() in this manner should be reserved for special cases. Please see the examples below.

Methods and Properties

AreLinesConfigura	Determines if the line(s) used in an indicator are

ble	configurable from within the indicator dialog window.
Line Class	Objects derived from the Line class are used to characterize how an oscillator line is visually displayed (plotted) on a chart.
Lines	A collection holding all of the Line objects that define the visualization characteristics oscillator lines of the indicator.

Syntax

```
AddLine(Brush brush, double value, string name)
AddLine(Stroke stroke, double value, string name)
```

Warning: This method should **ONLY** be called within the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Parameters

brush	A `Brush` object used to construct the line
name	A `string` value representing the name of the line
stroke	A `Stroke` object used to construct the line
value	A `double` value representing the value the line will be drawn at

Examples

Defining a single UI configurable static line

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";
      // Adds an oscillator line at a value of 30
      AddLine(Brushes.Gray, 30, "Lower");
   }
}
```

Indicator which dynamically adds a line in State.Configure

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name                  = "Examples Indicator";

      // logical property which user can set
      UseSpecialMode   = false;
      // Default brush selection pushed to the UI
      MyBrush = Brushes.Red;
   }
   else if (State == State.Configure)
   {
      // if user enables logical property
      if (UseSpecialMode)
      {
         // add line using default selected brush and special
line name
         AddLine(MyBrush, 40,  "My Special Line");
      }
      else
      {
         // otherwise use default selected brush and regular
line name
         AddLine(MyBrush, 60, "My Regular Line");
      }
   }
}


[XmlIgnore]
public Brush MyBrush { get; set; }

public bool UseSpecialMode { get; set; }
```

11.6.8.1.1 AreLinesConfigurable

Definition

Determines if the line(s) used in an indicator are configurable from within the indicator dialog window.

Property Value

A bool which **true** if any indicator line(s) are configurable; otherwise, **false**. Default set to **true**.

Syntax
AreLinesConfigurable

Examples
```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          AddLine(Brushes.Gray, 30, "Lower");
          AreLinesConfigurable = false; // Indicator lines are not
configurable
     }
}
```

11.6.8.1.2 Line Class

Definition
Objects derived from the Line class are used to characterize how an oscillator line is visually displayed (plotted) on a chart.

Properties

Brush	The System.Windows.Media.Brush used to construct the line (reference)
BrushDX	A SharpDX.Direct2D1.Brush used to actually render the line **Note**: To avoid and resolve access violation exceptions, please see Warning and examples remarked below
DashStyleDX	A SharpDX.Direct2D1.DashStyle used to render the stroke style **Note**: To avoid and resolve access violation exceptions,

	please see Warning and examples remarked below
DashStyleHelper	A dashstyle used to construct the stroke. Possible values are: • DashStyleHelper.Dash • DashStyleHelper.DashDot • DashStyleHelper.DashDotDot • DashStyleHelper.Dot • DashStyleHelper.Solid
Name	A `string` representing the name of the line
RenderTarget	The RenderTarget drawing context used for the line. **Note**: This property must be set before accessing a stroke's BrushDX property. Please see Warning and examples remarked below
StrokeStyle	A SharpDX.Direct2D1.StrokeStyle
Value	A `double` representing the value of the line
Width	A `float` representing the width in pixels

Examples

See the AddLine() method for examples.

11.6.8.1.3 Lines

Definition

A collection holding all of the Line objects that define the visualization characteristics oscillator lines of the indicator.

Property Value

A collection of Line objects.

Syntax

```
Lines[int index]
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Lines are added to the Lines collection in order
          AddLine(Brushes.Gray, 30, "Lower"); // Stored in Lines[0]
          AddLine(Brushes.Gray, 70, "Upper"); // Stored in Lines[1]
     }
}

// Dynamically change the upper line's color and thickness based on
the indicator value
protected override void OnBarUpdate()
{
   if(Value[0] > 70)
   {
      Lines[1].Brush = Brushes.Blue;
      Lines[1].Width = 3;
   }
   else
   {
      Lines[1].Brush = Brushes.Gray;
      Lines[1].Width = 1;
   }
}
```

11.6.8.2 AddPlot()

Definition

Adds plot objects that define how an indicator or strategy data series render on a chart. When this method is called to add a plot, an associated Series<double> object is created held in the Values collection.

Note: Plots are **ONLY** visible from the UI property grid when AddPlot() is called from **State.SetDefaults**. If your indicator or strategy dynamically adds plots during **State.Configure**, you will **NOT** have an opportunity to select the plot or to set the plot configuration via the UI. Alternatively, you may use custom public Brush, Stroke, or

PlotStyle properties which are accessible in **State.SetDefaults** and pass those values to AddPlot() during **State.Configure**. Calling AddPlot() in this manner should be reserved for special cases. Please see the examples below.

Methods and Properties

ArePlotsConfigurable	Determines if the plot(s) used in an indicator are configurable within the indicator dialog window.
Displacement	An offset value that shifts the visually displayed value of an indicator.
PlotBrushes	Holds an array of color series objects holding historical bar colors.
Plots	A collection holding all of the Plot objects that define their visualization characteristics.

Syntax

```
AddPlot(Brush brush, string name)
AddPlot(Stroke stroke, PlotStyle plotStyle, string name)
```

Warning: This method should **ONLY** be called within the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Parameters

brush	A `Brush` object used to construct the plot
name	A `string` representing the name of the plot

plotStyle	A PlotStyle object used to construct the style of the plot
	
	Possible values:
	
	`PlotStyle.Bar`
	`PlotStyle.Block`
	`PlotStyle.Cross`
	`PlotStyle.Dot`
	`PlotStyle.Hash`

	`PlotStyle.HLine`
	`PlotStyle.Line`
	`PlotStyle.PriceBox`
	`PlotStyle.Square`
	`PlotStyle.TriangleDown`
	`PlotStyle.TriangleLeft`
	`PlotStyle.TriangleRight`
	`PlotStyle.TriangleUp`

stroke	A `Stroke` object used to construct the plot

Tips:
1. We suggest using the NinjaScript wizard to generate your plots.
2. Plot objects **DO NOT** hold the actual script values. They simply define how the script's values are plotted on a chart.
3. A script may calculate multiple values and therefore hold multiple plots to determine the display of each calculated value. Script values are held in the script's Values collection.
4. If you script calls AddPlot() multiple times, then multiple Values series are added per the "three value series" example below
5. For MultiSeries scripts, plots are synched to the primary series of the NinjaScript object.
6. Plots will become visible once the script's BarsRequiredToPlot value has been satisfied. By default, the value is 20.

Examples

Indicator using various AddPlot() signatures

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";

      // Adds a blue line style plot
      AddPlot(Brushes.Blue, "MyPlot");

      // Adds a blue historgram style plot
      AddPlot(new Stroke(Brushes.Blue), PlotStyle.Bar, "MyPlot2");

      // Ensures that the width of the PlotStyle.Bar plot matches
the width of the data series
      Plots[1].AutoWidth = true;

      // Adds a blue Dash-Line style plot with 5pixel width and 50%
opacity
      AddPlot(new Stroke(Brushes.Blue, DashStyleHelper.Dash, 5,
50), PlotStyle.Line, "MyPlot3");
   }
}
```

Indicator using a public Series<double> to expose a plot with a friendly name. This is required for making plots accessible in the Strategy Builder
For an example on exposing other variables publicly, see Exposing Indicator values that are not plots

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";

      // Adds a blue line style plot
      AddPlot(Brushes.Blue, "MyPlot");

      // Adds a blue historgram style plot
      AddPlot(new Stroke(Brushes.Blue), PlotStyle.Bar, "MyPlot2");
   }
}

protected override void OnBarUpdate()
{
   MyPlot[0] = Close[0] + High[0] / 2;
   MyPlot[1] = Close[0] + High[0] / 2;
}

[Browsable(false)]
[XmlIgnore]
public Series<double> MyPlot
{
   get { return Values[0]; }
}

[Browsable(false)]
[XmlIgnore]
public Series<double> MyPlot2
{
   get { return Values[1]; }
}
```

Indicator which adds three value series

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Examples Indicator";

      // Add three plots and associated Series<double> objects
      AddPlot(Brushes.Blue, "PlotA");     // Defines the plot for
Values[0]
      AddPlot(Brushes.Red, "PlotB");      // Defines the plot for
Values[1]
      AddPlot(Brushes.Green, "PlotC");    // Defines the plot for
Values[2]
   }
}
protected override void OnBarUpdate()
{
   Values[0][0] = Median[0];    // Blue "Plot A"
   Values[1][0] = Low[0];       // Red "Plot B"
   Values[2][0] = High[0];      // Green "Plot C"
}
```

Indicator which dynamically adds a plot in State.Configure

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name                  = "Examples Indicator";

      // logical property which user can set
      UseSpecialMode    = false;
      // Default brush selection pushed to the UI
      MyBrush = Brushes.Red;
   }
   else if (State == State.Configure)
   {
      // if user enables logical property
      if (UseSpecialMode)
      {
         // add plot using default selected brush and special plot
name
         AddPlot(MyBrush, "My Special Plot");
      }
      else
      {
         // otherwise use default selected brush and regular plot
name
         AddPlot(MyBrush, "My Regular Plot");
      }
   }
}

protected override void OnBarUpdate()
{
   if (UseSpecialMode)
      Value[0] = Close[0] + High[0]  / 2;

   else Value[0] = Close[0] * TickSize / 2;
}

[XmlIgnore]
public Brush MyBrush { get; set; }

public bool UseSpecialMode { get; set; }
```

11.6.8.2.1 ArePlotsConfigurable

Definition

Determines if the plot(s) used in an indicator are configurable within the indicator dialog window.

Property Value
A bool which returns **true** if any indicator plot(s) are configurable; otherwise, **false**. Default set to **true**.

Syntax
ArePlotsConfigurable

Examples
```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          AddPlot(Brushes.Orange, "SMA");
          ArePlotsConfigurable = false; // Plots are not
configurable in the indicator dialog
     }
}
```

11.6.8.2.2 Displacement

Definition
An offset value that shifts the visually displayed value of an indicator.

Property Value
An int value that represents the number of bars ago to offset with.

Syntax
Displacement

Examples
```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          Displacement = 2; // Plots the indicator value from 2
bars ago on the current bar
          AddPlot(Brushes.Orange, "SMA");
     }
}
```

11.6.8.2.3 PlotBrushes

Definition

Holds an array of color series objects holding historical bar colors. A color series object is added to this array when calling the AddPlot() method in a custom Indicator for plots. Its purpose is to provide access to the color property of all bars.

Property Value

An array of color series objects.

Syntax

```
PlotBrushes[int PlotIndex][int barsAgo]
```

Examples

```
protected override void OnStateChange()
{
   if(State == State.SetDefaults)
   {
      Name = "Example Indicator";
          // Add two plots
        AddPlot(Brushes.Blue, "Upper");
        AddPlot(Brushes.Orange, "Lower");
     }
}

protected override void OnBarUpdate()
{
     // Sets values to our two plots
     Upper[0] = SMA(High, 20)[0];
     Lower[0] = SMA(Low, 20)[0];

     // Color the Upper plot based on plot value conditions
     if (IsRising(Upper))
          PlotBrushes[0][0] = Brushes.Blue;
     else if (IsFalling(Upper))
          PlotBrushes[0][0] = Brushes.Red;
     else
          PlotBrushes[0][0] = Brushes.Yellow;

     // Color the Lower plot based on plot value conditions
     if (IsRising(Lower))
          PlotBrushes[1][0] = Brushes.Blue;
     else if (IsFalling(Lower))
          PlotBrushes[1][0] = Brushes.Red;
     else
          PlotBrushes[1][0] = Brushes.Yellow;
}

public Series<double> Upper
{
   get { return Values[0]; }
}

public Series<double> Lower
{
   get { return Values[1]; }
}
```

11.6.8.2.4 Plots

Definition

A collection holding all of the Plot objects that define their visualization characteristics.

Property Value
A collection of Plot objects.

Syntax
`Plots[int index]`

> **Note**: The example code below will change the color of an entire plot series. See PlotBrushes for information on changing only specific segments of a plot instead.

Example

```
protected override void OnStateChange()
{
   if(State == State.SetDefaults)
   {
       Name = "Examples Indicator";
        // Lines are added to the Lines collection in order
       AddPlot(Brushes.Orange, "Plot1"); // Stored in Plots[0]
       AddPlot(Brushes.Blue, "Plot2");   // Stored in Plots[1]
      }
}

// Dynamically change the primary plot's color based on the
indicator value
protected override void OnBarUpdate()
{
   if (Value[0] > 70)
   {
      Plots[0].Brush = Brushes.Blue;
      Plots[0].Width = 2;
   }
   else
   {
      Plots[0].Brush = Brushes.Red;
      Plots[0].Width = 2;
   }
}
```

11.6.8.3 BarsRequiredToPlot

Definition
The number of bars on a chart required before the script plots. By default, the value is 20 bars.

> **Note**: This property is **NOT** the same as a minimum number of bars required to calculate the script values. OnBarUpdate will always start calculating for the first bar on the chart (CurrentBar 0)

Property Value
An `int` value that represents the minimum number of bars required.

Syntax
`BarsRequiredToPlot`

Examples
```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          BarsRequiredToPlot = 10; // Do not plot until the 11th
bar on the chart
          AddPlot(Brushes.Orange, "SMA");
     }
}
```

11.6.8.4 DisplayInDataBox

Definition
Determines if plot(s) display in the chart data box.

Property Value
This property returns **true** if the indicator plot(s) values display in the chart data box; otherwise, **false**. Default set to **true**.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax
`DisplayInDataBox`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          DisplayInDataBox = false;
          AddPlot(Brushes.Orange, "SMA");
     }
}
```

11.6.8.5 DrawHorizontalGridLines

Definition

Plots horizontal grid lines on the indicator panel.

> **Note**: The indicator panel's parent chart has a similar option 'Grid line - horizontal which if Visible property set to **false**, will override the indicator's local setting if **true**.

Property Value

This property returns **true** if horizontal grid lines are plotted on the indicator panel; otherwise, **false**. Default set to **true**.

> **Warning**: This property should **ONLY** be set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

`DrawHorizontalGridLines`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          DrawHorizontalGridLines = false; // Horizontal grid lines
will not plot on the indicator panel
          AddPlot(Brushes.Orange, "SMA");
     }
}
```

11.6.8.6 DrawOnPricePanel

Definition
Determines the chart panel the draw objects renders

Property Value
This property returns **true** if the indicator paints draw objects on the price panel; otherwise when false, draw objects are painted on the actual indicator panel itself. Default set to **true**.

> **Warning**: This property should **ONLY** be set from the OnStateChange() method during **State.SetDefaults.** Dynamically using DrawOnPricePanel in an indicator outside of State.SetDefaults may show issues when working with that indicator through a hosting strategy via AddChartIndicator().

Syntax
`DrawOnPricePanel`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          DrawOnPricePanel = false; // Draw objects now paint on
the indicator panel itself
          AddPlot(Brushes.Orange, "SMA");
     }
}
```

11.6.8.7 DrawVerticalGridLines

Definition
Plots vertical grid lines on the indicator panel.

> **Note**: The indicator panel's parent chart has a similar option 'Grid line - vertical which if Visible property set to **false**, will override the indicator's local setting if **true**.

Property Value
This property returns **true** if vertical grid lines are plotted on the indicator panel; otherwise, **false**. Default set to **true**.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

DrawVerticalGridLines

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          DrawVerticalGridLines = false; // Vertical grid lines
will not plot on the indicator panel
          AddPlot(Brushes.Orange, "SMA");
     }
}
```

11.6.8.8 IndicatorBaseConverter

Definition

A custom TypeConverter class handling the designed behavior of an indicator's property descriptor collection. Use this as a base class for any custom **TypeConverter** you are applying to an indicator class.

Notes:

- A working NinjaScript demo can be found through the reference sample on "Using a TypeConverter to Customize Property Grid Behavior"
- When applying the custom converter, you must fully qualify the name (e.g., "NinjaTrader.NinjaScript.Indicators.MyCustomConveter")
- Additional **TypeConverter** information can be found from the MSDN documentation
- See also TypeConverterAttribute
- For Strategies, see the StrategyBaseConverter class

Relevant base methods

TypeConverter.GetProperties()	When overriding **GetProperties()**, calling `base.GetProperties()` ensures

	that all default property grid behavior works as designed
TypeConverter.GetPropertiesSupported()	In your custom converter class, you must override **GetPropertiesSupported()** and return a value of **true** in order for your custom type converter to work

Syntax

```
public class IndicatorBaseConverter : TypeConverter
```

Warning: Failure to apply a type of **IndicatorBaseConverter** on an indicator class can result in unpredictable behavior of the standard NinjaTrader WPF property grid.

Tip: Common indicator functions like Print() are not available to a type converter instance. To debug a type converter class, you can use the AddOn Debug Concepts or attach to a debugger (recommended)

Examples

```
//This namespace holds Indicators in this folder and is required.
Do not change it.
namespace NinjaTrader.NinjaScript.Indicators
{
   // When applying the type converter, you must fully qualify the
name
   [TypeConverter("NinjaTrader.NinjaScript.Indicators.MyCustomConve
ter")]
   public class MyCustomIndicator : Indicator
   {
      protected override void OnStateChange()
      {
         if (State == State.SetDefaults)
         {
            Name    = "MyCustomIndicator";
         }
      }

      protected override void OnBarUpdate()
      {
         //Add your custom indicator logic here.
      }
   }

   public class MyCustomConveter : IndicatorBaseConverter
   {
      // A general TypeConveter method used for converting types
      public override PropertyDescriptorCollection
GetProperties(ITypeDescriptorContext context, object component,
Attribute[] attrs)
      {
         // sometimes you may need the indicator instance which
actually exists on the grid
         MyCustomIndicator indicator = component as
MyCustomIndicator;

         // base.GetProperties ensures we have all the properties
(and associated property grid editors)
         // NinjaTrader internal logic handles for a given
indicator
         PropertyDescriptorCollection propertyDescriptorCollection
= base.GetPropertiesSupported(context)
                  ? base.GetProperties(context, component, attrs) :
TypeDescriptor.GetProperties(component, attrs);

         if (indicator == null || propertyDescriptorCollection ==
null)
            return propertyDescriptorCollection;

         // example of why you may need the instance that exists on
the grid....
         if (indicator.EntryHandling ==
EntryHandling.UniqueEntries)
```

11.6.8.9 IsChartOnly

Definition

If true, any indicator will be only available for charting usage - indicators with this property enabled would for example not be expected to show if called in a SuperDOM or MarketAnalyzer window.

Property Value

This property returns **true** if the indicator can only be used on a chart; otherwise, **false**. Default set to **false**.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

```
IsChartOnly
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IsChartOnly = true; // Allow the indicator to work in
charting environment only
     }
}
```

11.6.8.10 IsSuspendedWhileInactive

Definition

Prevents OnBarUpdate from being raised while the indicators display is not in use. Enabling this property in your indicator helps save CPU cycles while the indicator is suspended and not in use by a user. Once the indicator is in a state that would no longer be considered suspended, the historical OnBarUpdate() events will be triggered allowing the indicator to catch up to current real-time values.

Suspension occurs in the following scenarios:

- Minimized Chart
- Minimized Market Analyzer
- Minimized Hot List Analyzer
- Minimized SuperDOM

- Background tabs of above features are considered "minimized"
- Inactive workspaces in the background

> **Note**: Since events in OnBarUpdate() will not be processed while the indicator is suspended, internal NinjaScript functions such as Alert(), PlaySound(), Share(), Print(), etc - or any other method that would be used to notify a user of activity will **NOT** be processed until the indicator is un-suspended.

Scenarios where suspension will not occur

The **IsSuspendedWhileInactive** property will be ignored and real-time events will be processed as normal under the following cases:

- Indicators running in Automated NinjaScript Strategies
- Indicators which have manually configured alerts
- Indicators which have been manually attached to orders

Property Value

This property returns **true** if indicator can take advantage of suspension optimization; otherwise, **false**. Default set to **false**.

> **Note**: This property is overridden to "**true**" automatically by the NinjaScript Code Wizard. You will need to remove the property to return to the default value or manually set it to false to disable this behavior

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

```
IsSuspendedWhileInactive
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IsSuspendedWhileInactive = true;
     }
}
```

11.6.8.11 PaintPriceMarkers

Definition

If true, any indicator plot values display price markers in the y-axis.

Property Value

This property returns **true** if the indicator plot values display in the y-axis; otherwise, **false**. Default set to **true**.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

`PaintPriceMarkers`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          PaintPriceMarkers = true; // Indicator plots values
display in the y-axis
          AddPlot(Brushes.Orange, "SMA");
     }
}
```

11.6.8.12 ShowTransparentPlotsInDataBox

Definition

Determines if plot(s) values which are set to a Transparent brush display in the chart data box. The default behavior is to hide any plots which have been configured as a Transparent brush, however this behavior can be changed by setting **ShowTransparentPlotsInDataBox** to **true** on the indicator.

Property Value
This property returns **true** if transparent indicator plot(s) values display in the chart data box; otherwise, **false**. Default set to **false**.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax
```
ShowTransparentPlotsInDataBox
```

Examples
```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          ShowTransparentPlotsInDataBox = true;
          AddPlot(Brushes.Transparent, "MyPlot");
     }
}
```

11.6.9 Market Analyzer Column

Custom Market Analyzer columns can be used to further enhance your Market Analyzer experience by providing custom columns displaying values of your choosing. The methods and properties covered in this section are unique to custom Market Analyzer Column development.

In this section

CurrentText	Sets text to be displayed in the Market Analyzer column.
CurrentValue	The value to be displayed in the Market Analyzer Column.
DataType	Determines the data type displayed in a Market Analyzer Column.

FormatDecimals	Rounds the value contained in CurrentValue to a specified number of decimal places before displaying it in the Market Analyzer column.
IsEditable	Determines if a Market Analyzer Column is editable.
PriorValue	Contains the last value of CurrentValue. `PriorValue` is assigned the value of `CurrentValue` immediately before `CurrentValue` is updated.

11.6.9.1 CurrentText

Definition
Sets text to be displayed in the Market Analyzer column.

> **Note**: `CurrentText` will overrule any value set for CurrentValue. If both `CurrentValue` and `CurrentText` have assigned values, the value of `CurrentText` will display in the column.

Property Value
A `string` representing text to display in the column

Syntax
```
CurrentText
```

Example


```
protected override void OnMarketData(MarketDataEventArgs
marketDataUpdate)
{
    // Print "Ask" in the column if an Ask price update is received
    if(marketDataUpdate.MarketDataType == MarketDataType.Ask)
        CurrentText = "Ask";
}
```

11.6.9.2 CurrentValue

Definition
The value to be displayed in the Market Analyzer Column

Property Value
A `double` representing the value to be displayed in the column

Syntax
`CurrentValue`

Example

```
protected override void OnMarketData(Data.MarketDataEventArgs
marketDataUpdate)
{
     CurrentValue = marketDataUpdate.Price;
}
```

11.6.9.3 DataType

Definition
Determines the data type displayed in a Market Analyzer Column.

Syntax
`DataType`

Example

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          DataType    = typeof(string);
          IsEditable = true;
     }
}
```

11.6.9.4 FormatDecimals

Definition
Rounds the value contained in CurrentValue to a specified number of decimal places before displaying it in the Market Analyzer column.

Property Value
An `int` representing a number of decimal places to which to round `CurrentValue`

Syntax
`FormatDecimals`

Example

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Round CurrentValue to one decimal place
        FormatDecimals = 1;
    }
}

protected override void OnMarketData(MarketDataEventArgs
marketDataUpdate)
{
    CurrentValue = marketDataUpdate.Price;
}
```

11.6.9.5 IsEditable

Definition

Determines if a Market Analyzer Column is editable.

Property Value

This property returns **true** if the Market Analyzer Column can be edited; otherwise, **false**.

Syntax

```
IsEditable
```

Example

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          DataType   = typeof(string);
          IsEditable = true;
     }
}
```

11.6.9.6 OnRender()

Definition

Used to draw custom content to a **Market Analyzer Column**, such as a graph.

This method is called during the following conditions:

- The Market Analyzer is scrolled

- The user changes the Market Analyzer's properties through the Properties menu
- The Market Analyzer first loads (e.g. restoring from a workspace)
- The height / width of the Market Analyzer window changes
- A user re-sizes the content area by dragging the splitter between the columns

Note: While similar to a Chart Indicator's OnRender() method, the **Market Analyzer Column** uses WPF Drawing Context class, rather than the **SharpDX** library used for chart rendering. Concepts between these two methods are guaranteed to be different.

Method Return Value

This method does not return a value.

Syntax

You must override the method in your **Market Analyzer column** with the following syntax:

```
protected override void OnRender(DrawingContext dc, System.Windows.Size renderSize)
{

}
```

Method Parameters

dc	The drawing context for the column
renderSize	The rendering size for the column

Tip: In order to force **OnRender()** to be called under a specific condition, call the OnPropertyChanged() method which will force the entire column to repaint. This approach should be used instead of calling **OnRender()** directly.

Examples

```
protected override void OnRender(DrawingContext dc,
System.Windows.Size renderSize)
{
   // Rendering logic for our Market Analyzer column
}
```

11.6.9.7 PriorValue

Definition

Contains the last value of CurrentValue. `PriorValue` is assigned the value of `CurrentValue` immediately before `CurrentValue` is updated.

Property Value

A `double` containing the last value contained in `CurrentValue` before its most recent update

Syntax

`PriorValue`

Example

```
protected override void OnMarketData(MarketDataEventArgs
marketDataUpdate)
{
    if (marketDataUpdate.MarketDataType == MarketDataType.Last)
    {
        CurrentValue = marketDataUpdate.Price;

        // Trigger an alert if the current Last price update is
greater than the previous one
        if(CurrentValue > PriorValue)
            Alert("MA Alert", Priority.High, "Check Market
Analyzer", "", 30, Brushes.Black, Brushes.White);
    }
}
```

11.6.10 Optimization Fitness

Custom Optimization Fitnesses can be used when optimizing to help you choose custom metrics your Strategy can be measured against. The methods and properties covered in this section are unique to custom Optimization Fitness development.

In this section

OnCalculatePerformanceValue()	This method calculates the value for the Optimization Fitness.
Value	The value an optimization would be calculating against when using this Optimization Fitness.

11.6.10.1 OnCalculatePerformanceValue()

Definition
This method calculates the value for the Optimization Fitness.

Syntax
```
protected override void OnCalculatePerformanceValue(StrategyBase strategy)
{

}
```

Examples


```
protected override void OnCalculatePerformanceValue(StrategyBase
strategy)
{
     Value =
strategy.SystemPerformance.AllTrades.TradesPerformance.Percent.Draw
down;
}
```

11.6.10.2 Value

Definition
The value an optimization would be calculating against when using this Optimization Fitness.

Property Value
A `double` value.

Syntax
```
Value
```

Examples


```
protected override void OnCalculatePerformanceValue(StrategyBase
strategy)
{
     Value =
strategy.SystemPerformance.AllTrades.TradesPerformance.Percent.Draw
down;
}
```

11.6.11 Optimizer

Custom Optimizers can be used to optimize your Strategy through different algorithms. These may allow you to make trade offs like being able to find adequate results quickly as opposed to trying to find the absolute best result but through a time consuming process. The methods and properties covered in this section are unique to custom Optimizer development.

In this section

NumberOfIterations	Informs the Strategy Analyzer how many iterations of optimizing it needs to do.
OnOptimize()	This method must be overridden in order to optimize a strategy.
OptimizationParameters	The optimization parameters selected for the optimization run.
RunIteration()	Runs an iteration of backtesting for the optimizer.
SupportsMultiObjectiveOptimization	Informs the Strategy Analyzer if this Optimizer can do multi-objective optimizations.

11.6.11.1 NumberOfIterations

Definition

Informs the Strategy Analyzer how many iterations of optimizing it needs to do.

Property Value

An `int` value.

Syntax

```
NumberOfIterations
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
          Name = "MyOptimizer";
     else if (State == State.Configure && Strategies.Count > 0)
          NumberOfIterations = 1;
}
```

11.6.11.2 OnOptimize()

Definition

This method must be overridden in order to optimize a strategy. This method is called once per optimization run (not once per iteration).

Method Return Value

This method does not return a value.

Syntax

You must override the method in your Optimizer with the following syntax.

```
protected override void OnOptimize()
{

}
```

Examples

```
protected override void OnOptimize()
{
     // If there is no optimization objective, return
     if (Strategies[0].OptimizationParameters.Count == 0)
          return;

     // Optimizer logic
}
```

11.6.11.3 OptimizationParameters

Definition

The optimization parameters selected for the optimization run. (e.g. user parameters or Data Series)

Property Value

A bool value.

Syntax
```
Strategies[0].OptimizationParameters
```

Examples
```
protected override void OnOptimize()
{
     // If there are no optimization parameters to optimize, return
     if (Strategies[0].OptimizationParameters.Count == 0)
          return;

     // Do something with the optimization parameter
     Parameter parameter = Strategies[0].OptimizationParameters[0];
}
```

11.6.11.4 RunIteration()

Definition
Runs an iteration of backtesting for the optimizer

Method Return Value
This method does not return a value.

Syntax
```
RunIteration()
```

Examples
```
protected override void OnOptimize()
{
     // Optimizer logic
     RunIteration();
}
```

11.6.11.5 SupportsMultiObjectiveOptimization

Definition
Informs the Strategy Analyzer if this Optimizer can do multi-objective optimizations.

Property Value
A bool value.

Syntax
SupportsMultiObjectiveOptimization

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          Name = "MyOptimizer";
          SupportsMultiObjectiveOptimization = true;
     }
}
```

11.6.12 Performance Metrics

Custom Performance Metrics can be used when generating Trade Performance statistics.

Once custom performance metrics are created be sure to enable their usage in Tools > Options > General or else they will not be available in the Strategy Analyzer or Trade Performance windows.

In this section

Format()	This method allows you to customize the rendering of the performance value on the Summary grid.
OnAddTrade()	This method is called as each trade is added.
OnCopyTo()	Called as the values of a trade metric are saved.
OnMergePerformanceMetric()	This method is called when the Performance Metric would be aggregated and merged together.
PerformanceUnit	Enumeration defining each type of PerformanceUnit calculated by NinjaTrader. Used to store a value for this performance type in PerformanceMetrics.

Values	The Values array holds an 5 values corresponding to each Cbi.PerformanceUnit.

11.6.12.1 Format()

Definition

This method allows you to customize the rendering of the performance value on the Summary grid.

Syntax

```
public override string Format(object value, Cbi.PerformanceUnit unit, string
propertyName)
{

}
```

Examples

```
public override string Format(object value, Cbi.PerformanceUnit
unit, string propertyName)
{
     double[] tmp = value as double[];
     if (tmp != null && tmp.Length == 5)
          switch (unit)
          {
               case Cbi.PerformanceUnit.Currency : return
Core.Globals.FormatCurrency(tmp[0], denomination);
               case Cbi.PerformanceUnit.Percent   : return
tmp[1].ToString("P");
               case Cbi.PerformanceUnit.Pips : return
Math.Round(tmp[
2]).ToString(Core.Globals.GeneralOptions.CurrentCulture);
               case Cbi.PerformanceUnit.Points : return
Math.Round(tmp[
3]).ToString(Core.Globals.GeneralOptions.CurrentCulture);
               case Cbi.PerformanceUnit.Ticks : return
Math.Round(tmp[
4]).ToString(Core.Globals.GeneralOptions.CurrentCulture);
          }
     return value.ToString();
}
```

11.6.12.2 OnAddTrade()

Definition
This method is called as each trade is added. You would add any custom math you wanted to do here.

> **Note**: If your performance metric only needs to iterate through all trades at the end to perform its calculation and does not need to be calculated on each trade then using the property approach (On demand example) will have less of a performance impact.

Syntax
```
protected override void OnAddTrade(Cbi.Trade trade)
{

}
```

Examples
```
protected override void OnAddTrade(Cbi.Trade trade)
{
     Values[(int)Cbi.PerformanceUnit.Currency] +=
trade.ProfitCurrency;
     Values[(int)Cbi.PerformanceUnit.Percent]  +=
trade.ProfitPercent;
     Values[(int)Cbi.PerformanceUnit.Pips]     += trade.ProfitPips;
     Values[(int)Cbi.PerformanceUnit.Points]   +=
trade.ProfitPoints;
     Values[(int)Cbi.PerformanceUnit.Ticks]    +=
trade.ProfitTicks;
}
```

11.6.12.3 OnCopyTo()

Definition
Called as the values of a trade metric are saved.

Syntax
```
protected override void OnCopyTo(PerformanceMetricBase target)
{

}
```

Examples

```
protected override void OnCopyTo(PerformanceMetricBase target)
{
   // You need to cast, in order to access the right type
   SampleCumProfit targetMetrics = (target as SampleCumProfit);

   if (targetMetrics != null)
      Array.Copy(Values, targetMetrics.Values, Values.Length);
}
```

11.6.12.4 OnMergePerformanceMetric()

Definition

This method is called when the Performance Metric would be aggregated and merged together (E.g. on the Strategy Analyzer's total row).

Syntax

```
protected override void OnMergePerformanceMetric(PerformanceMetricBase merge)
{

}
```

Examples

```
protected override void
OnMergePerformanceMetric(PerformanceMetricBase target)
{
   // You need to cast, in order to access the right type
   SampleCumProfit targetMetrics = (target as SampleCumProfit);

   // This is just a simple weighted average sample
   if (targetMetrics != null && TradesPerformance.TradesCount +
targetMetrics.TradesPerformance.TradesCount > 0)
      for (int i = 0; i < Values.Length; i++)
         targetMetrics.Values[i] = (targetMetrics.Values[i] *
targetMetrics.TradesPerformance.TradesCount + Values[i] *
TradesPerformance.TradesCount) / (TradesPerformance.TradesCount +
targetMetrics.TradesPerformance.TradesCount);
}
```

11.6.12.5 PerformanceUnit

Definition

Enumeration defining each type of PerformanceUnit calculated by NinjaTrader. Used to store a value for this performance type in PerformanceMetrics.

Syntax

```
PerformanceUnit.Currency
PerformanceUnit.Percent
PerformanceUnit.Pips
PerformanceUnit.Points
PerformanceUnit.Ticks
```

Examples

```
//Prints unrealized PnL in ticks at the close of each bar
Print(Position.GetUnrealizedProfitLoss(PerformanceUnit.Ticks,
Close[0]));
```

11.6.12.6 Values

Definition

The Values array holds an 5 values corresponding to each Cbi.PerformanceUnit. NinjaTrader will then access the Values property to display the calculated performance metric in the UI. You can also access these performance metrics for a NinjaScript strategy.

Syntax

```
public double[] Values
{ get; private set; }
```

Calculating Values OnAddTrade Example

```
protected override void OnAddTrade(Cbi.Trade trade)
{
        Values[(int)Cbi.PerformanceUnit.Currency]  +=
trade.ProfitCurrency;
        Values[(int)Cbi.PerformanceUnit.Percent]    = (1.0 +
Values[(int)Cbi.PerformanceUnit.Percent]) * (1.0 +
trade.ProfitPercent) - 1;
        Values[(int)Cbi.PerformanceUnit.Pips]      +=
trade.ProfitPips;
        Values[(int)Cbi.PerformanceUnit.Points]    +=
trade.ProfitPoints;
        Values[(int)Cbi.PerformanceUnit.Ticks]     +=
trade.ProfitTicks;
}

// The attribute determines the name of the performance value on
the grid
[Display("MyPerformanceMetric", Order = 0)]
public double[] Values
{ get; private set; }
```

Calculating Values On Demand Example

```
// The attribute determines the name of the performance value on
the grid
[Display("MyPerformanceMetric", Order = 0)]
public double[] Values
{
   get
   {
      return /*Your custom math here*/
   }

   private set;
}
```

11.6.13 Share Service

Custom **Share Services** can be developed in order to enable users to share content from the NinjaTrader application to various websites and social media networks via the Sharing Services dialog. NinjaTrader comes pre-configured with **Share Services** for an Email adapter and Test message via email adapter, however a custom adapter can be developed for any website, forum, or social media network by following their public API documentation and guidelines.

In this section

CharacterLimit	Determines the maximum number of characters the social network allows.
CharactersReservedPerMedia	Sets the number of characters allowed when attaching an image to ensure that character count is properly calculated.
Icon	The shape which displays within the Share window when sharing content.
UseOAuth	If this property is set to true, a Connect button will appear in the dialogue for configuring the adapter that will call OnAuthorizeAccount() when the user clicks it.
IsConfigured	Sets when the Share Service is correctly configured.

IsDefault	Sets the default Share Service used when the type of sharing service is selected (email for example).
IsImageAttachmentSupported	Determines if the Share Service will allow for images as attachments.
OnAuthorizeAccount()	If the UseOAuth property is set to true, this method will be called when the user clicks the Connect button in the Share Services dialogue under Tools -> Options.
OnShare()	This method is called when the user clicks OK on the Share window in NinjaTrader. This method can also be called by Alerts and general NinjaScript objects.
Signature	Sets the text appended to the end of the user's message.

11.6.13.1 CharacterLimit

Definition

Determines the maximum number of characters the social network allows. Signature, text, and links all contribute to this character count displayed on the share window.

A value of `int.MaxValue` determines no practical limit and will make the character count not appear on the Share window.

Property Value

A `int` value that represents the maximum number of characters the social network allows.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

```
CharacterLimit
```

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        CharacterLimit       = 280;
    }
}
```

11.6.13.2 CharactersReservedPerMedia

Definition

Sets the number of characters allowed when attaching an image to ensure that character count is properly calculated.

> **Note**: Social networks which limit the number of characters for each post, will have a defined number of characters that are reserved when an image or other media is attached.

Property Value

A `int` value that represents the number of characters reserved when attaching an image or other media.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

`CharactersReservedPerMedia`

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        CharactersReservedPerMedia  = 40;
    }
}
```

11.6.13.3 Icon

Definition

The shape which displays within the Share window when sharing content. Since this is a standard object, any type of icon can be used (unicode characters, custom image file resource, geometry path, etc). For more information on using images to create icons, see the Using Images with Custom Icons page.

> **Note**: When using UniCode characters, first ensure that the desired characters exist in the icon pack for the font family used in NinjaTrader.

Property Value

A generic virtual `object` representing the drawing tools menu icon. This property is read-only.

Syntax

You must override this property using the following syntax:

```
public override object Icon
```

Examples

```
public override object Icon
{
   get
   {
      //use a unicode character as our string which will render an
arrow
      string uniCodeArrow = "\u279A";
      return uniCodeArrow;
   }
}
```

11.6.13.4 UseOAuth

Definition

If this property is set to true, a Connect button will appear in the dialogue for configuring the adapter that will call OnAuthorizeAccount() when the user clicks it.

Property Value

A `bool` value determining if the OnAuthorizeAccount() method should be called in order to authorize the account to the social service.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults**

Syntax
UseOAuth

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      UseOAuth     = true;
   }
}
```

11.6.13.5 IsConfigured

Definition
Sets when the Share Service is correctly configured. Typically this would be set after the account is authorized, at which point the adapter will allow for the user to share content to the sharing service.

> **Note**: It is not required for a **Share Service** to authorize a user, therefore it is possible to set **IsConfigured** to true in State.SetDefaults which will bypass any sort of authorization or additional setup that may be needed for the share adapter.

Property Value
A `bool` value when **true** determines if the Share Adapter is properly configured.

Syntax
IsConfigured

Examples

```
public override void OnAuthorizeAccount()
{
   //Authorization logic would be here, after success, set
IsConfigured to true.

   IsConfigured = true;
}
```

11.6.13.6 IsDefault

Definition

Sets the default Share Service used when the type of sharing service is selected.

For example, if you are using two different email adapters, you may set one to be the default when the user selects the email sharing service. Setting this property as the default would only apply to any email adapters and would not apply to any other types of sharing services which have their own respective default adapter.

Property Value

A `bool` value that represents if the current adapter is default **Share Service** used for that type of sharing service.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults**

Syntax

`Default`

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        Default       = false;
    }
}
```

11.6.13.7 IsImageAttachmentSupported

Definition
Determines if the Share Service will allow for images as attachments.

Property Value
A `bool` value when false, screenshots will be unable to be sent to the social network.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax
```
IsImageAttachmentSupported
```

Examples
```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      IsImageAttachmentSupported    = false;
   }
}
```

11.6.13.8 OnAuthorizeAccount()

Definition
If the IsAuthorizationRequired property is set to true, this method will be called when the user clicks the Connect button in the Share Services dialogue under Tools -> Options. When this method is called, it will allow you go through the handshake process for authorizing the account to a sharing service. For example, you can obtain user tokens for posting on their behalf to social networks using OAuth authentication.

Documentation on the OAuth handshake process can be found from the official OAuth website: http://oauth.net/code/

Specific documentation for the authorization process for a particular sharing service can be found on it's public API resources located on their website.

Method Return Value
An asynchronous Task

Parameters
This method does not require any parameters

Syntax
This method is not required to be overridden. You may override the method in your **Share Service** with the following syntax if needed:

```
public override async Task OnAuthorizeAccount()
{

}
```

Examples

```
public override async Task OnAuthorizeAccount()
{
   //MyShareServicesToken() is a place holder for an actual API's
token method
   string result = await MyShareServicesToken("myToken");

   // result is also a place holder
   if(result == "APIErrorCode123")
   {
      Print("Unable to authorize token");
      return;
   }

   // please see the your API's OUATH documenation for proper
handshake usage
   else Print("Success!");
}
```

11.6.13.9 OnShare()

Definition
This method is called when the user clicks OK on the Share window in NinjaTrader. This method can also be called by Alerts and general NinjaScript objects.

Method Return Value
This method does not return a value

Parameters

text	The message being sent to the social network or other Share provider. This is what appears in the textbox of the Share window

imageFilePath	Optional path to screenshot or other image to be sent to the social network or other Share provider

Syntax

You must override the method in your Share Service with the following syntax.

public override void OnShare(string *text*, string *imageFilePath*)
{

}

Examples

```
public override void OnShare(string text, string imgFilePath)
{
	// place your share service logic here
}
```

11.6.13.1 Signature

Definition

Sets the text appended to the end of the user's message. It is uneditable by the user, and contributes to the character count of the overall message.

You can set it to an empty string if it does not apply to your adapter. In that case, the Signature label will not appear in the Share window.

Property Value

A `string` value which is appended to the end of the user's message.

Syntax

`Signature`

Examples

```
//example #1, adds text "This message was sent from NinjaTrader" at
the end of the message".
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Signature   = "This message was sent from NinjaTrader";
   }
}

//example #2, uses an empty string which does not add any
additional text to the message
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Signature   = string.Empty;
   }
}
```

11.6.14 Strategy

The methods and properties covered in this section are unique to custom strategy development.

In this section

Account	Represents the real-world or simulation **Account** configured for the strategy.
AddChartIndicator()	Adds an indicator to the strategy only for the purpose of displaying it on a chart.
AddPerformanceMetric()	Adds an instance of custom Performance Metric to a strategy used in strategy calculations.
ATM Strategy Methods	Adds ATM strategies to manage your position
BarsRequiredToTrade	The number of historical bars required before the strategy starts processing order methods called in the OnBarUpdate() method.

BarsSinceEntryExecution()	Returns the number of bars that have elapsed since the last specified entry.
BarsSinceExitExecution()	Returns the number of bars that have elapsed since the last specified exit.
ChartIndicators	Contains a collection of Indicators which have been added to the strategy instance using AddChartIndicator().
CloseStrategy()	Cancels all working orders, closes any existing positions, and finally disables the strategy.
ConnectionLossHandling	Sets the manner in which your strategy will behave when a connection loss is detected.
DaysToLoad	Determines the number of trading days which will be configured when loading the strategy from the **Strategies Grid**.
DefaultQuantity	An order size variable that can be set either programmatically or overriden via the Strategy that determines the quantity of an entry order.
DisconnectDelaySeconds	Determines the amount of time a disconnect would have to last before connection loss handling takes action.
EntriesPerDirection	Determines the maximum number of entries allowed per direction while a position is active based on the EntryHandling property.
EntryHandling	Sets the manner in how entry orders will handle.
Execution	Represents a read only interface that exposes information regarding an execution (filled order) resulting from an order and is passed as a parameter in the OnExecutionUpdate() method.
ExitOnSessionCloseSeconds	The number of seconds before the actual session end time that the "IsExitOnSessionCloseStrategy" function will trigger.

IncludeCommission	Determines if the strategy performance results will include commission on a historical backtest.
IncludeTradeHistoryInBacktest	Determines if the strategy will save orders, trades, and execution history.
IsAdoptAccountPositionAware	Determines if the strategy is programmed in a manner capable of handling real-world account positions.
IsExitOnSessionCloseStrategy	Determines if the strategy will cancel all strategy generated orders and close all open strategy positions at the close of the session.
IsFillLimitOnTouch	Determines if the strategy will use a more liberal fill algorithm for back-testing purposes only.
IsInstantiatedOnEachOptimizationIteration	Determines if the strategy should be re-instantiated (re-created) after each optimization run when using the Strategy Analyzer Optimizer.
IsInStrategyAnalyzer	Determines if the current NinjaScript Strategy is run from a Strategy Analyzer chart.
IsTradingHoursBreakLineVisible	Plots trading hours break lines on the indicator panel.
IsWaitUntilFlat	Indicates the strategy is currently waiting until a flat position is detected before submitting live orders.
NumberRestartAttempts	Determines the maximum number of restart attempts allowed within the last x minutes defined in RestartsWithinMinutes when the strategy experiences a connection loss.
OnAccountItemUpdate()	An event driven method used for strategies which is called for each AccountItem update for the account on which the strategy is running.

OnExecutionUpdate()	An event driven method which is called on an incoming execution of an order managed by a strategy.
OnOrderTrace()	An event driven method used for strategies which will allow you to customize the output of TraceOrders.
OnOrderUpdate()	An event driven method which is called each time an order managed by a strategy changes state.
OnPositionUpdate()	An event driven method which is called each time the position of a strategy changes state.
OptimizationPeriod	Reserved for Walk-Forward Optimization, this property determines the number of days used for the "in sample" backtest period for a given strategy. See also TestPeriod.
Order	Represents a read only interface that exposes information regarding an order.
Order Methods	NinjaScript provides several approaches you can use for order placement within your NinjaScript strategy.
OrderFillResolution	Determines how strategy orders are filled during historical states.
OrderFillResolutionType	Determines the bars type which will be used for historical fill processing.
OrderFillResolutionValue	Determines the bars period interval value which will be used for historical fill processing.
PerformanceMetrics	Holds an array of PerformanceMetrics objects that represent custom metrics that can be used for strategy calcuations.
Plots	A collection holding all of the Plot objects that define their visualization characteristics.
Position	Represents position related information that pertains to an instance of a strategy.

PositionAccount	Represents position related information that pertains to real-world account (live or simulation).
Positions	Holds an array of Position objects that represent positions managed by the strategy.
PositionsAccount	Holds an array of PositionAccount objects that represent positions managed by the strategy's account.
RealtimeErrorHandling	Defines the behavior of a strategy when a strategy generated order is returned from the broker's server in a "Rejected" state.
RestartsWithinMinutes	Determines within how many minutes the strategy will attempt to restart.
SetOrderQuantity	Determines how order sizes are calculated for a given strategy.
Slippage	Sets the amount of slippage in ticks per execution used in performance calculations during backtests.
StartBehavior	Sets the start behavior of the strategy. See Syncing Account Positions for more information.
StopTargetHandling	Determines how stop and target orders are submitted during an entry order execution.
StrategyBaseConverter	A custom TypeConverter class handling the designed behavior of an strategy's property descriptor collection.
SystemPerformance	The SystemPerformance object holds all trades and trade performance data generated by a strategy.
TestPeriod	Reserved for Walk-Forward Optimization, this property determines the number of days used for the "out of sample" backtest period for a given strategy.
TimeInForce	Sets the time in force property for all orders generated by a strategy.

TraceOrders	Determines if OnOrderTrace() would be called for a given strategy.
Trade	A Trade is a completed buy/sell or sell/buy transaction. It consists of an entry and exit execution.
TradeCollection	A collection of Trade objects.
TradesPerformanceValues	Performance values of a collection of Trade objects.
WaitForOcoClosingBracket	Determines if the strategy will submit both legs of an OCO bracket before submitting the pair to the broker.

11.6.14.1 Account

Definition
Represents the real-world or simulation **Account** configured for the strategy.

Property Value
An Account object configured for the strategy

Syntax
```
Account
```

Examples
```
//Displays text on chart indicating what account the strategy is
applied to
Draw.TextFixed(this, "tag1", "Strategy is applied to " +
Account.Name, TextPosition.BottomRight);
```

11.6.14.2 AddChartIndicator()

Definition
Adds an indicator to the strategy only for the purpose of displaying it on a chart.

Notes:
- Only the Plot properties of an indicator added by AddChartIndicator() will be accessible in the Indicators dialogue on charts. Other properties must be set in code.

- To add Bars objects to your strategy for calculation purposes see the AddDataSeries() method.
- An indicator being added via AddChartIndicator() cannot use any additional data series hosted by the calling strategy, but can only use the strategy's primary data series. If you wish to use a different data series for the indicator's input, you can add the series in the indicator itself and explicitly reference it in the indicator code (please make sure though the hosting strategy has the same AddDataSeries() call included as well)
  - If a secondary or null Bars series is specified by the calling strategy (not the indicator itself), the strategy's primary series will be substituted instead.
- Dynamically using DrawOnPricePanel in an indicator outside of State.SetDefaults may show issues when working with that indicator through a hosting strategy via AddChartIndicator().

Method Return Value

This method does not return a value.

Syntax

```
AddChartIndicator(IndicatorBase indicator)
```

Warning: This method should **ONLY** be called from the OnStateChange() method during **State.DataLoaded**

Parameters

indicator	An indicator object

Examples

```
protected override void OnStateChange()
{
    if (State == State.DataLoaded)
    {
        // Charts a 20 period simple moving average to the chart
        AddChartIndicator(SMA(20));
    }
}
```

Tip: If you are adding an indicator which is dependent on the correct State of the indicator, you will need to ensure that you are also calling the indicator from the strategy in OnBarUpdate(), otherwise your indicator will only process in **State.RealTime** for performance optimizations.

```
protected override void OnStateChange()
{
   if (State == State.DataLoaded)
   {
      // Charts a 20 period simple moving average to the chart
      AddChartIndicator(SMA(20));
   }
}

protected override void OnBarUpdate()
{
   // call SMA() historically to ensure the indicator processes its
historical states as well
   double sma = SMA(20)[0];
}
```

11.6.14.3 AddPerformanceMetric()

Definition

Adds an instance of custom Performance Metric to a strategy used in strategy calculations.

Method Return Value

This method does not return a value.

Syntax

```
AddPerformanceMetric(PerformanceMetricBase performanceMetric)
```

Warning: This method should **ONLY** be called from the OnStateChange() method during **State.Configure**

Parameters

performanceMetric	The performance metric object to be added

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          AddPerformanceMetric(new
NinjaTrader.NinjaScript.PerformanceMetrics.SampleCumProfit());
     }
}
```

11.6.14.4 ATM Strategy Methods

ATM Strategy Methods

From a NinjaScript strategy it is possible to use ATM strategies to manage your positions. Benefit of such an approach is that you can use the NinjaScript strategy to generate automated entry signals and once entered, you can delegate exit management to an ATM strategy which allows you degrees of manual control over how to close out of a trade.

For more information please see the Using ATM Strategies section.

ATM Strategy Management

- AtmStrategyCancelEntryOrder()
- AtmStrategyChangeEntryOrder()
- AtmStrategyChangeStopTarget()
- AtmStrategyClose()
- AtmStrategyCreate()

ATM Strategy Monitoring

- GetAtmStrategyEntryOrderStatus()
- GetAtmStrategyMarketPosition()
- GetAtmStrategyPositionAveragePrice()
- GetAtmStrategyPositionQuantity()
- GetAtmStrategyRealizedProfitLoss()
- GetAtmStrategyStopTargetOrderStatus()
- GetAtmStrategyUniqueId()
- GetAtmStrategyUnrealizedProfitLoss()

11.6.14.4.1 AtmStrategyCancelEntryOrder()

Definition

Cancels the specified entry order determined by the string "orderId" parameter.

Notes:

1. This method is intended **ONLY** for orders submitted as Atm Entry Orders and assumes the OrderState is **NOT** terminal (i.e., Cancelled, Filled, Rejected, Unknown).

2. If the specified order does not exist, the method returns false and an error is logged.

Method Return Value
Returns **true** if the specified order was found; otherwise **false**.

Syntax
`AtmStrategyCancelEntryOrder(string orderId)`

Warning: This method should **ONLY** be called once the strategy State has reached **State.Realtime**

Parameters

orderId	The unique identifier for the entry order

Examples

```
protected override void OnBarUpdate()
{
   // ATM strategy methods only work during real-time
   if (State != State.Realtime)
      return;

   string[] entryOrder = GetAtmStrategyEntryOrderStatus("orderId");

   // checks if the entry order exists
   // and the order state is not already cancelled/filled/rejected
   if (entryOrder.Length > 0 && entryOrder[2] == "Working")
   {
      AtmStrategyCancelEntryOrder("orderId");
   }
}
```

11.6.14.4.2 AtmStrategyChangeEntryOrder()

Definition
Changes the price of the specified entry order.

Method Return Value
Returns **true** if the specified order was found; otherwise **false**.

Syntax

```
AtmStrategyChangeEntryOrder(double limitPrice, double stopPrice, string orderId)
```

Parameters

limitPrice	Order limit price
stopPrice	Order stop price
orderId	The unique identifier for the entry order

Examples

```
protected override void OnBarUpdate()
{
     AtmStrategyChangeEntryOrder(GetCurrentBid(), 0,
"orderIdValue");
}
```

11.6.14.4.3 AtmStrategyChangeStopTarget()

Definition

Changes the price of the specified order of the specified ATM strategy.

Method Return Value

Returns **true** if the specified order was found; otherwise **false**.

Syntax

```
AtmStrategyChangeStopTarget(double limitPrice, double stopPrice, string orderName,
string atmStrategyId)
```

Parameters

limitPrice	Order limit price
stopPrice	Order stop price
orderName	The order name such as "Stop1" or "Target2"
atmStrategyId	The unique identifier for the ATM strategy

Examples

```
protected override void OnBarUpdate()
{
     AtmStrategyChangeStopTarget(0, SMA(10)[0], "Stop1",
"AtmIdValue");
}
```

11.6.14.4.4 AtmStrategyClose()

Definition

Cancels any working orders and closes any open position of a strategy using the default ATM strategy close behavior.

Method Return Value

Returns **true** if the specified ATM strategy was found; otherwise **false**.

> **Note**: A method return value of **true** in **NO WAY** indicates that the strategy in fact is closed. It indicates that the the specified ATM strategy was found and the internal close routine was triggered.

Syntax

```
AtmStrategyClose(string atmStrategyId)
```

Parameters

atmStrategyId	The unique identifier for the ATM strategy

Examples

```
protected override void OnBarUpdate()
{
     // Check for valid condition and create an ATM Strategy
     if (GetAtmStrategyUnrealizedProfitLoss("idValue") > 500)
          AtmStrategyClose("idValue");
}
```

11.6.14.4.5 AtmStrategyCreate()

Definition

Submits an entry order that will execute a specified ATM Strategy.

Notes:

- Please review the section on using ATM Strategies
- This method is **NOT** backtestable and will **NOT** execute on historical data
- See the AtmStrategyCancelEntryOrder() to cancel an entry order
- See the AtmStrategyChangeEntryOrder() to change the price of the entry order
- The ATM Strategy will be created asyncronous on the hosting NinjaScripts UI Thread, a callback is provided **solely** to check when the ATM Strategy is started on that thread - accessing for example price data in that outside OnBarUpdate() context **is not possible**.
- Please see the SampleATMStrategy build into NinjaTrader for example usage.

Method Return Value

This method does not return a value

Syntax

```
AtmStrategyCreate(OrderAction action, OrderType orderType, double limitPrice, double stopPrice, TimeInForce timeInForce, string orderId, string strategyTemplateName, string atmStrategyId, Action<Cbi.ErrorCode, string> callback)
```

Parameters

action	Sets if the entry order is a buy or sell order Possible values are: • OrderAction.Buy • OrderAction.Sell
orderType	Sets the order type of the entry order Possible values are: • OrderType.Limit • OrderType.Market • OrderType.MIT

	• OrderType.StopMarket • OrderType.StopLimit
limitPrice	The limit price of the order
stopPrice	The stop price of the order
timeInForce	Sets the time in force of the entry order Possible values are: • TimeInForce.Day • TimeInForce.Gtc
orderId	The unique identifier for the entry order
strategyTemplateName	Specifies which strategy template will be used
atmStrategyId	The unique identifier for the ATM strategy
callback	The callback action is used to check that the ATM Strategy is successfully started

Tip: Unlike NinjaScript Strategy orders (both managed and unmanaged), ATM strategies generated by the AtmStrategyCreate() method can then be managed manually by any order entry window such as the SuperDOM or within your NinjaScript strategy.

Examples

```
private string atmStrategyId;
private string atmStrategyOrderId;
private bool   isAtmStrategyCreated = false;

protected override void OnBarUpdate()
{
    if (State < State.Realtime)
        return;

    if (Close[0] > SMA(20)[0])
    {
        atmStrategyId = GetAtmStrategyUniqueId();
        atmStrategyOrderId = GetAtmStrategyUniqueId();

        AtmStrategyCreate(OrderAction.Buy, OrderType.Market, 0, 0,
TimeInForce.Day,
            atmStrategyOrderId, "MyTemplate", atmStrategyId,
(atmCallbackErrorCode, atmCallbackId) => {

            // checks that the call back is returned for the
current atmStrategyId stored
            if (atmCallbackId == atmStrategyId)
            {
                // check the atm call back for any error codes
                if (atmCallbackErrorCode == Cbi.ErrorCode.NoError)
                {
                    // if no error, set private bool to true to
indicate the atm strategy is created
                    isAtmStrategyCreated = true;
                }
            }
        });
    }

    if(isAtmStrategyCreated)
    {
        // atm logic
    }

    else if(!isAtmStrategyCreated)
    {
        // custom handling for a failed atm Strategy
    }
}
```

11.6.14.4.6 GetAtmStrategyEntryOrderStatus()

Definition

Gets the current state of the specified entry order.

> **Note**: If the method can't find the specified order, an empty array is returned.

Method Return Value

A `string[]` array holding three elements that represent average fill price, filled amount and order state.

Syntax

`GetAtmStrategyEntryOrderStatus(string orderId)`

Parameters

orderId	The unique identifier for the entry order

Examples

```
protected override void OnBarUpdate()
{
     string[] entryOrder =
GetAtmStrategyEntryOrderStatus("orderId");

     // Check length to ensure that returned array holds order
information
     if (entryOrder.Length > 0)
     {
          Print("Average fill price is " +
entryOrder[0].ToString());
          Print("Filled amount is " + entryOrder[1].ToString());
          Print("Current state is " + entryOrder[2].ToString());
     }
}
```

11.6.14.4.7 GetAtmStrategyMarketPosition()

Definition

Gets the current market position of the specified ATM Strategy.

> **Notes**:

1. Changes to positions will not be reflected till at least the next OnBarUpdate() event after an order fill.
2. If the ATM Strategy does not exist then MarketPosition.Flat returns
3. Please note this provides access to the current ATM strategy position, which should not be confused with the NinjaScript strategy position or account position. For more information please see the Using ATM Strategies section.

Method Return Value

MarketPosition.Flat
MarketPosition.Long
MarketPosition.Short

Syntax

```
GetAtmStrategyMarketPosition(string atmStrategyId)
```

Parameters

atmStrategyId	The unique identifier for the ATM strategy

Examples

```
protected override void OnBarUpdate()
{
     // Check if flat
     if (GetAtmStrategyMarketPosition("id") == MarketPosition.Flat)
          Print("ATM Strategy position is currently flat");
}
```

11.6.14.4.8 GetAtmStrategyPositionAveragePrice()

Definition

Gets the current position's average price of the specified ATM Strategy.

Note: Changes to positions will not be reflected till at least the next OnBarUpdate() event after an order fill.

Method Return Value

A `double` value representing the average price.

Syntax

```
GetAtmStrategyPositionAveragePrice(string atmStrategyId)
```

Parameters

atmStrategyId	The unique identifier for the ATM strategy

Examples

```
protected override void OnBarUpdate()
{
     // Check if flat
     if (GetAtmStrategyMarketPosition("id") != MarketPosition.Flat)

          Print("Average price is " +
GetAtmStrategyPositionAveragePrice("id").ToString());
}
```

11.6.14.4.9 GetAtmStrategyPositionQuantity()

Definition

Gets the current position quantity of the specified ATM Strategy.

Note: Changes to positions will not be reflected till at least the next OnBarUpdate() event after an order fill.

Method Return Value

An `int` value representing the quantity.

Syntax

```
GetAtmStrategyPositionQuantity(string atmStrategyId)
```

Parameters

atmStrategyId	The unique identifier for the ATM strategy

Examples

```
protected override void OnBarUpdate()
{
     // Check if flat
     if (GetAtmStrategyMarketPosition("idValue") !=
MarketPosition.Flat)
          Print("Position size is " +
GetAtmStrategyPositionQuantity("id").ToString());
}
```

11.6.14.4.10 GetAtmStrategyRealizedProfitLoss()

Definition

Gets the realized profit and loss value of the specified ATM Strategy.

Method Return Value

A `double` value representing the realized profit and loss.

Syntax

`GetAtmStrategyRealizedProfitLoss(string atmStrategyId)`

Parameters

atmStrategyId	The unique identifier for the ATM strategy

Examples

```
protected override void OnBarUpdate()
{
     Print("PnL is " +
GetAtmStrategyRealizedProfitLoss("id").ToString());
}
```

11.6.14.4.11 GetAtmStrategyStopTargetOrderStatus()

Definition

Gets the current order state(s) of the specified stop or target order of a still-active ATM strategy.

Notes:
1. If the method can't find the specified order(s), an empty array is returned.

2. A specified stop or target within an ATM strategy can actually hold multiple orders. For example, if your ATM strategy submits a stop and target and you receive multiple partial fills on entry with a delay of a few seconds or more between entry fills, the ATM strategy will submit stop and target orders for each partial fill all with the same price and order type.

Method Return Value

A `string[,]` multi-dimensional array holding three dimensions that represent average fill price, filled amount and order state. The length (number of elements) represents the number of orders that represent the specified name.

Syntax

```
GetAtmStrategyStopTargetOrderStatus(string orderName, string atmStrategyId)
```

Parameters

orderName	The order name such as "Stop1" or "Target2"
atmStrategyId	The unique identifier for the ATM strategy

Examples

```
protected override void OnBarUpdate()
{
     string[,] orders =
GetAtmStrategyStopTargetOrderStatus("Target1", "idValue");

     // Check length to ensure that returned array holds order
information
     if (orders.Length > 0)
     {
          for (int i = 0; i < orders.GetLength(0); i++)
          {
               Print("Average fill price is " + orders[i,
0].ToString());
               Print("Filled amount is " + orders[i,
1].ToString());
               Print("Current state is " + orders[i,
2].ToString());
          }
     }
}
```

11.6.14.4.12 GetAtmStrategyUnrealizedProfitLoss()

Definition

Gets the unrealized profit and loss value of the specified ATM Strategy.

Method Return Value

A `double` value representing the unrealized profit and loss.

Syntax

`GetAtmStrategyUnrealizedProfitLoss(string atmStrategyId)`

Parameters

atmStrategyId	The unique identifier for the ATM strategy

Examples

```
protected override void OnBarUpdate()
{
     Print("Unrealized PnL is " +
GetAtmStrategyUnrealizedProfitLoss("id").ToString());
}
```

11.6.14.4.13 GetAtmStrategyUniqueId()

Definition
Generates a unique ATM Strategy ID value.

Method Return Value
A `string` value representing a unique id value.

Syntax
`GetAtmStrategyUniqueId()`

Parameters
This method does use take any parameters.

Examples

```
protected override void OnBarUpdate()
{
     string orderId = GetAtmStrategyUniqueId();
}
```

11.6.14.5 BarsRequiredToTrade

Definition
The number of historical bars required before the strategy starts processing order methods called in the OnBarUpdate() method. This property is generally set via the UI when starting a strategy.

Note: In a multi-series strategy this restriction applies only for the primary Bars object. This means your can run into situations where the primary bars required to trade have been reached, but the additional bars required have not. Should your strategy logic intertwine calculations across different Bars objects please ensure all Bars objects have met the BarsRequiredToTrade requirement before proceeding. This can be done via checks on the CurrentBars array.

Property Value
An `int` value representing the number of historical bars. Default value is set to 20.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

BarsRequiredToTrade

Tip: When working with a multi-series strategy, real-time bar update events for a particular Bars object are only received when that Bars object has satisfied the BarsRequiredToTrade requirement. To ensure this requirement is met, please use the CurrentBars array.

Examples

Setting the default BarsRequiredToTrade value

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          BarsRequiredToTrade = 20;
     }
}
```

Checking BarsRequiredToTrade againt a CurrentBars array

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      BarsRequiredToTrade = 20;
   }
   else if (State == State.Configure)
   {
      // add 30 minute series for calcuation logic
      AddDataSeries(BarsPeriodType.Minute, 30);
   }
}

protected override void OnBarUpdate()
{
   // do not process order logic until bars required to trade
is met
   // for both primary and 30-minute series have reached their
bars required to trade
   if (CurrentBars[0] < BarsRequiredToTrade || CurrentBars[1]
< BarsRequiredToTrade)
      return;

   //order logic
}
```

11.6.14.6 BarsSinceEntryExecution()

Definition

Returns the number of bars that have elapsed since the last entry. When a signal name is provided, the number of bars that have elapsed since that last specific entry will be returned.

Method Return Value

An `int` value that represents a number of bars. A value of -1 will be returned if a previous entry does not exist.

Syntax

```
BarsSinceEntryExecution()
BarsSinceEntryExecution(string signalName)
```

The following method signature should be used when working with multi-time frame and instrument strategies:

```
BarsSinceEntryExecution(int barsInProgressIndex, string signalName, int
entryExecutionsAgo)
```

> **Note**: When working with a multi-series strategy the `BarsSinceEntryExecution()` will return you the elapsed bars as determined by the first Bars object for the instrument specified by the barsInProgressIndex.

Parameters

signalName	The signal name of an entry order specified in an order entry method.
barsInProgressIndex	The index of the Bars object the entry order was submitted against. **Note**: See the BarsInProgress property.
entryExecutionsAgo	Number of entry executions ago. Pass in 0 for the number of bars since the last entry execution.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < BarsRequiredToTrade)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterLong();

}
```

11.6.14.7 BarsSinceExitExecution()

Definition

Returns the number of bars that have elapsed since the last exit. When a signal name is provided, the number of bars that have elapsed since that last specific exit will be returned.

Method Return Value

An `int` value that represents a number of bars. A value of -1 will be returned if a previous exit does not exist.

Syntax

```
BarsSinceExitExecution()
BarsSinceExitExecution(string signalName)
```

The following method signature should be used when working with multi-time frame and instrument strategies:

```
BarsSinceExitExecution(int barsInProgressIndex, string signalName, int exitExecutionsAgo)
```

> **Note**: When working with a multi-series strategy the `BarsSinceExitExecution()` will return you the elapsed bars as determined by the first Bars object for the instrument specified in the barsInProgressIndex.

Parameters

signalName	The signal name of an exit order specified in an order exit method.
barsInProgressIndex	The index of the Bars object the entry order was submitted against. **Note**: See the BarsInProgress property.
exitExecutionsAgo	Number of exit executions ago. Pass in 0 for the number of bars since the last exit execution.

> **Tip**: Please see SetStopLoss(), SetProfitTarget() or SetTrailStop() for their corresponding signal name

Examples

```
protected override void OnBarUpdate()
{
   if (CurrentBar < BarsRequiredToTrade)
        return;

   // Only enter if at least 10 bars has passed since our last exit
or if we have never traded yet
   if ((BarsSinceExitExecution() > 10 || BarsSinceExitExecution()
== -1) && CrossAbove(SMA(10), SMA(20), 1))
        EnterLong();
}
```

11.6.14.8 ChartIndicators

Definition

Contains a collection of Indicators which have been added to the strategy instance using AddChartIndicator().

Property Value

An Indicator object

Syntax

```
ChartIndicators[int index]
```

Examples

```
if (State == State.DataLoaded)
{
   AddChartIndicator(SMA(20));

   // Set the plots color for the added indicator
   ChartIndicators[0].Plots[0].Brush = Brushes.Blue;

   // Set the added indicator to panel 1 (specified index needs to
be >= 1)
   ChartIndicators[0].Panel = 1;
}
```

11.6.14.9 CloseStrategy()

Definition

Cancels all working orders, closes any existing positions, and finally disables the strategy. This behavior can also be overridden for a given strategy if desired.

Notes:
- If you choose to override this method using custom logic, the default behavior of the CloseStrategy() method will **NOT** be executed. For this reason, it is suggested to call the base implementation of CloseStrategy() method within the virtual override to ensure that the strategy is terminated as designed, otherwise it is your responsibility to correctly manage any working orders or positions.
- CloseStrategy() will work of the current strategy position and will not factor in any StartBehavior setting, i.e. calling CloseStrategy() while the script is in a virtual historical position could result in an unwanted position
- The default CloseStrategy() handling will be applied to all series of a MultiSeries NinjaScript strategy.

Method Return Value
This method does not return a value.

Syntax
```
CloseStrategy(string signalName)
```

Warning: This method can only be call before the State has reached State.Terminated and after the State reaches State.Realtime

You may choose to override this method using the following syntax:

```
public override void CloseStrategy(string signalName)
{

}
```

Parameters

signalName	The signal name which will be used to identify the closing order. If no signal name exists or is null, "Close" will be substituted instead.

Examples

Basic usage of CloseStrategy

```
DateTime StartTime = new DateTime();
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "ExampleStrategy";
   }

   else if (State == State.Transition)
      StartTime = Core.Globals.Now;
}

protected override void OnBarUpdate()
{
   // if we're still in position 45 minutes after the start time,
close strategy
   if(Position.MarketPosition != MarketPosition.Flat && Time[0] >=
StartTime.AddMinutes(45))
      CloseStrategy("My Custom Close");
}
```

Overriding the Default CloseStrategy logic

```
public override void CloseStrategy(string signalName)
{
   Print("Executing Custom Close Logic");
      // custom close logic

   // call default close action
   base.CloseStrategy(signalName);
}
```

11.6.14.1 ConnectionLossHandling

Definition

Sets the manner in which your strategy will behave when a connection loss is detected.

When using ConnectionLossHandling.Recalculate, recalculations will only occur if the strategy was stopped based on the conditions below. Should the connection be reestablished before the strategy was stopped, the strategy will continue running without recalculating as if no disconnect occurred.

- If data feed disconnects for longer than the time specified in DisconnectDelaySeconds, the strategy is stopped.
- If the order feed disconnects and the strategy places an order action while disconnected, the strategy is stopped.

- If both the data and order feeds disconnect for longer than the time specified in DisconnectDelaySeconds, the strategy is stopped.

Property Value

An `enum` determining how the strategy will behave. Default value is set to ConnectionLossHandling.Recalculate Possible values are:

ConnectionLossHandling.KeepRunning	Keeps the strategy running. When the connection is reestablished the strategy will resume as if no disconnect occurred.
ConnectionLossHandling.Recalculate	Strategies will attempt to recalculate its strategy position when a connection is reestablished.
ConnectionLossHandling.StopStrategy	Automatically stops the strategy when disconnected for more than DisconnectDelaySeconds. No action will be taken when a connection is reestablished.

Syntax

`ConnectionLossHandling`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Keeps the strategy running as if no disconnect
occurred
          ConnectionLossHandling =
ConnectionLossHandling.KeepRunning;
     }
}
```

11.6.14.11 DaysToLoad

Definition

Determines the number of trading days which will be configured when loading the strategy from the **Strategies Grid**.

> **Notes**:
> 1. This property does **NOT** affect a strategy configured of a Chart or the Strategy Analyzer.
> 2. A trading day is defined by a Trading Hour template

Property Value

An `int` value determining the number of trading days to load for historical data processing. Default value is 5, but can be configured and overridden from the UI.

Syntax

```
DaysToLoad
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          DaysToLoad = 15;
     }
}
```

11.6.14.12 DefaultQuantity

Definition

An order size variable that can be set either programmatically or overriden via the Strategy that determines the quantity of an entry order.

Property Value

An `int` value represents the number of contracts or shares to enter a position with. Default value is 1.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax
DefaultQuantity

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          DefaultQuantity = 1;
     }
}
```

11.6.14.13 DisconnectDelaySeconds

Definition
Determines the amount of time a disconnect would have to last before connection loss handling takes action.

Property Value
An int value represents the time required for a disconnect to last before connection loss handling actions will occur. Default value is 10.

Syntax
DisconnectDelaySeconds

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Disconnect has to be at least 10 seconds
          DisconnectDelaySeconds = 10;
     }
}
```

11.6.14.14 EntriesPerDirection

Definition
Determines the maximum number of entries allowed per direction while a position is active based on the EntryHandling property.

Note: This property **ONLY** applies to Managed order methods. When IsUnmanaged is set to **true**, Entry Handling properties will be hidden from the UI.

Property Value

An `int` value represents the maximum number of entries allowed. Default value is 1.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

```
EntriesPerDirection
```

Examples

If an open position already exists, subsequent EnterLong() calls are ignored.

```
// Example #1
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          EntriesPerDirection = 1;
          EntryHandling = EntryHandling.AllEntries;
     }
}

protected override void OnBarUpdate()
{
     if (CrossAbove(SMA(10), SMA(20), 1)
          EnterLong("SMA Cross Entry");

     if (CrossAbove(RSI(14, 3), 30, 1)
          EnterLong("RSI Cross Entry);
}
```

EnterLong() will be processed once for each uniquely named entry.

```
// Example #2
protected override void OnStateChange()
{
      EntriesPerDirection = 1;
      EntryHandling = EntryHandling.UniqueEntries;
}

protected override void OnBarUpdate()
{
     if (CrossAbove(SMA(10), SMA(20), 1)
          EnterLong("SMA Cross Entry");

     if (CrossAbove(RSI(14, 3), 30, 1)
          EnterLong("RSI Cross Entry);
}
```

11.6.14.1 EntryHandling

Definition

Sets the manner in how entry orders will handle.

Note: This property **ONLY** applies to Managed order methods. When IsUnmanaged is set to **true**, Entry Handling properties will be hidden from the UI.

Property Value

An `enum` which sets how the entry orders are handled. Default value is EntryHandling.AllEntries. Possible values include:

EntryHandling.AllEntries	NinjaScript will process all order entry methods until the maximum allowable entries set by the EntriesPerDirection property is reached while in an open position
EntryHandling.UniqueEntries	NinjaScript will process order entry methods until the maximum allowable entries set by the EntriesPerDirection property per each uniquely named entry

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

EntryHandling

Examples

Allow a maximum of two entries while a position is open

```
// Example #1
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          EntriesPerDirection = 2;
          EntryHandling = EntryHandling.AllEntries;
     }
}

protected override void OnBarUpdate()
{
     if (CrossAbove(SMA(10), SMA(20), 1)
          EnterLong("SMA Cross Entry");
}
```

EnterLong() will be processed once for each uniquely named entry.

```
// Example #2
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          EntriesPerDirection = 1;
          EntryHandling = EntryHandling.UniqueEntries;
     }
}

protected override void OnBarUpdate()
{
     if (CrossAbove(SMA(10), SMA(20), 1)
          EnterLong("SMA Cross Entry");

     if (CrossAbove(RSI(14, 3), 30, 1)
          EnterLong("RSI Cross Entry");
}
```

11.6.14.1 Execution

Definition

Represents a read only interface that exposes information regarding an execution (filled order) resulting from an order and is passed as a parameter in the OnExecutionUpdate() method.

Note: Not all executions will have associated Order objects (e.g ExitOnSessionClose executions or AtmStrategyCreate() executions)

Methods and Properties

Account	The Account the execution occurred
BarsInProgress	An `int` value representing the BarsArray in which the execution occurred
Commission	A `double` value representing the commission of an execution

ExecutionId	A `string` value representing the exchange generated execution id
Instrument	An Instrument value representing the instrument of an order
MarketPosition	The position of the execution. Possible values are: • MarketPosition.Long • MarketPosition.Short
Name	A `string` representing the name of an order which can be provided by the entry or exit signal name
Order	An Order value representing an order associated to the execution.
OrderId	A `string` representing the unique id of the order which was executed
Position	An `int` value represents the current quantity of account position at the time of execution
PositionStrategy	An `int` value represents the current quantity of strategy position at the time of execution
Price	A `double` value representing the price of an execution
Quantity	An `int` value representing quantity of an execution
Rate	A `double` value representing the exchange rate calculated for non-

	USD base products (1 if no rate was applied)
Slippage	A `double` value representing the number of ticks calculated between the last trade price and the execution price
Time	A DateTime structure representing the time the execution occurred
ToString()	A `string` representation of an execution

Examples

Finding the executions of a particular Order object

```
// Example #1
private Order entryOrder = null;

protected override void OnBarUpdate()
{
     if (entryOrder == null && Close[0] > Open[0])
         EnterLong("myEntryOrder");
}

protected override void OnExecutionUpdate(Execution execution,
string executionId, double price, int quantity, MarketPosition
marketPosition, string orderId, DateTime time)
{
    // Assign entryOrder in OnExecutionUpdate() to ensure the
assignment occurs when expected.
    // This is more reliable than assigning Order objects in
OnBarUpdate, as the assignment is not guaranteed to be complete if
it is referenced immediately after submitting
    if (execution.Order.Name == "myEntryOrder" &&
execution.Order.OrderState == OrderState.Filled)
        entryOrder = execution.order;

     if (entryOrder != null && entryOrder == execution.Order)
        Print(execution.ToString());
}
```

Generic execution logic not specific to a particular Order object

```
// Example #2
protected override void OnExecutionUpdate(Execution execution,
string executionId, double price, int quantity, MarketPosition
marketPosition, string orderId, DateTime time)
{
     // Remember to check the underlying Order object for null
before trying to access its properties
     if (execution.Order != null && execution.Order.OrderState ==
OrderState.Filled)
          Print(execution.ToString());
}
```

11.6.14.17 ExitOnSessionCloseSeconds

Definition

The number of seconds before the actual session end time that the "IsExitOnSessionCloseStrategy" function will trigger.
The time from which this property will be calculated is taken from the Trading Hours EOD property set in the strategy's Trading Hours template. The **ExitOnSessionCloseSeconds** property can either be set programatically in the OnStateChange() method or be driven by the UI at run time.

Note: This is a real-time only property, it will not have any effect on your ExitOnSessionClose time in backtesting processing historical data.

Property Value

An `int` representing the number of seconds. Default value is 30.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

`ExitOnSessionCloseSeconds`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Triggers the exit on close function 30 seconds prior
to trading day end
          IsExitOnSessionCloseStrategy = true;
          ExitOnSessionCloseSeconds = 30;
     }
}
```

11.6.14.18 IncludeCommission

Definition

Determines if the strategy performance results will include commission on a historical backtest. When **true**, the Commission Template applied to the account on which the strategy is running will be used.

Property Value

A `bool` value which returns **true** if the strategy includes commission on a historical backtest; otherwise, **false**. Default value is set to **false**.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

`IncludeCommission`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IncludeCommission = true;
     }
}
```

11.6.14.19 IncludeTradeHistoryInBacktest

Definition

Determines if the strategy will save orders, trades, and execution history. When this property is set to **false** you will see significant memory savings at the expense of having access to the detailed trading information.

Notes:

- Since trade information is not stored you will only see entry/exit executions plotted on the chart with no connecting PnL trade lines.
- This property is always defaulted to **true**, except when the strategy is running on the strategy tab.

Property Value

This property returns **true** if the strategy will include trade history; otherwise, **false**. Default is set to **true**.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.Configure** (or **State.SetDefaults** when adding the script from the strategy tab)

Syntax

```
IncludeTradeHistoryInBacktest
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Exclude trade history in a backtest to benefit from
memory savings
          IncludeTradeHistoryInBacktest = false;
     }
}

protected override void OnBarUpdate()
{
    // Stop taking trades after 10 trades have been taken since the
strategy was enabled
    if(SystemPerformance.AllTrades.Count >= 10)
        return;
}
```

11.6.14.2 IsAdoptAccountPositionAware

Definition

Determines if the strategy is programmed in a manner capable of handling real-world account positions. Once set to **true**, your strategy's "Start behavior" options will include an additional parameter named "Adopt account position" which can bet set at run-time. Only set to **true** if you have specifically programmed your strategy to be able to adopt account positions.

Property Value

This property returns **true** if the strategy can adopt account positions; otherwise, **false**. Default is set to **false**.

> **Note**: This property should **ONLY** be set from the OnStateChange() method during State.SetDefaults.

Syntax

`IsAdoptAccountPositionAware`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IsAdoptAccountPositionAware = true;
     }
}
```

11.6.14.2 IsExitOnSessionCloseStrategy

Definition

Determines if the strategy will cancel all strategy generated orders on all strategy instruments and close all open strategy positions at the close of ANY session for multi-time frame/multi-instrument strategies. This property can be set programatically in the OnStateChange() method or be driven by the UI at run time. See also "ExitOnSessionCloseSeconds".

Property Value

This property returns **true** if the strategy will exit on close; otherwise, **false**. Default value is set to **true**.

Warnings:

- This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**
- On historical data, IsExitOnSessionCloseStrategy will cause positions to be exited at the close of the last bar of the session. If you are using a non time-based Bar Type, such as Renko, and have "Break at EOD" set to False on the Data Series, this means that IsExitOnSessionCloseStrategy could trigger *after* the session close, since the last bar of the session can extend beyond the session close time in this scenario.
- Even if you're backtesting with a historical order fill resolution set to be more granular than your base primary series, the **ExitOnSessionCloseSeconds** will still be tied to the primary higher timeframe series bar. IsExitOnSessionCloseStrategy should not be used in combination with Daily Bars and **High Order Fill Resolution** since it will cause the position to close as the same time as the daily bar updates (at session close)
- This property is designed to be only used on intraday strategies

Syntax

`IsExitOnSessionCloseStrategy`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Triggers the exit on session close function 30 seconds
prior to real-time trading day end
          IsExitOnSessionCloseStrategy = true;
          ExitOnSessionCloseSeconds = 30;
     }
}
```

11.6.14.21 IsFillLimitOnTouch

Definition

Determines if the strategy will use a more liberal fill algorithm for back-testing purposes only. The default behavior of the strategy's fill algorithm is to fill a limit order once price has penetrated the limit price. However this behavior can be changed by setting **IsFillLimitOnTouch** to **true**, in which case the strategy's fill algorithm will consider a limit order filled once price has reached the limit price, but does not necessarily need to trade through the limit price

Property Value

This property returns **true** if the strategy will fill limit orders when touched; otherwise, **false**. Default is set to **false**.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

```
IsFillLimitOnTouch
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IsFillLimitOnTouch = true;
     }
}
```

11.6.14.21 IsInstantiatedOnEachOptimizationIteration

Definition

Determines if the strategy should be re-instantiated (re-created) after each optimization run when using the Strategy Analyzer Optimizer.

The **default behavior** is to re-instantiate the strategy for each optimization backtest run. However, the process of re-instantiating a strategy requires more time and computer resources to return results, which could impact the amount of time it takes to run an optimization. When **false**, the strategy is re-used to save time and computer resources. Under this design, internal properties are reset to default values after each iteration, but it is possible that user-defined properties and other custom resources may carry their state over from the previous iteration into a new backtest run. To take advantage of performance optimizations, developers may need to reset class level variables in the strategy otherwise unexpected results can occur.

Note: If you choose to take advantage of the performance benefits during strategy optimization by setting the **IsInstantiatedOnEachOptimizationIteration** property to **false**, any objects you create in your code **MUST** be reset during the appropriate **State** within the OnStateChange() method. Please see the example below on "*Manually resetting class level variables to take advantage of Strategy Analyzer optimizer performance benefits*".

Property Value

This property returns **true** if the strategy is not recycled; otherwise, **false**. Default set to **true**.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

```
IsInstantiatedOnEachOptimizationIteration
```

Tip: The default NinjaTrader indicators and strategies have been optimized to take advantage of performance optimizations as their resources are setup >= **State.Configure**. Please see the default system indicators and strategies for an idea of how you may improve your strategy and indicator performance, or you may also reference the example code below.

Examples

Using IsInstantiatedOnEachOptimizationIteration to reset class level variables

```
// A custom trades dictionary is created when strategy is
instantiated
// since we later set "IsInstantiatedOnEachOptimizationIteration"
to true,
// we are guaranteed to start with a new object on each
optimization run
private Dictionary<DateTime, string> myTrades = new
Dictionary<DateTime, string>();

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name        = "My Optimization Test 1";
      Description = "Demonstrates using
IsInstantiatedOnEachOptimizationIteration to reset a class level
variable";
      Fast        = 10;
      Slow        = 25;

      // setting to true so our custom trades dictionary is reset
on each optimization run (comes with a performance penalty)
      // This is the default behavior.
      IsInstantiatedOnEachOptimizationIteration = true;
   }

   else if (State == State.Terminated)
   {
      // Print the number of trades at the end of the optimization
      if (myTrades != null)
      {
         // if we set "IsInstantiatedOnEachOptimizationIteration"
to false (so not using the default of true), the values here would
be unexpected
         // since the custom trade dictionary was never explicitly
reset at the end of each optimization
         Print(myTrades.Count);
      }
   }
}

protected override void OnBarUpdate()
{
   if (CurrentBar < BarsRequiredToTrade)
      return;

   if (CrossAbove(SMA(Fast), SMA(Slow), 1))
   {
      EnterLong();
      myTrades.Add(Time[0], "long");

   }
```

Manually resetting class level variables to take advantage of Strategy Analyzer optimizer performance benefits

```
// A custom trades dictionary is declared when strategy is first
optimized,
// but not instantiated until later in State.DataLoaded,
private Dictionary<DateTime, string> myTrades;

// examples of other fields which need to be reset
private double myDouble;
private bool myBool;
private DateTime myDateTime;
private Order myOrderObject;
private Brush myBrushObject;
private SMA mySMAIndicator;
private Array myIntArray;
private List<object> myList;
private Series<double> mySeries;

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "My Optimization Test 2";
      Description = "Demonstrates manually resetting a class level
variable without re-instantiating the strategy";
      Fast = 10;
      Slow = 25;

      // in this case, we do not need to re-instantiate the
strategy after each optimization
      // because we are explicitly resetting the custom trade
dictionary in State.DataLoaded
      // This design of re-using the strategy instance comes with
performance benefits
      IsInstantiatedOnEachOptimizationIteration = false;
   }

   else if (State == State.DataLoaded)
   {
      // re-create custom trade dictionary on each optimization run
      // we are guaranteed to start with a new object on each
optimization run
      if (myTrades != null)
        myTrades.Clear();
      else
        myTrades = new Dictionary<DateTime, string>();

      //Any strategy defaults which are maintained do not need to
be reset if they are not mutable as the strategy runs.
      //Any strategy state that would be mutable after
State.SetDefaults needed to be reset for the next run.
      myDouble = double.MinValue;
      myBool = false;
      myDateTime = DateTime.MinValue;
```

11.6.14.24 IsInStrategyAnalyzer

Definition

Determines if the current NinjaScript Strategy is run from a Strategy Analyzer chart.

Property Value

A `bool` value when **true** the strategy is being run from the Strategy Analyzer chart; otherwise will return **false.**

Syntax

`IsInStrategyAnalyzer`

Examples

```
protected override void OnBarUpdate()
{
    // Only draw the ArrowUp on our condition if we're not in
the Strategy Analyzer chart
    if (Close[0] > SMA(High, 14)[0] && !IsInStrategyAnalyzer)
       Draw.ArrowUp(this, CurrentBar.ToString(), true, 0,
High[0] + TickSize, Brushes.Blue);
}
```

11.6.14.25 IsTradingHoursBreakLineVisible

Definition

Plots trading hours break lines on the indicator panel.

> **Note**: The indicator panel's parent chart has a similar property 'Plot session break line' which if set to **false**, will override the indicator's local setting if **true**.

Property Value

This property returns **true** if trading hours break lines are plotted on the indicator panel; otherwise, **false**. Default set to **true**.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

`IsTradingHoursBreakLineVisible`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IsTradingHoursBreakLineVisible = true;
          AddPlot(Brushes.Orange, "SMA");
     }
}
```

11.6.14.2 IsWaitUntilFlat

Definition

Indicates the strategy is currently waiting until a flat position is detected before submitting live orders.

> **Note**: This property would only apply if the strategy StartBehavior was set to StartBehavior.WaitUntilFlat or StartBehavior.WaitUntilFlatSynchronizeAccount.

Property Value

This property returns **true** if the strategy has detected it is either in a long or short position during State.Transition; otherwise **false**. Default value is set to **false**.

Syntax

```
IsWaitUntilFlat
```

Examples

```
// If a strategy is waiting for a flat position, return and print a
message
if (!IsWaitUntilFlat)
{
   Print("This strategy is currently waiting for a flat account
position to begin placing trades");
   return;
}
```

11.6.14.27 NumberRestartAttempts

Definition
Determines the maximum number of restart attempts allowed within the last x minutes defined in RestartsWithinMinutes when the strategy experiences a connection loss. If restart attempts exceeds this property within a time span shorter than or equal to RestartsWithinMinutes, then the strategy will be stopped and no further attempts will occur. The purpose of these settings is to stop the strategy should your connection be unstable and incapable of maintaining a consistent connected state.

Property Value
An `int` value represents the maximum number of restart attempts. Default value is set to 4.

Syntax
```
NumberRestartAttempts
```

Examples
```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Only allow the strategy to restart 4 times within the
MaxRestartMinutes time span
          // If disconnected more than 4 times within that time
span, stop the strategy and do not
          // attempt any further restarts.
          NumberRestartAttempts = 4;
     }
}
```

11.6.14.28 OnAccountItemUpdate()

Definition
An event driven method used for strategies which is called for each AccountItem update for the account on which the strategy is running.

> **Note**: OnAccountItemUpdate() will be called continually in real-time if a position exists on the account on which the strategy is running. This is to provide updates on the current Unrealized Profit and Loss and associated risk values.

Method Return Value
This method does not return a value.

Syntax

You must override the method in your strategy with the following syntax:

```
protected override void OnAccountItemUpdate(Account account, AccountItem accountItem,
double value)
{

}
```

Method Parameters

account	The Account updated
accountItem	The AccountItem updated
value	The value of the AccountItem updated

Examples

```
protected override void OnAccountItemUpdate(Account account,
AccountItem accountItem, double value)
{
   Print(string.Format("{0} {1} {2}", account.Name, accountItem,
value));

   // output:
   // Sim101 BuyingPower 103962.5
   // Sim101 CashValue 103962.5
   // Sim101 GrossRealizedProfitLoss 3962.5
   // Sim101 RealizedProfitLoss 3962.5
}
```

11.6.14.28.1 AccountItemEventArgs

Definition

AccountItemEventArgs contains Account-related information to be passed as an argument to the OnAccountItemUpdate() event.

Note: For a complete, working example of this class in use, download framework example located on our Developing AddOns Overview

The properties listed below are accessible from an instance of AccountItemEventArgs:

Account	The Account for which OnAccountItemUpdate() was called
AccountItem	The AccountItem which has updated, resulting in the call to OnAccountItemUpdate()
Currency	The currency of the Account in question
Time	A DateTime object representing the time at which the change occurred
Value	The new value of the updated AccountItems

Example

```
// This method is fired on any change of an AccountItem
private void OnAccountItemUpdate(object sender,
AccountItemEventArgs e)
{
        /* Dispatcher.InvokeAsync() is needed for multi-threading
considerations. When processing events outside of the UI thread,
and we want to
        influence the UI .InvokeAsync() allows us to do so. It can
also help prevent the UI thread from locking up on long operations.
*/
        Dispatcher.InvokeAsync(() =>
        {
            //Print which AccountItem changed, on which account,
and the new value, using
            outputBox.AppendText(string.Format("{0}Account: {1}{0}
AccountItem: {2}{0}Value: {3}",
                Environment.NewLine,
                e.Account.Name,
                e.AccountItem,
                e.Value));
        });

}
```

11.6.14.25 OnExecutionUpdate()

Definition

An event driven method which is called on an incoming execution of an order managed by a strategy. An execution is another name for a fill of an order.

- An order can generate multiple executions (partial fills)
- OnExecutionUpdate is typically called after OnOrderUpdate() is called
- Only orders which have been submitted and managed by the strategy will call OnExecutionUpdate()
- Executions drive the strategy Position object, which is updated when this method is called

Notes:

- Programming in this environment is reserved for the more advanced user. If you are for example looking to protect a strategy managed position with a basic stop and target, then the Set() methods would be more convenient.
- When connected to the Playback connection, it is possible for OnExecutionUpdate() to trigger in the middle of a call to OnBarUpdate(). The Sim101 account adds a simulated random delay for processing execution events, but the Playback connection triggers executions immediately, for the sake of consistency in backtesting. Because of this, OnExecutionUpdate() can appear to be triggered earlier than it would in live trading, or when simulation trading on a live connection.
- Please also review Multi-Thread Considerations for NinjaScript
- Its best practice to only work with the passed by value parameters and not reference parameters. This insures that you process each change of the underlying state.
- Rithmic and Interactive Brokers Users: When using a NinjaScript strategy it is best practice to only work with passed by value data from OnExecutionUpdate(). Instances of multiple fills at the same time for the same instrument might result in an incorrect OnPositionUpdate, as sequence of events are not guaranteed due to provider API design.

Method Return Value

This method does not return a value.

Syntax

You must override the method in your strategy with the following syntax:

```
protected override void OnExecutionUpdate(Execution execution, string executionId,
double price, int quantity, MarketPosition marketPosition, string orderId, DateTime
time)
{

}
```

Parameters

execution	An Execution object passed by reference representing the execution
executionId	A `string` value representing the execution id
price	A `double` value representing the execution price
quantity	An `int` value representing the execution quantity
marketPosition	A MarketPosition object representing the position of the execution. Possible values are: • MarketPosition.Long • MarketPosition.Short
orderId	A `string` representing the order id
time	A DateTime value representing the time of the execution

OnExecutionUpdate Example (See SampleOnOrderUpdate for complete example)

```
private Order entryOrder = null; // This variable holds an object
representing our entry order
private Order stopOrder = null; // This variable holds an object
representing our stop loss order
private Order targetOrder = null; // This variable holds an object
representing our profit target order
private int sumFilled = 0; // This variable tracks the quantities
of each execution making up the entry order

protected override void OnExecutionUpdate(Execution execution,
string executionId, double price, int quantity, MarketPosition
marketPosition, string orderId, DateTime time)
{
    /* We advise monitoring OnExecutionUpdate() to trigger
submission of stop/target orders instead of OnOrderUpdate() since
OnExecution() is called after OnOrderUpdate()
       which ensures your strategy has received the execution which
is used for internal signal tracking. */
    if (entryOrder != null && entryOrder == execution.Order)
    {
        if (execution.Order.OrderState == OrderState.Filled ||
execution.Order.OrderState == OrderState.PartFilled ||
(execution.Order.OrderState == OrderState.Cancelled &&
execution.Order.Filled > 0))
        {
            // We sum the quantities of each execution making up
the entry order
            sumFilled += execution.Quantity;

            // Submit exit orders for partial fills
            if (execution.Order.OrderState ==
OrderState.PartFilled)
            {
                stopOrder = ExitLongStopMarket(0, true,
execution.Order.Filled, execution.Order.AverageFillPrice - 4 *
TickSize, "MyStop", "MyEntry");
                targetOrder = ExitLongLimit(0, true,
execution.Order.Filled, execution.Order.AverageFillPrice + 8 *
TickSize, "MyTarget", "MyEntry");
            }
            // Update our exit order quantities once orderstate
turns to filled and we have seen execution quantities match order
quantities
            else if (execution.Order.OrderState ==
OrderState.Filled && sumFilled == execution.Order.Filled)
            {
                // Stop-Loss order for OrderState.Filled
                stopOrder = ExitLongStopMarket(0, true,
execution.Order.Filled, execution.Order.AverageFillPrice - 4 *
TickSize, "MyStop", "MyEntry");
                targetOrder = ExitLongLimit(0, true,
execution.Order.Filled, execution.Order.AverageFillPrice + 8 *
TickSize, "MyTarget", "MyEntry");
            }

            // Resets the entryOrder object and the sumFilled
counter to null / 0 after the order has been filled
            if (execution.Order.OrderState != OrderState.PartFilled
&& sumFilled == execution.Order.Filled)
            {
                entryOrder = null;
                sumFilled = 0;
```

Using Execution information to calculate Average Entry Price (Rithmic/Interactive Brokers Friendly Approach)

```
private Order targetLong1   = null;
private Order stopLossLong1 = null;
private int sumFilledLong1 = 0; // This variable tracks the
quantities of each execution making up the entry order
private List<double> LongEntry1Prices; // This List is used to
track the fill prices of the entry order

protected override void OnExecutionUpdate(Execution execution,
string executionId, double price, int quantity, MarketPosition
marketPosition, string orderId, DateTime time)
{
    // Use execution.Name to identify the order, so we are not
using execution.Order, which may not be up to date if an
ExecutionUpdate is seen before an OrderUpdate in a partial fill
    if (execution.Name == "Long limit entry 1")
    {
        // We sum the quantities of each execution making up the
entry order
        sumFilledLong1 += execution.Quantity;

        if (LongEntry1Prices.IsNullOrEmpty())
            LongEntry1Prices = new List<double>();

        for (int i = 0; i < execution.Quantity; i++)
            LongEntry1Prices.Add(execution.Price);

        // Now we can calculate the average entry price, and use it
to protect the specifc entry
        double averageEntryPrice = 0;
        for (int i = 0; i < LongEntry1Prices.Count; i++)
            averageEntryPrice += LongEntry1Prices[i];
        averageEntryPrice /= LongEntry1Prices.Count;

        if (stopLossLong1 == null && targetLong1 == null)
        {
            // Directly assign order objects from the method's
return value. This prevents us from overprotecting the position by
making sure our code changes the orders, instead of submitting new
orders
            stopLossLong1 = ExitLongStopMarket(0, true,
sumFilledLong1, averageEntryPrice - StopDistance * TickSize,
"StopLossLong1", "Long limit entry 1");
            targetLong1 = ExitLongLimit(0, true, sumFilledLong1,
averageEntryPrice + ProfitDistance * TickSize, "TargetLong1", "Long
limit entry 1");
        }
        else
        {
            ChangeOrder(stopLossLong1, sumFilledLong1, 0,
averageEntryPrice - StopDistance * TickSize);
            ChangeOrder(targetLong1, sumFilledLong1,
averageEntryPrice + ProfitDistance * TickSize, 0);
```

Additional Reference Samples

Additional reference code samples are available the NinjaScript Educational Resources section of our support forum.

11.6.14.3 OnOrderTrace()

Definition

An event driven method used for strategies which will allow you to customize the output of TraceOrders.

> **Warning**: Overriding this method with disable the default order tracing that is generated by the NinjaTrader core. It is then up to you to pass the message generated to the NinjaTrader output window using the Print() method. Generally, overriding this method is not required.

Method Return Value

This method does not return a value.

Syntax

You must override the method in your strategy with the following syntax:

```
protected override void OnOrderTrace(DateTime timestamp, string message)
{

}
```

Method Parameters

`timestamp`	The time that the order trace was generated
`message`	The message that is generated

Examples

```
protected override void OnOrderTrace(DateTime timestamp, string
message)
{
    // The below print would give us the default tracing
    Print(string.Format("{0} {1}", timestamp, message));

    // The extended example would also include the instrument
fullname from our primary bars object
    if (BarsArray[0] != null)
        Print(string.Format("{0} {1} {2}", timestamp, message,
BarsArray[0].Instrument.FullName));
}
```

Additional Reference Samples

Additional reference code samples are available the NinjaScript Educational Resources section of our support forum.

11.6.14.3 OnOrderUpdate()

Definition

An event driven method which is called each time an order managed by a strategy changes state. An order will change state when a change in order quantity, price or state (working to filled) occurs. You can use this method to program your own order rejection handling.

Notes:

- Only orders which have been submitted and managed by the strategy will call OnOrderUpdate().
- Programming in this environment is reserved for the more advanced user. If you are for example looking to protect a strategy managed position with a basic stop and target, then the Set() methods would be more convenient.
- For triggering actions such as the submission of a stop loss order and target order using custom OCO logic when your entry order is filled, we recommend working directly in OnExecutionUpdate() instead.
- OnOrderUpdate() will run inside of order methods such as EnterLong() or SubmitOrderUnmanaged(), therefore attempting to assign an order object outside of OnOrderUpdate() may not return as soon as expected. If your strategy is dependent on tracking the order object from the very first update, you should try to match your order objects by the order.Name (signal name) from during the OnOrderUpdate() as the order is first updated.
- Rithmic and Interactive Brokers Users: When using a NinjaScript strategy it is best practice to only work with passed by value data from OnExecutionUpdate(). Instances of multiple fills at the same time for the same instrument might result in an incorrect OnPositionUpdate, as sequence of events are not guaranteed due to provider API

design. For an example on protecting positions with this approach, see OnExecutionUpdate()

Critical: If you want to drive your strategy logic based on order fills you must use OnExecutionUpdate() instead of OnOrderUpdate(). OnExecutionUpdate() is always triggered after OnOrderUpdate(). There is internal strategy logic that is triggered after OnOrderUpdate() is called but before OnExecutionUpdate() that can adversely affect your strategy if you are relying on tracking fills within OnOrderUpdate().

Playback Connection

When connected to the Playback Connection, calling market order based methods such as EnterLong() and EnterShort() will result in order state events being fired prior to the order method return an Order object. This is done to ensure that all events are in sync at high speed playback.

Method Return Value

This method does not return a value.

Syntax

You must override the method in your strategy with the following syntax:

```
protected override void OnOrderUpdate(Order order, double limitPrice, double
stopPrice, int quantity, int filled, double averageFillPrice, OrderState orderState,
DateTime time, ErrorCode error, string comment)
{

}
```

Method Parameters

order	An Order object passed by reference representing the order object
limitPrice	A `double` value representing the limit price of the order update
stopPrice	A `double` value representing the stop price of the order update
quantity	An `int` value representing the quantity of the order update

filled	An `int` value representing the filled amount of the order update
averageFillPrice	A `double` value representing the average fill price of the order update
orderState	An **OrderState** value representing the state of the order (e.g., filled, canceled, rejected, etc) **Note**: See order state values table below
time	A DateTime structure representing the last time the order changed state
error	An **ErrorCode** value which categorizes an error received from the broker Possible values are: ErrorCode.LoginExpired ErrorCode.LogOnFailed ErrorCode.NoError ErrorCode.OrderRejected ErrorCode.OrderRejectedByRisk ErrorCode.Panic ErrorCode.UnableToCancelOrder ErrorCode.UnableToChangeOrder ErrorCode.UnableToSubmitOrder ErrorCode.UserAbort
comment	A `string` representing the error message provided directly from the broker

OrderState Values

OrderState.Initialized	Order is initialized in NinjaTrader
OrderState.Submitted	Order is submitted to the broker

OrderState.Accepted	Order is accepted by the broker or exchange
OrderState.TriggerPending	Order is pending submission
OrderState.Working	Order is working in the exchange queue
OrderState.ChangePending	Order change is pending in NinjaTrader
OrderState.ChangeSubmitted	Order change is submitted to the broker
OrderState.CancelPending	Order cancellation is pending in NinjaTrader
OrderState.CancelSubmitted	Order cancellation is submitted to the broker
OrderState.Cancelled	Order cancellation confirm received from broker
OrderState.Rejected	Order is rejected
OrderState.PartFilled	Order is partially filled
OrderState.Filled	Order is completely filled
OrderState.Unknown	An unknown order state. Default if broker does not report current order state.

Examples

Understanding the order object parameter vs updating value parameter (Multi-Thread Considerations for NinjaScript)

```
protected override void OnOrderUpdate(Cbi.Order order, double
limitPrice, double stopPrice,
                                      int quantity, int filled,
double averageFillPrice,
                                      Cbi.OrderState orderState,
DateTime time, Cbi.ErrorCode error, string comment)
{
   Print("The most current order state is: " +
order.OrderState);   // OrderState.PartFilled
   Print("This particular order update state is: " +
orderState); // OrderState.Working
}
```

Properly assigning order object values

```
private Order entryOrder = null;

protected override void OnBarUpdate()
{
    if (entryOrder == null && Close[0] > Open[0])
        EnterLong("entryOrder");
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled, double
averageFillPrice, OrderState orderState, DateTime time, ErrorCode
error, string nativeError)
{
    // check if the current order matches the orderName passed in
"EnterLong"()
    // Assign entryOrder in OnOrderUpdate() to ensure the
assignment occurs when expected.
    // This is more reliable than assigning Order objects in
OnBarUpdate, as the assignment is not guaranteed to be complete if
it is referenced immediately after submitting
    if (order.Name == "entryOrder")
        entryOrder = order;

    // if entry order exists
    if (entryOrder != null && entryOrder == order)
    {
        Print(order.ToString());
        if (order.OrderState == OrderState.Cancelled)
        {
            // Do something here
            entryOrder = null;
        }
    }
}
```

Additional Reference Samples

Additional reference code samples are available the NinjaScript Educational Resources section of our support forum.

11.6.14.3 OnPositionUpdate()

Definition

An event driven method which is called each time a PositionUpdate is received for the strategy.

- This method is typically called after OnExecutionUpdate()

- OnPositionUpdate() will update with PositionUpdates that are filtered for the strategy. The strategy Position object is driven by Executions, and is updated as early as OnExecutionUpdate()

Notes:

- You will **NOT** receive position updates for manually placed orders, or orders managed by other strategies (including any ATM strategies) in OnPositionUpdate(). The Account class contains a pre-built event handler (PositionUpdate) which can be used to filter position updates on a specified account.
- Its best practice to only work with the passed by value parameters and not reference parameters. This insures that you process each change of the underlying state.
- Rithmic and Interactive Brokers Users: When using a NinjaScript strategy it is best practice to only work with passed by value data from OnExecution. Instances of multiple fills at the same time for the same instrument might result in an incorrect OnPositionUpdate, as sequence of events are not guaranteed due to provider API design. For an example on protecting positions with this approach, see OnExecutionUpdate()

Method Return Value

This method does not return a value.

Syntax

You must override the method in your strategy with the following syntax:

```
protected override void OnPositionUpdate(Position position, double averagePrice, int
quantity, MarketPosition marketPosition)
{

}
```

Method Parameters

position	A Position object passed by reference representing the current position object
averageFillPrice	A `double` value representing the updating average fill price of a position

quantity	An `int` value representing the updating quantity of a position
marketPosition	A MarketPosition object representing the updating position update provided directly from the broker. This is not the actual Position core position object, but the last change of the market position Possible values are: • MarketPosition.Flat • MarketPosition.Long • MarketPosition.Short

Examples

```
protected override void OnPositionUpdate(Cbi.Position position,
double averagePrice,
        int quantity, Cbi.MarketPosition marketPosition)
{
   if (position.MarketPosition == MarketPosition.Flat)
   {
      // Do something like reset some variables here
   }
}
```

Understanding the order object parameter vs updating value parameter (Multi-Thread Considerations for NinjaScript)

```
protected override void OnPositionUpdate(Cbi.Position position,
double averagePrice,
        int quantity, Cbi.MarketPosition marketPosition)
{
   Print("The most current MarketPosition is: " +
position.MarketPosition);   // Flat
   Print("This particular position update marketPosition is: " +
marketPosition); // Long
}
```

Additional Reference Samples

Additional reference code samples are available the NinjaScript Educational Resources section of our support forum.

11.6.14.33 OptimizationPeriod

Definition
Reserved for Walk-Forward Optimization, this property determines the number of days used for the "in sample" backtest period for a given strategy. See also TestPeriod.

> **Note**: This property should **ONLY** be called from the OnStateChange() method during State.SetDefaults

Property Value
An `int` value representing the number of "in sample" days used for walk-forward optimization; Default value is set to 10.

Syntax
`OptimizationPeriod`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          //set the default optimization period to 20 days for WFOs
          OptimizationPeriod = 20;
     }
}
```

11.6.14.34 Order

Definition
Represents a read only interface that exposes information regarding an order.

- An Order object returned from calling an order method is dynamic in that its properties will always reflect the current state of an order
- The property <Order>.OrderId is **NOT** a unique value, since it can change throughout an order's lifetime. Please see the Advance Order Handling section on "*Transitioning order references from historical to live"* for details on how to handle.
- The property <Order>.Oco **WILL** be appended with a suffix when the strategy transitions from historical to real-time to ensure the OCO id is unique across multiple strategies for live orders
- To check for equality you can compare Order objects directly

Methods and Properties

Account	The Account the order resides
AverageFillPrice	A `double` value representing the average fill price of an order
Filled	An `int` value representing the filled amount of an order
FromEntrySignal	A `string` representing the user defined fromEntrySignal parameter on an order
Gtd	A DateTime structure representing when the order will be canceled
HasOverfill	A `bool` value representing if the order is an overfill. For use when using Unmanaged orders and IgnoreOverFill
Instrument	An Instrument value representing the instrument of an order
IsBacktestOrder	A `bool` that indicates if the order was generated while processing historical data. For use with GetRealtimeOrder() when transitioning historical order objects to live order objects when strategies transition to from State.Historical to State.Realtime.
IsLiveUntilCancelled	A `bool` that when true, indicates the order will be canceled by managed order handling at expiration
IsTerminalState()	A static method used to determine if the an order's

	OrderState is in considered terminal and no longer active
LimitPrice	A `double` value representing the limit price of an order
LimitPriceChanged	A `double` value representing the new limit price of an order. Used with Account.Change()
Name	A `string` representing the name of an order which can be provided by the entry or exit signal name
Oco	A `string` representing the OCO (one cancels other) id of an order
OrderAction	Represents the action of the order. Possible values are: OrderAction.Buy OrderAction.BuyToCover OrderAction.Sell OrderAction.SellShort
OrderId	A `string` representing the broker issued order id value (this value can change)
OrderState	The current state of the order. See the order state values table below
OrderType	The type of order submitted. Possible values are: OrderType.Limit OrderType.Market OrderType.MIT OrderType.StopMarket OrderType.StopLimit

Quantity	An `int` value representing the quantity of an order
QuantityChanged	An `int` value representing the new quantity of an order. Used with Account.Change()
StopPrice	A `double` value representing the stop price of an order
StopPriceChanged	A `double` value representing the new stop price of an order. Used with Account.Change()
Time	A DateTime structure representing the last time the order changed state
TimeInForce	Determines the life of the order. Possible values are: TimeInForce.Day TimeInForce.Gtc
ToString()	A `string` representation of an order

OrderState Values

OrderState.Initialized	Order is initialized in NinjaTrader
OrderState.Submitted	Order is submitted to the broker
OrderState.Accepted	Order is accepted by the broker or exchange
OrderState.TriggerPending	Order is pending submission

OrderState.Working	Order is working in the exchange queue
OrderState.ChangePending	Order change is pending in NinjaTrader
OrderState.ChangeSubmitted	Order change is submitted to the broker
OrderState.CancelPending	Order cancellation is pending in NinjaTrader
OrderState.CancelSubmitted	Order cancellation is submitted to the broker
OrderState.Cancelled	Order cancellation is confirmed by the exchange
OrderState.Rejected	Order is rejected
OrderState.PartFilled	Order is partially filled
OrderState.Filled	Order is completely filled
OrderState.Unknown	An unknown order state. Default if broker does not report current order state.

Critical: In a historical backtest, orders will always reach a "Working" state. In real-time, some stop orders may only reach "Accepted" state if they are simulated/held on a brokers server

Examples

```
private Order entryOrder = null;

protected override void OnBarUpdate()
{
    if (entryOrder == null && Close[0] > Open[0])
        EnterLong("myEntryOrder");
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled, double
averageFillPrice, OrderState orderState, DateTime time, ErrorCode
error, string nativeError)
{
    // Assign entryOrder in OnOrderUpdate() to ensure the
assignment occurs when expected.
    // This is more reliable than assigning Order objects in
OnBarUpdate, as the assignment is not guaranteed to be complete if
it is referenced immediately        after submitting
    if (order.Name == "myEntryOrder")
        entryOrder = order;

    if (entryOrder != null && entryOrder == order)
    {
        Print(order.ToString());
        if (order.OrderState == OrderState.Filled)
            entryOrder = null;
    }
}
```

11.6.14.34.1 IsTerminalState()

Definition

A static method used to determine if the an order's **OrderState** is considered terminal and no longer active.

> **Note**: This is a static method and is compared against an order state, **NOT** the order itself. Please see the example below for correct syntax an usage.

Method Return Value

A `bool` value which will return **true** when an **OrderState** is equal to **OrderState.Cancelled**, **OrderState.Filled**, **OrderState.Rejected**, **OrderState.Unknown**; otherwise **false**.

Syntax

`IsTerminalState(OrderState orderState)`

Parameters

orderState	The **OrderState** to compare

Examples

```
private Order entryOrder = null;

protected override void OnBarUpdate()
{
   // submit order under valid condition
   // note that the order assignment and handling is done in
OnOrderUpdate()
   if (entryOrder == null && Close[0] > Open[0])
      EnterLongLimit(Close[0] - 1, "myEntryOrder");
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled, double
averageFillPrice, OrderState orderState, DateTime time, ErrorCode
error, string nativeError)
{
   // assign incoming order
   if (entryOrder == null)
   {
      // check that order matches by signal name, that order is not
in terminal state
      if (order.Name == "myEntryOrder" &&
!Order.IsTerminalState(order.OrderState))
         entryOrder = order;
   }

   if (entryOrder != null && entryOrder == order)
   {
      // set "entryOrder" to null if it is Cancelled, Filled,
Rejected, Unknown
      if (Order.IsTerminalState(entryOrder.OrderState))
         entryOrder = null;
   }
}
```

11.6.14.3 Order Methods

> **Note**: You will not be able to mix and match the two approaches. If you decide to go with the Managed approach you will only be able to use the Managed order methods. If you decide to go with the Unmanaged approach you will only be able to use the Unmanaged order methods.

Order Methods Overview

NinjaScript provides several approaches you can use for order placement within your NinjaScript strategy. The main approaches can be categorized as a Managed approach and an Unmanaged approach.

Managed

The Managed approach offers you order methods that are wrapped with an invisible convenience layer that allows you to focus on your system's trading rules leaving the underlying mechanics of order management and the relationships between entry and exit orders and positions to NinjaTrader. The cost for having the convenience layer is that there are order handling rules that must be followed to prevent order errors.

- Understanding the Managed approach
- Advanced Order Handling
- CancelOrder()
- EnterLong()
- EnterLongLimit()
- EnterLongMIT()
- EnterLongStopMarket()
- EnterLongStopLimit()
- EnterShort()
- EnterShortLimit()
- EnterShortMIT()
- EnterShortStopMarket()
- EnterShortStopLimit()
- ExitLong()
- ExitLongLimit()
- ExitLongMIT()
- ExitLongStopMarket()
- ExitLongStopLimit()
- ExitShort()
- ExitShortLimit()
- ExitShortMIT()
- ExitShortStopMarket()
- ExitShortStopLimit()
- GetRealtimeOrder()
- SetProfitTarget()
- SetStopLoss()

Unmanaged

The Unmanaged approach offers you more flexible order methods without the convenience layer. This means you are not restricted to any order handling rules besides those imposed by the brokerage/exchange. With such flexibility though, you will have to ensure to program your strategy to handle any and all issues that may arise with placing orders.

- Understanding the Unmanaged approach
- CancelOrder()
- ChangeOrder()
- GetRealtimeOrder()
- IgnoreOverfill
- IsUnmanaged
- SubmitOrderUnmanaged()

11.6.14.35.1 Managed Approach

The Managed approach in NinjaScript is designed to offer the greatest ease of use for beginner to intermediate programmers. The order methods are wrapped in a convenience layer that allows you to focus on your system's trading rules, leaving the underlying mechanics of order management and the relationships between entry orders, exit orders, and positions to NinjaTrader. This approach is best suited for simple to moderate order complexity, and can be further broken down into a Basic/Common Managed approach and a more Advanced Managed approach. The following section will discuss the use of the Basic/Common approach.

A few key points to keep in mind:

- Orders are submitted as live and working when a strategy is running in real-time
- Profit target, stop loss and trail stop orders are submitted immediately when an entry order is filled, and are tied together via OCO (One Cancels Other)
- Order changes and cancellations are queued in the event that the order is in a state where it can't be cancelled or modified
- By default, orders submitted via Entry() and Exit() methods automatically cancel at the end of a bar if not re-submitted
- Entry() methods will reverse the position automatically. For example if you are in a 1 contract long position and now call EnterShort() -> you will see 2 executions, one to close the prior long position and the other to get you into the desired 1 contract short position.

* Via the SetProfitTarget(), SetStopLoss(), SetTrailStop() and SetParabolicStop methods

Order submission for entry and exit methods - basic operation

Orders are primarily submitted from within the OnBarUpdate() method when a specific order method is called. By default, orders are kept alive, provided they are re-submitted on each call of the OnBarUpdate() method. If an order is not re-submitted, it is then canceled. Orders can be modified by re-submitting them with changed parameters (a new limit price, for example).

In the example below, a Buy Limit order is working at the bid price, provided that the Close price of the current bar is greater than the current value of the 20 period Simple Moving Average. If the entry condition is no longer true and the order is still active, it will be immediately canceled.

```
protected override void OnBarUpdate()
{
     // Entry condition
     if (Close[0] > SMA(20)[0])
          EnterLongLimit(GetCurrentBid());
}
```

This technique allows you the quickest and easiest order submission method suitable for programmers of all levels. Should you want to submit an order and not have to keep re-submitting it to keep it alive you can use an advanced approach reserved for experienced programmers, which includes an option to keep orders alive until specifically canceled in code.

Order Entry Methods

Order Entry Methods

Order entry methods are used to submit orders of different types. Methods exist to submit Market, Market-if-Touched, Limit, Stop Market, and Stop Limit orders. See the order-entry method pages listed in the help guide table of contents under this page for more information on a specific method.

Signal Names on Entry Methods

You can optionally tag an entry order with a signal name. Signal names are used to identify executions resulting from the order on a chart and in performance reports. Market positions created from a tagged entry method are marked with the signal name which serves two purposes:

- Used to tie an exit method to a specific position
- Used to identify unique entries in a strategy

Below is an example of placing an Market entry order and an associated Limit exit order, tied together by the signal name of the entry order.

```
protected override void OnBarUpdate()
{
    if (CurrentBar <  ) return;

    if (Close[0] > Close[1])
    {
        // Place a Market order to enter long
        EnterLong("longEntry");

        // Manually place a Profit Target 10 ticks above
the current price, tied to the entry order's SignalName
        ExitLongLimit(Close[0] + (10 * TickSize),
"longEntry");
    }
}
```

Defining how Entry Methods are Processed in a Strategy

You can limit how many entry methods are processed by determining the maximum number of entries in a single direction across all entry methods, or across unique signal names. The following properties can be set in the Strategies window when adding a strategy to a chart or to the Strategies tab of the Control Center window.

- EntriesPerDirection property - Sets the maximum number of entries in a single direction
- EntryHandling property - Determines if EntriesPerDirection applies across all entries or for entries with specified signal names

The example code below illustrates how the above properties control the processing of entry methods. The code contains two entry conditions and two EnterLong methods, each tagged with unique signal names.

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        EntriesPerDirection = 1;
        EntryHandling = EntryHandling.AllEntries;
    }
}

protected override void OnBarUpdate()
{
    // Entry condition 1
    if (CrossAbove(SMA(10), SMA(20), 1))
        EnterLong("Condition 1 Entry");

    // Entry condition 2
    if (CrossAbove(RSI(14, 3), 30, 1))
        EnterLong("Condition 2 Entry");
}
```

Entry Methods on Multi-Instrument Strategies

When running strategies that submit orders to multiple instruments, entry methods will submit orders to the instrument referenced by the BarsInProgress. The following example assumes that the strategy is running on a 1 minute E-Mini S&P 500 chart. It adds an NQ data series, then enters a position on both instruments.

```
protected override void OnStateChange()
{
     AddDataSeries("NQ 09-14", BarsPeriodType.Minute, 1);
}

protected override void OnBarUpdate()
{
     if (BarsInProgress == 0)
          EnterLong("ES Trade");
     else if (BarsInProgress == 1)
          EnterLong("NQ Trade");
}
```

More information on using `BarsInProgress` to filter instruments can be found in the Advanced Order Handling page.

Quantity Type and TIF

You can set the entry order quantity and order type directly in code via the following properties:

- QuantityType - Sets the order quantity is taken from the entry method quantity property or the default strategy quantity size
- TimeInForce propery - Sets the time in force of the order

How to close a position

Closing a Position using a Stop Loss, Trailing Stop and/or Profit Target

You can predefine a stop loss, trailing stop and/or profit target in a strategy by calling the SetStopLoss(), SetTrailStop(), SetParabolicStop() or SetProfitTarget() methods from inside the OnStateChange() event handler. When these methods are called, they submit live working orders in real-time as executions are reported as a result of calling an entry method. These orders are also tied via OCO (One Cancels Other).

Stop losses and profit target can be generated for each fill or each position. This is determined by the "Stop & target submission" property which is set in the Strategies window. Possible values are listed below:

ByStrategyPosition - When this is selected, only one stop loss, trail stop and/or profit target order is submitted. As entry executions come in, the order size is amended. The downside of this approach is that if you receive partial fills, the orders are re-inserted into the exchange order queue. The upside is that if you broker charges you commission per order (not per quantity), you will not incur additional commission expenses.

PerEntryExecution - When this is selected, a stop loss, trail stop and/or profit target order is submitted for each partial fill received. The downside is that if your broker charges commission per order, you can incur very expensive commission costs if you receive partial fills. The upside is that orders are submitted as soon as possible, giving you the advantage of getting into the order queue immediately.

Closing a Position using an Exit Method

Exit methods submit orders to close out a position in whole or in part. When closing a position with Exit orders, the order quantity will be reduced as the strategy position reduces - for example, if we use ExitLongStopMarket() and

ExitLongStopLimit() to protect a position and one of those orders gets filled, the other order associated with exiting that position will reduce their quantity.

As with entry methods, more information about specific exit methods can be found in this Help Guide's table of contents, beneath this page.

Closing a Partial Position using an Exit Method

You can close out a partial position by specifying the exit quantity. The following example first enters long for three contracts. Then, each subsequent bar update submits a market order to exit one contract until the position is completely closed. "ExitLong(1)" will be ignored if a long market position does not exist.

```
protected override void OnBarUpdate()
{
     if (CrossAbove(SMA(10), SMA(20), 1))
          EnterLong(3);

     ExitLong(1);
}
```

FromEntrySignal -- Using Signal Names in Exit Methods

Identifying entries with a signal name allows you to place multiple unique entries within a single strategy and call exit methods with specified signal names, so that only a position created with the specified signal name is closed. In the example below, there are two entry conditions which create positions, and two exit conditions specifying which position to close based on the signal name.

```
protected override void OnBarUpdate()
{
     // Entry condition 1
     if (CrossAbove(SMA(10), SMA(20), 1))
          EnterLong("Condition 1 Entry");

     // Entry condition 2
     if (CrossAbove(RSI(14, 3), 30, 1))
          EnterLong("Condition 2 Entry");

     // Closes the position created by entry condition 1
     if (CrossBelow(SMA(10), SMA(20), 1))
          ExitLong("Condition 1 Entry");

     // Closes the position created by entry condition 2
     if (CrossBelow(RSI(14, 3), 70, 1))
          ExitLong("Condition 2 Entry");
}
```

Tip: If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat.

```
protected override void OnBarUpdate()
{

   if (Position.MarketPosition == MarketPosition.Flat)
   {
      // Entry condition 1
      if (CrossAbove(SMA(10), SMA(20), 1))
         EnterLong("Condition 1 Entry");
   }

   if (Position.MarketPosition != MarketPosition.Flat)
   {
      // Scale in condition 2 for position management
      if (CrossAbove(RSI(14, 3), 30, 1))
         EnterLong("Condition 2 Entry");

      // Exit all positions using an empty string (could
also use string.Empty)
      if (CrossBelow(SMA(10), SMA(20), 1))
         ExitLong("Exit All", "");

   }
}
```

Understanding core order objects

When using order methods such as EnterLong(), ExitShortLimit(), etc, a direct order object is returned for the NinjaTrader Core. These objects can be used throughout the lifetime of your strategy to provide additional metadata concerning your strategy, as well as apply advanced concepts such as CancelOrder() and ChangeOrder(). More information about this advanced concept which is discussed under the Advanced Order Handling section

Internal Order Handling Rules that Reduce Unwanted Positions

To prevent situations in real-time in which you may have multiple orders working to accomplish the same task, there are some "under the hood" rules that a NinjaScript strategy follows when Managed order methods are called. For example, if your strategy had a limit order for 1 contract working as a Profit Target, but then your strategy was also programmed to reverse the position at the price very close to the target limit order, then submitting both orders can be risky, since

it could lead to a larger position than the strategy is designed to enter if both orders got filled in quick succession by the exchange.

> **Note**: These rules do not apply to market orders, such as ExitLong() or ExitShort().

For the most part, you do not need to be intimately familiar with these rules as you develop your strategies. It is all taken care of for you internally within a strategy. If a rule is violated, you will be notified through an error log in the Control Center Log tab.

> **Note**: To prevent excessive logging which could degrade performance, you will only be notified of the very first order which has violated an order handling rule. Subsequent orders which violate a rule will not be notified through the error log.

The following rules are true *per unique signal name*:

Methods that generate orders to **enter** a position will be ignored if:

- A position is open and an order submitted by a non market order exit method (ExitLongLimit() for example) is active and the order is used to open a position in the opposite direction
- A position is open and an order submitted by a set method (SetStopLoss() for example) is active and the order is used to open a position in the opposite direction
- A position is open and two or more Entry methods to reverse the position are entered together. In this case the second Entry order will be ignored.
- The strategy position is flat and an order submitted by an enter method (EnterLongLimit() for example) is active and the order is used to open a position in the opposite direction
- The entry signal name is not unique

Methods that generate orders to **exit** a position will be ignored if:

- A position is open and an order submitted by an enter method (EnterLongLimit() for example) is active and the order is used to open a position in the opposite direction
- A position is open and an order submitted by a set method (SetStopLoss() for example) is active

Set() methods that generate orders to **exit** a position will be ignored if:

- A position is open and an order submitted by an enter method (EnterLongLimit() for example) is active and the order is used to open a position in the opposite direction
- A position is open and an order submitted by a non market order exit method (ExitLongLimit() for example) is active

Advanced Order Handling	Through advanced order handling you can submit, change and cancel orders at your discretion through any event-driven method within a strategy.
CancelOrder()	Cancels a specified order.
ChangeOrder()	Amends a specified Order.
EnterLong()	Generates a buy market order to enter a long position.
EnterLongLimit()	Generates a buy limit order to enter a long position.
EnterLongMIT()	Generates a buy MIT order to enter a long position.
EnterLongStopLimit()	Generates a buy stop limit order to enter a long position.
EnterLongStopMarket()	Generates a buy stop market order to enter a long position.
EnterShort()	Generates a sell short market order to enter a short position.

EnterShortLimit()	Generates a sell short stop limit order to enter a short position.
EnterShortMIT()	Generates a sell MIT order to enter a short position.
EnterShortStopLimit()	Generates a sell short stop limit order to enter a short position.
EnterShortStopMarket()	Generates a sell short stop order to enter a short position.
ExitLong()	Generates a sell market order to exit a long position.
ExitLongLimit()	Generates a sell limit order to exit a long position.
ExitLongMIT()	Generates a sell MIT order to exit a long position.
ExitLongStopLimit()	Generates a sell stop limit order to exit a long position.
ExitLongStopMarket()	Generates a sell stop market order to exit a long position.
ExitShort()	Generates a buy to cover market order to exit a short position.
ExitShortLimit()	Generates a buy to cover limit order to exit a short position.
ExitShortMIT()	Generates a buy to cover MIT order to exit a short position.
ExitShortStopLimit	Generates a buy to cover stop limit order to exit a short position.

()	
ExitShortStopMarket()	Generates a buy to cover stop market order to exit a short position.
GetRealtimeOrder()	Returns a matching real-time order object based on a specified historical order object reference.
SetParabolicStop()	Generates a parabolic type trail stop order with the signal name "Parabolic stop" to exit a position.
SetProfitTarget()	Generates a profit target order with the signal name "Profit target" to exit a position.
SetStopLoss()	Generates a stop loss order with the signal name "Stop loss" used to exit a position.
SetTrailStop()	Generates a trail stop order with the signal name "Trail stop" to exit a position.

11.6.14.35.1.1 Advanced Order Handling

Advanced order handling is reserved for **EXPERIENCED** programmers. Through advanced order handling you can submit, change and cancel orders at your discretion through any event-driven method within a strategy. Each order method within the "Managed Approach" section has a method overload designed for advanced handling.

Live Until Cancelled Orders

Orders can remain live until you call the CancelOrder() method, or until the order's time in force has expired, whichever comes first. This flexibility allows you to control exactly when an order should be cancelled instead of relying on the close of a bar. Each order method, such as EnterLongLimit(), has a method overload designed to submit a "live until canceled" order. When using this overload, it is important to retain a reference to the Order object, so that it can be canceled via CancelOrder() at a later time.

The Order Class

All order methods return an Order object. There are several important items to note:

- An Order object returned from calling an order method contains dynamic properties which will always reflect the current state of the associated order
- The property <Order>.OrderId is **NOT** a unique value, since it can change throughout an order's lifetime. Please see the section below on "*Transitioning order references from historical to live"* for details on how to handle.
- To check for equality, you can compare Order objects directly

The following example code demonstrates the submission of an order and the assignment of the Order return object to the variable "entryOrder." After this, the object is checked in the OnOrderUpdate() method for equality, and then checked for the `Filled` state.

Examples

```
private Order entryOrder = null;
protected override void OnBarUpdate()
{
     if (entryOrder == null && Close[0] > Open[0])
          EnterLong("myEntryOrder");
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled,
double averageFillPrice, OrderState orderState, DateTime
time, ErrorCode error, string nativeError)
{
     // Assign entryOrder in OnOrderUpdate() to ensure the
assignment occurs when expected.
     // This is more reliable than assigning Order objects
in OnBarUpdate, as the assignment is not gauranteed to be
complete if it is referenced immediately after submitting
     if (order.Name == "myEntryOrder" && orderState !=
OrderState.Filled)
         entryOrder = order;

    // Null Entry order if filled or cancelled. We do not
use the Order objects after the order is filled, so we can
null it here
    if (entryOrder != null && entryOrder == order)
    {
        if (order.OrderState == OrderState.Cancelled &&
order.Filled == 0)
            entryOrder = null;
        if (order.OrderState == OrderState.Filled)
            entryOrder = null;
    }
}
```

Transitioning order references from historical to live

When starting a strategy on real-time data, the starting behavior will renew any active historical orders and resubmit these orders to your live or simulation account. This process includes updating the historical/backtest generated order ID to the account generated order ID, and any associated OCO IDs. If you are tracking order objects, is critical that you update the order reference to ensure that it is now using the correct order details.

This should be done in OnOrderUpdate() to ensure all cases of order transitions are handled.

Critical: If you **DO NOT** update a historical order reference, and then attempt to cancel/change that order *after* it has been submitted in real-time, your strategy will be disabled with a message similar to: *"Strategy has been disabled because it attempted to modify a historical order that has transitioned to a live order."*

Tip: When the real-time order is submitted, there is a generic Order object passed into the OnOrderUpdate() method containing the live order details which can be used for debugging. It is recommended you use the helper GetRealtimeOrder() when your strategy transitions to real-time to update your order references

Example

```
private Order entryOrder = null;

protected override void OnBarUpdate()
{
   if (entryOrder == null && Close[0] > Open[0])
      entryOrder = EnterLongLimit("myEntryOrder", Low[0]);
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled,
double averageFillPrice, OrderState orderState, DateTime
time, ErrorCode error, string nativeError)
{
    // One time only, as we transition from historical
    // Convert any old historical order object references
to the live order submitted to the real-time account
    if (entryOrder != null && entryOrder.IsBacktestOrder &&
State == State.Realtime)
        entryOrder = GetRealtimeOrder(entryOrder);

    // Null entryOrder if filled or cancelled. We do not
use the Order objects after the order is filled, so we can
null it here
    if (entryOrder != null && entryOrder == order)
    {
        if (order.OrderState == OrderState.Cancelled &&
order.Filled == 0)
            entryOrder = null;
        if (order.OrderState == OrderState.Filled)
            entryOrder = null;
    }
}
```

Working with a Multi-Instrument Strategy

With advanced order handling, you can submit an order in the context of any Bars object by designating the "BarsInProgress" index. For example, if your primary bar series is "MSFT" and your secondary series added to the strategy through the AddDataSeries() method is "AAPL", you can submit an order for either "MSFT" or "AAPL" from anywhere within the strategy. In addition to the information found in the multi-time frame and instrument strategies page, this section specifically covers order submission.

As an example, consider the EnterLongLimit() method and one of its method overloads designed for advanced order handling:

```
EnterLongLimit(int barsInProgressIndex, bool isLiveUntilCancelled, int
quantity, double limitPrice, string signalName)
```

In this example, an "MSFT 1 minute" chart is the primary bar series on which the strategy is running. A secondary bar series is added for "AAPL 1 minute" via the AddDataSeries() method in the OnStateChange() event handler. After adding the secondary `Bars` object, MSFT has a BarsInProgress index of 0 and AAPL has an index value of 1.

The following example code demonstrates how to monitor for bar update events on the first instrument, while submitting orders to the second instrument.

Example

```
private Order entryOrder = null;

protected override void OnStateChange()
{
    if (State == State.Configure)
    {
        AddDataSeries("AAPL", BarsPeriodType.Minute, 1);
    }
}

protected override void OnBarUpdate()
{
     // Check if the MSFT series triggered an bar update
event
     if (BarsInProgress == 0)
     {
          // Submit an order for AAPL in the context of
MSFT bar update event
          if (entryOrder == null)
               EnterLongLimit(1, true, 1, Lows[1][0], "AAPL
Order");
     }
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled,
double averageFillPrice, OrderState orderState, DateTime
time, ErrorCode error, string nativeError)
{
     // Assign entryOrder in OnOrderUpdate() to ensure the
assignment occurs when expected.
     // This is more reliable than assigning Order objects
in OnBarUpdate, as the assignment is not gauranteed to be
complete if it is referenced immediately after submitting
     if (order.Name == "AAPL Order" && orderState !=
OrderState.Filled)
        entryOrder = order;
}
```

11.6.14.35.1.2 CancelOrder()

Definition

Cancels a specified order. This method is reserved for experienced programmers that fully understanding the concepts of advanced order handling.

Notes:
1. This method sends a cancel request to the broker and does not guarantee that an order is completely cancelled. Most of the time you can expect your order to come back 100% cancelled.
2. An order can be completely filled or part filled in the time that you send the cancel request and the time the exchange receives the request. Check the OnOrderUpdate() method for the state of an order you attempted to cancelled.

Syntax

```
CancelOrder(Order order)
```

Warning: If you have existing **historical order references** which have transitioned to real-time, you **MUST** update the **order object reference** to the newly submitted **real-time** order; otherwise errors may occur as you attempt to cancel the order. You may use the GetRealtimeOrder() helper method to assist in this transition.

Parameters

order	An Order object representing the order you wish to cancel.

Examples

```
private Order myEntryOrder = null;
private int barNumberOfOrder = 0;

protected override void OnBarUpdate()
{
    // Submit an entry order at the low of a bar
    if (myEntryOrder == null)
    {
        // use 'live until canceled' limit order to prevent default
managed order handling which would expire at end of bar
        EnterLongLimit(0, true, 1, Low[0], "Long Entry");
        barNumberOfOrder = CurrentBar;
    }

    // If more than 5 bars has elapsed, cancel the entry order
    if (CurrentBar > barNumberOfOrder + 5)
        CancelOrder(myEntryOrder);
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled,
    double averageFillPrice, OrderState orderState, DateTime time,
ErrorCode error, string nativeError)
{
    // Assign entryOrder in OnOrderUpdate() to ensure the
assignment occurs when expected.
    // This is more reliable than assigning Order objects in
OnBarUpdate, as the assignment is not gauranteed to be complete if
it is referenced immediately after submitting
    if (order.Name == "Long Entry")
        myEntryOrder = order;

    // Evaluates for all updates to myEntryOrder.
    if (myEntryOrder != null && myEntryOrder == order)
    {
        // Check if myEntryOrder is cancelled.
        if (myEntryOrder.OrderState == OrderState.Cancelled)
        {
            // Reset myEntryOrder back to null
            myEntryOrder = null;
        }
    }
}
```

11.6.14.35.1.3 ChangeOrder()

Definition

Amends a specified Order.

Note: This method is only relevant for Managed orders with IsLiveUntilCancelled set to true and Unmanaged orders.

Syntax

```
ChangeOrder(Order order, int quantity, double limitPrice, double stopPrice)
```

Warning: If you have existing **historical order references** which have transitioned to real-time, you **MUST** update the **order object reference** to the newly submitted **real-time order;** otherwise errors may occur as you attempt to change the order. You may use the GetRealtimeOrder() helper method to assist in this transition.

Parameters

order	Order object of the order you wish to amend
quantity	Order quantity
limitPrice	Order limit price. Use "0" should this parameter be irrelevant for the OrderType being submitted.
stopPrice	Order stop price. Use "0" should this parameter be irrelevant for the OrderType being submitted.

Examples

```
private Order stopOrder = null;

protected override void OnBarUpdate()
{
     // Raise stop loss to breakeven when you are at least 4 ticks
in profit
     if (stopOrder != null && stopOrder.StopPrice <
Position.AveragePrice && Close[0] >= Position.AveragePrice + 4 *
TickSize)
          ChangeOrder(stopOrder, stopOrder.Quantity, 0,
Position.AveragePrice);
}
```

11.6.14.35.1.4 EnterLong()

Definition

Generates a buy market order to enter a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterLong()
EnterLong(string signalName)
EnterLong(int quantity)
EnterLong(int quantity, string signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterLong(int barsInProgressIndex, int quantity, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100)
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterLong(5, "SMA Cross Entry");
}
```

11.6.14.35.1.5 EnterLongLimit()

Definition

Generates a buy limit order to enter a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterLongLimit(double limitPrice)
EnterLongLimit(double limitPrice, string signalName)
EnterLongLimit(int quantity, double limitPrice)
EnterLongLimit(int quantity, double limitPrice, string signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterLongLimit(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double limitPrice, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.

limitPrice	The limit price of the order.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100).
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar, but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterLongLimit(GetCurrentBid(), "SMA Cross Entry");
}
```

11.6.14.35.1.6 EnterLongMIT()

Definition

Generates a buy MIT order to enter a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterLongMIT(double stopPrice)
EnterLongMIT(double stopPrice, string signalName)
EnterLongMIT(int quantity, double stopPrice)
EnterLongMIT(int quantity, double stopPrice, string signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterLongMIT(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity, double stopPrice, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
stopPrice	The stop price of the order.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100).
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar, but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterLongMIT(GetCurrentBid() + TickSize, "SMA Cross
Entry");
}
```

11.6.14.35.1.7 EnterLongStopLimit()

Definition

Generates a buy stop limit order to enter a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterLongStopLimit(double limitPrice, double stopPrice)
EnterLongStopLimit(double limitPrice, double stopPrice, string signalName)
EnterLongStopLimit(int quantity, double limitPrice, double stopPrice)
EnterLongStopLimit(int quantity, double limitPrice, double stopPrice, string
signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterLongStopLimit(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double limitPrice, double stopPrice, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
limitPrice	The limit price of the order.
stopPrice	The stop price of the order.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100).
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar, but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterLongStopLimit(High[0] + 2 * TickSize, High[0], "SMA
Cross Entry");
}
```

11.6.14.35.1.8 EnterLongStopMarket()

Definition

Generates a buy stop market order to enter a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterLongStopMarket(double stopPrice)
EnterLongStopMarket(double stopPrice, string signalName)
EnterLongStopMarket(int quantity, double stopPrice)
EnterLongStopMarket(int quantity, double stopPrice, string signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterLongStopMarket(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double stopPrice, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
stopPrice	The stop price of the order.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100).
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar, but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterLongStopMarket(GetCurrentAsk() + TickSize, "SMA
Cross Entry");
}
```

11.6.14.35.1.9 EnterShort()

Definition

Generates a sell short market order to enter a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterShort()
EnterShort(string signalName)
EnterShort(int quantity)
EnterShort(int quantity, string signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterShort(int barsInProgressIndex, int quantity, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100).
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterShort("SMA Cross Entry");
}
```

11.6.14.35.1.10 EnterShortLimit()

Definition

Generates a sell short limit order to enter a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterShortLimit(double limitPrice)
EnterShortLimit(double limitPrice, string signalName)
EnterShortLimit(int quantity, double limitPrice)
EnterShortLimit(int quantity, double limitPrice, string signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterShortLimit(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double limitPrice, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
limitPrice	The limit price of the order.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100).
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar, but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterShortLimit(GetCurrentAsk(), "SMA Cross Entry");
}
```

11.6.14.35.1.11 EnterShortMIT()

Definition

Generates a sell MIT order to enter a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterShortMIT(double stopPrice)
EnterShortMIT(double stopPrice, string signalName)
EnterShortMIT(int quantity, double stopPrice)
EnterShortMIT(int quantity, double stopPrice, string signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterShortMIT(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity, double
 stopPrice, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.

stopPrice	The stop price of the order.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100).
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar, but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterShortMIT(GetCurrentAsk() + TickSize, "SMA Cross
Entry");
}
```

11.6.14.35.1.12 EnterShortStopLimit()

Definition

Generates a sell short stop limit order to enter a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterShortStopLimit(double limitPrice, double stopPrice)
EnterShortStopLimit(double limitPrice, double stopPrice, string signalName)
EnterShortStopLimit(int quantity, double limitPrice, double stopPrice)
EnterShortStopLimit(int quantity, double limitPrice, double stopPrice, string
signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterShortStopLimit(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double limitPrice, double stopPrice, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
limitPrice	The limit price of the order.
stopPrice	The stop price of the order.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100).
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar, but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterShortStopLimit(Low[0] - 2 * TickSize, Low[0], "SMA
Cross Entry");
}
```

11.6.14.35.1.13 EnterShortStopMarket()

Definition

Generates a sell short stop order to enter a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
EnterShortStopMarket(double stopPrice)
EnterShortStopMarket(double stopPrice, string signalName)
EnterShortStopMarket(int quantity, double stopPrice)
EnterShortStopMarket(int quantity, double stopPrice, string signalName)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
EnterShortStopMarket(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double stopPrice, string signalName)
```

Note: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy

Parameters

signalName	User defined signal name identifying the order

	generated. Max 50 characters.
stopPrice	The stop price of the order.
quantity	Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100).
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar, but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterShortStopMarket(GetCurrentBid() - TickSize, "SMA
Cross Entry");
}
```

11.6.14.35.1.14 ExitLong()

Definition

Generates a sell market order to exit a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
ExitLong()
ExitLong(int quantity)
ExitLong(string fromEntrySignal)
ExitLong(string signalName, string fromEntrySignal)
ExitLong(int quantity, string signalName, string fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitLong(int barsInProgressIndex, int quantity, string signalName, string
fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
quantity	Entry order quantity.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterLong("SMA Cross Entry");

     // Exits position
     if (CrossBelow(SMA(10), SMA(20), 1))
          ExitLong();
}
```

Tips (also see Overview):

- This method is ignored if a long position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.15 ExitLongLimit()

Definition

Generates a sell limit order to exit a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
ExitLongLimit(double limitPrice)
ExitLongLimit(int quantity, double limitPrice)
ExitLongLimit(double limitPrice, string fromEntrySignal)
ExitLongLimit(double limitPrice, string signalName, string fromEntrySignal)
ExitLongLimit(int quantity, double limitPrice, string signalName, string
fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitLongLimit(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity, double
 limitPrice, string signalName, string fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
limitPrice	The limit price of the order.
quantity	Entry order quantity.
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
          EnterLong("SMA Cross Entry");

     // Exits position
     if (CrossBelow(SMA(10), SMA(20), 1))
          ExitLongLimit(GetCurrentBid());
}
```

Tips (also see Overview):

- This method is ignored if a long position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.16 ExitLongMIT()

Definition

Generates a sell MIT order to exit a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
ExitLongMIT(double stopPrice)
ExitLongMIT(int quantity, double stopPrice)
ExitLongMIT(double stopPrice, string fromEntrySignal)
ExitLongMIT(double stopPrice, string signalName, string fromEntrySignal)
ExitLongMIT(int quantity, double stopPrice, string signalName, string fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitLongMIT(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity, double stopPrice, string signalName, string fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
stopPrice	The stop price of the order.
quantity	Entry order quantity.
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
private double stopPrice = 0;

protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
     {
          EnterLong("SMA Cross Entry");
          stopPrice = High[0];
     }

     // Exits position
     ExitLongMIT(stopPrice);
}
```

Tips (also see Overview):

- This method is ignored if a long position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.17 ExitLongStopLimit()

Definition

Generates a sell stop limit order to exit a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

`ExitLongStopLimit(double limitPrice, double stopPrice)`

```
ExitLongStopLimit(int quantity, double limitPrice, double stopPrice)
ExitLongStopLimit(double limitPrice, double stopPrice, string fromEntrySignal)
ExitLongStopLimit(double limitPrice, double stopPrice, string signalName, string
fromEntrySignal)
ExitLongStopLimit(int quantity, double limitPrice, double stopPrice, string
signalName, string fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitLongStopLimit(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double limitPrice, double stopPrice, string signalName, string fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
limitPrice	The limit price of the order
stopPrice	The stop price of the order.
quantity	Entry order quantity.
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
private double stopPrice = 0;

protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
     {
          EnterLong("SMA Cross Entry");
          stopPrice = Low[0] - 5 * TickSize;
     }

     // Exits position
     ExitLongStopLimit(stopPrice - (10 * TickSize), stopPrice);
}
```

Tips (also see Overview):

- This method is ignored if a long position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.18 ExitLongStopMarket()

Definition

Generates a sell stop market order to exit a long position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

`ExitLongStopMarket(double stopPrice)`

```
ExitLongStopMarket(int quantity, double stopPrice)
ExitLongStopMarket(double stopPrice, string fromEntrySignal)
ExitLongStopMarket(double stopPrice, string signalName, string fromEntrySignal)
ExitLongStopMarket(int quantity, double stopPrice, string signalName, string
fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitLongStopMarket(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double stopPrice, string signalName, string fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
stopPrice	The stop price of the order.
quantity	Entry order quantity.
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
private double stopPrice = 0;

protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossAbove(SMA(10), SMA(20),
1))
     {
          EnterLong("SMA Cross Entry");
          stopPrice = Low[0];
     }

     // Exits position
     ExitLongStopMarket(stopPrice);
}
```

Tips (also see Overview):

- This method is ignored if a long position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.19 ExitShort()

Definition

Generates a buy to cover market order to exit a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

`ExitShort()`

```
ExitShort(int quantity)
ExitShort(string fromEntrySignal)
ExitShort(string signalName, string fromEntrySignal)
ExitShort(int quantity, string signalName, string fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitShort(int barsInProgressIndex, int quantity, string signalName, string
fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
quantity	Entry order quantity.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossBelow(SMA(10), SMA(20),
1))
          EnterShort("SMA Cross Entry");

     // Exits position
     if (CrossBelow(SMA(10), SMA(20), 1))
          ExitShort();
}
```

Tips (also see Overview):

- This method is ignored if a short position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.20 ExitShortLimit()

Definition

Generates a buy to cover limit order to exit a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
ExitShortLimit(double limitPrice)
ExitShortLimit(int quantity, double limitPrice)
ExitShortLimit(double limitPrice, string fromEntrySignal)
ExitShortLimit(double limitPrice, string signalName, string fromEntrySignal)
```

```
ExitShortLimit(int quantity, double limitPrice, string signalName, string
fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitShortLimit(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double limitPrice, string signalName, string fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
limitPrice	The limit price of the order.
quantity	Entry order quantity.
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossBelow(SMA(10), SMA(20),
1))
          EnterShort("SMA Cross Entry");

     // Exits position
     if (CrossAbove(SMA(10), SMA(20), 1))
          ExitShortLimit(GetCurrentAsk());
}
```

Tips (also see Overview):

- This method is ignored if a short position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.21 ExitShortMIT()

Definition

Generates a buy to cover MIT order to exit a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

```
ExitShortMIT(double stopPrice)
ExitShortMIT(int quantity, double stopPrice)
ExitShortMIT(double stopPrice, string fromEntrySignal)
ExitShortMIT(double stopPrice, string signalName, string fromEntrySignal)
```

```
ExitShortMIT(int quantity, double stopPrice, string signalName, string
fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitShortMIT(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity, double
stopPrice, string signalName, string fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
stopPrice	The stop price of the order.
quantity	Entry order quantity.
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
private double stopPrice = 0;

protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossBelow(SMA(10), SMA(20),
1))
     {
          EnterShort("SMA Cross Entry");
          stopPrice = Low[0];
     }

     // Exits position
     ExitShortMIT(stopPrice);
}
```

Tips (also see Overview):

- This method is ignored if a short position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.22 ExitShortStopLimit()

Definition

Generates a buy to cover stop limit order to exit a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

`ExitShortStopLimit(double limitPrice, double stopPrice)`

```
ExitShortStopLimit(int quantity, double limitPrice, double stopPrice)
ExitShortStopLimit(double limitPrice, double stopPrice, string fromEntrySignal)
ExitShortStopLimit(double limitPrice, double stopPrice, string signalName, string
fromEntrySignal)
ExitShortStopLimit(int quantity, double limitPrice, double stopPrice, string
signalName, string fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitShortStopLimit(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double limitPrice, double stopPrice, string signalName, string fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
limitPrice	The limit price of the order
stopPrice	The stop price of the order.
quantity	Entry order quantity.
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
private double stopPrice = 0;

protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossBelow(SMA(10), SMA(20),
1))
     {
          EnterShort("SMA Cross Entry");
          stopPrice = Low[0] + 5 * TickSize;
     }

     // Exits position
     ExitShortStopLimit(stopPrice + (10 * TickSize), stopPrice);
}
```

Tips (also see Overview):

- This method is ignored if a short position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.23 ExitShortStopMarket()

Definition

Generates a buy to cover stop market order to exit a short position.

Method Return Value

An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Advanced Order Handling section.

Syntax

`ExitShortStopMarket(double stopPrice)`

```
ExitShortStopMarket(int quantity, double stopPrice)
ExitShortStopMarket(double stopPrice, string fromEntrySignal)
ExitShortStopMarket(double stopPrice, string signalName, string fromEntrySignal)
ExitShortStopMarket(int quantity, double stopPrice, string signalName, string
fromEntrySignal)
```

The following method variation is for experienced programmers who fully understand Advanced Order Handling concepts:

```
ExitShortStopMarket(int barsInProgressIndex, bool isLiveUntilCancelled, int quantity,
double stopPrice, string signalName, string fromEntrySignal)
```

Parameters

signalName	User defined signal name identifying the order generated. Max 50 characters.
fromEntrySignal	The entry signal name. This ties the exit to the entry and exits the position quantity represented by the actual entry. **Note**: Using an empty string will attach the exit order to all entries.
stopPrice	The stop price of the order.
quantity	Entry order quantity.
isLiveUntilCancelled	The order will **NOT** expire at the end of a bar but instead remain live until the CancelOrder() method is called or its time in force is reached.
barsInProgressIndex	The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for. See the BarsInProgress property.

Examples

```
private double stopPrice = 0;

protected override void OnBarUpdate()
{
     if (CurrentBar < 20)
          return;

     // Only enter if at least 10 bars has passed since our last
entry
     if ((BarsSinceEntryExecution() > 10 ||
BarsSinceEntryExecution() == -1) && CrossBelow(SMA(10), SMA(20),
1))
     {
          EnterShort("SMA Cross Entry");
          stopPrice = Low[0];
     }

     // Exits position
     ExitShortStopMarket(stopPrice);
}
```

Tips (also see Overview):

- This method is ignored if a short position does not exist
- It is helpful to provide a signal name if your strategy has multiple exit points to help identify your exits on a chart
- You can tie an exit to an entry by providing the entry signal name in the parameter "fromEntrySignal"
- If you do not specify a quantity the entire position is exited rendering your strategy flat
- If you do not specify a "fromEntrySignal" parameter the entire position is exited rendering your strategy flat

11.6.14.35.1.24 GetRealtimeOrder()

Definition

Returns a matching real-time order object based on a specified historical order object reference.

Note: This method is only needed if you have historical order references which you wish to transition and manage in real-time (i.e., you had a working order which was submitted historically and re-submitted in real-time as the strategy is enabled). This method only

needs to be called once per order object, and should be done in OnOrderUpdate to handle all scenarios. Please see the Advanced Order Handling section on transition orders for more details.

Method Return Value

Returns a real-time order reference associated with the historical order object. If no associated order exists (i.e. OrderState is Filled, Canceled, Rejected, Unknown), a null value returns

Syntax

```
GetRealtimeOrder(Order historicalOrder)
```

Parameters

historicalOrder	The historical order object to update to real-time

Examples

```
private Order myOrder;

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled, double
averageFillPrice, OrderState orderState, DateTime time, ErrorCode
error, string nativeError)
{
    // One time only, as we transition from historical
    // Convert any old historical order object references to the
live order submitted to the real-time account
    if (myOrder != null && myOrder.IsBacktestOrder && State ==
State.Realtime)
        myOrder = GetRealtimeOrder(myOrder);

    // Assign Order objects here
    // This is more reliable than assigning Order objects in
OnBarUpdate, as the assignment is not guaranteed to be complete if
it is referenced immediately after submitting
    if (order.Name == "myOrder Signal Name")
        myOrder = order;

    // Null Entry order if filled or cancelled. We do not use the
Order objects after the order is filled, so we can null it here
    if (myOrder != null && myOrder == order)
    {
        if (order.OrderState == OrderState.Cancelled &&
order.Filled == 0)
            myOrder = null;
        if (order.OrderState == OrderState.Filled)
            myOrder = null;
    }
}
```

11.6.14.35.1.25 SetParabolicStop

Definition

Generates a parabolic type trail stop order with the signal name "Parabolic stop" to exit a position. Parabolic stops are amended on a bar update basis, so dependent upon the Calculate setting of the parent strategy. Parabolic stop orders are real working orders (unless simulated is specified in which case the stop order is locally simulated and submitted as market once triggered) submitted immediately to the market upon receiving an execution from an entry order.

Although logic wise very similiar, this technique works different from the ParablicSAR indicator. The indicator will provide trailing stop levels 'always in the market' assuming a constant market position switch, either long or short (reversing). The SetParabolicStop() method in contrast will apply the same parabolic trailing technique sensitive to price acceleration to the custom strategy entry signal / position it is associated with.

Notes:

- The SetParabolicStop() method can NOT be used concurrently with the SetStopLoss() or SetTrailStop() method for the same position, if any of methods are called for the same position (fromEntrySignal) the SetStopLoss() will always take precedence. You can however, use all three methods in the same strategy if they reference different signal names.
- Parabolic stop orders are submitted in real-time on incoming executions from entry orders
- Since they are submitted upon receiving an execution, the Set method should be called **prior** to submitting the associated entry order to ensure an initial level is set.
- A strategy will either generate a trail stop order for each partial fill of an entry order or one order for all fills. See additional information under the Strategies tab of the Options dialog window.
- If a profit target order is generated in addition to a trail stop order, they are submitted as OCO (one cancels other)
- Parabolic stop orders are submitted as stop-market orders
- A parabolic stop order is automatically canceled if the managing position is closed by another strategy generated exit order
- Should you have multiple Bars objects of the same instrument while using SetParabolicStop() in your strategy, you should only submit orders for this instrument to the first Bars context of that instrument. This is to ensure your order logic is processed correctly and any necessary order amendments are done properly.
- Parabolic stop orders are modified based on the strategies 'Calculate' settings. In the case of 'Calculate' on bar close, when the bar closes the parabolic stop order modification will occur using the closing price of the bar as the reference price to apply the trail offset. Subsequently if the open price of the next bar is significantly higher or lower then the current close price then there is a possibility that the calculated parabolic stop price is now an invalid stop price. This is a risk with modifying any stop order closer to the current market price since any modification above/below the current price would be rejected.

Syntax

```
SetParabolicStop(CalculationMode mode, double value)
SetParabolicStop(string fromEntrySignal, CalculationMode mode, double value, bool
isSimulatedStop, double acceleration, double accelerationMax, double accelerationStep)
```

Warnings:

- This method **CANNOT** be called from the OnStateChange() method during **State.SetDefaults**
- CalculationMode.**Price** is irrelevant for trail stops. Attempting to use this mode will log a message and the stop order be ignored. Please use SetStopLoss() for this mode instead.

Parameters

mode	Determines the manner in which the value parameter is calculated Possible values are: CalculationMode.Currency: Initial PnL away from average entry. Calculated by the dollar per tick value for the order quantity used. When this mode is used, StopTargetHandling will automatically be set to **ByStrategyPosition.** The Stop loss will then continue to update following each parabolic step. CalculationMode.Percent: Percentage away from the average entry, based on the average entry price. CalculationMode.Pips: Pips away from average entry. CalculationMode.Ticks: Ticks away from entry

	average entry. Please note in percentage calculation mode a value of 1 is equal to 100%, a value of 0.1 is equal to 10%, and a value of 0.01 will be 1%
isSimulatedStop	If true, will simulate the stop order and submit as market once triggered
value	The value the trail stop order is offset from the position entry price
fromEntrySignal	The entry signal name. This ties the trail stop exit to the entry and exits the position quantity represented by the actual entry. Using an empty string will attach the exit order to all entries.
acceleration	Sets the acceleration value
accelerationMax	Sets the maximum acceleration value
accelerationStep	Sets the step value used to increment acceleration value

Tips (also see Overview):

- It is suggested to call this method from within the strategy OnStateChange() method if your stop loss price/offset is static
- You may call this method from within the strategy OnBarUpdate() method should you wish to dynamically change the stop loss price while in an open position
- Should you call this method to dynamically change the stop loss price in the strategy OnBarUpdate() method, you should always reset the stop loss price / offset value when your strategy is flat otherwise, the last price/offset value set will be used to generate your stop loss order on your next open position
- The signal name generated internally by this method is `"Parabolic stop"` which can be used with various methods such as BarsSinceExitExecution(), or other order concepts which rely on identifying a signal name

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Sets a parabolic stop using default acceleration
(0.02), accelerationMax (0.2), accelerationStep (0.02) settings and
a floor value of 12 ticks
          SetParabolicStop(CalculationMode.Ticks, 12);

          // Sets a parabolic stop of with a currency floor of 500
          SetParabolicStop("MyLongEntry", CalculationMode.Currency,
 500, false, 0.03, 0.3, 0.01);
     }
}
```

11.6.14.35.1.26 SetProfitTarget()

Definition

Generates a profit target order with the signal name "Profit target" to exit a position. Profit target orders are real working orders submitted immediately to the market upon receiving an execution from an entry order.

Notes:

- Profit target orders are submitted in real-time on incoming executions from entry orders
- Since they are submitted upon receiving an execution, the Set method should be called **prior** to submitting the associated entry order to ensure an initial level is set.
- A strategy will either generate a target order for each partial fill of an entry order or one order for all fills. See additional information under the Strategies tab of the Options dialog window.
- If a stop loss or trail stop order is generated in addition to a profit target order, they are submitted as OCO (one cancels other)
- A profit target order is automatically cancelled if the managing position is closed by another strategy generated exit order
- Should you have multiple Bars objects of the same instrument while using SetProfitTarget() in your strategy, you should only submit orders for this instrument to the first Bars context of that instrument. This is to ensure your order logic is processed correctly and any necessary order amendments are done properly.

Syntax

```
SetProfitTarget(CalculationMode mode, double value)
SetProfitTarget(CalculationMode mode, double value, bool isMIT)
SetProfitTarget(string fromEntrySignal, CalculationMode mode, double value)
```

```
SetProfitTarget(string fromEntrySignal, CalculationMode mode, double value, bool
isMIT)
```

Warning: This method **CANNOT** be called from the OnStateChange() method during **State.SetDefaults**

Parameters

currency	Sets the profit target amount in currency ($500 profit for example)
isMIT	Sets the profit target as a market-if-touched order
mode	Determines the manner in which the value parameter is calculated Possible values are: CalculationMode.Currency: PnL away from average entry. Calculated by the dollar per tick value for the order quantity used. When this mode is used, StopTargetHandling will automatically be set to **ByStrategyPosition** CalculationMode.Percent: Percentage away from the average entry, based on the average entry price.

	<table><tr><td>CalculationMode.Pips</td><td>Pips away from average entry.</td></tr><tr><td>CalculationMode.Price</td><td>The absolute price point specified.</td></tr><tr><td>CalculationMode.Ticks</td><td>Ticks away from entry average entry.</td></tr></table> Please note in percentage calculation mode a value of 1 is equal to 100%, a value of 0.1 is equal to 10%, and a value of 0.01 will be 1%
value	The value the profit target order is offset from the position entry price (exception is using .Price mode where 'value' will represent the actual price)
fromEntrySignal	The entry signal name. This ties the profit target exit to the entry and exits the position quantity represented by the actual entry. Using an empty string will attach the exit order to all entries.

Tips (also see Overview):

- It is suggested to call this method from within the strategy OnStateChange() method if your profit target price/offset is static
- You may call this method from within the strategy OnBarUpdate() method should you wish to dynamically change the target price while in an open position
- Should you call this method to dynamically change the target price in the strategy OnBarUpdate() method, you should always reset the target price / offset value when your strategy is flat otherwise, the last price/offset value set will be used to generate your profit target order on your next open position
- The signal name generated internally by this method is `"Profit target"` which can be used with various methods such as BarsSinceExitExecution(), or other order concepts which rely on identifying a signal name

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Submits a profit target order 10 ticks away from the
avg entry price
          SetProfitTarget(CalculationMode.Ticks, 10);
     }
}
```

11.6.14.35.1.27 SetStopLoss()

Definition

Generates a stop loss order with the signal name "Stop loss" used to exit a position. Stop loss orders are real working orders (unless simulated is specified in which case the stop order is locally simulated and submitted as market once triggered) submitted immediately to the market upon receiving an execution from an entry order.

Notes:

- Stop loss orders are submitted in real-time on incoming executions from entry orders
- Since they are submitted upon receiving an execution, the Set method should be called **prior** to submitting the associated entry order to ensure an initial level is set.
- A strategy will either generate a stop loss order for each partial fill of an entry order or one order for all fills. See additional information under the Strategies tab of the Options dialog window.
- If a profit target order is generated in addition to a stop loss order, they are submitted as OCO (one cancels other)
- Stop loss orders are submitted as stop-market orders
- A stop loss order is automatically canceled if the managing position is closed by another strategy generated exit order
- Should you have multiple Bars objects of the same instrument while using SetStopLoss() in your strategy, you should only submit orders for this instrument to the first Bars context of that instrument. This is to ensure your order logic is processed correctly and any necessary order amendments are done properly.
- The SetStopLoss() method can **NOT** be used concurrently with the SetTrailStop() or SetParabolicStop() method for the same position, if any methods are called for the same position (fromEntrySignal) the SetStopLoss() will always take precedence. You can however, use all three methods in the same strategy if they reference different signal names.

Syntax

```
SetStopLoss(CalculationMode mode, double value)
SetStopLoss(string fromEntrySignal, CalculationMode mode, double value, bool isSimulatedStop)
```

Warning: This method **CANNOT** be called from the OnStateChange() method during **State.SetDefaults**

Parameters

mode	Determines the manner in which the value parameter is calculated Possible values are: CalculationMode.Currency: PnL away from average entry. Calculated by the dollar per tick value for the order quantity used. When this mode is used, StopTargetHandling will automatically be set to **ByStrategyPosition** CalculationMode.Percent: Percentage away from the average entry, based on the average entry price.

	<table><tr><td>CalculationMode.Pips</td><td>Pips away from average entry.</td></tr><tr><td>CalculationMode.Price</td><td>The absolute price point specified.</td></tr><tr><td>CalculationMode.Ticks</td><td>Ticks away from entry average entry.</td></tr></table> Please note in percentage calculation mode a value of 1 is equal to 100%, a value of 0.1 is equal to 10%, and a value of 0.01 will be 1%
isSimulatedStop	If true, will simulate the stop order and submit as market once triggered
value	The value the stop loss order is offset from the position entry price (exception is using .Price mode where 'value' will represent the actual price)
fromEntrySignal	The entry signal name. This ties the stop loss exit to the entry and exits the position quantity represented by the actual entry. Using an empty string will attach the exit order to all entries.

Tips (also see Overview):

- It is suggested to call this method from within the strategy OnStateChange() method if your stop loss price/offset is static
- You may call this method from within the strategy OnBarUpdate() method should you wish to dynamically change the stop loss price while in an open position
- Should you call this method to dynamically change the stop loss price in the strategy OnBarUpdate() method, you should always reset the stop loss price / offset value when your strategy is flat otherwise, the last price/offset value set will be used to generate your stop loss order on your next open position

- The signal name generated internally by this method is `"Stop loss"` which can be used with various methods such as BarsSinceExitExecution(), or other order concepts which rely on identifying a signal name

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Submits a stop loss of $500
          SetStopLoss(CalculationMode.Currency, 500);
     }
}
```

11.6.14.35.1.28 SetTrailStop()

Definition

Generates a trail stop order with the signal name "Trail stop" to exit a position. Trail stops are amended on a bar update basis, so dependent upon the Calculate setting of the parent strategy. Trail stop orders are real working orders (unless simulated is specified in which case the stop order is locally simulated and submitted as market once triggered) submitted immediately to the market upon receiving an execution from an entry order.

Notes:

- The SetTrailStop() method can NOT be used concurrently with the SetStopLoss() or SetParabolicStop() method for the same position, if any of methods are called for the same position (fromEntrySignal) the SetStopLoss() will always take precedence. You can however, use all three methods in the same strategy if they reference different signal names.
- Trail stop orders are submitted in real-time on incoming executions from entry orders
- Since they are submitted upon receiving an execution, the Set method should be called **prior** to submitting the associated entry order to ensure an initial level is set.
- A strategy will either generate a trail stop order for each partial fill of an entry order or one order for all fills. See additional information under the Strategies tab of the Options dialog window.
- If a profit target order is generated in addition to a trail stop order, they are submitted as OCO (one cancels other)
- Trail stop orders are submitted as stop-market orders

- A trail stop order is automatically canceled if the managing position is closed by another strategy generated exit order
- Should you have multiple Bars objects of the same instrument while using SetTrailStop() in your strategy, you should only submit orders for this instrument to the first Bars context of that instrument. This is to ensure your order logic is processed correctly and any necessary order amendments are done properly.
- Trail stop orders are modified based on the strategies 'Calculate' settings. In the case of 'Calculate' on bar close, when the bar closes the trail stop order modification will occur using the lowest/highest price of the bar as the reference price to apply the trail offset. Subsequently if the open price of the next bar is significantly higher or lower than this price then there is a possibility that the calculated trail stop price is now an invalid stop price. This is a risk with modifying any stop order closer to the current market price since any modification above/below the current price would be rejected.

Syntax

```
SetTrailStop(CalculationMode mode, double value)
SetTrailStop(string fromEntrySignal, CalculationMode mode, double value, bool isSimulatedStop)
```

Warnings:

- This method **CANNOT** be called from the OnStateChange() method during **State.SetDefaults**
- CalculationMode.**Price** and CalculationMode.**Currency** are irrelevant for trail stops. Attempting to use one of these modes will log a message and the stop order be ignored. Please use SetStopLoss() for these modes instead.

Parameters

mode	Determines the manner in which the value parameter is calculated Possible values are: CalculationMode.Percent — Percentage away from the average entry, based on the average entry price.

	<table><tr><td>CalculationMode.Pips</td><td>Pips away from average entry.</td></tr><tr><td>CalculationMode.Ticks</td><td>Ticks away from entry average entry.</td></tr></table> Please note in percentage calculation mode a value of 1 is equal to 100%, a value of 0.1 is equal to 10%, and a value of 0.01 will be 1%
isSimulatedStop	If true, will simulate the stop order and submit as market once triggered
value	The value the trail stop order is offset from the position entry price (exception is using .Price mode where 'value' will represent the actual price)
fromEntrySignal	The entry signal name. This ties the trail stop exit to the entry and exits the position quantity represented by the actual entry. Using an empty string will attach the exit order to all entries.

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // Sets a trail stop of 12 ticks
          SetTrailStop(CalculationMode.Ticks, 12);
     }
}
```

Tips (also see Overview):

- It is suggested to call this method from within the strategy OnStateChange() method if your trail stop price/offset is static

- You may call this method from within the strategy OnBarUpdate() method should you wish to dynamically change the trail stop price while in an open position
- Should you call this method to dynamically change the trail stop price in the strategy OnBarUpdate() method, you should always reset the trail stop price / offset value when your strategy is flat otherwise, the last price/offset value set will be used to generate your trail stop order on your next open position
- The signal name generated internally by this method is `"Trail stop"` which can be used with various methods such as BarsSinceExitExecution(), or other order concepts which rely on identifying a signal name

11.6.14.35.2 Unmanaged Approach

The Unmanaged approach is reserved for **VERY EXPERIENCED** programmers. In place of the convenience layer that the Managed approach offered, the Unmanaged approach instead offers ultimate flexibility in terms of order submission and management. This section will discuss some of the basics of working with Unmanaged order methods.

Getting started with Unmanaged order methods

To be able to offer you the flexibility required to achieve more complex order submission techniques, NinjaTrader needs to be able to know if you are going to be using the Unmanaged approach beforehand.

In the OnStateChange() method designating the IsUnmanaged property as true signifies to NinjaTrader that you will be using the Unmanaged approach. Setting this will effectively prevent any of the signal tracking and internal order handling rules that were present in the Managed approach.

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          IsUnmanaged = true;
     }
}
```

Please note that you will ***not be able*** to mix order methods from the two approaches. When setting IsUnmanaged to true, you can only use Unmanaged order methods in the strategy.

Working with Unmanaged order methods

Order Submission

Order submission with the Unmanaged approach is done solely from a single order method. Parameterizing the SubmitOrderUnmanaged() method differently will determine what kind of order you will be submitting. Please note that these orders are live until cancelled. Should you want to cancel these orders you will need to use the CancelOrder() method or wait till the orders expire due to the strategy's time in force setting.

In the example below, a buy limit order to enter a long position is working at the bid price provided that the close price of the current bar is greater than the current value of the 20 period simple moving average.

```
protected override void OnBarUpdate()
{
     // Entry condition
     if (Close[0] > SMA(20)[0] && entryOrder == null)
          entryOrder = SubmitOrderUnmanaged(0,
OrderAction.Buy, OrderType.Limit, 1, GetCurrentBid(), 0,
"", "Long Limit");
}
```

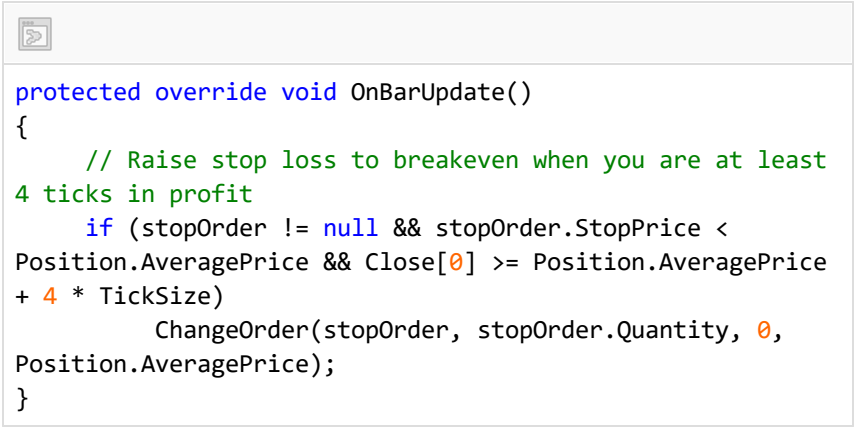
It is critical to assign an Order object to keep track of your order or else you will not be able to identify it in your code later since there is no signal tracking when using Unmanaged order methods. Please be aware of the following information about Order objects:

- An Order object returned from calling an order method is dynamic in that its properties will always reflect the current state of an order
- The property <Order>.OrderId is **NOT** a unique value since it can change throughout an order's lifetime
- To check for equality you can compare Order objects directly

Order Modification

Unlike the Managed approach where you could modify a working order by calling the entry order method again with your new parameters, the Unmanaged approach requires the utilization of the ChangeOrder() method. The ChangeOrder() method requires you to have access to the Order object you wish

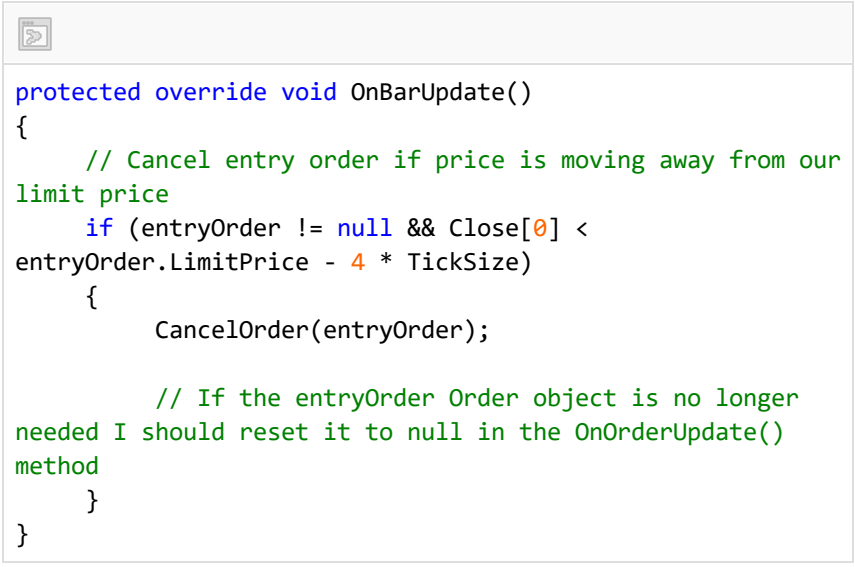
to modify so it is important to hold onto those for any active order you have in your strategy.



```
protected override void OnBarUpdate()
{
     // Raise stop loss to breakeven when you are at least
4 ticks in profit
     if (stopOrder != null && stopOrder.StopPrice <
Position.AveragePrice && Close[0] >= Position.AveragePrice
+ 4 * TickSize)
          ChangeOrder(stopOrder, stopOrder.Quantity, 0,
Position.AveragePrice);
}
```

Order Cancellation

Similar to the live until canceled technique from the Managed approach, canceling orders can be done through the CancelOrder() method.



```
protected override void OnBarUpdate()
{
     // Cancel entry order if price is moving away from our
limit price
     if (entryOrder != null && Close[0] <
entryOrder.LimitPrice - 4 * TickSize)
     {
          CancelOrder(entryOrder);

          // If the entryOrder Order object is no longer
needed I should reset it to null in the OnOrderUpdate()
method
     }
}
```

Signal Tracking

Since the Unmanaged approach does not utilize NinjaScript's signal tracking the features associated with it will no longer be relevant. The following properties and their associated concept cannot be used with Unmanaged order methods:

EntriesPerDirection

EntryHandling
SetOrderQuantity

Methods utilizing signal names like BarsSinceEntryExecution() and BarsSinceExitExecution() can still be used though.

Critical considerations when using Unmanaged order methods

When using the Unmanaged approach it is imperative to understand that NinjaTrader has many safety mechanisms that were present in the Managed approach turned off. There are critical issues that must be considered and your strategy must be programmed in a manner that addresses these concerns. Failure to do so may result in a serious adverse affect on your trading account.

Overfills

Overfills is a serious issue that can occur when using complex entry conditions that bracket the market in both directions end up with both entries being filled instead of one being canceled. Overfills can also occur when you place a trade quickly hoping to close a position while a prior order to close the same position already had an in-flight execution. The exact scenarios in which an overfill can occur is highly dependent on the specific strategy programming. By default, NinjaTrader will protect against overfills even though you are using the Unmanaged approach by halting the strategy, but should you decide to custom program your own overfill handling it is up to you to either prevent overfills from being a possibility in your code or by introducing logic to address overfills should one occur.

Order rejections

Order rejections are not local to using Unmanaged order methods, but the impact of improper rejection management is just as detrimental. Please be sure the strategy has significant contingency programming to handle order rejections so as to prevent your strategy from being left in some sort of limbo state. This is especially important if you decide to turn off RealtimeErrorHandling protection.

Connection Loss

Even though NinjaTrader provides connection loss handling features it is still important to ensure your recovered strategy's internal state is not in limbo. Should you have internal variables tracking various information it may be necessary for you to program your own additional connection loss handling into OnConnectionStatusUpdate() to properly recover all aspects of your strategy in the manner you desired.

CancelOrder()	Cancels a specified order.
ChangeOrder()	Amends a specified Order.
IgnoreOverfill	An unmanaged order property which defines the behavior of a strategy when an overfill is detected.
IsUnmanaged	Determines if the strategy will be using Unmanaged order methods.
SubmitOrderUnmanaged()	Generates an Unmanaged order.

11.6.14.35.2.1 CancelOrder()

Please see the "CancelOrder()" section under the "Managed Approach".

11.6.14.35.2.2 ChangeOrder()

Please see the "ChangeOrder()" section under the "Managed Approach".

11.6.14.35.2.3 IgnoreOverfill

Definition

An unmanaged order property which defines the behavior of a strategy when an overfill is detected. An overfill is categorized as when an order returns a "Filled" or "PartFilled" state after the order was already marked for cancellation. The cancel request could have been induced by an explicit CancelOrder() call, from more implicit cancellations like those that occur when another order sharing the same OCO ID is filled, or from things like order expiration.

Critical:
- Setting this property value to **true** can have **serious** adverse affects on a running strategy unless you have programmed your own overfill handling
- User defined overfill handling is advanced and should **ONLY** be addressed by **experienced programmers**. Additional information can be found on overfills in the Unmanaged approach section

Property Value

This property returns **true** if the strategy will ignore overfills; otherwise, **false**. Default is set to **false**.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

```
IgnoreOverfill
```

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        // Allows for custom overfill handling
        IgnoreOverfill = true;
    }
}
```

11.6.14.35.2.4 IsUnmanaged

Definition

Determines if the strategy will be using Unmanaged order methods.

> **Note**: Unmanaged order methods and Managed order methods **CANNOT** be used interchangeably. When IsUnmanaged is set to **true**, calling managed order methods such as EnterLong(), SetStopLoss(), etc, will generate an error which will be displayed on the Log tab of the Control Center.

Property Value

This property returns **true** if the strategy will use Unmanaged order methods; otherwise, **false**. Default is set to **false**.

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax
IsUnmanaged

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Use Unmanaged order methods
          IsUnmanaged = true;
     }
}
```

11.6.14.35.2.5 SubmitOrderUnmanaged()

Definition
Generates an Unmanaged order.

Method Return Value
An Order read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the Unmanaged Approach section.

Syntax
```
SubmitOrderUnmanaged(int selectedBarsInProgress, OrderAction orderAction, OrderType
orderType, int quantity)
SubmitOrderUnmanaged(int selectedBarsInProgress, OrderAction orderAction, OrderType
orderType, int quantity, double limitPrice)
SubmitOrderUnmanaged(int selectedBarsInProgress, OrderAction orderAction, OrderType
orderType, int quantity, double limitPrice, double stopPrice)
SubmitOrderUnmanaged(int selectedBarsInProgress, OrderAction orderAction, OrderType
orderType, int quantity, double limitPrice, double stopPrice, string oco)
SubmitOrderUnmanaged(int selectedBarsInProgress, OrderAction orderAction, OrderType
orderType, int quantity, double limitPrice, double stopPrice, string oco, string
signalName)
```

Parameters

selectedBarsInProgress	The index of the Bars object the order is to be submitted against. This determines what instrument the order is submitted for. **Note**: See the BarsInProgress property.

orderAction	Determines if the order is a buy or sell order Possible values: OrderAction.Buy OrderAction.BuyToCover OrderAction.Sell OrderAction.SellShort
orderType	Determines the type of order submitted Possible values: OrderType.Limit OrderType.Market OrderType.MIT OrderType.StopMarket OrderType.StopLimit
quantity	Sets the number of contracts to submit with the order
limitPrice	Order limit price. Use "0" should this parameter be irrelevant for the OrderType being submitted.
stopPrice	Order stop price. Use "0" should this parameter be irrelevant for the OrderType being submitted.
oco	A string representing the OCO ID used to link OCO orders together **Note**: OCO strings should not be reused. Use unique strings for each OCO group, and reset after orders in that group are filled/canceled
signalName	A string representing the name of the order. Max 50 characters.

Examples

```
private Order entryOrder = null;

protected override void OnBarUpdate()
{
     // Entry condition
     if (Close[0] > SMA(20)[0] && entryOrder == null)
          SubmitOrderUnmanaged(0, OrderAction.Buy,
OrderType.Market, 1, 0, 0, "", "Enter Long");
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled, double
averageFillPrice, OrderState orderState, DateTime time, ErrorCode
error, string nativeError)
{
    // Assign entryOrder in OnOrderUpdate() to ensure the
assignment occurs when expected.
    // This is more reliable than assigning Order objects in
OnBarUpdate, as the assignment is not gauranteed to be complete if
it is referenced immediately after submitting
    if (order.Name == "Enter Long" && orderState ==
OrderState.Filled)
        entryOrder = order;
}
```

11.6.14.3 OrderFillResolution

Definition

Determines how strategy orders are filled during historical states.

Please see Understanding Historical Fill Processing for general information on historical fill processing.

Property Value

An enum value that determines how the strategy orders are filled. Default value is set to OrderFillResolution.Standard. Possible values are:

OrderFillResolution.Standard	Faster - Uses the existing bar type and interval that you are running the backtest on to fill your orders.
OrderFillResolution.High	More granular - Allows you to set a secondary bar series to be used as the price data to fill your orders. (See also OrderFillResolutionType and OrderFillResolutionValue)

Syntax
`OrderFillResolution`

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during State.SetDefaults

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "ExampleStrategy";
      OrderFillResolution = OrderFillResolution.Standard;
   }
}
```

11.6.14.3 OrderFillResolutionType

Definition
Determines the bars type which will be used for historical fill processing.

> **Note:** This property will only be valid if the OrderFillResolution is set to OrderFillResolution.High

Property Value
A BarsPeriodType representing the type of bars during historical order processing. Default value is set to BarsPeriodType.Minute.

Syntax
`OrderFillResolutionType`

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during State.SetDefaults

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "ExampleStrategy";

      // use one second bars for filling orders
      OrderFillResolution        = OrderFillResolution.High;

      OrderFillResolutionType    = BarsPeriodType.Second;
      OrderFillResolutionValue   = 1;
   }

}
```

11.6.14.3 OrderFillResolutionValue

Definition

Determines the bars period interval value which will be used for historical fill processing.

> **Note:** This property will only be valid if the OrderFillResolution is set to OrderFillResolution.High

Property Value

A `int` representing the interval used for the bars period during historical order processing. Default value is set to 1.

Syntax

`OrderFillResolutionValue`

> **Warning**: This property should **ONLY** bet set from the OnStateChange() method during State.SetDefaults

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "ExampleStrategy";

      // use one second bars for filling orders
      OrderFillResolution        = OrderFillResolution.High;

      OrderFillResolutionType    = BarsPeriodType.Second;
      OrderFillResolutionValue   = 1;
   }

}
```

11.6.14.3 PerformanceMetrics

Definition

Holds an array of PerformanceMetrics objects that represent custom metrics that can be used for strategy calcuations.

Index value is based on the the array of Bars objects added via the AddPerformanceMetric method.

Property Value

An array of PerformanceMetrics objects.

Syntax

```
PerformanceMetrics[int index]
```

Examples

```
// Define a new SampleCumProfit object
NinjaTrader.NinjaScript.PerformanceMetrics.SampleCumProfit
myProfit;

protected override void OnStateChange()
{
   if (State == State.Configure)
   {
      // Instantiate myProfit to a new instance of SampleCumProfit
      myProfit = new
NinjaTrader.NinjaScript.PerformanceMetrics.SampleCumProfit();

      // Use AddPerformanceMetric to add myProfit to the strategy
      AddPerformanceMetric(myProfit);
   }
}

protected override void OnBarUpdate()
{
   // Print a string representing the Type of the performance
metric at Index 0 of the PerformanceMetrics collection
   Print(PerformanceMetrics[0]);
}
```

11.6.14.4 Plots

Plotting functionality for NinjaScript Strategies is largely identical to the framework for Indicators. Please review the Plots / AddPlot() page under the Indicators section.

An overview of the draw or plotting related methods / properties available to NinjaScript Strategies vs. Indicators is listed below -

Method or Property	Strategy	Indicator
AddChartIndicator()	✓	✗
AddLine()	✓	✓
AddPlot()	✓	✓
AllowRemovalOfDrawObjects	✓	✓
AreLinesConfigurable	✓	✓
ArePlotsConfigurable	✓	✓
BackBrush	✓	✓
BackBrushAll	✓	✓
BackBrushes	✓	✓
BackBrushesAll	✓	✓
BarBrush	✓	✓

BarBrushes	✓	✓
CandleOutlineBrush	✓	✓
CandleOutlineBrushes	✓	✓
ChartBars	✓	✓
ChartControl	✓	✓
ChartIndicators[]	✓	✗
ChartObjects	✗	✓
ChartPanel	✓	✓
DisplayInDataBox	✓	✓
Draw.Methods()	✓	✓
DrawHorizontalGridLines	✗	✓
DrawObjects	✓	✓
DrawOnPricePanel	✓	✓
DrawVerticalGridLines	✗	✓
ForceRefresh()	✗	✓
FormatPriceMarker()	✓	✓
GetValueAt()	✓	✓
IsAutoScale	✓	✓
IsOverlay	✓	✓
IsTradingHoursBreakLineVisible	✗	✓
IsValidDataPoint()	✓	✓
Lines[]	✓	✓
MaxValue	✓	✓
MinValue	✓	✓
OnCalculateMinMax()	✓	✓
OnRender()	✓	✓
OnRenderTargetChanged()	✓	✓
PaintPriceMarkers	✗	✓
Panel	✓	✓
PanelUI	✓	✓
PlotBrushes[]	✓	✓
Plots[]	✓	✓
RemoveDrawObject()	✓	✓
RemoveDrawObjects()	✓	✓
RenderTarget	✓	✓
ScaleJustification	✓	✓
SetZOrder()	✓	✓
ShowTransparentPlotsInDataBox	✓	✓

UserControlCollection[]	✓	✓
ZOrder	✓	✓

11.6.14.4 Position

Definition

Represents position related information that pertains to an instance of a strategy.

Tips:
- For multi-instrument scripts, please see Positions object which holds an array of all instrument positions managed by the strategy's account
- For a real-world Account Position, please see PositionAccount.

Methods and Properties

Account	An Account object which corresponds to the position
AveragePrice	Gets the average entry price of the strategy position
GetUnrealizedProfitLoss()	Gets the unrealized PnL
Instrument	An Instrument value representing the instrument of an order
MarketPosition	Gets the current market position Possible values: MarketPosition.Flat MarketPosition.Long MarketPosition.Short
Quantity	Gets the current position size
ToString()	A `string` representation of a position

Examples

```
protected override void OnBarUpdate()
{
     // Print out the average entry price
     Print("The average entry price is " + Position.AveragePrice);
}
```

11.6.14.41.1 AveragePrice

Definition
Gets the average price of a strategy position.

Property Value
A `double` value representing the position's average price per unit.

Syntax
`Position.AveragePrice`

Examples

```
protected override void OnBarUpdate()
{
     // Raise stop loss to breakeven when there is at least 10
ticks in profit
     if (Close[0] >= Position.AveragePrice + 10 * TickSize)
          ExitLongStopMarket(Position.Quantity,
Position.AveragePrice);
}
```

11.6.14.41.2 GetUnrealizedProfitLoss()

Definition
Calculates the unrealized PnL for the strategy position.

Method Return Value
A `double` value representing the unrealized PnL.

Syntax
`Position.GetUnrealizedProfitLoss(PerformanceUnit unit, [double price])`

Note:

- If no double argument is provided in the call, the current (real-time) Last price will be substituted in. In case Tools > Options > Trading > 'Use last price for Pnl' is unchecked or the instrument type is Forex / CFD the bid (for a long position) / ask (for a short position) would be used as a substitute.
- For back-testing a double price to compare against should be provided like in our example below.

Parameters

unit	Possible values: PerformanceUnit.Currency PerformanceUnit.Percent PerformanceUnit.Pips PerformanceUnit.Points PerformanceUnit.Ticks
price	Optional price passed in used to calculate the PnL such as Close[0]. This value is used as the current price and compared against your entry price for the PnL.

Examples

```
protected override void OnBarUpdate()
{
     // If not flat print our unrealized PnL
     if (Position.MarketPosition != MarketPosition.Flat)
          Print("Open PnL: " +
Position.GetUnrealizedProfitLoss(PerformanceUnit.Points,
Close[0]));
}
```

11.6.14.41.3 Instrument

Definition

Gets the instrument of a strategy position.

Property Value

An Instrument representing the position's instrument.

Syntax

```
Position.Instrument
```

Examples

```
protected override void OnPositionUpdate(Position position, double
averagePrice, int quantity, MarketPosition marketPosition)
{
     // If the position is an AAPL position
     if (position.Instrument.MasterInstrument.Name == "AAPL")
    {
         //do something
    }
}
```

11.6.14.41.4 MarketPosition

Definition

Gets the strategy's current market position

Property Value

MarketPosition.Flat
MarketPosition.Long
MarketPosition.Short

Syntax

`Position.MarketPosition`

Examples

```
protected override void OnBarUpdate()
{
     // If not flat print our open PnL
     if (Position.MarketPosition != MarketPosition.Flat)
          Print("Open PnL: " +
Position.GetUnrealizedProfitLoss(PerformanceUnit.Points,
Close[0]));
}
```

11.6.14.41.5 Quantity

Definition

Gets the strategy's current position size.

Property Value

An `int` value representing the position size.

Syntax

```
Position.Quantity
```

Examples

```
protected override void OnBarUpdate()
{
     // Prints out the current market position
     Print(Position.MarketPosition.ToString() + " " +
Position.Quantity.ToString());
}
```

11.6.14.4 PositionAccount

Definition

Represents position related information that pertains to real-world account (live or simulation).

Tips:

- For multi-instrument scripts, please see PositionsAccount object which holds an array of all instrument positions managed by the strategy's account
- For a Strategy Position, please see Position

Methods and Properties

Account	An Account object which corresponds to the position
AveragePrice	Gets the average entry price of the account position
GetUnrealizedProfitLoss()	Gets the unrealized PnL for the account
Instrument	An Instrument value representing the instrument of an order
MarketPosition	Gets the current market position of the account Possible values: MarketPosition.Flat

	MarketPosition.Long MarketPosition.Short
Quantity	Gets the current account position size
ToString()	A `string` representation of an account position

Examples

```
protected override void OnBarUpdate()
{
     // Print out the average entry price
     Print("The average entry price is " +
PositionAccount.AveragePrice);
}
```

11.6.14.42.1 AveragePrice

Definition

Gets the average price of an account position.

Property Value

A `double` value representing the account position's average price per unit.

Syntax

`PositionAccount.AveragePrice`

Examples

```
protected override void OnBarUpdate()
{
     // Raise stop loss to breakeven when there is at least 10
ticks in profit
     if (Close[0] >= PositionAccount.AveragePrice + 10 * TickSize)
          ExitLongStopMarket(PositionAccount.Quantity,
PositionAccount.AveragePrice);
}
```

11.6.14.42.2 GetUnrealizedProfitLoss()

Definition

Calculates the unrealized PnL for the account position.

Method Return Value

A `double` value representing the account's unrealized PnL.

Syntax

```
PositionAccount.GetUnrealizedProfitLoss(PerformanceUnit unit, [double price])
```

Note:

- If no double argument is provided in the call, the current (real-time) Last price will be substituted in. In case Tools > Options > Trading > 'Use last price for Pnl' is unchecked or the instrument type is Forex / CFD the bid (for a long position) / ask (for a short position) would be used as a substitute.
- For back-testing a double price to compare against should be provided like in our example below.

Parameters

unit	Possible values: PerformanceUnit.Currency PerformanceUnit.Percent PerformanceUnit.Pips PerformanceUnit.Points PerformanceUnit.Ticks
price	Optional price passed in used to calculate the PnL such as Close[0]. This value is used as the current price and compared against your entry price for the PnL.

Examples

```
protected override void OnBarUpdate()
{
     // If not flat print our unrealized PnL
     if (PositionAccount.MarketPosition != MarketPosition.Flat)
          Print("Open PnL: " +
PositionAccount.GetUnrealizedProfitLoss(PerformanceUnit.Points,
Close[0]));
}
```

11.6.14.42.3 Instrument

Definition

Gets the instrument of an account position.

Property Value

An Instrument representing the account's instrument position

Syntax

`PositionAccount.Instrument`

Examples

```
protected override void OnPositionUpdate(Position position, double
averagePrice, int quantity, MarketPosition marketPosition)
{
     // If the position is an AAPL position
     if (PositionAccount.Instrument.MasterInstrument.Name ==
"AAPL")
    {
          //do something
    }
}
```

11.6.14.42.4 MarketPosition

Definition

Gets the account's current market position

Property Value

MarketPosition.Flat
MarketPosition.Long
MarketPosition.Short

Syntax

```
PositionAccount.MarketPosition
```

Examples

```
protected override void OnBarUpdate()
{
     // If not flat print our open PnL
     if (PositionAccount.MarketPosition != MarketPosition.Flat)
          Print("Open PnL: " +
PositionAccount.GetUnrealizedProfitLoss(PerformanceUnit.Points,
Close[0]));
}
```

11.6.14.42.5 Quantity

Definition

Gets the current account's position size.

Property Value

An `int` value representing the account's position size.

Syntax

```
PositionAccount.Quantity
```

Examples

```
protected override void OnBarUpdate()
{
     // Prints out the current market position
     Print(PositionAccount.MarketPosition.ToString() + " " +
PositionAccount.Quantity.ToString());
}
```

11.6.14.43 Positions

Definition

Holds an array of Position objects that represent positions managed by the strategy. This property should only be used when your strategy is executing orders against multiple instruments.

Index value is based on the the array of Bars objects added via the AddDataSeries() method. For example:

First Bars is ES 1 Minute
Secondary Bars is ES 5 Minute
Third Bars is NQ 5 Minute

Positions[0] == ES position
Positions[1] == Always a flat position, ES position will always be Positions[0]
Positions[2] == NQ position

Tips:
- For single instrument scripts, please see Position object
- For a real-world Account Positions, please see PositionsAccount

Property Value
An array of Position objects.

Syntax
```
Positions[int index]
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          AddDataSeries("ES 09-14", BarsPeriodType.Minute, 5);
          AddDataSeries("NQ 09-14", BarsPeriodType.Minute, 5);
     }
}

protected override void OnBarUpdate()
{
     Print("ES position is " + Positions[0].MarketPosition);
     Print("NQ positions is " + Positions[2].MarketPosition);

     // Alternative approach. By checking what Bars object is
calling the OnBarUpdate()
     // method, we can just use the Position property since its
pointing to the correct
     // position.
     if (BarsInProgress == 0)
          Print("ES position is " + Position.MarketPosition);
     else if (BarsInProgress = 2)
          Print("NQ position is " + Position.MarketPosition);
}
```

11.6.14.4 PositionsAccount

Definition

Holds an array of PositionAccount objects that represent positions managed by the strategy's account. This property should only be used when your strategy is executing orders against multiple instruments.

Index value is based on the the array of Bars objects added via the AddDataSeries() method. For example:

First Bars is ES 1 Minute
Secondary Bars is ES 5 Minute
Third Bars is NQ 5 Minute

PositionsAccount[0] == ES position
PositionsAccount[1] == Always a flat position, ES position will always be PositionsAccount[0]
PositionsAccount[2] == NQ position

Tips:
- For single instrument scripts, please see PositionAccount object

- For Strategy Positions, please see Positions

Property Value
An array of PositionAccount objects.

Syntax
```
PositionsAccount[int index]
```

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "ExampleStrategy";

   }

   else if (State == State.Configure)
   {
      AddDataSeries("ES 03-15", BarsPeriodType.Minute, 5);
      AddDataSeries("NQ 03-15", BarsPeriodType.Minute, 5);

   }
}

protected override void OnBarUpdate()
{
     Print("ES account position is " +
PositionsAccount[0].MarketPosition);
     Print("NQ account position is " +
PositionsAccount[2].MarketPosition);

     // Alternative approach. By checking what Bars object is
calling the OnBarUpdate()
     // method, we can just use the Position property since its
pointing to the correct
     // position.
     if (BarsInProgress == 0)
          Print("ES account position is " +
PositionAccount.MarketPosition);
     else if (BarsInProgress == 2)
          Print("NQ account position is " +
PositionAccount.MarketPosition);
}
```

11.6.14.4 RealtimeErrorHandling

Definition

Defines the behavior of a strategy when a strategy generated order is returned from the broker's server in a "Rejected" state. Default behavior is to stop the strategy, cancel any remaining working orders, and then close any open positions managed by the strategy by submitting one "Close" order for each unique position.

Critical:

- Setting this property value to **IgnoreAllErrors** can have **serious** adverse affects on a running strategy unless you have programmed your own order rejection handling in the OnOrderUpdate() method
- User defined rejection handling is advanced and should **ONLY** be addressed by experienced programmers

Property Value

An `enum` value determining how the strategy behaves. Default value is set to RealtimeErrorHandling.StopCancelClose. Possible values include:

RealtimeErrorHandling.IgnoreAllErrors	Ignores any order errors received by the strategy and will continue running.
RealtimeErrorHandling.StopCancelClose	Default behavior of a strategy
RealtimeErrorHandling.StopCancelCloseIgnoreRejects	Will perform default behavior on all errors except order rejections

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax

```
RealtimeErrorHandling
```

Examples

```
private Order stopLossOrder = null;
private Order entryOrder = null;

protected override void OnStateChange()
{
   if (State == State.Configure)
   {
      RealtimeErrorHandling =
RealtimeErrorHandling.IgnoreAllErrors;
   }
}

protected override void OnBarUpdate()
{
   if (entryOrder == null && Close[0] > Open[0])
      EnterLong("myEntryOrder");

   if (stopLossOrder == null)
      stopLossOrder = ExitLongStopMarket(Position.AveragePrice - 10
 * TickSize, "myStopLoss", "myEntryOrder");
}

protected override void OnOrderUpdate(Order order, double
limitPrice, double stopPrice, int quantity, int filled, double
averageFillPrice,
                                      OrderState orderState,
DateTime time, ErrorCode error, string nativeError)
{
   // Assign stopLossOrder in OnOrderUpdate() to ensure the
assignment occurs when expected.
   // This is more reliable than assigning Order objects in
OnBarUpdate,
   // as the assignment is not guaranteed to be complete if it is
referenced immediately after submitting
   if (order.Name == "myStopLoss" && orderState ==
OrderState.Filled)
      stopLossOrder = order;

   if (stopLossOrder != null && stopLossOrder == order)
   {
      // Rejection handling
      if (order.OrderState == OrderState.Rejected)
      {
         // Stop loss order was rejected !!!!
         // Do something about it here
      }
   }
}
```

11.6.14.44 RestartsWithinMinutes

Definition

Determines within how many minutes the strategy will attempt to restart. The strategy will only restart off a reestablished connection when there have been fewer restart attempts than NumberRestartAttempts in the last NumberRestartAttempts time span. The purpose of these settings is to stop the strategy should your connection be unstable and incapable of maintaining a consistent connected state.

Property Value

An `int` value representing the maximum number of minutes in the time span in which restart attempts have to be less than NumberRestartAttempts for a strategy to be restarted when a connection is reestablished. Default value is set to 5.

Syntax

```
RestartsWithinMinutes
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          /* Allow for restarting the strategy only if there were
less restart attempts than
          MaxRestartAttempts within the last 5 minutes */
          RestartsWithinMinutes = 5;
     }
}
```

11.6.14.45 SetOrderQuantity

Definition

Determines how order sizes are calculated for a given strategy.

Property Value

An `enum` determining how order quantities are set. Default value is set to SetOrderQuantity.Strategy.

Possible values are:

SetOrderQuantity.DefaultQuantity	User defined order size based on

	the DefaultQuantity property
SetOrderQuantity.Strategy	Takes the order size specified programmatically within the strategy

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax
`SetOrderQuantity`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          SetOrderQuantity = SetOrderQuantity.DefaultQuantity; //
calculate orders based off default size
     }
}
```

11.6.14.4 Slippage

Definition
Sets the amount of slippage in ticks per execution used in performance calculations during backtests.

Property Value
An `int` value representing the number ticks. Default value is set to 0.

Syntax
`Slippage`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          Slippage = 2;
     }
}
```

11.6.14.4 StartBehavior

Definition

Sets the start behavior of the strategy. See Syncing Account Positions for more information.

> **Note**: In order to use **AdoptAccountPosition** you will need to first set IsAdoptAccountPositionAware to **true**. Please be sure that your strategy is specifically programmed in a manner that can accommodate account positions before using this mode.

Property Value

An `enum` value that determines how the strategy behaves; Default value is set to StartBehavior.WaitUntilFlat. Possible values are:

StartBehavior.AdoptAccountPosition
StartBehavior.ImmediatelySubmit
StartBehavior.ImmediatelySubmitSynchronizeAccount
StartBehavior.WaitUntilFlat
StartBehavior.WaitUntilFlatSynchronizeAccount

Syntax

`StartBehavior`

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          StartBehavior = StartBehavior.WaitUntilFlat;
     }
}
```

11.6.14.5 StopTargetHandling

Definition

Determines how stop and target orders are submitted during an entry order execution.

Property Value

An `enum` value that determines how the strategy behaves. Default value is set to StopTargetHandling.PerEntryExecution. Possible values are:

StopTargetHandling.ByStrategyPosition	Stop and Target order quantities will match the current strategy position. (Stops and targets may result in "stacked" orders on partial fills)
StopTargetHandling.PerEntryExecution	Stop and Target orders will match the total entry execution. (Stops and targets order quantities may not match strategy position under a partial fill scenario)

Warning: If your strategy executes to an Interactive Brokers or TD Ameritrade account, the StopTargetHandling will always be forced to .ByStrategyPosition

Syntax

`StopTargetHandling`

Tip: The default strategy behavior is to match the order quantity used for the stops and targets to the total entry execution. However in cases where the strategy's entry order is partially filled, StopTargetHandling.PerEntryExecution will result in a new set of stop loss and profit target orders for each entry execution. If you would prefer all of your stops and targets to be placed at the same time within the same order, it is suggested to use StopTargetHandling.ByStrategyPosition. However this may result in more stop and target

orders being submitted than the overall strategy position in a scenario in which the strategy's entire entry orders are not filled in one fill.

Example

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          StopTargetHandling =
StopTargetHandling.PerEntryExecution;
     }
}
```

11.6.14.5 StrategyBaseConverter

Definition

A custom TypeConverter class handling the designed behavior of a strategy's property descriptor collection. Use this as a base class for any custom **TypeConverter** you are applying to a strategy class.

Notes:

- A working NinjaScript demo can be found through the reference sample on "Using a TypeConverter to Customize Property Grid Behavior"
- When applying the custom converter, you must fully qualify the name (e.g., "NinjaTrader.NinjaScript.Strategies.MyCustomConveter")
- Additional **TypeConverter** information can be found from the MSDN documentation
- See also TypeConverterAttribute
- For Indicators, see the IndicatorBaseConverter class

Relevant base methods

TypeConverter.GetProperties()	When overriding **GetProperties()**, calling `base.GetProperties()` ensures that all default property grid behavior works as designed
TypeConverter.GetPropertiesSupported()	In your custom converter class, you must override

	GetPropertiesSupported() and return a value of **true** in order for your custom type converter to work

Syntax

```
public class StrategyBaseConverter : TypeConverter
```

Warning: Failure to apply a type of **StrategyBaseConverter** on an strategy class can result in unpredictable behavior of the standard NinjaTrader WPF property grid.

Tip: Common strategy functions like Print() are not available to a type converter instance. To debug a type converter class, you can use the AddOn Debug Concepts or attach to a debugger (recommended)

Examples

```
//This namespace holds Strategies in this folder and is required.
Do not change it.
namespace NinjaTrader.NinjaScript.Strategies
{
   // When applying the type converter, you must fully qualify the
name
   [TypeConverter("NinjaTrader.NinjaScript.Strategies.MyCustomConve
ter")]
   public class MyCustomStrategy : Strategy
   {
      protected override void OnStateChange()
      {
         if (State == State.SetDefaults)
         {
            Name                                = "MyCustomStrategy";
         }
      }

      protected override void OnBarUpdate()
      {
         //Add your custom strategy logic here.
      }
   }

   // custom converter class for strategies
   public class MyCustomConveter : StrategyBaseConverter
   {
      // A general TypeConveter method used for converting types
      public override PropertyDescriptorCollection
GetProperties(ITypeDescriptorContext context, object component,
Attribute[] attrs)
      {
         // sometimes you may need the strategy instance which
actually exists on the grid
         MyCustomStrategy strategy = component as MyCustomStrategy;

         // base.GetProperties ensures we have all the properties
(and associated property grid editors)
         // NinjaTrader internal logic handles for a given strategy
         PropertyDescriptorCollection propertyDescriptorCollection
= base.GetPropertiesSupported(context)
                  ? base.GetProperties(context, component, attrs) :
TypeDescriptor.GetProperties(component, attrs);

         if (strategy == null || propertyDescriptorCollection ==
null)
            return propertyDescriptorCollection;

         // example of why you may need the instance that exists on
the grid....
         if (strategy.EntryHandling == EntryHandling.UniqueEntries)
         {
            // do something in the event a property contains some
```

11.6.14.5 SystemPerformance

Definition

The SystemPerformance object holds all trades and trade performance data generated by a strategy.

Notes:

- A NinjaScript strategy can generate both synthetic trades (historical backtest trades) and real-time trades executed on a real-time data stream. If you wish to access only real-time trades, access the "RealTimeTrades" collection
- The first trade of the "RealTimeTrades" collection will contain a synthetic entry execution if the strategy was **NOT** flat at the time you start the strategy.
- These properties require that IncludeTradeHistoryInBacktest be set to `true`.

Methods and Properties

AllTrades	Gets a TradeCollection object of all trades generated by the strategy
LongTrades	Gets a TradeCollection object of long trades generated by the strategy
RealTimeTrades	Gets a TradeCollection object of real-time trades generated by the strategy
ShortTrades	Gets a TradeCollection object of short trades generated by the strategy

Examples

```
protected override void OnBarUpdate()
{
     // Print out the number of long trades
     Print("The strategy has taken " +
SystemPerformance.LongTrades.Count + " long trades.");
}
```

11.6.14.52.1 AllTrades

Definition

A TradeCollection object of all trades generated by a strategy.

Syntax
SystemPerformance.AllTrades

Examples

```
protected override void OnBarUpdate()
{
     // Print out the number of long trades
     Print("The strategy has taken " +
SystemPerformance.AllTrades.Count + " trades.");
}
```

11.6.14.52.2 LongTrades

Definition
LongTrades is a TradeCollection object of long trades generated by a strategy.

Syntax
SystemPerformance.LongTrades

Examples

```
protected override void OnBarUpdate()
{
     // Print out the number of long trades
     Print("The strategy has taken " +
SystemPerformance.LongTrades.Count + " long trades.");
}
```

11.6.14.52.3 RealTimeTrades

Definition
RealTimeTrades is a TradeCollection object of real-time trades generated by a strategy.

Syntax
SystemPerformance.RealTimeTrades

Examples

```
protected override void OnBarUpdate()
{
     // Print out the number of real-time trades
     Print("The strategy has taken " +
SystemPerformance.RealTimeTrades.Count + " real-time trades.");
}
```

11.6.14.52.4 ShortTrades

Definition

ShortTrades is a TradeCollection object of short trades generated by a strategy.

Syntax

```
SystemPerformance.ShortTrades
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the number of short trades
     Print("The strategy has taken " +
SystemPerformance.ShortTrades.Count + " short trades.");
}
```

11.6.14.53 TestPeriod

Definition

Reserved for Walk-Forward Optimization, this property determines the number of days used for the "out of sample" backtest period for a given strategy. See also OptimizationPeriod.

> **Note**: This property should **ONLY** be called from the OnStateChange() method during State.SetDefaults

Property Value

An `int` value representing the number of "out of sample" days used for walk-forward optimization; Default value is set to 28

Syntax

```
TestPeriod
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          //set the default TestPeriod to 31 days for WFOs
          TestPeriod = 31;
     }
}
```

11.6.14.5 TimeInForce

Definition

Sets the time in force property for all orders generated by a strategy. The selected TIF parameter is sent to your broker on order submission and will instruct how long you would like the order to be active before it is cancelled.

Note: This property is dependent on what time in force your broker may or may not support. If a brokerage / exchange combination is not compatible with a particular time in force, the order will be rejected by the broker. NinjaTrader does not have a method to prevent an unsupported TIF to be sent to a particular exchange. For questions about what TIF may be supported, please contact your broker directly.

Property Value

An `enum` value that determines the time in force. Default value is set to TimeInForce.Gtc. Possible values are:

TimeInForce.Day	Orders will be canceled by the broker at the end of the trading session
TimeInForce.Gtc	Order will remain working until the order is explicitly cancelled.
TimeInForce.Gtd	Order will remain working until the specified date

Syntax

`TimeInForce`

Examples

	Setting default TIF for all strategy orders

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          TimeInForce = TimeInForce.Day;
     }
}
```

	Setting TIF conditionally

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
       if (Instrument != null)
       {
              if (Instrument.Exchange == Exchange.Nybot)
                     TimeInForce = TimeInForce.Day;
              else if (Instrument.Exchange == Exchange.Globex)
                     TimeInForce = TimeInForce.Gtc;
       }
     }
}
```

11.6.14.5 TraceOrders

Definition

Determines if OnOrderTrace() would be called for a given strategy. When enabled, traces are generated and displayed in the NinjaScript Output window for each call of an order method providing confirmation that the method is entered and providing information if order methods are ignored and why. This is valuable for debugging if you are not seeing expected behavior when calling an order method. This property can be set programatically in the OnStateChange() method.

The output will reference a method "PlaceOrder()" which is an internal method that all Enter() and Exit() methods use.

Property Value

This property returns **true** if the strategy will output trace information; otherwise, **false**. Default value is **false**.

Syntax

TraceOrders

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          TraceOrders = true;
     }
}
```

Tips
1. See this article for more examples of how to utilize this property.
2. You can override the default output by using OnOrderTrace() in your strategy.

11.6.14.5 Trade

Definition

A Trade is a completed buy/sell or sell/buy transaction. It consists of an entry and exit execution.

Example 1	Example 2
Buy 1 contract at a price of 1000 and sell 1 contract at a price of 1001 is one complete trade.	Buy 2 contracts at a price of 1000 and sell the 1st contract at a price of 1001, then sell the 2nd contract at a price of 1002 are **two** completed trades.

In the second example above, two trade objects are created to represent each individual trade. Each trade object will hold the **same** entry execution for two contracts since this single execution was the opening execution for both individual trades.

Methods and Properties

Commission	A `double` value representing the commission of

	the trade
Entry	Gets an Execution object representing the entry
EntryEfficiency	A `double` value representing the entry efficiency of the trade
Exit	Gets an Execution object representing the exit
ExitEfficiency	A `double` value representing the exit efficiency of the trade
MaeCurrency	A `double` value representing max adverse excursion in currency
MaePercent	A `double` value representing max adverse excursion as a percentage
MaePips	A `double` value representing max adverse excursion in pips
MaePoints	A `double` value representing max adverse excursion in points
MaeTicks	A `double` value representing max adverse excursion in ticks
MfeCurrency	A `double` value representing max favorable excursion in currency
MfePercent	A `double` value representing max favorable excursion as a percentage
MfePips	A `double` value representing max favorable excursion in pips
MfePoints	A `double` value representing max favorable excursion in points
MfeTicks	A `double` value representing max favorable excursion in ticks

ProfitCurrency	A `double` value representing profit quoted in currency.
ProfitPercent	A `double` value representing profit as a percentage
ProfitPips	A `double` value representing profit in pips
ProfitPoints	A `double` value representing profit in points
ProfitTicks	A `double` value representing profit in ticks
Quantity	An `int` value representing the quantity of the trade
TotalEfficiency	A `double` value representing the total efficiency of the trade
TradeNumber	An `int` value representing the trade numbed by the sequence it occurred
ToString()	A `string` representation of the Trade object

Examples

```
protected override void OnBarUpdate()
{
    if (SystemPerformance.RealTimeTrades.Count > 0)
    {
        // Check to make sure there is at least one trade in the
collection
        Trade lastTrade =
SystemPerformance.RealTimeTrades[SystemPerformance.RealTimeTrades.C
ount - 1];

        // Calculate the PnL for the last completed real-time trade
        double lastProfitCurrency = lastTrade.ProfitCurrency;

        // Store the quantity of the last completed real-time trade
        double lastTradeQty = lastTrade.Quantity;

        // Pring the PnL to the NinjaScript Output window
        Print("The last trade's profit in currency is " +
lastProfitCurrency);
        // The trade profit is quantity aware, we can easily print
the profit per traded unit as well
        Print("The last trade's profit in currency per traded unit
is " + (lastProfitCurrency / lastTradeQty));
    }
}
```

11.6.14.5 TradeCollection

Definition

A collection of Trade objects. You can access a trade object by providing an index value. Trades are indexed sequentially meaning the oldest trade taken in a strategy will be at an index value of zero. The most recent trade taken will be at an index value of the total trades in the collection minus 1.

Methods and Properties

TradesCount	An `int` value representing the number of trades in the collection
EvenTrades	Gets a TradeCollection object of even trades
GetTrades()	Gets a TradeCollection object representing a specified position
LosingTrades	Gets a TradeCollection object of losing trades

TradesPerformance	Gets a TradesPerformance object
WinningTrades	Gets a TradeCollection object of winning trades

Examples

Example 1

```
protected override void OnBarUpdate()
{
    // Accesses the first/last trade in the strategy (oldest trade
is at index 0)
    // and prints out the profit as a percentage to the output
window
    if (SystemPerformance.AllTrades.Count > 1)
    {
        Trade lastTrade =
SystemPerformance.AllTrades[SystemPerformance.AllTrades.Count - 1];
        Trade firstTrade = SystemPerformance.AllTrades[0];

        Print("The last trade profit is " +
lastTrade.ProfitPercent);
        Print("The first trade profit is " +
firstTrade.ProfitPercent);
    }
}
```

Example 2

```
protected override void OnBarUpdate()
{
    // Once the strategy has executed 20 trades loop through the
losing trades
    // collection and print out the PnL on only long trades
    if (SystemPerformance.AllTrades.Count == 20)
    {
        Print("There are " +
SystemPerformance.AllTrades.LosingTrades.Count + " losing
trades.");
        foreach (Trade myTrade in
SystemPerformance.AllTrades.LosingTrades)
        {
            if (myTrade.Entry.MarketPosition ==
MarketPosition.Long)
                Print(myTrade.ProfitCurrency);
        }
    }
}
```

11.6.14.57.1 TradesCount

Definition

Indicates the number of trades in the collection.

Property Value

An `int` value that represents the number of trades in the collection.

Syntax

`<TradeCollection>.Count`

Examples

```
protected override void OnBarUpdate()
{
     // Print out the number of long trades
     Print("The strategy has taken " +
SystemPerformance.LongTrades.TradesCount + " long trades.");
}
```

11.6.14.57.2 EvenTrades

Definition

A subcollection of Trade objects consisting of only the non-winning and non-losing trades in a TradeCollection.

Note: You can access a trade object by providing an index value. Trades are indexed sequentially meaning the oldest trade taken in a strategy will be at an index value of zero. The most recent trade taken will be at an index value of the total trades in the collection minus 1.

Methods and Properties

Count	An `int` value representing the number of trades in the collection
GetTrades()	Gets a TradeCollection object representing a specified position
TradesPerformance	Gets a TradesPerformance object

Syntax

```
<TradeCollection>.EvenTrades
```

Examples

```
protected override void OnBarUpdate()
{
     // Accesses the first/last losing trade in the strategy
(oldest trade is at index 0)
     // and prints out the quantity NinjaScript Output window
     if (SystemPerformance.AllTrades.EvenTrades.Count > 1)
     {
          Trade lastTrade =
SystemPerformance.AllTrades.EvenTrades[SystemPerformance.AllTrades.
Count - 1];
          Trade firstTrade =
SystemPerformance.AllTrades.EvenTrades[0];

          Print("The last even trade's quantity was " +
lastTrade.Quantity);
          Print("The first even trade's quantity was " +
firstTrade.Quantity);
     }
}
```

11.6.14.57.3 GetTrades()

Definition
Returns a TradeCollection object representing all trades that make up the specified position.

Method Return Value
A TradeCollection object.

Syntax
```
<TradeCollection>.GetTrades(string instrument, string entrySignalName, int instance)
```

Parameters

instrument	An instrument name such as "MSFT"
entrySignalName	The name of your entry signal
instance	The occurrence to check for (1 is the most recent, 2 is the 2nd most recent position, etc...)

Examples

```
protected override void OnBarUpdate()
{
     TradeCollection myTrades =
SystemPerformance.AllTrades.GetTrades("MSFT", "myEntrySignal", 1);
     Print("The last position was comprised of " + myTrades.Count +
 " trades.");
}
```

11.6.14.57.4 LosingTrades

Definition
A subcollection of Trade objects consisting of only the losing trades in a TradeCollection. You can access a trade object by providing an index value. Trades are indexed sequentially meaning the oldest trade taken in a strategy will be at an index value of zero. The most recent trade taken will be at an index value of the total trades in the collection minus 1.

Methods and Properties

Count	An `int` value representing the number of trades in the collection

GetTrades()	Gets a TradeCollection object representing a specified position
TradesPerformance	Gets a TradesPerformance object

Syntax

```
<TradeCollection>.LosingTrades
```

Examples

```
protected override void OnBarUpdate()
{
     // Accesses the first/last losing trade in the strategy
(oldest trade is at index 0)
     // and prints out the profit as a percentage to the output
window
     if (SystemPerformance.AllTrades.LosingTrades.Count > 1)
     {
          Trade lastTrade =
SystemPerformance.AllTrades.LosingTrades[SystemPerformance.AllTrade
s.Count - 1];
          Trade firstTrade =
SystemPerformance.AllTrades.LosingTrades[0];

          Print("The last losing trade's profit was " +
lastTrade.ProfitPercent);
          Print("The first losing trade's profit was " +
firstTrade.ProfitPercent);
     }
}
```

11.6.14.57.5 TradesPerformance

Definition

Performance profile of a collection of Trade objects.

Methods and Properties

AverageBarsInTrade	A `double` value representing the average number of bars per trade

AverageEntryEfficiency	A `double` value representing the average entry efficiency
AverageExitEfficiency	A `double` value representing the average exit efficiency
AverageTimeInMarket	A `TimeSpan` value representing quantity-weighted average duration of a trade
AverageTotalEfficiency	A `double` value representing the average total efficiency
TotalCommission	A `double` value representing the total commission
Currency	Gets a TradesPerformanceValues object in currency
GrossLoss	A `double` value representing the gross loss
GrossProfit	A `double` value representing the gross profit
LongestFlatPeriod	A `TimeSpan` value representing longest duration of being flat
MaxConsecutiveLoser	An `int` value representing the maximum number of consecutive losses seen
MaxConsecutiveWinner	An `int` value representing the maximum number of consecutive winners seen
MaxTime2Recover	A `TimeSpan` value representing maximum time to recover from a draw down
MonthlyStdDev	A `double` value representing the monthly standard deviation
MonthlyUlcer	A `double` value representing the monthly Ulcer index
NetProfit	A `double` value representing the net profit

Percent	Gets a TradesPerformanceValues object in percent
PerformanceMetrics	An array of custom NinjaScript performance metrics
Pips	Gets a TradesPerformanceValues object in pips
Points	Gets a TradesPerformanceValues object in points
ProfitFactor	A `double` value representing the profit factor
R2	A `double` value representing the R-squared value
RiskFreeReturn	A `double` value representing the risk free return rate
SharpeRatio	A `double` value representing the Sharpe Ratio
SortinoRatio	A `double` value representing the Sortino Ratio
Ticks	Gets a TradesPerformanceValues object in ticks
TotalQuantity	An `int` value representing the total quantity
TotalSlippage	An `double` value representing the total slippage. This is presented in points, I.E. 0.25 for 1 execution on E-mini S&P 500 Futures.
TradesCount	An `int` value representing the trades count
TradesPerDay	An `int` value representing the avg trades per day

Examples

```
protected override void OnBarUpdate()
{
    // Only trade if you have less than 5 consecutive losers in a
row
    if
(SystemPerformance.RealTimeTrades.TradesPerformance.MaxConsecutiveL
oser < 5)
    {
        // Trade logic here
    }
}
```

11.6.14.57.5.1 AverageBarsInTrade

Definition
Returns the average number of bars per trade.

Property Value
A `double` value that represents the average number of bars per trade.

Syntax
```
<TradeCollection>.TradesPerformance.AverageBarsInTrade
```

Examples
```
protected override void OnBarUpdate()
{
     // Print out the average number of bars per trade of all
trades
     Print("Average # bars per trade is: " +
SystemPerformance.AllTrades.TradesPerformance.AverageBarsInTrade);
}
```

11.6.14.57.5.2 AverageEntryEfficiency

Definition
Returns the average entry efficiency.

Property Value
A `double` value that represents the average entry efficiency.

Syntax
```
<TradeCollection>.TradesPerformance.AverageEntryEfficiency
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the average entry efficiency
     Print("Average entry efficiency is: " +
SystemPerformance.AllTrades.TradesPerformance.AverageEntryEfficienc
y);
}
```

11.6.14.57.5.3 AverageExitEfficiency

Definition

Returns the average exit efficiency.

Property Value

A `double` value that represents the average exit efficiency.

Syntax

```
<TradeCollection>.TradesPerformance.AverageExitEfficiency
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the average exit efficiency
     Print("Average exit efficiency is: " +
SystemPerformance.AllTrades.TradesPerformance.AverageExitEfficiency
);
}
```

11.6.14.57.5.4 AverageTimeInMarket

Definition

Returns the average duration of a trade weighted by quantity.

Property Value

A `TimeSpan` value that represents the quantity-weighted average duration of a trade.

Syntax

```
<TradeCollection>.TradesPerformance.AverageTimeInMarket
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the quantity-weighted average duration of all
trades
     Print("Average time in market: " +
SystemPerformance.AllTrades.TradesPerformance.AverageTimeInMarket);
}
```

11.6.14.57.5.5 AverageTotalEfficiency

Definition

Returns the average total efficiency.

Property Value

A `double` value that represents the average total efficiency.

Syntax

```
<TradeCollection>.TradesPerformance.AverageTotalEfficiency
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the average total efficiency
     Print("Average total efficiency is: " +
SystemPerformance.AllTrades.TradesPerformance.AverageTotalEfficienc
y);
}
```

11.6.14.57.5.6 Currency

Definition

Returns a TradesPerformanceValues object in currency.

Property Value

A TradesPerformanceValues object that is represented in currency.

Syntax

```
<TradeCollection>.TradesPerformance.Currency
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the avg. profit of all trades in currency
     Print("Average profit: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.AverageProfi
t);
}
```

11.6.14.57.5.7 GrossLoss

Definition

Returns the gross loss.

Property Value

A `double` value that represents the gross loss.

Syntax

```
<TradeCollection>.TradesPerformance.GrossLoss
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the gross loss of all trades
     Print("Gross loss is: " +
SystemPerformance.AllTrades.TradesPerformance.GrossLoss);
}
```

11.6.14.57.5.8 GrossProfit

Definition

Returns the gross profit.

Property Value

A `double` value that represents the gross profit.

Syntax

```
<TradeCollection>.TradesPerformance.GrossProfit
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the gross profit of all trades
     Print("Gross profit is: " +
SystemPerformance.AllTrades.TradesPerformance.GrossProfit);
}
```

11.6.14.57.5.9 LongestFlatPeriod

Definition

Returns the longest duration of being flat.

Property Value

A `TimeSpan` value that represents the longest duration of being flat.

Syntax

```
<TradeCollection>.TradesPerformance.LongestFlatPeriod
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the longest duration of being flat
     Print("Longest flat period: " +
SystemPerformance.AllTrades.TradesPerformance.LongestFlatPeriod);
}
```

11.6.14.57.5.10 MaxConsecutiveLoser

Definition

Returns the maximum number of consecutive losers seen.

Property Value

An `int` value that represents the maximum number of consecutive losers seen.

Syntax

```
<TradeCollection>.TradesPerformance.MaxConsecutiveLoser
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the max consecutive losers of all trades
     Print("Max # of consecutive losers is: " +
SystemPerformance.AllTrades.TradesPerformance.MaxConsecutiveLoser);
}
```

11.6.14.57.5.11 MaxConsecutiveWinner

Definition

Returns the maximum number of consecutive winners seen.

Property Value

An `int` value that represents the maximum number of consecutive winners seen.

Syntax

```
<TradeCollection>.TradesPerformance.MaxConsecutiveWinner
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the max consecutive winners of all trades
     Print("Max # of consecutive winners is: " +
SystemPerformance.AllTrades.TradesPerformance.MaxConsecutiveWinner)
;
}
```

11.6.14.57.5.12 MaxTimeToRecover

Definition

Returns the maximum time to recover from a draw down.

Property Value

A `TimeSpan` value that represents the maximum time to recover from a draw down.

Syntax

```
<TradeCollection>.TradesPerformance.MaxTimeToRecover
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the maximum time to recover from a draw down
     Print("Max time to recover is: " +
SystemPerformance.AllTrades.TradesPerformance.MaxTimeToRecover);
}
```

11.6.14.57.5.13 MonthlyStdDev

Definition
Returns the monthly standard deviation.

Property Value
A `double` value that represents the monthly standard deviation.

Syntax
`<TradeCollection>.TradesPerformance.MonthlyStdDev`

Examples

```
protected override void OnBarUpdate()
{
     // Print out the monthly standard deviation
     Print("Monthly standard deviation is: " +
SystemPerformance.AllTrades.TradesPerformance.MonthlyStdDev);
}
```

11.6.14.57.5.14 MonthlyUlcer

Definition
Returns the monthly Ulcer index.

Property Value
A `double` value that represents the monthly Ulcer index.

Syntax
`<TradeCollection>.TradesPerformance.MonthlyUlcer`

Examples

```
protected override void OnBarUpdate()
{
     // Print out the monthly Ulcer index
     Print("Monthly Ulcer index is: " +
SystemPerformance.AllTrades.TradesPerformance.MonthlyUlcer);
}
```

11.6.14.57.5.15 NetProfit

Definition
Returns the net profit.

Property Value
A `double` value that represents the net profit.

Syntax
```
<TradeCollection>.TradesPerformance.NetProfit
```

Examples
```
protected override void OnBarUpdate()
{
     // Print out the net profit of all trades
     Print("Net profit is: " +
SystemPerformance.AllTrades.TradesPerformance.NetProfit);
}
```

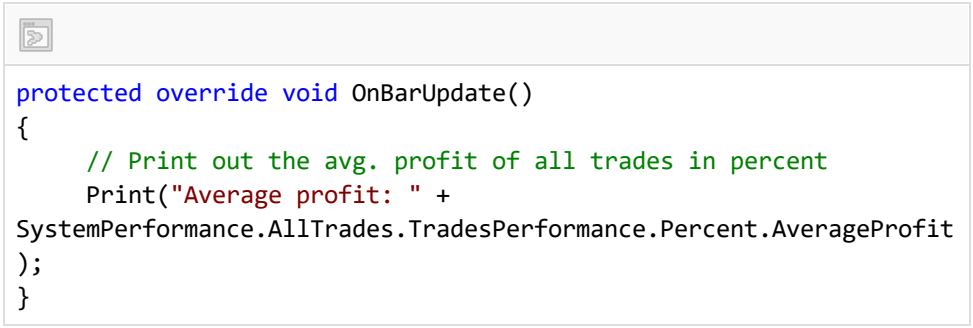
11.6.14.57.5.16 Percent

Definition
Returns a TradesPerformanceValues object in percent.

Property Value
A TradesPerformanceValues object that is represented in percent.

Syntax
```
<TradeCollection>.TradesPerformance.Percent
```

Examples



```
protected override void OnBarUpdate()
{
     // Print out the avg. profit of all trades in percent
     Print("Average profit: " +
SystemPerformance.AllTrades.TradesPerformance.Percent.AverageProfit
);
}
```

11.6.14.57.5.17 PerformanceMetrics

Definition

Returns a collection of custom Performance Metrics. These need to have been enabled in Tools > Options > General to be able to use them.

Syntax

<TradeCollection>.TradesPerformance.PerformanceMetrics

Examples

```
protected override void OnBarUpdate()
{
     // Print out the number of enabled custom Performance Metrics
     Print("Number of Performance Metrics: "
          +
SystemPerformance.AllTrades.TradesPerformance.PerformanceMetrics.Le
ngth);

     // Find a the value of a specific custom Performance Metric
named "MyPerformanceMetric"
     for (int i = 0; i <
SystemPerformance.AllTrades.TradesPerformance.PerformanceMetrics.Le
ngth; i++)
     {
          if
(SystemPerformance.AllTrades.TradesPerformance.PerformanceMetrics[i
] is
               NinjaTrader.NinjaScript.PerformanceMetrics.MyPerform
anceMetric)
          {
               Print((SystemPerformance.AllTrades.TradesPerformance
.PerformanceMetrics[i] as
                    NinjaTrader.NinjaScript.PerformanceMetrics.MyPe
rformanceMetric).Values[0]);
          }
     }
}
```

11.6.14.57.5.18 Pips

Definition

Returns a TradesPerformanceValues object in pips.

Property Value

A TradesPerformanceValues object that is represented in pips.

Syntax

```
<TradeCollection>.TradesPerformance.Pips
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the avg. profit of all trades in pips
     Print("Average profit: " +
SystemPerformance.AllTrades.TradesPerformance.Pips.AverageProfit);
}
```

11.6.14.57.5.19 Points

Definition
Returns a TradesPerformanceValues object in points.

Property Value
A TradesPerformanceValues object that is represented in points.

Syntax
```
<TradeCollection>.TradesPerformance.Points
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the avg. profit of all trades in points
     Print("Average profit: " +
SystemPerformance.AllTrades.TradesPerformance.Points.AverageProfit)
;
}
```

11.6.14.57.5.20 ProfitFactor

Definition
Returns the profit factor.

Property Value
A `double` value that represents the profit factor.

Syntax
```
<TradeCollection>.TradesPerformance.ProfitFactor
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the profit factor of all trades
     Print("Profit factor is: " +
SystemPerformance.AllTrades.TradesPerformance.ProfitFactor);
}
```

11.6.14.57.5.21 RSquared

Definition

Returns the trade performance R-Squared value.

Property Value

A `double` value that represents the R-Squared (R2)

Syntax

```
<TradeCollection>.TradesPerformance.RSquared
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the R2 value of all trades
     Print("R-Squared is: " +
SystemPerformance.AllTrades.TradesPerformance.RSquared);
}
```

11.6.14.57.5.22 RiskFreeReturn

Definition

The risk free return used in calculations of Sharpe and Sortino ratios.

Property Value

A `double` value that represents the risk free return.

Syntax

```
<TradeCollection>.TradesPerformance.RiskFreeReturn
```

Examples

```
protected override void OnBarUpdate()
{
     // Set a 3.5% risk free return
     SystemPerformance.AllTrades.TradesPerformance.RiskFreeReturn =
 0.035;

     // Print out the Sharpe ratio of all trades based on a 3.5%
risk free return
     Print("Sharpe ratio is: " +
SystemPerformance.AllTrades.TradesPerformance.SharpeRatio);
}
```

11.6.14.57.5.23 SharpeRatio

Definition

Returns the Sharpe ratio using a risk free return.

Property Value

A `double` value that represents the Sharpe ratio using a risk free return.

Syntax

```
<TradeCollection>.TradesPerformance.SharpeRatio
```

Examples

```
protected override void OnBarUpdate()
{
     // Set a 0% risk free return
     SystemPerformance.AllTrades.TradesPerformance.RiskFreeReturn =
 0;

     // Print out the Sharpe ratio of all trades based on a zero
risk free return
     Print("Sharpe ratio is: " +
SystemPerformance.AllTrades.TradesPerformance.SharpeRatio);
}
```

11.6.14.57.5.24 SortinoRatio

Definition

Returns the Sortino ratio using a risk free return.

Property Value

A `double` value that represents the Sortino ratio using a risk free return.

Syntax
`<TradeCollection>.TradesPerformance.SortinoRatio`

Examples

```
protected override void OnBarUpdate()
{
     // Set a 0% risk free return
     SystemPerformance.AllTrades.TradesPerformance.RiskFreeReturn =
 0;

     // Print out the Sortino ratio of all trades based on a zero
risk free return
     Print("Sortino ratio is: " +
SystemPerformance.AllTrades.TradesPerformance.SortinoRatio);
}
```

11.6.14.57.5.25 Ticks

Definition
Returns a TradesPerformanceValues object in ticks.

Property Value
A TradesPerformanceValues object that is represented in ticks.

Syntax
`<TradeCollection>.TradesPerformance.Ticks`

Examples

```
protected override void OnBarUpdate()
{
     // Print out the avg. profit of all trades in ticks
     Print("Average profit: " +
SystemPerformance.AllTrades.TradesPerformance.Ticks.AverageProfit);
}
```

11.6.14.57.5.26 TotalCommission

Definition
Returns the total commission.

Property Value
A `double` value that represents the total commission.

Syntax
```
<TradeCollection>.TradesPerformance.TotalCommission
```

Examples


```
protected override void OnBarUpdate()
{
     // Print out the total commission of all trades
     Print("Total commission is: " +
SystemPerformance.AllTrades.TradesPerformance.TotalCommission);
}
```

11.6.14.57.5.27 TotalQuantity

Definition
Returns the total quantity.

Property Value
A `double` value that represents the total quantity.

Syntax
```
<TradeCollection>.TradesPerformance.TotalQuantity
```

Examples


```
protected override void OnBarUpdate()
{
     // Print out the total quantity of all trades
     Print("Total quantity is: " +
SystemPerformance.AllTrades.TradesPerformance.TotalQuantity);
}
```

11.6.14.57.5.28 TotalSlippage

Definition
Returns the total slippage.

Property Value
A `double` value that represents the total slippage. This is presented in points, I.E. 0.25 for 1 execution on E-mini S&P 500 Futures.

Syntax
```
<TradeCollection>.TradesPerformance.TotalSlippage
```

Examples
```
protected override void OnBarUpdate()
{
     // Print out the total slippage of all trades
     Print("Total slippage is: " +
SystemPerformance.AllTrades.TradesPerformance.TotalSlippage);
}
```

11.6.14.57.5.29 TradesCount

Definition
Returns the total # of trades.

Property Value
A `double` value that represents the total # of trades.

Syntax
```
<TradeCollection>.TradesPerformance.TradesCount
```

Examples
```
protected override void OnBarUpdate()
{
     // Print out the total # of trades
     Print("Trades count is: " +
SystemPerformance.AllTrades.TradesPerformance.TradesCount);
}
```

11.6.14.57.5.30 TradesPerDay

Definition
Returns the average number of trades per day.

Property Value
An `int` value that represents the average number of trades per day.

Syntax
```
<TradeCollection>.TradesPerformance.TradesPerDay
```

Examples
```
protected override void OnBarUpdate()
{
     // Print out the average number of trades per day of all
trades
     Print("Average # of trades per day is: " +
SystemPerformance.AllTrades.TradesPerformance.TradesPerDay);
}
```

11.6.14.57.6 WinningTrades

Definition
A subcollection of Trade objects consisting of only the winning trades in a TradeCollection. You can access a trade object by providing an index value. Trades are indexed sequentially meaning the oldest trade taken in a strategy will be at an index value of zero. The most recent trade taken will be at an index value of the total trades in the collection minus 1.

Methods and Properties

Count	An `int` value representing the number of trades in the collection
GetTrades()	Gets a TradeCollection object representing a specified position
TradesPerformance	Gets a TradesPerformance object

Syntax
```
<TradeCollection>.WinningTrades
```

Examples

```
protected override void OnBarUpdate()
{
     // Accesses the first/last winning trade in the strategy
(oldest trade is at index 0)
     // and prints out the profit as a percentage to the output
window
     if (SystemPerformance.AllTrades.WinningTrades.Count > 1)
     {
          Trade lastTrade =
SystemPerformance.AllTrades.WinningTrades[SystemPerformance.AllTrad
es.Count - 1];
          Trade firstTrade =
SystemPerformance.AllTrades.WinningTrades[0];

          Print("The last winning trade's profit was " +
lastTrade.ProfitPercent);
          Print("The first winning trade's profit was " +
firstTrade.ProfitPercent);
     }
}
```

11.6.14.5 TradesPerformanceValues

Definition

Performance values of a collection of Trade objects.

- Currency and Point based calculations are per trade
- Percent based calculations are per traded unit

Methods and Properties

AverageEtd	A `double` value representing avg end trade draw down
AverageMae	A `double` value representing avg maximum adverse excursion
AverageMfe	A `double` value representing avg maximum favorable excursion
AverageProfit	A `double` value representing avg profit
CumProfit	A `double` value representing cumulative profit (percent is compounded)

Drawdown	A `double` value representing draw down
LargestLoser	A `double` value representing largest loss
LargestWinner	A `double` value representing largest gain
ProfitPerMonth	A `double` value representing profit per month always as a percent
StdDev	A `double` value representing standard deviation on a per unit basis
Turnaround	A `double` value representing the turnaround
Ulcer	A `double` value representing the Ulcer value

Examples

```
protected override void OnBarUpdate()
{
     // If the profit on real-time trades is > $1000 stop trading
     if
(SystemPerformance.RealTimeTrades.TradesPerformance.Currency.CumPro
fit > 1000)
          return;
}
```

11.6.14.58.1 AverageEtd

Definition

Returns the average ETD (end trade draw down) of the collection.

Property Value

A `double` value that represents the average ETD of the collection.

Syntax

```
<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.AverageEtd
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the average ETD of all trades in currency
     Print("Average ETD of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.AverageEtd);
}
```

11.6.14.58.2 AverageMae

Definition

Returns the average MAE (max adverse excursion) of the collection.

Property Value

A `double` value that represents the average MAE of the collection.

Syntax

`<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.AverageMae`

Examples

```
protected override void OnBarUpdate()
{
     // Print out the average MAE of all trades in currency
     Print("Average MAE of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.AverageMae);
}
```

11.6.14.58.3 AverageMfe

Definition

Returns the average MFE (max favorable excursion) of the collection.

Property Value

A `double` value that represents the average MFE of the collection.

Syntax

`<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.AverageMfe`

Examples

```
protected override void OnBarUpdate()
{
     // Print out the average MFE of all trades in currency
     Print("Average MFE of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.AverageMfe);
}
```

11.6.14.58.4 AverageProfit

Definition
Returns the average profit of the collection.

Property Value
A `double` value that represents the average profit of the collection.

Syntax
```
<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.AverageProfit
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the average profit of all trades in currency
     Print("Average profit of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.AverageProfi
t);
}
```

11.6.14.58.5 CumProfit

Definition
Returns the cumulative profit of the collection.

Property Value
A `double` value that represents the cumulative profit of the collection.

Syntax
```
<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.CumProfit
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the cumulative profit of all trades in currency
     Print("Average cumulative profit of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.CumProfit);
}
```

11.6.14.58.6 Drawdown

Definition

Returns the draw down of the trade collection.

Property Value

A `double` value that represents the average ETD of the collection.

Syntax

```
<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.Drawdown
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the draw down of all trades in currency
     Print("Draw down of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.Drawdown);
}
```

11.6.14.58.7 LargestLoser

Definition

Returns the largest loss amount of the collection.

Property Value

A `double` value that represents the largest loss amount of the collection.

Syntax

```
<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.LargestLoser
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the largest loss of all trades in currency
     Print("Largest loss of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.LargestLoser
);
}
```

11.6.14.58.8 LargestWinner

Definition

Returns the largest win amount of the collection.

Property Value

A `double` value that represents the largest win amount of the collection.

Syntax

```
<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.LargestWinner
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the largest win of all trades in currency
     Print("Largest win of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.LargestWinne
r);
}
```

11.6.14.58.9 ProfitPerMonth

Definition

Returns the profit per month of the collection. This value is always returned as a percentage.

Property Value

A `double` value that represents the profit per month of the collection as a percentage.

Syntax

```
<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.ProfitPerMonth
```

Examples

```
protected override void OnBarUpdate()
{
     // Print out the profit per month of all trades
     Print("Profit per month of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.ProfitPerMon
th);
}
```

11.6.14.58.10 StdDev

Definition
Returns the standard deviation of the collection on a per unit basis.

Property Value
A `double` value that represents the standard deviation of the collection on a per unit basis.

Syntax
`<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.StdDev`

Examples

```
protected override void OnBarUpdate()
{
     // Print out the standard deviation of all trades
     Print("Standard deviation of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.StdDev);
}
```

11.6.14.58.11 Turnaround

Definition
Returns the amount of turnaround.

Property Value
A `double` value that represents the amount of turnaround.

Syntax
`<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.Turnaround`

Examples

```
protected override void OnBarUpdate()
{
     // Print out the turnaround of all trades
     Print("Turnaround of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.Turnaround);
}
```

11.6.14.58.12 Ulcer

Definition
Returns the Ulcer.

Property Value
A `double` value that represents the Ulcer.

Syntax
`<TradeCollection>.TradesPerformance.<TradesPerformanceValues>.Ulcer`

Examples

```
protected override void OnBarUpdate()
{
     // Print out the Ulcer index of all trades
     Print("Turnaround of all trades is: " +
SystemPerformance.AllTrades.TradesPerformance.Currency.Ulcer);
}
```

11.6.14.5 WaitForOcoClosingBracket

Definition
Determines if the strategy will submit both legs of an OCO bracket before submitting the pair to the broker.

Why would this be needed?
There may be brokers who require that OCO orders are submitted simultaneously in a single API call vs sending them in sequence with an include user defined OCO identifier. For brokers that require OCO orders to be submitted in a single function call, a NinjaScript strategy must wait until it has both legs of the OCO pair generated by SetStopLoss(), SetTrailStop() and SetProfitTarget().

Warning:

- If you only wish to send a stop loss or profit target (but not both) via any of the Set...() methods mentioned above, when WaitForOcoClosingBracket is **enabled**, your exit orders will **NOT** be sent since NinjaTrader needs to wait until it has both orders of the OCO bracket. Disabling WaitForOcoClosingBracket NinjaTrader will immediately submit a stop or profit target order, whichever is submitted first.
- This property only effects Set...() methods and does not affect other order methods (i.e. ExitLongLimit() / SubmitOrderUnmanaged()).

Property Value

This property returns **true** if the strategy will wait for both legs of an OCO bracket to be called in a strategy before submitting the order pair to the broker; otherwise, **false**. Default value is set to **true**.

Note: Current affected brokers: **TD AMERITRADE**. For any other broker, this property has no effect.

Syntax

```
WaitForOcoClosingBracket
```

Examples

```
protected override void OnStateChange()
{
    if (State == State.Configure)
    {
        WaitForOcoClosingBracket = false;
    }
}
```

11.6.15 SuperDOM Column

Custom **SuperDOM Columns** can be used to add additional functionality to the SuperDOM window. The methods and properties covered in this section are unique to custom SuperDOM column development.

Tip: The system **SuperDOM Columns** which ship with NinjaTrader are open source and you can review their implementation from the NinjaScript Editor **SuperDOMColumn**

folder, or by using the text editor of your choice by reviewing the source code located in *Documents\NinjaTrader 8\bin\Custom\SuperDomColumns*

In this section

Market Depth	Provides Level 2 information for a **SuperDOMColumn**.
OnMarketData()	Called and guaranteed to be in the correct sequence for every change in level one market data for the underlying instrument. The **OnMarketData()** method updates can include but is not limited to the bid, ask, last price and volume.
OnOrderUpdate()	Called every time an order changes state. An order will change state when a change in order quantity, price or state (e.g. working to filled) occurs.
OnPositionUpdate()	Called every time a position changes state.
OnPropertyChanged()	This method should be used any time you wish to repaint the column instead of calling OnRender() directly.
OnRender()	Used to draw custom content to the **SuperDOM Column**, such as a Grid.
OnRestoreValues()	Called when the column is restored (e.g. from a workspace).

11.6.15.1 MarketDepth

Definition

Provides Level 2 information for a **SuperDOMColumn**

Note: In order to ensure you are using the same exact **MarketDepth** subscription that the SuperDOM's main price ladder is using, it is required that you create your own

MarketDepth handler. The NinjaScript Code Wizard was designed to automatically complete this process for you, and an example is outlined at the bottom of this page

Property Value

SuperDom.MarketDepth	A collection of MarketDepthRows
SuperDom.MarketDepth.Asks	A collection of orders on the ask side of the market
SuperDom.MarketDepth.Bids	A collection of orders on the bid side of the market
SuperDom.MarketDepth.Instrument	The instrument which is being updated

Syntax

```
SuperDom.MarketDepth
SuperDom.MarketDepth.Asks[int idx];
SuperDom.MarketDepth.Bids[int idx];
SuperDom.MarketDepth.Instrument
```

Examples

```
protected override void OnStateChange()
{
   if (State == State.Active)
   {
      // subscribe to the same market depth events as the primary
SuperDOM Price Ladder
      if (SuperDom.MarketDepth != null)
      {
         WeakEventManager<Data.MarketDepth<LadderRow>,
Data.MarketDepthEventArgs>.AddHandler(SuperDom.MarketDepth,
"Update", OnMarketDepthUpdate);
      }
   }
   else if (State == State.Terminated)
   {
      // unsubscribe to the same market depth events as the primary
SuperDOM Price Ladder
      if (SuperDom == null) return;

      if (SuperDom.MarketDepth != null)
      {
         WeakEventManager<Data.MarketDepth<LadderRow>,
Data.MarketDepthEventArgs>.RemoveHandler(SuperDom.MarketDepth,
"Update", OnMarketDepthUpdate);
      }
   }
}

// custom market depth handler
private void OnMarketDepthUpdate(object sender,
Data.MarketDepthEventArgs e)
{
   // Print some data to the Output window
   if (e.MarketDataType == MarketDataType.Ask && e.Operation ==
Operation.Update)
      Print(string.Format("The most recent ask change is {0} {1}",
e.Price, e.Volume));
}
```

11.6.15.2 OnMarketData()

Definition

Called and guaranteed to be in the correct sequence for every change in level one market data for the underlying instrument. The **OnMarketData()** method updates can include but is not limited to the bid, ask, last price and volume.

Method Return Value

This method does not return a value.

Syntax

```
protected override void OnMarketData(MarketDataEventArgs marketDataUpdate)
{

}
```

Parameters

marketDataUpdate	A MarketDataEventArgs representing the change in market data

Examples

```
protected override void OnMarketData(MarketDataEventArgs
marketDataUpdate)
{
   if (marketDataUpdate.MarketDataType == Data.MarketDataType.Last)
   {
      // Do something
   }
}
```

11.6.15.3 OnOrderUpdate()

Definition

Called every time an order changes state. An order will change state when a change in order quantity, price or state (e.g. working to filled) occurs.

> **Note**: The **OnOrderUpdate()** method is called on **ALL** order updates (e.g., any account and instrument combination) and **NOT** just the specific items which are selected in the **SuperDOM.**

Method Return Value

This method does not return a value.

Syntax

```
protected override void OnOrderUpdate(OrderEventArgs orderUpdate)
{

}
```

Method Parameters

orderUpdate	An OrderEventArgs representing the change in order state

Examples

```
protected override void OnOrderUpdate(OrderEventArgs orderUpdate)
{
   // Do not take action if the order update does not come from the
selected SuperDOM instrument/account
   if (orderUpdate.Order.Instrument != SuperDom.Instrument ||
orderUpdate.Order.Account != SuperDom.Account)
      return;

   // Do something
}
```

11.6.15.4 OnPositionUpdate()

Definition

Called every time a position changes state.

> **Note**: The **OnPositionUpdate()** method is called on **ALL** position updates (e.g., any account and instrument combination) and **NOT** just the specific items which are selected in the **SuperDOM.**

Method Return Value

This method does not return a value.

Syntax

```
protected override void OnPositionUpdate(PositionEventArgs positionUpdate)
{

}
```

Method Parameters

positionUpdate	A PositionEventArgs representing the change in position

Examples

```
protected override void OnPositionUpdate(PositionEventArgs
positionUpdate)
{
   // Do not take action if the position update does not come from
the selected SuperDOM instrument/account
   if (positionUpdate.Position.Instrument != SuperDom.Instrument
      || positionUpdate.Position.Account != SuperDom.Account)
      return;

   // Do something
}
```

11.6.15.5 OnPropertyChanged()

Definition

This method should be used any time you wish to repaint the column instead of calling OnRender() directly.

Method Return Value

This method does not return a value

Syntax

`OnPropertyChanged()`

Parameters

This method does not require any parameters

Examples

```
// Repaint the SuperDOM column
OnPropertyChanged();
```

11.6.15.6 OnRender()

Definition

Used to draw custom content to the **SuperDOM Column**, such as a Grid.

This method is called during the following conditions:

- The SuperDOM is centered (either automatically or when the user presses the Center button)
- The SuperDOM is scrolled

- All accounts are disconnected
- A simulation account is reset
- A position is updated
- The user changes the SuperDOM's properties through the Properties menu
- The SuperDOM first loads (e.g. restoring from a workspace)
- The user changes the PnL display unit by clicking on the Position display
- The height/width of the SuperDOM window changes
- A user resizes the content area by dragging the splitter between price ladder and the columns

Note: While similar to a Chart Indicator's OnRender() method, the **SuperDOM Column** uses WPF Drawing Context class, rather than the **SharpDX** library used for chart rendering. Concepts between these two methods are guaranteed to be different.

Method Return Value

This method does not return a value.

Syntax

You must override the method in your **SuperDOM column** with the following syntax:

```
protected override void OnRender(DrawingContext dc, double renderWidth)
{

}
```

Method Parameters

dc	The drawing context for the column
renderWidth	The rendering width for the column

Tip: In order to force **OnRender()** to be called under a specific condition, call the OnPropertyChanged() method which will force the entire column to repaint. This approach should be used instead of calling **OnRender()** directly.

Examples

```
protected override void OnRender(DrawingContext dc, double
renderWidth)
{
   // Rendering logic for our column
}
```

11.6.15.7 OnRestoreValues()

Definition

Called when the column is restored (e.g. from a workspace). All public properties in a **SuperDOM Column** are saved to the workspace upon closing and selecting save. You may choose to do something explicit with a certain property when the **OnRestoreValues()** method is called.

Method Return Value

This method does not return a value

Syntax

You may override the method in your SuperDOM column with the following syntax:

```
public override void OnRestoreValues()
{

}
```

Parameters

This method does not require any parameters

Examples

```
public override void OnRestoreValues()
{
    // Do something with the restored values. Can also trigger a
repaint via OnPropertyChanged()
}
```

11.7 SharpDX SDK Reference

Disclaimer: The **SharpDX SDK Reference** section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to

the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX / Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

SharpDX is an open-source managed .NET wrapper of the DirectX API allowing the development of high performance game, 2D and 3D graphics rendering as well as realtime sound application.

Tip: The concepts discussed in this section only apply to NinjaScript objects which use the Chart's OnRender() method. For code examples which demonstrate usage, please refer to the Using SharpDX for Custom Chart Rendering educational resource. You may also use view the source code of various ChartStyles, DrawingTools, and Indicators which come pre-installed in the **NinjaTrader.Custom** project (Documents\NinjaTrader 8\bin\Custom).

In this section

SharpDX	The SharpDX namespace contains fundamental classes used by SharpDX.
SharpDX.Direct2D1	The SharpDX.Direct2D1 namespace provides a managed Direct2D API. Direct2D is a hardware-accelerated, immediate-mode, 2-D graphics API that provides high performance and high-quality rendering for 2-D geometry, bitmaps, and text.
SharpDX.DirectWrite	The SharpDX.DirectWrite namespace provides a managed DirectWrite API. DirectWrite supports high-quality text rendering, resolution-independent outline fonts, and full Unicode text and layouts.

11.7.1 SharpDX

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

The **SharpDX** namespace contains fundamental classes used by SharpDX.

In this section

Color	Represents a 32-bit color (4 bytes) in the form of RGBA (in byte order: R, G, B, A).
Color3	Represents a color in the form of rgb.
Color4	Represents a color in the form of rgba.
DisposeBase	Base class for a System.IDisposable class.
Matrix3x2	Represents a 3x2 mathematical matrix.
RectangleF	Structure using similar layout as System.Drawing.RectangleF
Size2F	Structure using the same layout as System.Drawing.SizeF
Vector2	Represents a two dimensional mathematical vector.

11.7.1.1 Color

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Represents a 32-bit color (4 bytes) in the form of RGBA (in byte order: R, G, B, A).

Notes:

1. The color of each pixel is represented as a 32-bit number: 8 bits each for alpha, red, green, and blue (ARGB). Each of the four components is a number from 0 through 255, with 0 representing no intensity and 255 representing full intensity. The alpha component specifies the transparency of the color: 0 is fully transparent, and 255 is fully opaque. To determine the alpha, red, green, or blue component of a color, use the A, R, G, or B property, respectively.
2. Named colors are represented by using the properties of the **Color** structure. Please see the table of *Static Named Colors* below

Syntax

```
struct Color
```

Constructors

`new Color()`	Initializes a new instance of the **Color** struct
`new Color(float red, float green, float blue)`	Initializes a new instance of the **Color** struct using float values
`new Color(float red, float green, float blue, float alpha)`	Initializes a new instance of the **Color** struct using float values with alpha transparency

`new Color(int red, int green, int blue)`	Initializes a new instance of the **Color** struct using int values
`new Color(int red, int green, int blue, int alpha)`	Initializes a new instance of the **Color** struct using int values with alpha transparency
`new Color(byte red, byte green, byte blue)`	Initializes a new instance of the **Color** struct using byte values
`new Color(byte red, byte green, byte blue, byte alpha)`	Initializes a new instance of the **Color** struct using byte values with alpha transparency

Methods and Properties

R	The red component of the color
G	The green component of the color
B	The blue component of the color
A	The alpha component of the color
ToColor3()	Converts the color into a three component color
ToColor4()	Converts the color into a four component color

Static Named Colors

Colors by name

SharpDX.Color.Zero	Zero color
SharpDX.Color.Transparent	Transparent color

SharpDX.Color.AliceBlue	AliceBlue color
SharpDX.Color.AntiqueWhite	AntiqueWhite color
SharpDX.Color.Aqua	Aqua color
SharpDX.Color.Aquamarine	Aquamarine color
SharpDX.Color.Azure	Azure color
SharpDX.Color.Beige	Beige color
SharpDX.Color.Bisque	Bisque color
SharpDX.Color.Black	Black color
SharpDX.Color.BlanchedAlmond	BlanchedAlmond color
SharpDX.Color.Blue	Blue color
SharpDX.Color.BlueViolet	BlueViolet color
SharpDX.Color.Brown	Brown color
SharpDX.Color.BurlyWood	BurlyWood color
SharpDX.Color.CadetBlue	CadetBlue color
SharpDX.Color.Chartreuse	Chartreuse color
SharpDX.Color.Chocolate	Chocolate color
SharpDX.Color.Coral	Coral color

SharpDX.Color.CornflowerBlue	CornflowerBlue color
SharpDX.Color.Cornsilk	Cornsilk color
SharpDX.Color.Crimson	Crimson color
SharpDX.Color.Cyan	Cyan color
SharpDX.Color.DarkBlue	DarkBlue color
SharpDX.Color.DarkCyan	DarkCyan color
SharpDX.Color.DarkGoldenrod	DarkGoldenrod color
SharpDX.Color.DarkGray	DarkGray color
SharpDX.Color.DarkGreen	DarkGreen color
SharpDX.Color.DarkKhaki	DarkKhaki color
SharpDX.Color.DarkMagenta	DarkMagenta color
SharpDX.Color.DarkOliveGreen	DarkOliveGreen color
SharpDX.Color.DarkOrange	DarkOrange color
SharpDX.Color.DarkOrchid	DarkOrchid color
SharpDX.Color.DarkRed	DarkRed color
SharpDX.Color.DarkSalmon	DarkSalmon color

SharpDX.Color.DarkSeaGreen	DarkSeaGreen color
SharpDX.Color.DarkSlateBlue	DarkSlateBlue color
SharpDX.Color.DarkSlateGray	DarkSlateGray color
SharpDX.Color.DarkTurquoise	DarkTurquoise color
SharpDX.Color.DarkViolet	DarkViolet color
SharpDX.Color.DeepPink	DeepPink color
SharpDX.Color.DeepSkyBlue	DeepSkyBlue color
SharpDX.Color.DimGray	DimGray color
SharpDX.Color.DodgerBlue	DodgerBlue color
SharpDX.Color.Firebrick	Firebrick color
SharpDX.Color.FloralWhite	FloralWhite color
SharpDX.Color.ForestGreen	ForestGreen color
SharpDX.Color.Fuchsia	Fuchsia color
SharpDX.Color.Gainsboro	Gainsboro color
SharpDX.Color.GhostWhite	GhostWhite color

SharpDX.Color.Gold	Gold color
SharpDX.Color.Goldenrod	Goldenrod color
SharpDX.Color.Gray	Gray color
SharpDX.Color.Green	Green color
SharpDX.Color.GreenYellow	GreenYellow color
SharpDX.Color.Honeydew	Honeydew color
SharpDX.Color.HotPink	HotPink color
SharpDX.Color.IndianRed	IndianRed color
SharpDX.Color.Indigo	Indigo color
SharpDX.Color.Ivory	Ivory color
SharpDX.Color.Khaki	Khaki color
SharpDX.Color.Lavender	Lavender color
SharpDX.Color.LavenderBlush	LavenderBlush color
SharpDX.Color.LawnGreen	LawnGreen color
LemonChiffon	LemonChiffon color
SharpDX.Color.LightBlue	LightBlue color
SharpDX.Color.LightCoral	LightCoral color

SharpDX.Color.LightCyan	LightCyan color
SharpDX.Color.LightGoldenrodYellow	LightGoldenrodYellow color
SharpDX.Color.LightGray	LightGray color
SharpDX.Color.LightGreen	LightGreen color
SharpDX.Color.LightPink	LightPink color
SharpDX.Color.LightSalmon	LightSalmon color
SharpDX.Color.LightSeaGreen	LightSeaGreen color
SharpDX.Color.LightSkyBlue	LightSkyBlue color
SharpDX.Color.LightSlateGray	LightSlateGray color
SharpDX.Color.LightSteelBlue	LightSteelBlue color
SharpDX.Color.LightYellow	LightYellow color
SharpDX.Color.Lime	Lime color
SharpDX.Color.LimeGreen	LimeGreen color
SharpDX.Color.Linen	Linen color
SharpDX.Color.Magenta	Magenta color
SharpDX.Color.Maroon	Maroon color

SharpDX.Color.MediumAquamarine	MediumAquamarine color
SharpDX.Color.MediumBlue	MediumBlue color
SharpDX.Color.MediumOrchid	MediumOrchid color
SharpDX.Color.MediumPurple	MediumPurple color
SharpDX.Color.MediumSeaGreen	MediumSeaGreen color
SharpDX.Color.MediumSlateBlue	MediumSlateBlue color
SharpDX.Color.MediumSpringGreen	MediumSpringGreen color
SharpDX.Color.MediumTurquoise	MediumTurquoise color
SharpDX.Color.MediumVioletRed	MediumVioletRed color
SharpDX.Color.MidnightBlue	MidnightBlue color
SharpDX.Color.MintCream	MintCream color
SharpDX.Color.MistyRose	MistyRose color
SharpDX.Color.Moccasin	Moccasin color
SharpDX.Color.NavajoWhite	NavajoWhite color

SharpDX.Color.Navy	Navy color
SharpDX.Color.OldLace	OldLace color
SharpDX.Color.Olive	Olive color
SharpDX.Color.OliveDrab	OliveDrab color
SharpDX.Color.Orange	Orange color
SharpDX.Color.OrangeRed	OrangeRed color
SharpDX.Color.Orchid	Orchid color
SharpDX.Color.PaleGoldenrod	PaleGoldenrod color
SharpDX.Color.PaleGreen	PaleGreen color
SharpDX.Color.PaleTurquoise	PaleTurquoise color
SharpDX.Color.PaleVioletRed	PaleVioletRed color
SharpDX.Color.PapayaWhip	PapayaWhip color
SharpDX.Color.PeachPuff	PeachPuff color
SharpDX.Color.Peru	Peru color
SharpDX.Color.Pink	Pink color
SharpDX.Color.Plum	Plum color

SharpDX.Color.PowderBlue	PowderBlue color
SharpDX.Color.Purple	Purple color
SharpDX.Color.Red	Red color
SharpDX.Color.RosyBrown	RosyBrown color
SharpDX.Color.RoyalBlue	RoyalBlue color
SharpDX.Color.SaddleBrown	SaddleBrown color
SharpDX.Color.Salmon	Salmon color
SharpDX.Color.SandyBrown	SandyBrown color
SharpDX.Color.SeaGreen	SeaGreen color
SharpDX.Color.SeaShell	SeaShell color
SharpDX.Color.Sienna	Sienna color
SharpDX.Color.Silver	Silver color
SharpDX.Color.SkyBlue	SkyBlue color
SharpDX.Color.SlateBlue	SlateBlue color
SharpDX.Color.SlateGray	SlateGray color
SharpDX.Color.Snow	Snow color
SharpDX.Color.SpringGreen	SpringGreen color

SharpDX.Color.SteelBlue	SteelBlue color
SharpDX.Color.Tan	Tan color
SharpDX.Color.Teal	Teal color
SharpDX.Color.Thistle	Thistle color
SharpDX.Color.Tomato	Tomato color
SharpDX.Color.Turquoise	Turquoise color
SharpDX.Color.Violet	Violet color
SharpDX.Color.Wheat	Wheat color
SharpDX.Color.White	White color
SharpDX.Color.WhiteSmoke	WhiteSmoke color
SharpDX.Color.Yellow	Yellow color
SharpDX.Color.YellowGreen	YellowGreen color

11.7.1.2 Color3

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Represents a color in the form of rgb.

Syntax
```
struct Color3
```

Constructors

`new Color3()`	Initializes a new instance of the **Color3** struct.
`new Color3(float red, float green, float blue)`	Initializes a new instance of the **Color3** struct using float values for red, green, blue

Properties

Black	The Black color (0, 0, 0)
White	The White color (1, 1, 1)
Red	The red component of the color
Green	The green component of the color
Blue	The green component of the color

11.7.1.3 Color4

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Represents a color in the form of rgba.

Syntax

```
struct Color4
```

Constructor

`Color4()`	Initializes a new instance of the **Color4** struct
`Color4(Color3 color)`	Initializes a new instance of the **Color4** struct using a SharpDX.Color3 struct
`Color4(Color3 color, float alpha)`	Initializes a new instance of the **Color4** struct using a SharpDX.Color3 struct with a float for alpha values
`Color4(float red, float green, float blue, float alpha)`	Initializes a new instance of the **Color4** struct using float values for red, green, blue

Properties

Black	The Black color (0, 0, 0, 1)
White	The White color (1, 1, 1, 1)
Red	The red component of the color
Green	The green component of the color
Blue	The green component of the color
Alpha	The alpha component of the color

11.7.1.4 DisposeBase

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX**

SDK. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Base class for a System.IDisposable class.

Tip: For NinjaScript development purposes, the following **documented SharpDX** objects require Dispose() after they are used:

Brush, GeometrySink, GradientStopCollection, LinearGradientBrush, PathGeometry, RadialGradientBrush, SolidColorBrush, StrokeStyle, TextFormat, TextLayout

There are other **undocumented SharpDX** objects which are **NOT** included in this reference. Please be careful to dispose of *any* object (**SharpDX** or otherwise) which implements the **IDisposeable** interface - NinjaTrader is **NOT** guaranteed to dispose of these objects for you!

Methods and Properties

IsDisposed	Gets a value indicating whether this instance is disposed.
Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Implements IDisposable.Dispose())

11.7.1.4.1 Dispose()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some

of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Implements IDisposable.Dispose())

Tip: For NinjaScript development purposes, the following **documented SharpDX** objects require **Dispose()** after they are used:

Brush, GeometrySink, GradientStopCollection, LinearGradientBrush, PathGeometry, RadialGradientBrush, SolidColorBrush, StrokeStyle, TextFormat, TextLayout

There are other **undocumented SharpDX** objects which are **NOT** included in this reference. Please be careful to dispose of *any* object (**SharpDX** or otherwise) which implements the **IDisposeable** interface - NinjaTrader is **NOT** guaranteed to dispose of these objects for you!

Method return value

This method does not return a value

Syntax

```
<DisposeBaseObject>.Dispose()
```

11.7.1.4.2 IsDisposed

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the

DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Gets a value indicating whether this instance is disposed.

Property Value
A `bool` which is **true** if this instance is disposed; otherwise, **false**.

Syntax
```
<DisposeBaseObject>.IsDisposed
```

11.7.1.5 Matrix3x2

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Represents a 3x2 mathematical matrix.

Tip: For more information on Direct2D transforms, please see the MSDN Direct2D Transforms Overview

Syntax
```
struct Matrix3x2
```

Constructors

`new Matrix3x2()`	Initializes a new instance of the **Matrix3x2** struct

Methods and Properties

Identity	Gets the identity matrix.
M11	A `float` for the first element of the first row.
M12	A `float` for the second element of the first row.
M21	A `float` for the first element of the second row.
M22	A `float` for the second element of the second row.
M31	A `float` for the first element of the third row.
M32	A `float` for the second element of the third row.
TranslationVector	A SharpDX.Vector2 for the translation component of this matrix.
Matrix3x2.Rotation(float angle)	Creates a matrix that rotates.
Matrix3x2.Scaling(float scale)	Creates a matrix that uniformally scales along all three axis.
Translation(Vector2 value)	Creates a translation matrix using the specified offsets.

11.7.1.6 RectangleF

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this

reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Structure using similar layout as System.Drawing.RectangleF.

Note: This structure is slightly different from System.Drawing.RectangleF as It is internally storing Left,Top,Right,Bottom instead of Left,Top,Width,Height. Although automatic casting from a to System.Drawing.Rectangle is provided.

Syntax

```
struct RectangleF
```

Constructors

`new RectangleF()`	Initializes a new instance of the **RectangleF** struct.
`new RectangleF(float x, float y, float width, float height)`	Initializes a new instance of the **RectangleF** with specific dimensions

Properties

Bottom	Gets or sets the bottom.
Height	Gets or sets the height.
Left	Gets or sets the left.
Right	Gets or sets the right.
Top	Gets or sets the top.
Width	Gets or sets the width.

X	Gets or sets the left position.
Y	Gets or sets the top position.

11.7.1.7 Size2F

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Structure using the same layout as System.Drawing.SizeF

Syntax
```
struct Size2F
```

Constructors

`new Size2F()`	Initializes a new instance of the **SizeF** struct
`new Size2F(float width, float height)`	Initializes a new instance of the **SizeF** struct from the specified dimensions.

Properties

Height	Gets or sets the vertical component of this SizeF structure.
Width	Gets or sets the horizontal component of this SizeF

	structure.

11.7.1.8 Vector2

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX / Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Represents a two dimensional mathematical vector.

Syntax
```
struct Vector2
```

Tip: For NinjaScript Development Purposes, you can use the NinjaTrader.Gui.DxExtensions.ToVector2() helper method to convert a **System.Windows.Point** structure to a **SharpDX.Vector2** used for SharpDX rendering.

Constructors

`Vector2()`	Initializes a new instance of the **Vector2** struct.
`Vector2(float x, float y)`	Initializes a new instance of the **Vector2** struct using float values for x and y components

Properties

X	A `float` for the X component of the vector.

Y	A `float` for the Y component of the vector.

11.7.2 SharpDX.Direct2D1

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

The **SharpDX.Direct2D1** namespace provides a managed Direct2D API. Direct2D is a hardware-accelerated, immediate-mode, 2-D graphics API that provides high performance and high-quality rendering for 2-D geometry, bitmaps, and text.
(See also unmanaged API documentation)

In this section

AntialiasMode	Specifies how the edges of nontext primitives are rendered.
ArcSegment	Describes an elliptical arc between two points.
ArcSize	Specifies whether an arc should be greater than 180 degrees.
Brush	Defines an object that paints an area. Interfaces that derive from Brush describe how the area is painted.
BrushProperties	Describes the opacity and transformation of a brush.

DrawTextOptions	Specifies whether text snapping is suppressed or clipping to the layout rectangle is enabled. This enumeration allows a bitwise combination of its member values.
Ellipse	Contains the center point, x-radius, and y-radius of an ellipse.
FigureBegin	Indicates whether a specific GeometrySink figure is filled or hollow.
FigureEnd	Indicates whether a specific GeometrySink figure is open or closed.
FillMode	Specifies how the intersecting areas of geometries or figures are combined to form the area of the composite geometry.
GeometrySink	Describes a geometric path that can contain lines, arcs, cubic Bezier curves, and quadratic Bezier curves.
MeasuringMode	Indicates the measuring method used for text layout.
PathGeometry	Represents a complex shape that may be composed of arcs, curves, and lines.
RenderTarget	Represents an object that can receive drawing commands.
SolidColorBrush	Paints an area with a solid color.
StrokeStyle	Describes the caps, miter limit, line join, and dash information for

	a stroke.
SweepDirection	Defines the direction that an elliptical arc is drawn.

11.7.2.1 AntialiasMode

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Specifies how the edges of nontext primitives are rendered.
(See also unmanaged API documentation)

Syntax

```
enum AntialiasMode
```

Enumerators

PerPrimitive	Edges are antialiased using the Direct2D per-primitive method of high-quality antialiasing.
Aliased	Objects are aliased in most cases. Objects are antialiased only when they are drawn to a render target created by the CreateDxgiSurfaceRenderTarget method and Direct3D multisampling has been enabled on the backing DirectX Graphics Infrastructure (DXGI) surface.

11.7.2.2 ArcSegment

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Describes an elliptical arc between two points.
(See also unmanaged API documentation)

Syntax

```
struct ArcSegment
```

Properties

Point	The end point of the arc.
Size	The x-radius and y-radius of the arc.
RotationAngle	A value that specifies how many degrees in the clockwise direction the ellipse is rotated relative to the current coordinate system.
SweepDirection	A SweepDirection enum value that specifies whether the arc sweep is clockwise or counterclockwise.
ArcSize	A value that specifies whether the given arc is larger than 180 degrees.

11.7.2.3 ArcSize

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Specifies whether an arc should be greater than 180 degrees.
(See also unmanaged API documentation)

Syntax

```
enum ArcSize
```

Enumerators

Small	An arc's sweep should be 180 degrees or less.
Large	An arc's sweep should be 180 degrees or greater.

11.7.2.4 Brush

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Defines an object that paints an area. Interfaces that derive from **Brush** describe how the area is painted.
(See also unmanaged API documentation)

Notes:

1. An **Brush** is a device-dependent resource: your application should create brushes after it initializes the render target with which the brush will be used, and recreate the brush whenever the render target needs recreated. Please see the MSDN Direct2D Resources Overview for more information.
2. **Brush** space in **Direct2D** is specified differently than in XPS and Windows Presentation Foundation (WPF). In **Direct2D**, brush space is not relative to the object being drawn, but rather is the current coordinate system of the render target, transformed by the brush transform, if present. To paint an object as it would be painted by a **WPF brush**, you must translate the brush space origin to the upper-left corner of the object's bounding box, and then scale the brush space so that the base tile fills the bounding box of the object.
3. For convenience, **Direct2D** provides the BrushProperties function for creating new a Brush.

Syntax

```
class Brush
```

Tips:

1. For NinjaScript Development purposes, you can use the NinjaTrader.Gui.DxExtensions.ToDxBrush() helper method to convert a **System.Windows.Media.Brush** to a **SharpDX.Direct2D1.Brush** s
2. General information on **Direct2D brushes** can be found on the MSDN Direct2D Brushes Overview

Methods and Properties

Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)
IsDisposed	Gets a value indicating whether this instance is disposed.

	(Inherited from SharpDX.DisposeBase.)
Opacity	Gets or sets the degree of opacity of this brush.
Transform	Gets or sets the transform applied to this brush.

11.7.2.4.1 Opacity

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets or sets the degree of opacity of this brush.
(See also unmanaged API documentation)

Property Value

A `float` value between zero and 1 that indicates the opacity of the brush. This value is a constant multiplier that linearly scales the alpha value of all pixels filled by the brush. The opacity values are clamped in the range 0–1 before they are multipled together.

Syntax

```
<SolidColorBrush>.Opacity
```

11.7.2.4.2 Transform

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to

the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets or sets the transform applied to this brush.
(See also unmanaged API documentation)

Note: When the brush transform is the identity matrix, the brush appears in the same coordinate space as the render target in which it is drawn.

Property Value

A Matrix3x2 transform applied to this brush.

Syntax

```
<Brush>.Transform
```

11.7.2.5 BrushProperties

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Describes the opacity and transformation of a brush.
(See also unmanaged API documentation)

Syntax

```
struct BrushProperties
```

Constructors

`new BrushProperties()`	Initializes a new instance of the **BrushProperties** structure

Properties

Opacity	A value between 0.0f and 1.0f, inclusive, that specifies the degree of opacity of the brush.
Transform	The transformation that is applied to the brush.

11.7.2.6 CapStyle

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Describes the shape at the end of a line or segment.
(See also unmanaged API documentation)

Syntax

```
enum CapStyle
```

Enumerators

Flat	A cap that does not extend past the last point of the line. Comparable to cap used for objects other than lines.

Square	Half of a square that has a length equal to the line thickness.
Round	A semicircle that has a diameter equal to the line thickness.
Triangle	An isosceles right triangle whose hypotenuse is equal in length to the thickness of the line.

11.7.2.7 DrawTextOptions

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Specifies whether text snapping is suppressed or clipping to the layout rectangle is enabled. This enumeration allows a bitwise combination of its member values.
(See also unmanaged API documentation)

Syntax

```
enum DrawTextOptions
```

Enumerators

NoSnap	Text is not vertically snapped to pixel boundaries. This setting is recommended for text that is being animated.
Clip	Text is clipped to the layout rectangle.

None	Text is vertically snapped to pixel boundaries and is not clipped to the layout rectangle.

11.7.2.8 Ellipse

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Contains the center point, x-radius, and y-radius of an ellipse.
(See also unmanaged API documentation)

Syntax

```
struct Ellipse
```

Constructors

`new Ellipse()`	Initializes a new instance of the **Ellipse** struct
`new Ellipse(Vector2 center, float radiusX, float radiusY)`	Initializes a new instance of the **Ellipse** struct with specific dimensions

Properties

Point	A SharpDX.Vector for the center point of the ellipse
RadiusX	A `float` for the X-radius of the ellipse

RadiusY	A `float` for the Y-radius of the ellipse

11.7.2.9 FigureBegin

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Indicates whether a specific GeometrySink figure is filled or hollow.
(See also unmanaged API documentation)

Syntax

```
enum FigureBegin
```

Enumerators

Filled	Indicates the figure will be filled by the FillGeometry() method
Hollow	Indicates the figure will not be filled by the FillGeometry() method and will only consist of an outline. Moreover, the bounds of a hollow figure are zero. FigureBegin.Hollow should be used for stroking, or for other geometry operations.

11.7.2.10 FigureEnd

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this

section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Indicates whether a specific GeometrySink figure is open or closed
(See also unmanaged API documentation)

Syntax

```
enum FigureEnd
```

Enumerators

Open	The figure is open.
Closed	The figure is closed.

11.7.2.11 FillMode

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Specifies how the intersecting areas of geometries or figures are combined to form the area of the composite geometry.
(See also unmanaged API documentation)

Notes:
- Use the **FillMode** enumeration when creating an when modifying the fill mode of a GeometrySink with the SetFillMode() method.
- Direct2D fills the interior of a path by using one of the two fill modes specified by this enumeration: Alternate (alternate) or Winding (winding). Because the modes determine how to fill the interior of a closed shape, all shapes are treated as closed when they are filled. If there is a gap in a segment in a shape, draw an imaginary line to close it.

Syntax

```
enum FillMode
```

Enumerators

Alternate	Determines whether a point is in the fill region by drawing a ray from that point to infinity in any direction, and then counting the number of path segments within the given shape that the ray crosses. If this number is odd, the point is in the fill region; if even, the point is outside the fill region.
Winding	Determines whether a point is in the fill region of the path by drawing a ray from that point to infinity in any direction, and then examining the places where a segment of the shape crosses the ray. Starting with a count of zero, add one each time a segment crosses the ray from left to right and subtract one each time a path segment crosses the ray from right to left, as long as left and right are seen from the perspective of the ray. After counting the crossings, if the result is zero, then the point is outside the path. Otherwise, it is inside the path.

11.7.2.12 GeometrySink

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Describes a geometric path that can contain lines, arcs, cubic Bezier curves, and quadratic Bezier curves.
(See also unmanaged API documentation)

Notes:

1. To create a **GeometrySink**, describe a PathGeometry and retrive the object using the PathGeometry.Open() method
2. A geometry sink consists of one or more figures. Each figure is made up of one or more line, curve, or arc segments. To create a figure, call the BeginFigure method, specify the figure's start point, and then use its Add methods (such as AddLine) to add segments. When you are finished adding segments, call the EndFigure method. You can repeat this sequence to create additional figures. When you are finished creating figures, call the Close method.

Syntax

```
interface GeometrySink
```

Methods

AddArc()	Adds a single arc to the path geometry.
AddLine()	Creates a line segment between the current point and the specified end point and adds it to the geometry sink.

AddLines()	Creates a sequence of lines using the specified points and adds them to the geometry sink.
BeginFigure()	Starts a new figure at the specified point.
Close()	Closes the geometry sink, indicates whether it is in an error state, and resets the sink's error state.
Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)
EndFigure()	Ends the current figure; optionally, closes it.
SetFillMode()	Specifies the method used to determine which points are inside the geometry described by this geometry sink and which points are outside.

11.7.2.12.1 AddArc()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Adds a single arc to the path geometry.
(See also unmanaged API documentation)

Method Return Value
This method does not return a value

Syntax
```
<GeometrySink>.AddArc(ArcSegment arc)
```

Parameters

arc	The SharpDX.Direct2D1.ArcSegment segment to add to the figure.

11.7.2.12.2 AddLine()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Creates a line segment between the current point and the specified end point and adds it to the geometry sink.
(See also unmanaged API documentation)

Method Return Value
This method does not return a value

Syntax
```
<GeometrySink>.AddLine(Vector2 vector2)
```

Parameters

vector2	A SharpDX.Vector2 which represents the end point of the

	line to draw.

11.7.2.12.3 AddLines()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Creates a sequence of lines using the specified points and adds them to the geometry sink. (See also unmanaged API documentation)

Method Return Value

This method does not return a value

Syntax

```
<GeometrySink>.AddLines(Vector2[] pointsRef)
```

Parameters

`pointsRef`	A SharpDX.Vector2 array of one or more points that describe the lines to draw. A line is drawn from the geometry sink's current point (the end point of the last segment drawn or the location specified by BeginFigure() to the first point in the array. If the array contains additional points, a line is drawn from the first point to the second point in the array, from the second point to the third point, and so on.

11.7.2.12.4 BeginFigure()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Starts a new figure at the specified point.
(See also unmanaged API documentation)

Method Return Value

This method does not return a value

Syntax

```
<GeometrySink>.BeginFigure(Vector2 vector2, FigureBegin figureBegin)
```

Parameters

vector2	The SharpDX.Vector2 at which to begin the new figure.
figureBegin	The SharpDX.Direct2D1.FigureBegin which determines whether the new figure should be hollow or filled.

11.7.2.12.5 Close()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this

reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Closes the geometry sink, indicates whether it is in an error state, and resets the sink's error state.
(See also unmanaged API documentation)

Note: Do not close the geometry sink while a figure is still in progress; doing so puts the geometry sink in an error state. For the close operation to be successful, there must be one EndFigure() call for each call to BeginFigure(). After calling this method, the geometry sink might not be usable. Direct2D implementations of this interface do not allow the geometry sink to be modified after it is closed, but other implementations might not impose this restriction.

Method Return Value

This method does not return a value

Syntax

```
<GeometrySink>.Close()
```

Parameters

This method does not accept any parameters

11.7.2.12.6 EndFigure()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Ends the current figure; optionally, closes it.
(See also unmanaged API documentation)

Method Return Value
This method does not return a value

Syntax
```
<GeometrySink>.EndFigure(FigureEnd figureEnd)
```

Parameters

figureEnd	A SharpDX.Direct2D1.FigureEnd value that indicates whether the current figure is closed. If the figure is closed, a line is drawn between the current point and the start point specified by BeginFigure().

11.7.2.12.7 SetFillMode()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Specifies the method used to determine which points are inside the geometry described by this geometry sink and which points are outside.
(See also unmanaged API documentation)

Method Return Value
This method does not return a value

Syntax
```
<GeometrySink>.SetFillMode(FillMode fillMode)
```

Parameters

`fillMode`	The SharpDX.Direct2D1.FillMode used to determine whether a given point is part of the geometry.

11.7.2.13 GradientStop

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Contains the position and color of a gradient stop.
(See also unmanaged API documentation)

Notes:

1. Gradient stops can be specified in any order if they are at different positions. Two stops may share a position. In this case, the first stop specified is treated as the "low" stop (nearer 0.0f) and subsequent stops are treated as "higher" (nearer 1.0f). This behavior is useful if a caller wants an instant transition in the middle of a stop.
2. Typically, there are at least two points in a collection, although creation with only one stop is permitted. For example, one point is at position 0.0f, another point is at position 1.0f, and additional points are distributed in the [0, 1] range. Where the gradient progression is beyond the range of [0, 1], the stops are stored, but may affect the gradient.
3. When drawn, the [0, 1] range of positions is mapped to the brush, in a brush-dependent way. For details, see LinearGradientBrush and RadialGradientBrush.
4. Gradient stops with a position outside the [0, 1] range cannot be seen explicitly, but they can still affect the colors produced in the [0, 1] range. For example, a two-stop gradient 0.0f, Black}, {2.0f, White is indistinguishable visually from 0.0f, Black}, {1.0f, Mid-level gray. Also, the colors are clamped before interpolation.

Syntax
```
struct GradientStop
```

Properties

Position	A float value that indicates the relative position of the gradient stop in the brush. This value must be in the [0.0f, 1.0f] range if the gradient stop is to be seen explicitly.
Color	The SharpDX.Color of the gradient stop.

11.7.2.14 GradientStopCollection

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Describes an elliptical arc between two points.
(See also unmanaged API documentation)

Note: A gradient stop collection is a device-dependent resource: your application should create gradient stop collections after it initializes the render target with which the gradient stop collection will be used, and recreate the gradient stop collection whenever the render target needs recreated. Please see the MSDN Direct2D Resources Overview for more information.

Syntax
```
class GradientStopCollection
```

Constructors

new GradientStopCollection(RenderTarget renderTarget, GradientStop[] gradientStops)	Creates an **GradientStopCollection** from the specified gradient stops, a **Gamma.StandardRgb**, and ExtendMode.Clamp
new GradientStopCollection(RenderTarget renderTarget, GradientStop[] gradientStops, ExtendMode extendMode)	Creates an **GradientStopCollection** from the specified gradient stops, color **Gamma.StandardRgb**, and extend mode
new GradientStopCollection(RenderTarget renderTarget, GradientStop[] gradientStops, Gamma colorInterpolationGamma)	Creates an **GradientStopCollection** from the specified gradient stops, color interpolation gamma, and ExtendMode.Clamp
new GradientStopCollection(RenderTarget renderTarget, GradientStop[] *gradientStops*, Gamma *colorInterpolationGamma*, ExtendMode *extendMode*)	Creates an **GradientStopCollection** from the specified gradient stops, color interpolation gamma, and extend mode

Methods and Properties

ColorInterpolationGamma	Indicates the gamma space in which the gradient stops are interpolated
Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)
ExtendMode	Indicates the behavior of the gradient outside the normalized gradient range

GradientStopCount	Retrieves the number of gradient stops in the collection
IsDisposed	Gets a value indicating whether this instance is disposed. (Inherited from SharpDX.DisposeBase.)

11.7.2.14.1
ColorInterpolationGamma

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Indicates the gamma space in which the gradient stops are interpolated.
(See also unmanaged API documentation)

Note: Interpolating in a linear gamma space (**Gamma.Linear**) can avoid changes in perceived brightness caused by the effect of gamma correction in spaces where the gamma is not 1.0, such as the default sRGB color space, where the gamma is 2.2.

Property Value

A `SharpDX.Direct2D1.Gamma enum` value specifies which gamma is used for interpolation.

Possible values include:

StandardRgb	Interpolation is performed in the standard RGB (sRGB) gamma.
Linear	Interpolation is performed in the linear-gamma color space.

(see also unmanaged API documentation)

Syntax
```
<GradientStopCollection>.ColorInterpolationGamma
```

11.7.2.14.2 ExtendMode

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Indicates the behavior of the gradient outside the normalized gradient range.
(See also unmanaged API documentation)

Note: For an LinearGradientBrush, the brush's content area is the gradient axis. For an RadialGradientBrush, the brush's content is the area within the gradient ellipse

Property Value
A `SharpDX.ExtendMode enum` value which determines how a brush paints areas outside of its normal content area.

Possible values include:

Clamp	Repeat the edge pixels of the brush's content for all regions outside the normal content area.
Wrap	Repeat the brush's content.
Mirror	The same as Wrap, except that alternate tiles of the brush's

	content are flipped. (The brush's normal content is drawn untransformed.)

(see also unmanaged API documentation)

Syntax
```
<GradientStopCollection>.ExtendMode
```

11.7.2.14.3 GradientStopCount

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the number of gradient stops in the collection.
(See also unmanaged API documentation)

Note: For an LinearGradientBrush, the brush's content area is the gradient axis. For an RadialGradientBrush, the brush's content is the area within the gradient ellipse

Property Value
An `int` value representing the number of gradient stops in the collection.

Syntax
```
<GradientStopCollection>.GradientStopCount
```

11.7.2.15 LinearGradientBrush

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Paints an area with a linear gradient.
(See also unmanaged API documentation)

Notes:

1. An **LinearGradientBrush** paints an area with a linear gradient along a line between the brush start point and end point. The gradient, defined by the brush GradientStopCollection, is extruded perpendicular to this line, and then transformed by a brush transform (if specified).
2. The start point and end point are described in the brush space and are mappped to the render target when the brush is used. Note the starting and ending coordinates are absolute, not relative to the render target size. A value of (0, 0) maps to the upper-left corner of the render target, while a value of (1, 1) maps one pixel diagonally away from (0, 0). If there is a nonidentity brush transform or render target transform, the brush start point and end point are also transformed.
3. It is possible to specify a gradient axis that does not completely fill the area that is being painted. When this occurs, the ExtendMode, specified by the GradientStopCollection, determines how the remaining area is painted.
4. The **LinearGradientBrush** can only be used with the render target that created it or with the compatible targets for that render target.
5. A **LinearGradientBrush** is a device-dependent resource: your application should create linear gradient brushes after it initializes the render target with which the brushes will be used, and recreate the brushes whenever the render target needs recreated. Please see the MSDN Direct2D Resources Overview for more information.
6. For convenience, **Direct2D** provides the RadialGradientBrushProperties function for creating new a **LinearGradientBrush**.

Syntax

```
class SolidColorBrush
```

Tips:

1. For NinjaScript Development purposes, you can use the NinjaTrader.Gui.DxExtensions.ToDxBrush() helper method to convert a **System.Windows.Media.LinearGradientBrush** to a **SharpDX.Direct2D1.LinearGradientBrush**
2. General information on **Direct2D** brushes can be found on the MSDN Direct2D Brushes Overview

Constructors

new LinearGradientBrush(RenderTarget renderTarget, LinearGradientBrushProperties linearGradientBrushProperties, GradientStopCollection gradientStopCollection)	Creates an **LinearGradientBrush** that contains the specified gradient stops and has the specified transform and base opacity.
new LinearGradientBrush(RenderTarget renderTarget, LinearGradientBrushProperties linearGradientBrushProperties, Nullable<BrushProperties> brushProperties, GradientStopCollection gradientStopCollection)	Creates an **LinearGradientBrush** that contains the specified gradient stops and has the specified transform and base opacity.

Methods and Properties

Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)
EndPoint	Retrieves or sets the ending coordinates of the linear gradient.

GradientStopCollection	Retrieves the GradientStopCollection associated with this linear gradient brush.
IsDisposed	Gets a value indicating whether this instance is disposed. (Inherited from SharpDX.DisposeBase.)
Opacity	Gets or sets the degree of opacity of this brush. (Inherited from Brush.)
StartPoint	Retrieves or sets the starting coordinates of the linear gradient.
Transform	Gets or sets the transform applied to this brush. (Inherited from Brush.)

11.7.2.15.1 EndPoint

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves or sets the ending coordinates of the linear gradient.
(See also unmanaged API documentation)

Note: The start point and end point are described in the brush's space and are mapped to the render target when the brush is used. If there is a non-identity brush transform or

render target transform, the brush's start point and end point are also transformed.

Property Value
A SharpDX.Vector2 representing the ending two-dimensional coordinates of the linear gradient, in the brush's coordinate space.

Syntax
```
<LinearGradientBrush>.EndPoint
```

11.7.2.15.2 GradientStopCollection

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the GradientStopCollection associated with this linear gradient brush. (See also unmanaged API documentation)

Property Value
A SharpDX.Direct2D1.GradientStopCollection object associated with this linear gradient brush object. This parameter is passed uninitialized.

Syntax
```
<LinearGradientBrush>.GradientStopCollection
```

11.7.2.15.3 StartPoint

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this

reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves the starting coordinates of the linear gradient.
(See also unmanaged API documentation)

Note: The start point and end point are described in the brush's space and are mapped to the render target when the brush is used. If there is a non-identity brush transform or render target transform, the brush's start point and end point are also transformed.

Property Value

A SharpDX.Vector2 representing the starting two-dimensional coordinates of the linear gradient, in the brush's coordinate space.

Syntax

```
<LinearGradientBrush>.StartPoint
```

11.7.2.16 LinearGradientBrushProperties

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Contains the starting point and endpoint of the gradient axis for an LinearGradientBrush.
(See also unmanaged API documentationth)

Syntax

```
struct LinearGradientBrushProperties
```

Constructors

`new LinearGradientBrushProperties()`	Initializes a new instance of the **LinearGradientBrushProperties** structure

Properties

StartPoint	A SharpDX.Vector2 representing brush's coordinate space, the starting point of the gradient axis.
EndPoint	A SharpDX.Vector2 representing the brush's coordinate space, the endpoint of the gradient axis.

11.7.2.17 MeasuringMode

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Indicates the measuring method used for text layout.
(See also unmanaged API documentation)

Syntax

```
enum MeasuringMode
```

Enumerators

Natural	Specifies that text is measured

	using glyph ideal metrics whose values are independent to the current display resolution.
GdiClassic	Specifies that text is measured using glyph display-compatible metrics whose values tuned for the current display resolution.
GdiNatural	Specifies that text is measured using the same glyph display metrics as text measured by GDI using a font created with CLEARTYPE_NATURAL_QUALITY.

11.7.2.18 PathGeometry

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Represents a complex shape that may be composed of arcs, curves, and lines.
(See also unmanaged API documentation)

Notes:

1. A **PathGeometry** object enables you to describe a geometric path. To describe an **PathGeometry** object's path, use the object's Open method to retrieve an GeometrySink. Use the sink to populate the path geometry with figures and segments.
2. **PathGeometry** objects are device-independent resources created by Factory. In general, you should create geometries once and retain them for the life of the

application, or until they need to be modified. Please see the MSDN Direct2D Resources Overview for more information.

Syntax

```
class PathGeometry
```

Constructors

`new PathGeometry(Factory factory)`	Creates an empty **PathGeometry**.

Tips:
1. For NinjaScript development purposes, when creating a **PathGemeory** object you should use the NinjaTrader.Core.Globals.D2DFactory property
2. General information **Direct2D Path Geometries** can be found on the MSDN Path Geometries Overview

Methods and Properties

Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)
FigureCount	Retrieves the number of figures in the path geometry.
FillContainsPoint()	Indicates whether the area filled by the geometry would contain the specified point given the specified flattening tolerance.
GetBounds()	Retrieves the bounds of the geometry.
IsDisposed	Gets a value indicating whether this instance is disposed.

	(Inherited from SharpDX.DisposeBase.)
Open()	Retrieves the geometry sink that is used to populate the path geometry with figures and segments.
SegmentCount	Retrieves the number of segments in the path geometry.
StrokeContainsPoint()	Determines whether the geometry's stroke contains the specified point given the specified stroke thickness, style, and transform.

11.7.2.18.1 FigureCount

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves the number of figures in the path geometry.
(See also unmanaged API documentation)

Property Value

An `int` representing the number of figures

Syntax

```
<PathGeometry>.FigureCount
```

11.7.2.18.2 FillContainsPoint()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Indicates whether the area filled by the geometry would contain the specified point given the specified flattening tolerance.
(See also unmanaged API documentation)

Method Return Value

A `bool` value which is **true** if the area filled by the geometry contains point; otherwise, **false**.

Syntax

```
<PathGeometry>.FillContainsPoint(Vector2 point)
```

Parameters

point	The SharpDX.Vector2 point to test.

11.7.2.18.3 GetBounds()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the bounds of the geometry.
(See also unmanaged API documentation)

Method Return Value
A SharpDX.RectangleF which contains the bounds of this geometry. If the bounds are empty, this will be a rect where bounds.left > bounds.right.

Syntax
```
<PathGeometry>.GetBounds()
```

Parameters
This method does not accept any parameters

11.7.2.18.4 Open()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the geometry sink that is used to populate the path geometry with figures and segments.
(See also unmanaged API documentation)

Notes:
1. Because path geometries are immutable and can only be populated once, it is an error to call **Open()** on a path geometry more than once.
2. Note that the fill mode defaults to Alternate. To set the fill mode, call SetFillMode() before the first call to BeginFigure(). Failure to do so will put the geometry sink in an error state.

Method Return Value

A SharpDX.Direct2D1.GeometrySink which contains the address of a reference to the geometry sink that is used to populate the path geometry with figures and segments.

Syntax
```
<PathGeometry>.Open()
```

Parameters
This method does not accept any parameters

11.7.2.18.5 SegmentCount

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the number of segments in the path geometry.
(See also unmanaged API documentation)

Method Return Value
An `int` representing the number of segments

Syntax
```
<PathGeometry>.SegmentCount
```

11.7.2.18.6 StrokeContainsPoint()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and

DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Determines whether the geometry's stroke contains the specified point given the specified stroke thickness, style, and transform.
(See also unmanaged API documentation)

Method Return Value

A `bool` value set to **true** if the geometry's stroke contains the specified point; otherwise, **false**.

Syntax

```
<PathGeometry>.StrokeContainsPoint(Vector2 point, float strokeWidth)
<PathGeometry>.StrokeContainsPoint(Vector2 point, float strokeWidth, StrokeStyle
strokeStyle)
<PathGeometry>.StrokeContainsPoint(Vector2 point, float strokeWidth, StrokeStyle
strokeStyle, Matrix3x2 transform)
```

Parameters

point	The SharpDX.Vector2 point to test for containment.
strokeStyle	The SharpDX.Direct2D1.StrokeStyle style of stroke to apply.
strokeWidth	The thickness of the stroke to apply.
transform	The SharpDX.Matrix3x2 transform to apply to the stroked geometry.

11.7.2.19 RadialGradientBrush

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to

the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Paints an area with a radial gradient.
(See also unmanaged API documentation)

Notes:

1. The **RadialGradientBrush** is similar to the LinearGradientBrush in that they both map a collection of gradient stops to a gradient. However, the linear gradient has a start and an end point to define the gradient vector, while the **radial gradient** uses an ellipse and a gradient origin to define its gradient behavior. To define the position and size of the ellipse, use the Center, RadiusX, and RadiusY properties to specify the center, x-radius, and y-radius of the ellipse. The gradient origin is the center of the ellipse, unless a gradient offset is specified by using the GradientOriginOffset method.
2. The brush maps the **gradient stop** position 0.0f of the gradient origin, and the position 1.0f is mapped to the ellipse boundary. When the **gradient origin** is within the ellipse, the contents of the ellipse enclose the entire [0, 1] range of the brush gradient stops. If the **gradient origin** is outside the bounds of the ellipse, the brush still works, but its gradient is not well-defined.
3. The **start point** and **end point** are described in the brush space and are mappped to the render target when the brush is used. Note the starting and ending coordinates are absolute, not relative to the **render target** size. A value of (0, 0) maps to the upper-left corner of the render target, while a value of (1, 1) maps just one pixel diagonally away from (0, 0). If there is a nonidentity brush transform or render target transform, the brush ellipse and gradient origin are also transformed.
4. It is possible to specify an ellipse that does not completely fill area being painted. When this occurs, the ExtendMode and setting (specified by the brush GradientStopCollection) determines how the remaining area is painted.
5. A **RadialGradientBrush** brush may be used only with the render target that created it or with the compatible targets for that **render target**.
6. A **RadialGradientBrush** is a device-dependent resource: your application should create radial gradient brushes after it initializes the **render target** with which the brushes will be used, and recreate the brushes whenever the render target needs recreated. Please see the MSDN Direct2D Resources Overview for more information.
7. For convenience, Direct2D provides the RadialGradientBrushProperties function for creating new a RadialGradientBrush.

Syntax

```
class SolidColorBrush
```

Tips:
1. For NinjaScript Development purposes, you can use the NinjaTrader.Gui.DxExtensions.ToDxBrush() helper method to convert a **System.Windows.Media.LinearGradientBrush** to a **SharpDX.Direct2D1.LinearGradientBrush**
2. General information on **Direct2D** brushes can be found on the MSDN Direct2D Brushes Overview

Constructors

new RadialGradientBrush(RenderTarget renderTarget, RadialGradientBrushProperties radialGradientBrushProperties, GradientStopCollection gradientStopCollection)	Creates an RadialGradientBrush that contains the specified gradient stops and has the specified transform and base opacity.
new RadialGradientBrush(RenderTarget renderTarget, RadialGradientBrushProperties radialGradientBrushProperties, GradientStopCollection gradientStopCollection)	Creates an RadialGradientBrush that contains the specified gradient stops and has the specified transform and base opacity.
new RadialGradientBrush(RenderTarget renderTarget, RadialGradientBrushProperties radialGradientBrushProperties, BrushProperties brushProperties, GradientStopCollection gradientStopCollection)	Creates an RadialGradientBrush that contains the specified gradient stops and has the specified transform and base opacity.
new RadialGradientBrush(RenderTarget renderTarget, RadialGradientBrushProperties	Creates an RadialGradientBrush that contains the specified gradient stops and has the

radialGradientBrushProperties, Nullable<BrushProperties> brushProperties, GradientStopCollection gradientStopCollection)	specified transform and base opacity.

Methods and Properties

Center	Retrieves or sets the center of the gradient ellipse.
Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)
IsDisposed	Gets a value indicating whether this instance is disposed. (Inherited from SharpDX.DisposeBase.)
GradientOriginOffset	Retrieves or sets the offset of the gradient origin relative to the gradient ellipse's center.
GradientStopCollection	Retrieves the GradientStopCollection associated with this radial gradient brush object.
Opacity	Gets or sets the degree of opacity of this brush. (Inherited from Brush.)
RadiusX	Retrieves or sets the x-radius of the gradient ellipse.
RadiusY	Retrieves or sets the y-radius of the gradient ellipse.

Transform	Gets or sets the transform applied to this brush. (Inherited from Brush.)

11.7.2.19.1 Center

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves or sets the center of the gradient ellipse.
(See also unmanaged API documentation)

Property Value

A SharpDX.Vector2 representing the center of the gradient ellipse. This value is expressed in the brush's coordinate space.

Syntax

```
<RadialGradientBrush>.Center
```

11.7.2.19.2 GradientOriginOffset

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves or sets the offset of the gradient origin relative to the gradient ellipse's center. (See also unmanaged API documentation)

Property Value
A SharpDX.Vector2 representing the offset of the gradient origin from the center of the gradient ellipse. This value is expressed in the brush's coordinate space.

Syntax
```
<RadialGradientBrush>.GradientOriginOffset
```

11.7.2.19.3 GradientStopCollection

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the **GradientStopCollection** associated with this radial gradient brush object (See also unmanaged API documentation)

Note: The **GradientStopCollection** contains an array of SharpDX.GradientStopCollection structures and additional information, such as the extend mode and the color interpolation mode.

Property Value
The SharpDX.GradientStopCollection object associated with this linear gradient brush object. This parameter is passed uninitialized.

Syntax
```
<RadialGradientBrush>.GradientStopCollection
```

11.7.2.19.4 RadiusX

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves or sets the x-radius of the gradient ellipse.
(See also unmanaged API documentation)

Property Value

A `float` value representing the x-radius of the gradient ellipse. This value is expressed in the brush's coordinate space.

Syntax

```
<RadialGradientBrush>.RadiusX
```

11.7.2.19.5 RadiusY

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves or sets the y-radius of the gradient ellipse.
(See also unmanaged API documentation)

Property Value
A `float` value representing the y-radius of the gradient ellipse. This value is expressed in the brush's coordinate space.

Syntax
```
<RadialGradientBrush>.RadiusY
```

11.7.2.20 RadialGradientBrushProperties

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Contains the gradient origin offset and the size and position of the gradient ellipse for an RadialGradientBrush.
(See also unmanaged API documentation)

Syntax
```
struct RadialGradientBrushProperties
```

Constructors

`new RadialGradientBrushProperties()`	Initializes a new instance of the **RadialGradientBrushProperties** structure

Properties

Center	A SharpDX.Vector2 representing the brush's coordinate space, the center of the gradient ellipse.
GradientOriginOffset	A SharpDX.Vector2 representing brush's coordinate space, the

	offset of the gradient origin relative to the gradient ellipse's center.
RadiusX	A `float` in the brush's coordinate space, the x-radius of the gradient ellipse.
RadiusY	A `float` in the brush's coordinate space, the y-radius of the gradient ellipse.

11.7.2.21 RenderTarget

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Represents an object that can receive drawing commands.
(See also unmanaged API documentation)

Syntax

```
class RenderTarget
```

Tips:
1. For NinjaScript Development purposes, DrawingTools, ChartStyles, Indicators, and Strategies implement the Chart's RenderTarget ready to be used in the **OnRender()** method
2. General information on **Direct2D Render Targets** can be found on the MSDN Direct2D Render Targets Overview

Methods and Properties

AntialiasMode	Retrieves or sets the current antialiasing mode for nontext drawing operations.
DrawEllipse()	Draws the outline of the specified ellipse using the specified stroke style.
DrawGeometry()	Draws the outline of the specified geometry.
DrawLine()	Draws a line between the specified points.
DrawRectangle()	Draws the outline of a rectangle that has the specified dimensions.
DrawText()	Draws the specified text using the format information provided by an SharpDX.DirectWrite.TextFormat object.
DrawTextLayout()	Draws the formatted text described by the specified SharpDX.DirectWrite.TextLayout object.
FillEllipse()	Paints the interior of the specified ellipse.
FillGeometry()	Paints the interior of the specified geometry.
FillRectangle()	Paints the interior of the specified rectangle.
IsDisposed	Gets a value indicating whether this instance is disposed. (Inherited from SharpDX.DisposeBase.)

Transform	Gets or sets the current transform of the render target.

11.7.2.21.1 AntialiasMode

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves or sets the current antialiasing mode for nontext drawing operations.
(See also unmanaged API documentation)

Property Value

A SharpDX.Direct2D1.AntialiasMode enum value

Syntax

```
RenderTarget.AntialiasMode
```

11.7.2.21.2 Draw Ellipse()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Draws the outline of the specified ellipse using the specified stroke style.
(See also unmanaged API documentation)

> **Note**: This method doesn't return an error code if it fails.

Method Return Value

This method does not return a value

Syntax

```
RenderTarget.DrawEllipse(Ellipse ellipse, Brush brush)
RenderTarget.DrawEllipse(Ellipse ellipse, Brush brush, float strokeWidth)
RenderTarget.DrawEllipse(Ellipse ellipse, Brush brush, float strokeWidth, StrokeStyle
strokeStyle)
```

Parameters

ellipse	The SharpDX.Direct2D1.Ellipse position and radius of the ellipse to draw, in device-independent pixels.
brush	The SharpDX.Direct2D1.Brush used to paint the ellipse's outline.
strokeWidth	The thickness of the ellipse's stroke. The stroke is centered on the ellipse's outline.
strokeStyle	The SharpDX.Direct2D1.StrokeStyle to apply to the ellipse's outline, or null to paint a solid stroke.

11.7.2.21.3 Draw Geometry()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and

DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Draws the outline of the specified geometry using the specified stroke style.
(See also unmanaged API documentation)

Note: This method doesn't return an error code if it fails.

Method Return Value

This method does not return a value

Syntax

```
RenderTarget.DrawGeometry(Geometry geometry, Brush brush)
RenderTarget.DrawGeometry(Geometry geometry, Brush brush, float strokeWidth)
RenderTarget.DrawGeometry(Geometry geometry, Brush brush, float strokeWidth,
StrokeStyle strokeStyle)
```

Parameters

brush	An int which represents the method input
geometry	The SharpDX.Direct2D1.Geometry to draw
strokeStyle	The SharpDX.Direct2D1.StrokeStyle to apply to the geometry's outline, or null to paint a solid stroke.
strokeWidth	The thickness of the geometry's stroke. The stroke is centered on the geometry's outline.

11.7.2.21.4 Draw Line()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Draws a line between the specified points.
(See also unmanaged API documentation)

Note: This method doesn't return an error code if it fails.

Method Return Value

This method does not return a value

Syntax

```
RenderTarget.DrawLine(Vector2 point0, Vector2 point1, Brush brush)
RenderTarget.DrawLine(Vector2 point0, Vector2 point1, Brush brush, float strokeWidth)
RenderTarget.DrawLine(Vector2 point0, Vector2 point1, Brush brush, float strokeWidth,
StrokeStyle strokeStyle)
```

Parameters

brush	The SharpDX.Direct2D1.Brush brush used to paint the line's stroke.
point0	A SharpDX.Vector2 which determines the start point of the line, in device-independent pixels.
point1	A SharpDX.Vector2 which determines the end point of the

	line, in device-independent pixels.
strokeStyle	The SharpDX.Direct2D1.StrokeStyle to paint, or null to paint a solid line.
strokeWidth	A value greater than or equal to 0.0f that specifies the width of the stroke. If this parameter isn't specified, it defaults to 1.0f. The stroke is centered on the line.

11.7.2.21.5 Draw Rectangle()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Draws the outline of a rectangle that has the specified dimensions and stroke style. (See also unmanaged API documentation)

Note: This method doesn't return an error code if it fails.

Method Return Value

This method does not return a value

Syntax

```
RenderTarget.DrawRectangle(RectangleF rect, Brush brush)
RenderTarget.DrawRectangle(RectangleF rect, Brush brush, float strokeWidth)
RenderTarget.DrawRectangle(RectangleF rect, Brush brush, float strokeWidth,
StrokeStyle strokeStyle)
```

Parameters

brush	The SharpDX.Direct2D1.Brush used to paint the rectangle's stroke.
rect	The SharpDX.RectangleF which determines the dimensions of the rectangle to draw, in device-independent pixels.
strokeStyle	The SharpDX.Direct2D1.StrokeStyle used to paint, or null to paint a solid stroke.
strokeWidth	A value greater than or equal to 0.0f that specifies the width of the rectangle's stroke. The stroke is centered on the rectangle's outline.

11.7.2.21.6 Draw Text()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Draws the specified text using the format information provided by an SharpDX.DirectWrite.TextFormat object.
(See also unmanaged API documentation)

Note: This method doesn't return an error code if it fails.

Method Return Value

This method does not return a value.

Syntax

```
RenderTarget.DrawText(string text, TextFormat textFormat, RectangleF layoutRect, Brush
 defaultForegroundBrush)
RenderTarget.DrawText(string text, TextFormat textFormat, RectangleF layoutRect, Brush
 defaultForegroundBrush, DrawTextOptions options)
RenderTarget.DrawText(string text, TextFormat textFormat, RectangleF layoutRect, Brush
 defaultForegroundBrush, DrawTextOptions options,   MeasuringMode measuringMode)
RenderTarget.DrawText(string text, int stringLength, TextFormat textFormat, RectangleF
 layoutRect, Brush defaultForegroundBrush, RenderTarget.DrawTextOptions options,
MeasuringMode measuringMode)
```

Parameters

defaultForegroundBrush	The SharpDX.Direct2D1.Brush used to paint the text.
layoutRect	A SharpDX.RectangleF which determines size and position of the area in which the text is drawn.
measuringMode	A SharpDX.Direct2D1.MeasuringMode value that indicates how glyph metrics are used to measure text when it is formatted. The default value is DWRITE_MEASURING_MODE_NATURAL.
options	A SharpDX.Direct2D1.DrawTextOptions value that indicates whether the text should be snapped to pixel boundaries and whether the text should be clipped to the layout rectangle. The default value is None, which indicates that text should be snapped to pixel

	boundaries and it should not be clipped to the layout rectangle.
stringLength	An `int` value which represents the number of characters in string.
text	A `string` reference to an array of Unicode characters to draw.
textFormat	A SharpDX.DirectWrite.TextFormat object that describes formatting details of the text to draw, such as the font, the font size, and flow direction.

11.7.2.21.7 Draw TextLayout()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Draws the formatted text described by the specified SharpDX.DirectWrite.TextLayout object. (See also unmanaged API documentation)

Notes:

1. When drawing the same text repeatedly, using the **DrawTextLayout()** method is more efficient than using the DrawText() method because the text doesn't need to be formatted and the layout processed with each call.
2. This method doesn't return an error code if it fails.

Method Return Value

This method does not return a value

Syntax

```
RenderTarget.DrawTextLayout(Vector2 origin, TextLayout textLayout, Brush
defaultForegroundBrush)
RenderTarget.DrawTextLayout(Vector2 origin, TextLayout textLayout, Brush
defaultForegroundBrush, DrawTextOptions options)
```

Parameters

defaultForegroundBrush	The SharpDX.Direct2D1.Brush used to paint any text in textLayout that does not already have a brush associated with it as a drawing effect (specified by the SetDrawingEffect method).
options	A SharpDX.Direct2D1.DrawTextOptions value that indicates whether the text should be snapped to pixel boundaries and whether the text should be clipped to the layout rectangle. The default value is None, which indicates that text should be snapped to pixel boundaries and it should not be clipped to the layout rectangle.
origin	A SharpDX.Vector2 described in device-independent pixels, at which the upper-left corner of the text described by textLayout is drawn.
textLayout	A SharpDX.DirectWrite.TextLayout representing the formatted text to draw. Any drawing effects that do not inherit from Resource are ignored. If there are drawing effects that inherit from ID2D1Resource that are not brushes, this method fails and the

	render target is put in an error state.

11.7.2.21.8 FillEllipse()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Paints the interior of the specified ellipse.
(See also unmanaged API documentation)

Note: This method doesn't return an error code if it fails.

Method Return Value

This method does not return a value

Syntax

```
RenderTarget.FillEllipse(Ellipse ellipse, Brush brush)
```

Parameters

brush	A SharpDX.Direct2D1.Brush used to paint the interior of the ellipse.
ellipse	A SharpDX.Direct2D1.Ellipse which describes the position and radius, in device-independent pixels, of the ellipse to paint.

11.7.2.21.9 FillGeometry()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Paints the interior of the specified geometry.
(See also unamanged API documentation)

Note:
1. If the opacityBrush parameter is not null, the alpha value of each pixel of the mapped opacityBrush is used to determine the resulting opacity of each corresponding pixel of the geometry. Only the alpha value of each color in the brush is used for this processing; all other color information is ignored. The alpha value specified by the brush is multiplied by the alpha value of the geometry after the geometry has been painted by brush.
2. This method doesn't return an error code if it fails.

Method Return Value

This method does not return a value.

Syntax

```
RenderTarget.FillGeometry(Geometry geometry, Brush brush)
RenderTarget.FillGeometry(Geometry geometry, Brush brush, Brush opacityBrush)
```

Parameters

brush	The SharpDX.Direct2D1.Brush used to paint the geometry's interior.

geometry	The SharpDX.Direct2D1.Geometry to paint.
opacityBrush	The SharpDX.Direct2D1.Brush opacity mask to apply to the geometry, or null for no opacity mask. For more information, see the note section above

11.7.2.21.10 FillRectangle()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Paints the interior of the specified rectangle.
(See also unamanged API documentation)

Note: This method doesn't return an error code if it fails.

Method Return Value

This method does not return a value

Syntax

```
RenderTarget.FillRectangle(RectangleF rect, Brush brush)
```

Parameters

brush	The SharpDX.Direct2D1.Brush used to paint the rectangle's interior.

rect	A SharpDX.RectangleF describing the dimension of the rectangle to paint, in device-independent pixels.

11.7.2.21.11 Transform

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets or sets the current transform of the render target.
(See also unmanaged API documentation)

Property Value

A SharpDX.Matrix3x2

Syntax

```
RenderTarget.Transform
```

11.7.2.22 SolidColorBrush

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Paints an area with a solid color.
(See also unmanaged API documentation)

Notes:
1. The **SolidColorBrush** can only be used with the render target that created it or with the compatible targets for that render target.
2. A **SolidColorBrush** is a device-dependent resource. Please see the MSDN Direct2D Resources Overview for more information.
3. For convenience, **Direct2D** provides the BrushProperties function for creating new a **SolidColorBrush**.

Syntax

```
class SolidColorBrush
```

Tips:
1. For NinjaScript Development purposes, you can use the NinjaTrader.Gui.DxExtensions.ToDxBrush() helper method to convert a **System.Windows.Media.SolidColorBrush** to a **SharpDX.Direct2D1.SolidColorBrush**
2. General information on **Direct2D** brushes can be found on the MSDN Direct2D Brushes Overview

Constructors

`new SolidColorBrush(RenderTarget renderTarget, Color4 color)`	Creates a new **SolidColorBrush** that has the specified color and opacity.
`new SolidColorBrush(RenderTarget renderTarget, Color4 color, Nullable<BrushProperties> brushProperties)`	Creates a new **SolidColorBrush** that has the specified color and opacity.

Methods and Properties

Color	Retrieves or sets the color of the solid color brush.
Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)
IsDisposed	Gets a value indicating whether this instance is disposed. (Inherited from SharpDX.DisposeBase.)
Opacity	Gets or sets the degree of opacity of this brush. (Inherited from Brush.)
Transform	Gets or sets the transform applied to this brush. (Inherited from Brush.)

11.7.2.22.1 Color

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves the color of the solid color brush.
(See also unmanaged API documentation)

Property Value

The SharpDX.Color4 of this solid color brush.

Syntax
```
<SolidColorBrush>.Color
```

11.7.2.23 StrokeStyle

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Describes the caps, miter limit, line join, and dash information for a stroke.
(See also unmanaged API documentation)

Notes:
1. A stroke style is a device-independent resource; you can create it once then retain it for the life of your application. Please see the MSDN Direct2D Resources Overview for more information.
2. For convenience, **Direct2D** provides the StrokeStyleProperties function for creating new a **StrokeStyle**.

Syntax
```
class StrokeStyle
```

Constructors

`new StrokeStyle(Factory factory, StrokeStyleProperties properties)`	Creates an **StrokeStyle** that describes start cap, dash pattern, and other features of a stroke.
`new StrokeStyle(Factory factory, StrokeStyleProperties properties, float[] dashes)`	Creates an **StrokeStyle** that describes start cap, dash pattern, and other features of a stroke.

Tip: For NinjaScript development purposes, when creating a **StrokeStyle** object you should use the NinjaTrader.Core.Globals.D2DFactory property

Method and Properties

DashCap	Gets a value that specifies how the ends of each dash are drawn.
DashesCount	Retrieves the number of entries in the dashes array.
DashOffset	Retrieves a value that specifies how far in the dash sequence the stroke will start.
DashStyle	Gets a value that describes the stroke's dash pattern.
Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)
EndCap	Retrieves the type of shape used at the end of a stroke.
GetDashes()	Copies the dash pattern to the specified array.
IsDisposed	Gets a value indicating whether this instance is disposed. (Inherited from DisposeBase.)
LineJoin	Retrieves the type of joint used at the vertices of a shape's outline.
MiterLimit	Retrieves the limit on the ratio of the miter length to half the stroke's thickness.

StartCap	Retrieves the type of shape used at the beginning of a stroke.

11.7.2.23.1 DashCap

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets a value that specifies how the ends of each dash are drawn.
(See also unmanaged API documentation)

Property Value

A SharpDX.Direct2D1.CapStyle value that specifies how the ends of each dash are drawn.

Syntax

```
<StrokeStyle>.DashCap
```

11.7.2.23.2 DashesCount

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the number of entries in the dashes array.
(See also unmanaged API documentation)

Property Value
An `int` for the number of entries in the dashes array if the stroke is dashed; otherwise, 0.

Syntax
```
<StrokeStyle>.DashesCount
```

11.7.2.23.3 DashOffset

> **Disclaimer**: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the number of entries in the dashes array.
(See also unmanaged API documentation)

Property Value
A `float` value that specifies how far in the dash sequence the stroke will start.

Syntax
```
<StrokeStyle>.DashesCount
```

11.7.2.23.4 DashStyle

> **Disclaimer**: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and

DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets a value that describes the stroke's dash pattern.
(See also unmanaged API documentation)

Note: If a custom dash style is specified, the dash pattern is described by the dashes array, which can be retrieved by calling the GetDashes() method.

Property Value

A `SharpDX.Direct2D1.DashStyle enum` value that describes the predefined dash pattern used, or `DashStyle.Custom` if a custom dash style is used.

Possible Values are:

Solid	A solid line with no breaks.
Dash	A dash followed by a gap of equal length. The dash and the gap are each twice as long as the stroke thickness. The equivalent dash array for Dash is {2, 2}.
Dot	A dot followed by a longer gap. The equivalent dash array for Dot is {0, 2}.
DashDot	A dash, followed by a gap, followed by a dot, followed by another gap. The equivalent dash array for DashDot is {2, 2, 0, 2}.
DashDotDot	A dash, followed by a gap, followed by a dot, followed by another gap, followed by another dot, followed by another gap. The equivalent dash array for DashDotDot is {2, 2, 0, 2, 0, 2}.

Custom	The dash pattern is specified by an array of floating-point values.

(See also unmanaged API documentation)

Syntax
```
<StrokeStyle>.DashStyle
```

11.7.2.23.5 EndCap

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the type of shape used at the end of a stroke.
(See also unmanaged API documentation)

Property Value
A SharpDX.Direct2D1.CapStyle value that specifies the type of joint used at the vertices of a shape's outline.

Syntax
```
<StrokeStyle>.EndCap
```

11.7.2.23.6 GetDashes()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to

the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Copies the dash pattern to the specified array.
(See also unmanaged API documentation)

Note: The dashes are specified in units that are a multiple of the stroke width, with subsequent members of the array indicating the dashes and gaps between dashes: the first entry indicates a filled dash, the second a gap, and so on.

Method return value

This method does not return a value.

Syntax

```
<StrokeStyle>.GetDashes(float[] dashes, int dashesCount)
```

dashes	A `float` pointer to an array that will receive the dash pattern. The array must be able to contain at least as many elements as specified by dashesCount. You must allocate storage for this array.
dashesCount	The `int` number of dashes to copy. If this value is less than the number of dashes in the stroke style's dashes array, the returned dashes are truncated to dashesCount. If this value is greater than the number of dashes in the stroke style's dashes array, the extra dashes are set to 0.0f. To obtain the actual number of dashes in the

	stroke style's dashes array, use the DashesCount property.

11.7.2.23.7 LineJoin

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves the type of joint used at the vertices of a shape's outline.
(See also unmanaged API documentation)

Note: A miter limit affects how sharp miter joins are allowed to be. If the line join style is MiterOrBevel, then the join will be mitered with regular angular vertices if it doesn't extend beyond the miter limit; otherwise, the line join will be beveled.

Property Value

A `SharpDX.Direct2D1.LineJoin enum` value that specifies the type of joint used at the vertices of a shape's outline.

Possible values are:

Miter	Regular angular vertices.
Bevel	Beveled vertices.
Round	Rounded vertices.
MiterOrBevel	Regular angular vertices unless the join would extend beyond the

	miter limit; otherwise, beveled vertices.

(See also unmanaged API documentation)

Syntax
```
<StrokeStyle>.LineJoin
```

11.7.2.23.8 MiterLimit

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the limit on the ratio of the miter length to half the stroke's thickness.
(See also unmanaged API documentation)

Property Value
A positive `float` value greater than or equal to 1.0f that describes the limit on the ratio of the miter length to half the stroke's thickness.

Syntax
```
<StrokeStyle>.MiterLimit
```

11.7.2.23.9 StartCap

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to

the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Retrieves the type of shape used at the beginning of a stroke.
(See also unmanaged API documentation)

Property Value
A SharpDX.Direct2D1.CapStyle value for the type of shape used at the beginning of a stroke.

Syntax
```
<StrokeStyle>.StartCap
```

11.7.2.24 StrokeStyleProperties

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Describes the stroke that outlines a shape.
(See also unmanaged API documentation)

Syntax
```
struct StrokeStyleProperties
```

Properties

StartCap	The StartCap value applied to the start of all the open figures in a stroked geometry.
EndCap	The EndCap value applied to the end of all the open figures in a stroked geometry.
DashCap	The DashCap value for the shape at either end of each dash segment.
LineJoin	A LineJoin value that describes how segments are joined. This value is ignored for a vertex if the segment flags specify that the segment should have a smooth join.
MiterLimit	The MeterLlmit value of the thickness of the join on a mitered corner. This value is always treated as though it is greater than or equal to 1.0f.
DashStyle	A DashStyle value that specifies whether the stroke has a dash pattern and, if so, the dash style.
DashOffset	A DashOffset value that specifies an offset in the dash sequence. A positive dash offset value shifts the dash pattern, in units of stroke width, toward the start of the stroked geometry. A negative dash offset value shifts the dash pattern, in units of stroke width, toward the end of the stroked geometry.

11.7.2.25 SweepDirection

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Defines the direction that an elliptical arc is drawn.
(See also unmanaged API documentation)

Syntax

```
enum SweepDirection
```

Enumerators

CounterClockwise	Arcs are drawn in a counterclockwise (negative-angle) direction.
Clockwise	Arcs are drawn in a clockwise (positive-angle) direction.

11.7.3 SharpDX.DirectWrite

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

The **SharpDX.DirectWrite** namespace provides a managed **DirectWrite** API. **DirectWrite** supports high-quality text rendering, resolution-independent outline fonts, and full Unicode text and layouts.
(See also unmanaged API documentation)

In this section

TextFormat	The TextFormat interface describes the font and paragraph properties used to format text, and it describes locale information.
TextLayout	The TextLayout interface represents a block of text after it has been fully analyzed and formatted.

11.7.3.1 TextFormat

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

The TextFormat interface describes the font and paragraph properties used to format text, and it describes locale information.
(See also unmanaged API documentation)

Notes:
1. These properties cannot be changed after the **TextFormat** object is created. To change these properties, a new **TextFormat** object must be created with the desired properties.

2. The **TextFormat** interface is used to draw text with a single format. To draw text with multiple formats, or to use a custom text renderer, use the TextLayout interface. **TextLayout** enables the application to change the format for ranges of text within the string.
3. This object may not be thread-safe, and it may carry the state of text format change.
4. To draw simple text with a single format, Direct2D provides the DrawText() method, which draws a string using the format information provided by an **TextFormat** object.

Syntax

```
class TextFormat
```

Constructors

`new TextFormat(Factory factory, string fontFamilyName, float fontSize)`	Creates a text format object used for text layout with normal weight, style and stretch.
`new TextFormat(Factory factory, string fontFamilyName, FontWeight fontWeight, FontStyle fontStyle, float fontSize)`	Creates a text format object used for text layout with normal stretch.
`new TextFormat(Factory factory, string fontFamilyName, FontWeight fontWeight, FontStyle fontStyle, FontStretch fontStretch, float fontSize)`	Creates a text format object used for text layout.

Tip: For NinjaScript development purposes, when creating a **TextFormat** object you should use the NinjaTrader.Core.Globals.DirectWriteFactory property

Methods and Properties

Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)

FlowDirection	Gets or sets the direction that text lines flow.
FontFamilyName	Creates a text format object used for text layout with normal weight, style and stretch.
FontSize	Creates a text format object used for text layout with normal stretch.
FontStretch	Creates a text format object used for text layout.
FontStyle	Gets the font style of the text.
FontWeight	Gets the font weight of the text.
IsDisposed	Gets a value indicating whether this instance is disposed. (Inherited from SharpDX.DisposeBase.)
ParagraphAlignment	Gets or sets the alignment option of a paragraph which is relative to the top and bottom edges of a layout box.
ReadingDirection	Gets or sets the current reading direction for text in a paragraph.
TextAlignment	Gets or sets the alignment option of text relative to the layout box's leading and trailing edge.
WordWrapping	Gets or sets the word wrapping option.

11.7.3.1.1 Flow Direction

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this

section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets the direction that text lines flow.
(See also unmanaged API documentation)

Property Value

A `SharpDX.DirectWrite.FlowDirection enum` which determines text lines flow within their parent container.

Possible values are:

TopToBottom	Specifies that text lines are placed from top to bottom.

Syntax

```
<TextLayout>.FlowDirection
```

11.7.3.1.2 FontFamilyName

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets a copy of the font family name.
(See also unmanaged API documentation)

Property Value
A `string` value representing the current font family name

Syntax
```
<TextLayout>.FontFamilyName
```

11.7.3.1.3 FontSize

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Gets the font size in DIP unites.
(See also unmanaged API documentation)

Property Value
A `float` representing the current font size in DIP units.

Syntax
```
<TextLayout>.FontSize
```

11.7.3.1.4 FontStretch

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the

DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets the font stretch of the text.
(See also unmanaged API documentation)

Note:
1. A font stretch describes the degree to which a font form is stretched from its normal aspect ratio, which is the original width to height ratio specified for the glyphs in the font.
2. Values other than the ones defined in the enumeration are considered to be invalid, and are rejected by font API functions.

Property Value

A `SharpDX.DirectWrite.FontStretch enum` value which indicates the type of font stretch (such as normal or condensed). See table below

Syntax

```
<TextLayout>.FontStretch
```

Possible values are:

Undefined	Predefined font stretch : Not known (0).
UltraCondensed	Predefined font stretch : Ultra-condensed (1).
ExtraCondensed	Predefined font stretch : Extra-condensed (2).
Condensed	Predefined font stretch : Condensed (3).
SemiCondensed	Predefined font stretch : Semi-condensed (4).
Normal	Predefined font stretch : Normal (5).

Medium	Predefined font stretch : Medium (5).
SemiExpanded	Predefined font stretch : Semi-expanded (6).
Expanded	Predefined font stretch : Expanded (7).
ExtraExpanded	Predefined font stretch : Extra-expanded (8).
UltraExpanded	Predefined font stretch : Ultra-expanded (9).

11.7.3.1.5 FontStyle

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets the font style of the text.
(See also unmanaged API documentation)

Property Value

A `SharpDX.DirectWrite.FontStyle enum` value which indicates the type of font style (such as slope or incline).

Possible values are:

Normal	The characters in a normal, or roman, font are upright.

Oblique	The characters in an oblique font are artificially slanted.
Italic	The characters in an italic font are truly slanted and appear as they were designed.

Syntax

```
<TextLayout>.FontStyle
```

11.7.3.1.6 FontWeight

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets the font weight of the text.
(See also unmanaged API documentation)

Notes:
1. Weight differences are generally differentiated by an increased stroke or thickness that is associated with a given character in a typeface, as compared to a "normal" character from that same typeface.
2. Not all weights are available for all typefaces. When a weight is not available for a typeface, the closest matching weight is returned.
3. Font weight values less than 1 or greater than 999 are considered invalid, and they are rejected by font API functions.

Property Value

A `SharpDX.DirectWrite.FontWeight enum` value that indicates the type of weight (such as normal, bold, or black). See table below

Syntax

```
<TextLayout>.FontWeight
```

Possible values are:

Thin	Predefined font weight : Thin (100).
ExtraLight	Predefined font weight : Extra-light (200).
UltraLight	Predefined font weight : Ultra-light (200).
Light	Predefined font weight : Light (300).
Normal	Predefined font weight : Normal (400).
Regular	Predefined font weight : Regular (400).
Medium	Predefined font weight : Medium (500).
DemiBold	Predefined font weight : Demi-bold (600).
SemiBold	Predefined font weight : Semi-bold (600).
Bold	Predefined font weight : Bold (700).
ExtraBold	Predefined font weight : Extra-bold (800).
UltraBold	Predefined font weight : Extra-bold (800).

Black	Predefined font weight : Black (900).
Heavy	Predefined font weight : Heavy (900).
ExtraBlack	Predefined font weight : Extra-black (950).
UltraBlack	Predefined font weight : Ultra-black (950).
SemiLight	Predefined font weight : Normal (400).

11.7.3.1.7 ParagraphAlignment

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets or sets the alignment option of a paragraph which is relative to the top and bottom edges of a layout box.
(See also unmanaged API documentation)

Property Value

A `SharpDX.DirectWrite.ParagraphAlignment enum` value that indicates the current paragraph alignment option.

Possible values are:

Near	The top of the text flow is aligned to the top edge of the layout box.

Far	The bottom of the text flow is aligned to the bottom edge of the layout box.
Center	The center of the flow is aligned to the center of the layout box.

Syntax
`<TextLayout>.ParagraphAlignment`

11.7.3.1.8 ReadingDirection

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Gets or sets the current reading direction for text in a paragraph.
(See also unmanaged API documentation)

Property Value
A `SharpDX.DirectWrite.ReadingDirection enum` value that indicates the current reading direction for text in a paragraph.

Possible values are:

LeftToRight	Indicates that reading progresses from left to right.
RightToLeft	Indicates that reading progresses from right to left.

Syntax
`<TextLayout>.ReadingDirection`

11.7.3.1.9 TextAlignment

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets or sets the alignment option of text relative to the layout box's leading and trailing edge. (See also unmanaged API documentation)

Property Value

A `SharpDX.DirectWrite.TextAlignment enum` value of the current paragraph.

Possible values are:

Leading	The leading edge of the paragraph text is aligned to the leading edge of the layout box.
Trailing	The trailing edge of the paragraph text is aligned to the trailing edge of the layout box.
Center	The center of the paragraph text is aligned to the center of the layout box.
Justified	Align text to the leading side, and also justify text to fill the lines.

Syntax

```
<TextLayout>.TextAlignment
```

11.7.3.1.10 WordWrapping

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Gets or sets the word wrapping option.
(See also unmanaged API documentation)

Property Value

The `SharpDX.DirectWrite.WordWraping enum` value which determines the word wrapping option.

Possible values are:

Wrap	Indicates that words are broken across lines to avoid text overflowing the layout box.
NoWrap	Indicates that words are kept within the same line even when it overflows the layout box. This option is often used with scrolling to reveal overflow text.

Syntax

```
<TextLayout>.WordWrapping
```

11.7.3.2 LineMetrics

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some

of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Contains information about a formatted line of text.
(See also unmanaged API documentation)

Syntax

```
LineMetrics[int idx]
```

Properties

Baseline	A `float` for the distance from the top of the text line to its baseline.
Height	A `float` for the height of the text line.
IsTrimmed	A `bool` indicating the line is trimmed.
Length	An `int` value for the number of text positions in the text line. This includes any trailing whitespace and newline characters.
NewlineLength	An `int` value for the number of characters in the newline sequence at the end of the text line. If the count is zero, then the text line was either wrapped or it is the end of the text.
TrailingWhitespaceLength	Ant `int` value for the number of whitespace positions at the end of the text line. Newline sequences are considered whitespace.

11.7.3.3 TextLayout

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

The TextLayout interface represents a block of text after it has been fully analyzed and formatted.
(See also unmanaged API documentation)

Note: To draw a formatted string represented by an TextLayout object, Direct2D provides the DrawTextLayout method.

Syntax

```
class TextLayout
```

Constructors

`new TextLayout(Factory factory, string text, TextFormat textFormat, float maxWidth, float maxHeight)`	Takes a string, text format, and associated constraints, and produces an object that represents the fully analyzed and formatted result.

Tip: For NinjaScript development purposes, when creating a **TextLayout** object you should use the NinjaTrader.Core.Globals.DirectWriteFactory property

Methods and Properties

Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from SharpDX.DisposeBase.)
FlowDirection	Gets or sets the direction that text lines flow. (Inherited from TextFormat.)
FontFamilyName	Gets a copy of the font family name.(Inherited from TextFormat.)
FontSize	Gets the font size in DIP unites. (Inherited from TextFormat.)
FontStretch	Gets the font stretch of the text. (Inherited from TextFormat.)
FontStyle	Gets the font style of the text. (Inherited from TextFormat.)
FontWeight	Gets the font weight of the text. (Inherited from TextFormat.)
IsDisposed	Gets a value indicating whether this instance is disposed. (Inherited from SharpDX.DisposeBase.)
MaxHeight	Gets or sets the layout maximum height.
MaxWidth	Gets or sets the layout maximum width.
Metrics	Contains the metrics associated with text after layout. All coordinates are in device independent pixels (DIPs).

ParagraphAlignment	Gets or sets the alignment option of a paragraph which is relative to the top and bottom edges of a layout box.(Inherited from TextFormat.)
ReadingDirection	Gets or sets the current reading direction for text in a paragraph. (Inherited from TextFormat.)
TextAlignment	Gets or sets the alignment option of text relative to the layout box's leading and trailing edge. (Inherited from TextFormat.)
WordWrapping	Gets or sets the word wrapping option. (Inherited from TextFormat.)

11.7.3.3.1 GetLineMetrics()

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition

Retrieves the information about each individual text line of the text string.
(See also unmanaged API documentation)

Method Return Value

A LineMetrics[] contains a pointer to an array of structures containing various calculated length values of individual text lines.

Syntax

```
<TextLayout>.GetLineMetrics()
```

Parameters
This method does not accept any parameters

11.7.3.3.2 MaxHeight

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Gets or sets the layout maximum height.
(See also unmanaged API documentation)

Property Value
A `float` representing the text layout maximum height.

Syntax
```
<TextLayout>.MaxHeight
```

11.7.3.3.3 MaxWidth

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Gets or sets the layout maximum width.
(See also unmanaged API documentation)

Property Value
A `float` representing the text layout maximum width.

Syntax
```
<TextLayout>.MaxWidth
```

11.7.3.3.4 Metrics

Disclaimer: The SharpDX SDK Reference section was compiled from the official SharpDX Documentation and was **NOT** authored by NinjaTrader. The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**. This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly. Please refer to the official **SharpDX Documentation** for additional members not covered in this reference. For more seasoned graphic developers, the original **MSDN** Direct2D1 and DirectWrite unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only **essential** members in the structure of this page.*

Definition
Contains the metrics associated with text after layout. All coordinates are in device independent pixels (DIPs).
(See also unmanaged API documentation)

Syntax
```
<TextLayout>.Metrics
```

Properties

Left	A `float` value that indicates the left-most point of formatted text relative to the layout box, while excluding any glyph overhang.
Top	A `float` value that indicates the top-most point of formatted text

	relative to the layout box, while excluding any glyph overhang.
Width	A `float` value that indicates the width of the formatted text, while ignoring trailing whitespace at the end of each line.
WidthIncludingTrailingWhitespace	A `float` value that indicates width of the formatted text, taking into account the trailing whitespace at the end of each line.
Height	A `float` value that indicates the height of the formatted text. The height of an empty string is set to the same value as that of the default font.
LayoutWidth	A `float` value that indicates the initial width given to the layout. It can be either larger or smaller than the text content width, depending on whether the text was wrapped.
LayoutHeight	A `float` value that indicates the initial height given to the layout. Depending on the length of the text, it may be larger or smaller than the text content height.
MaxBidiReorderingDepth	An `int` value representing the maximum reordering count of any line of text, used to calculate the most number of hit-testing boxes needed. If the layout has no bidirectional text, or no text at all, the minimum level is 1.
LineCount	An `int` value representing total number of lines.

Endnotes 2... (after index)