Table of Contents

© 2017 NinjaTrader, LLC

© 2017 NinjaTrader, LLC

1 Welcome to NinjaTrader

NINJATRADER® 8

NinjaTrader is free to use for advanced charting, market analytics, backtesting and trade simulation.

The NinjaTrader 8 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 What's New in NinjaTrader 8

NinjaTrader 8 is our next generation trading platform redesigned using modern design techniques, allowing us to achieve greater performance and flexibility than ever before. Out of the box, NinjaTrader 8 incorporates over 500 changes and enhancements, largely collected from client feedback. The new version is ready to deliver the most advanced trading features for discretionary and automated traders of all levels trading stocks, futures, and forex

We have enjoyed incredible success with our pioneering strategy of offering a standards based (.NET) programming environment for indicators and strategies. The types of add-ons created by our vibrant developer community wildly surpassed our expectations and prompted us to re-think how the NinjaTrader platform could evolve. With NinjaTrader 8 we have created a true trading application development platform, allowing developers to build incredibly rich and integrated applications limited only by the imagination. We are confident that by providing formal support for deeper access into our core framework we will energize the community to build even better tools, adding significant value to our ecosystem (www.ninjatraderecosystem.com).

Although we have made large advancements with NinjaTrader 8, existing users will feel right at home as general usability has remained intact. We hope that you are as excited as we are about NinjaTrader 8. This document is a high level overview of the most significant changes in NinjaTrader 8.

▽ General

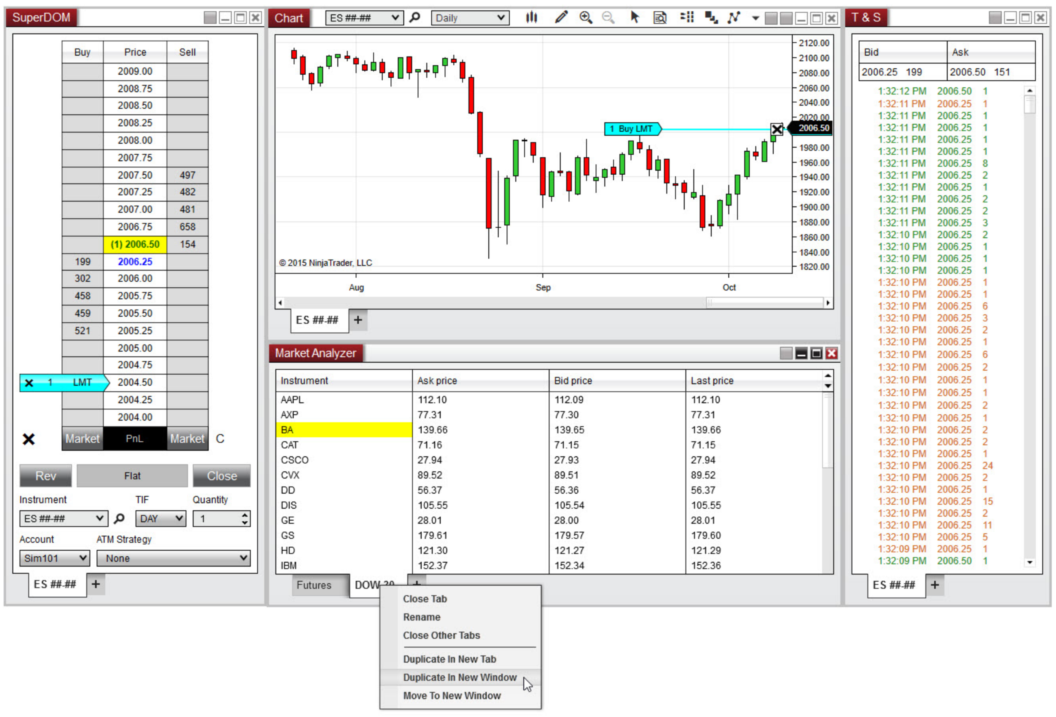
Performance Enhancements

- Upgraded to the latest Microsoft .NET 4.5 runtime environment
- NinjaTrader 8 core and UI is now fully multi-threaded, which adds significant performance increases across the entire platform
- Connectivity adapters now run in their own thread, which permits these events to run independent of the main application thread
- Changed the way data is saved in the database resulting in significant performance enhancements
- Replaced the Windows Forms UI with Windows Presentation Foundation (WPF), allowing us to take advantage of the latest UI concepts and models
- Improved optimizations in terms of open workspaces resources for efficient CPU usage

- Significant improvements on Strategy Optimization, resulting in 10x performance gains in our benchmarks
- Added support for concurrent historical bar requests, greatly improving data load time compared to NinjaTrader 7

New Tabbed Interface

We developed a new tabbed interface which is available from all trading and market analysis interfaces, optimizing the amount of screen real estate and the number of workspaces used to monitor and trade several markets at a time. Tabs can be dynamically named based on the Tab Content, such as Instrument Names, Accounts, ATM Strategies, and more, or with custom user-defined text. Existing tab content can also be duplicated into a new tab or into an entirely new window. Tabs can optionally be disabled on any individual window in order to maximize the display.



Buy	Price	Sell
	2009.00	
	2008.75	
	2008.50	
	2008.25	
	2008.00	
	2007.75	
	2007.50	497
	2007.25	482
	2007.00	481
	2006.75	658
	(1) 2006.50	154
199	2006.25	
302	2006.00	
458	2005.75	
459	2005.50	
521	2005.25	
	2005.00	
	2004.75	
1 LMT	2004.50	
	2004.25	
	2004.00	
Market	PnL	Market

Rev Flat Close

Instrument: ES ##-## TIF: DAY Quantity: 1

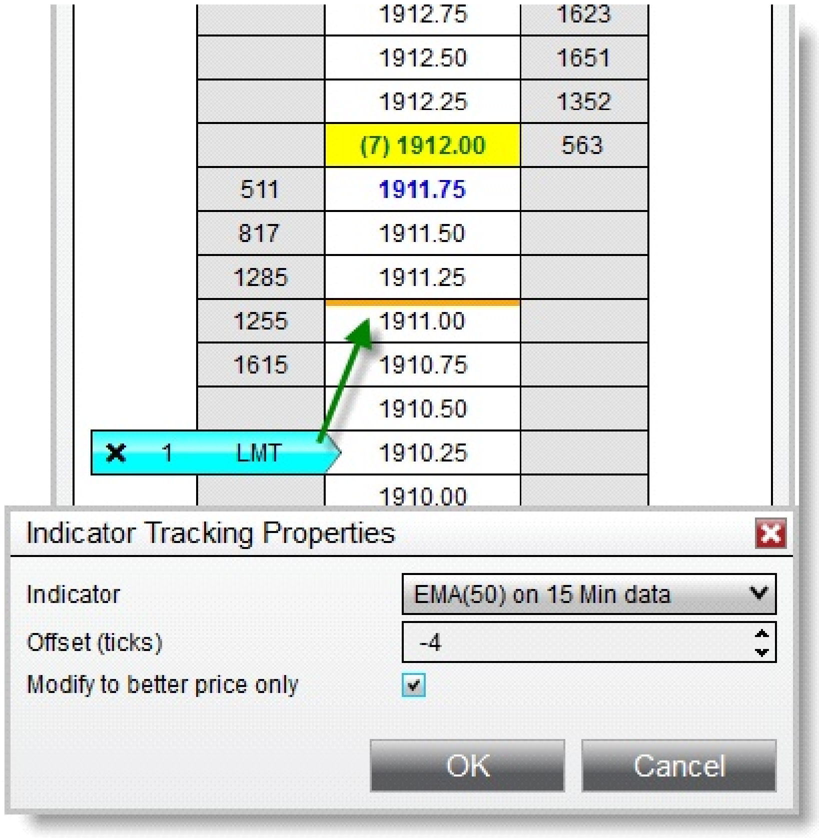
Account: Sim101 ATM Strategy: None

Instrument	Ask price	Bid price	Last price
AAPL	112.10	112.09	112.10
AXP	77.31	77.30	77.31
BA	139.66	139.65	139.66
CAT	71.16	71.15	71.15
CSCO	27.94	27.93	27.94
CVX	89.52	89.51	89.52
DD	56.37	56.36	56.37
DIS	105.55	105.54	105.55
GE	28.01	28.00	28.01
GS	179.61	179.57	179.60
HD	121.30	121.27	121.29
IBM	152.37	152.34	152.36

Bid	Ask
2006.25 199	2006.50 151

```
1:32:12 PM  2006.50  1
1:32:11 PM  2006.25  1
1:32:11 PM  2006.25  1
1:32:11 PM  2006.25  1
1:32:11 PM  2006.25  1
1:32:11 PM  2006.25  1
1:32:11 PM  2006.25  8
1:32:11 PM  2006.25  2
1:32:11 PM  2006.25  1
1:32:11 PM  2006.25  2
1:32:11 PM  2006.25  2
1:32:11 PM  2006.25  3
1:32:10 PM  2006.25  2
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  6
1:32:10 PM  2006.25  3
1:32:10 PM  2006.25  2
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  6
1:32:10 PM  2006.25  2
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  2
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  2
1:32:10 PM  2006.25  2
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  24
1:32:10 PM  2006.25  2
1:32:10 PM  2006.25  1
1:32:10 PM  2006.25  15
1:32:10 PM  2006.25  2
1:32:10 PM  2006.25  11
1:32:10 PM  2006.25  5
1:32:09 PM  2006.25  1
1:32:09 PM  2006.50  1
```

New "Attach Orders to Indicators" Feature



	1912.75	1623
	1912.50	1651
	1912.25	1352
	(7) 1912.00	563
511	1911.75	
817	1911.50	
1285	1911.25	
1255	1911.00	
1615	1910.75	
	1910.50	
× 1 LMT	1910.25	
	1910.00	

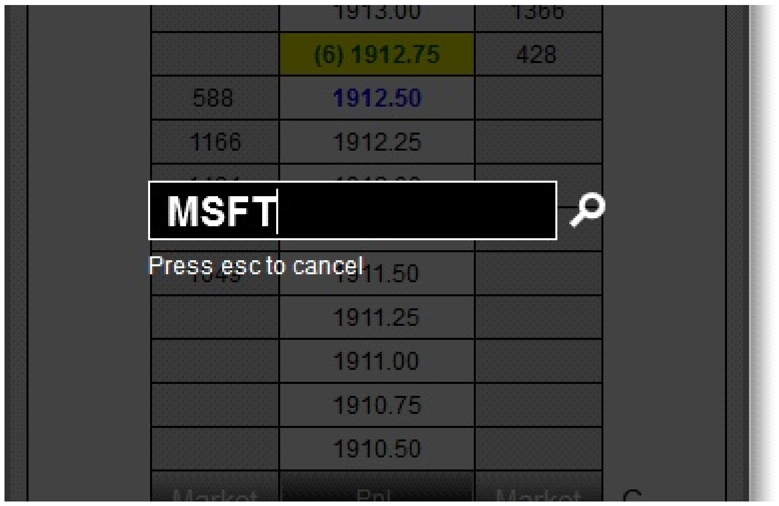
Indicator Tracking Properties

Indicator: EMA(50) on 15 Min data
Offset (ticks): -4
Modify to better price only: ✓

We introduced a new hands-free trade management concept which allows you to attach manual orders placed via Chart Trader or the SuperDOM to indicators, so that the orders will automatically follow the indicator values as they change. Configure your favorite indicator, such as an EMA, and watch as your working orders are modified to follow the price of the indicator precisely on each tick, on price change, or on bar close. Available for both manual entry and exit orders as well as ATM Strategy Stop Loss and Profit Target orders.

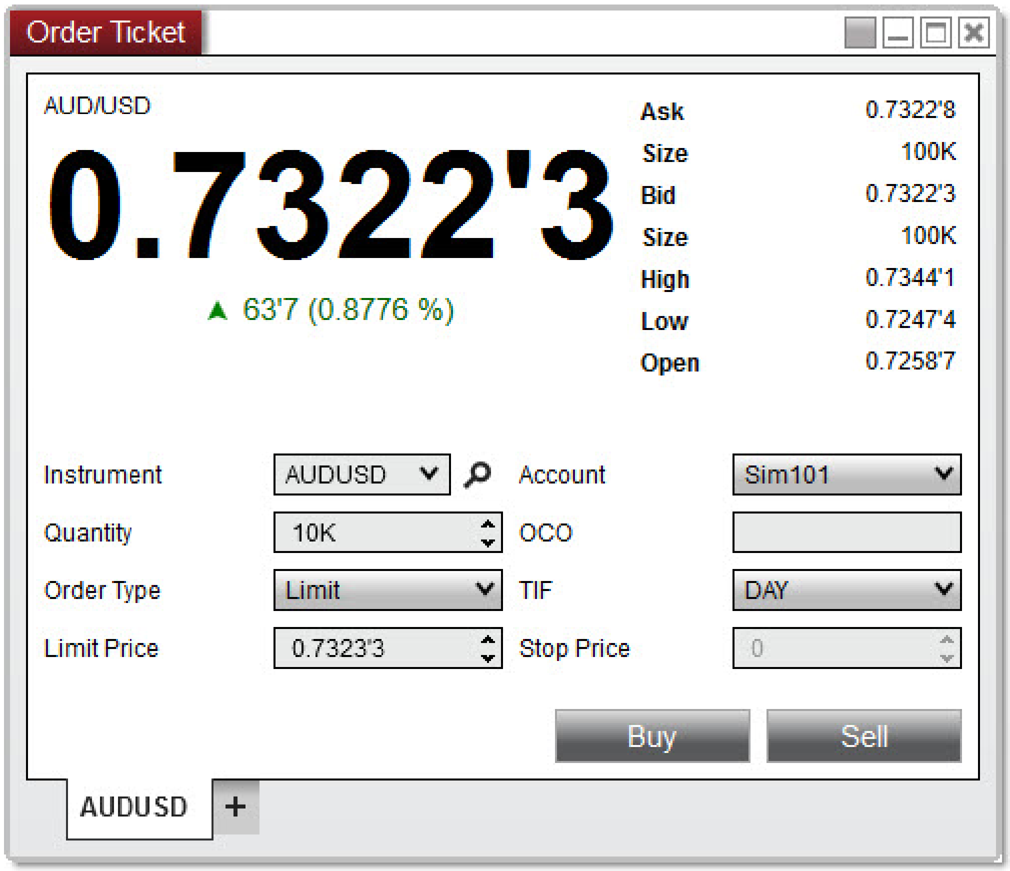
- No programming needed and defined completely through the user-interface
- Configure a tick offset to track changes below or above the indicator value
- Option to determine should the order modify to a better price only, or alternatively follow price change in either direction

New Instrument Overlay Selector



Instant instrument switching has been added to all trading and trade-analysis windows, significantly reducing the time it takes to switch an interface from one market to the next. This feature also includes a quick search button to easily navigate to the Instrument Window to look up instruments directly from the Instrument Overlay Selector. Simply start typing into any active window, and the Instrument Overlay Selector will automatically appear.

New Order Ticket Window



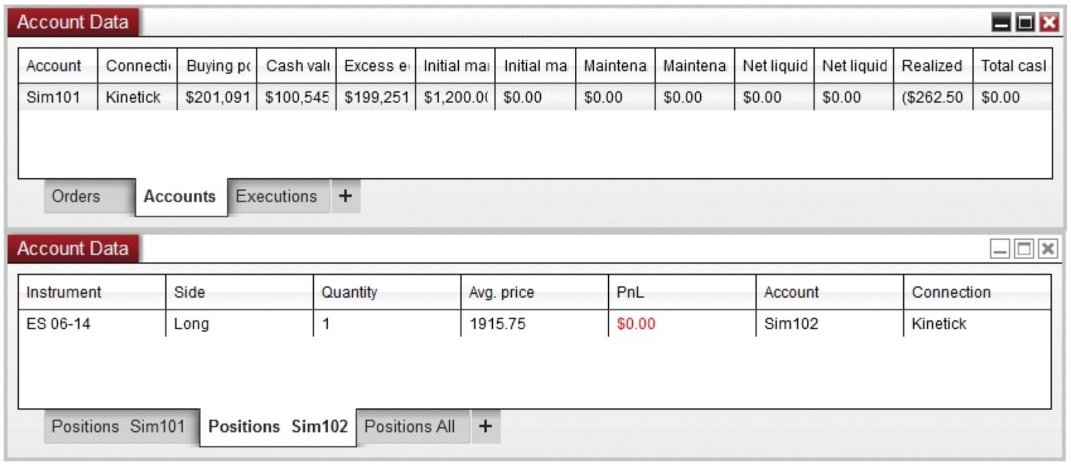
Ask	0.7322'8
Size	100K
Bid	0.7322'3
Size	100K
High	0.7344'1
Low	0.7247'4
Open	0.7258'7

Instrument	AUDUSD	Account	Sim101
Quantity	10K	OCO	
Order Type	Limit	TIF	DAY
Limit Price	0.7323'3	Stop Price	0

We designed a new Order Ticket Window which replaces the Order Entry panel from the Control Center Orders Tab. This interface is designed to work with all supported order types and includes an option to close the Order Ticket window after order submission to help keep your workspace clean.

New Account Data Window

Designed as a sister window to the Control Center, the new Account Data window works as a supplementary account data display feature giving you the ability to organize multiple tabs or windows for better account tracking and management. Tabs can be duplicated from the Control Center to a new Account Data window, which performs user defined filtering based on connected accounts. You can filter each window or tab independently, allowing multiple windows and tabs to show data for individual accounts, including Orders, Strategies, Executions, Positions and Account Balances.



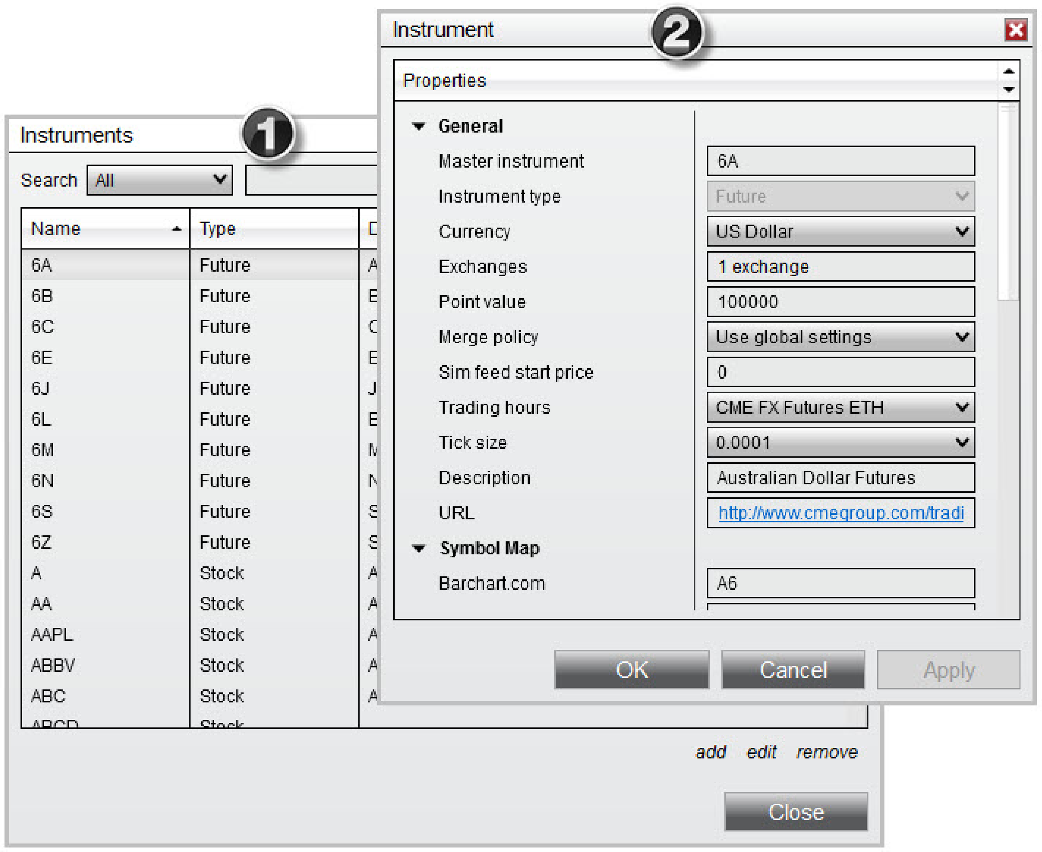
Account	Connecti	Buying pc	Cash valı	Excess e	Initial mai	Initial ma	Maintena	Maintena	Net liquid	Net liquid	Realized	Total casl
Sim101	Kinetick	$201,091	$100,545	$199,251	$1,200.0(	$0.00	$0.00	$0.00	$0.00	$0.00	($262.50	$0.00

Instrument	Side	Quantity	Avg. price	PnL	Account	Connection
ES 06-14	Long	1	1915.75	$0.00	Sim102	Kinetick

Streamlined Instrument Management

The methods used to manage instruments in NinjaTrader 8 have been renovated to evolve into a much more mature system.

We've introduced the flexibility to pull instrument updates directly from our servers, ensuring users always have the latest broker and data feed symbol mapping, as well as any other exchange mandated changes, such as exchange hours and rollover dates. Any custom changes made by the user will not be impacted by server changes, which give users the flexibility to customize their own local instrument database, unless they optionally select to reset these settings to the latest server defaults.



Search: All

Name	Type	D
6A	Future	A
6B	Future	E
6C	Future	C
6E	Future	E
6J	Future	J
6L	Future	E
6M	Future	M
6N	Future	N
6S	Future	S
6Z	Future	S
A	Stock	A
AA	Stock	A
AAPL	Stock	A
ABBV	Stock	A
ABC	Stock	A
ABCD	Stock	

General

Master instrument	6A
Instrument type	Future
Currency	US Dollar
Exchanges	1 exchange
Point value	100000
Merge policy	Use global settings
Sim feed start price	0
Trading hours	CME FX Futures ETH
Tick size	0.0001
Description	Australian Dollar Futures
URL	http://www.cmegroup.com/tradi

Symbol Map

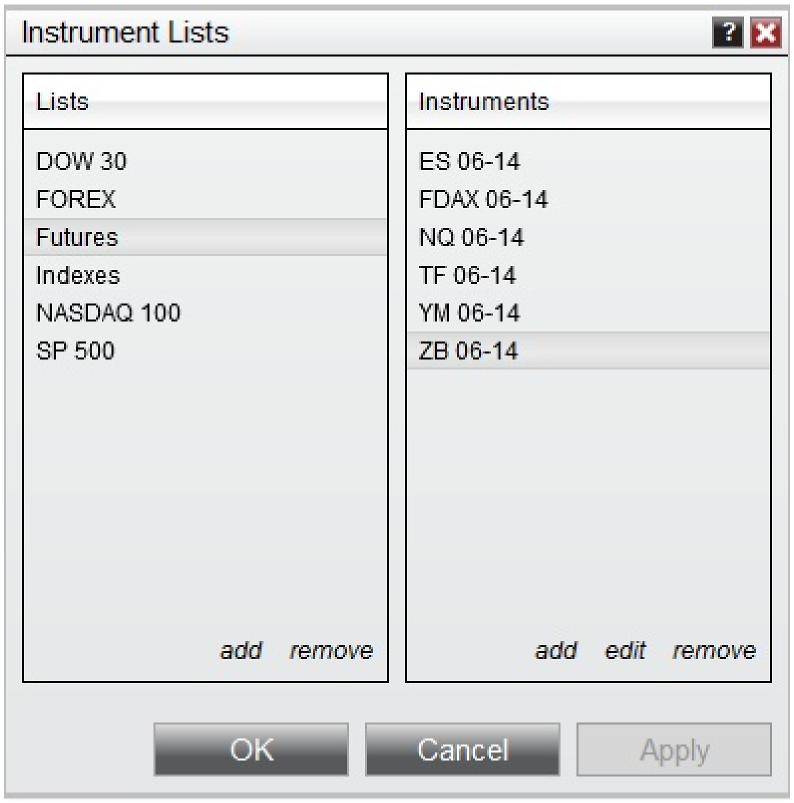
Barchart.com	A6

The Instrument Manager Window ❶ and Instrument Editor ❷ were both modernized and simplified.

- Auto search while typing, streamlining the search process
- Added multi-select capability to allow for bulk editing and management of instruments
- Multi-select available to interfaces which would support multiple instruments, such as the Market Analyzer which allows for faster selection of desired instruments

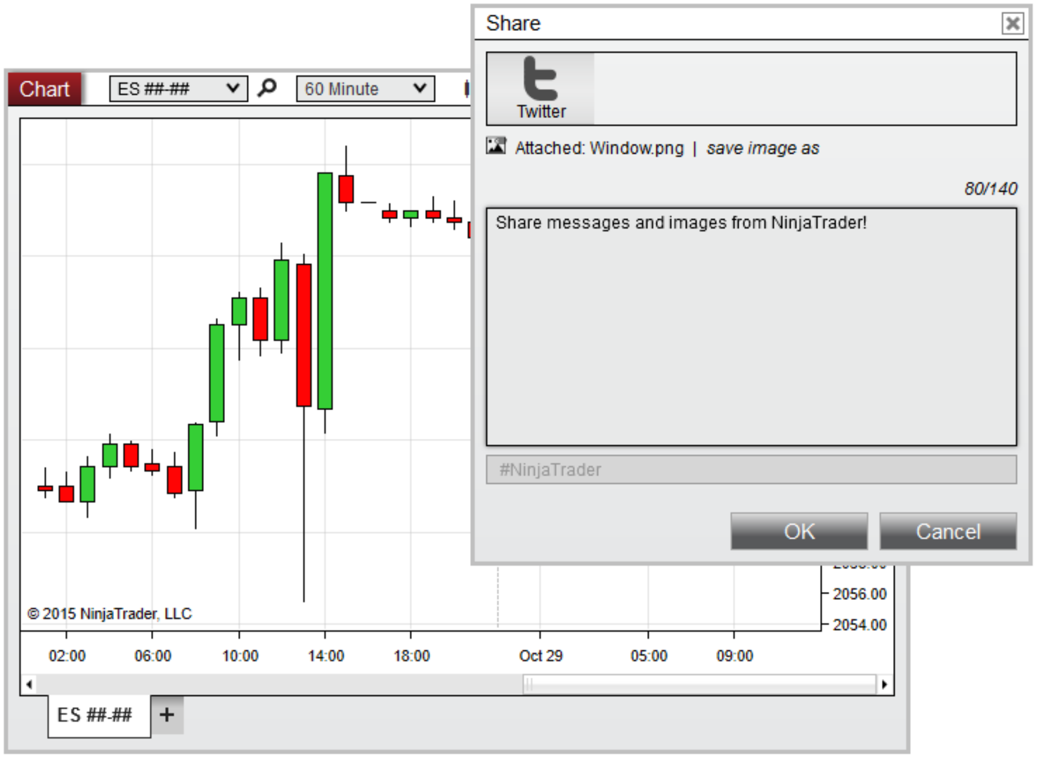
New Instrument Lists Window

We designed a new menu for improved management of Instrument Lists. The multi-select feature allows you to dynamically add, edit, or remove multiple instruments to a list at once.



New Sharing Services

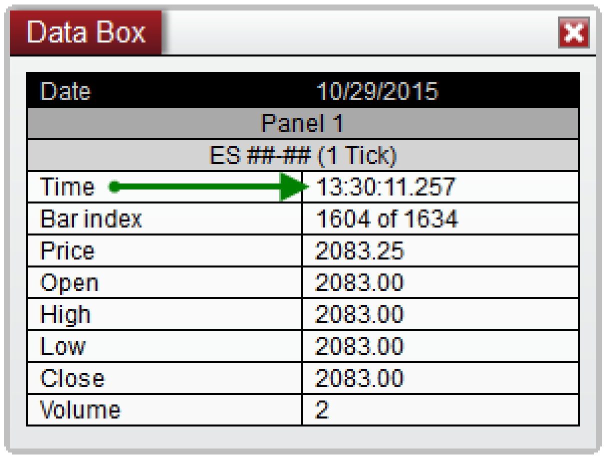
We’ve added an application-wide interface to share NinjaTrader 8 content via various social-media outlets. Users can share custom messages, images, and other content of any window, chart, or grid using the Share interface accessible from the right-click menu in any window. Services for Facebook, Twitter, Stocktwits, and Email come pre-built, and NinjaScript developers can build their own Service through NinjaScript.



Share messages and images from NinjaTrader!

New Sub-Second Granularity

Market data time stamps are now processed and stored to the .NET 'Tick' which is the equivalent of 100 nanosecond resolution, allowing for much finer granularity when timing orders and working with price data.



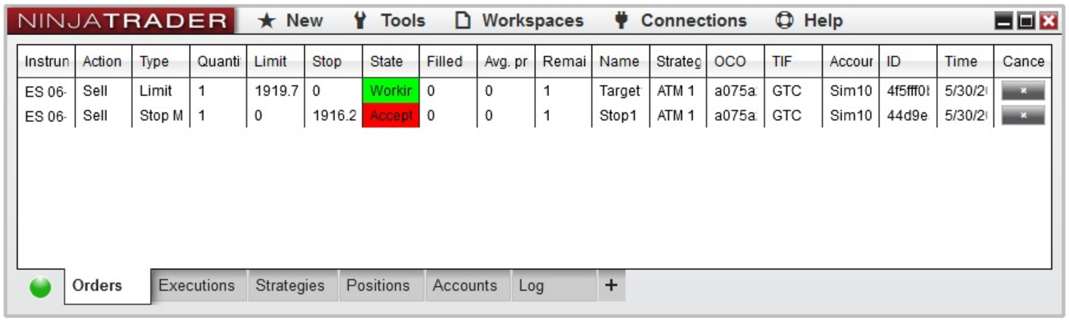
Date	10/29/2015
Panel 1	
ES ##-## (1 Tick)	
Time	13:30:11.257
Bar index	1604 of 1634
Price	2083.25
Open	2083.00
High	2083.00
Low	2083.00
Close	2083.00
Volume	2

New Historical Bid/Ask Data Per Tick

Historical Bid/Ask data is now stored with each last trade tick data. For NinjaScript developers, you can now add data series with a Bid or Ask price type into your indicators or strategies, allowing you to access historical Bid/Ask data per tick.

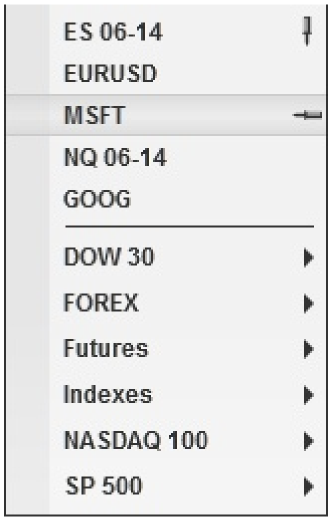
Control Center

We have redesigned the NinjaTrader 8 Control Center, which allows for quicker and more intuitive navigation, as well as a more efficient workflow. Connections and Workspaces menus have been moved directly to the Main Menu bar for easy access, and the Account Performance tab has been moved to its own dedicated Window (Trade Performance). In addition, the tabbed interface of the Control Center has been replicated across all trading and market-data windows, allowing for greater flexibility in comparing instruments or other data within a single window.



Instrun	Action	Type	Quanti	Limit	Stop	State	Filled	Avg. pr	Remai	Name	Strateg	OCO	TIF	Accour	ID	Time	Cance
ES 06-	Sell	Limit	1	1919.7	0	Workir	0	0	1	Target	ATM 1	a075a	GTC	Sim10	4f5fff0	5/30/2	×
ES 06-	Sell	Stop M	1	0	1916.2	Accept	0	0	1	Stop1	ATM 1	a075a	GTC	Sim10	44d9e	5/30/2	×

Enhanced Instrument Selector



We updated our Instrument Selector to be standardized across all order-entry windows, and it now saves recently viewed instruments for quick access later. We also introduced a new feature to pin your favorite instruments to the quick-access list, replacing the former concept of a "default" instrument list. Likewise, we removed the requirement for the default instrument list, which means you can now access any custom list and your favorite instruments from any interface.

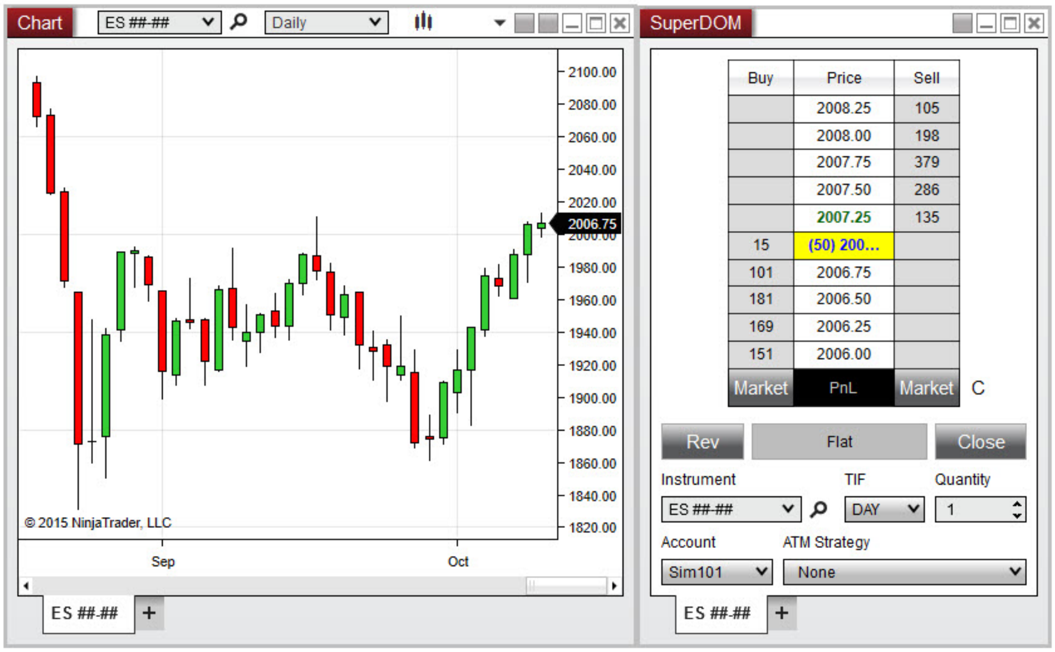
Application Options

- Improved “Sounds” preferences to allow users to add their own custom sound files
- Option for sounds to “Play consecutively,” to prevent synchronized sound triggers from overlapping each other
- “Simulation Color” is now a global property, adding consistency to trading interfaces
- Enhanced the “Auto Close” feature to enable users to specify a list of instruments on which to close positions at a specific time
- Added “Confirm on Window/Tab Close” option, which will prevent the unintentional closing of windows
- Added support for multiple languages. German, Spanish, and Russian will be included at launch, with the ability to add more languages in the future

Application Skins

We've added the ability to apply different "skins" to customize the look of NinjaTrader 8. In addition to the five pre-built skins available in the platform, users can create their own custom skins for endless customizability of colors, margins, and other layout elements.

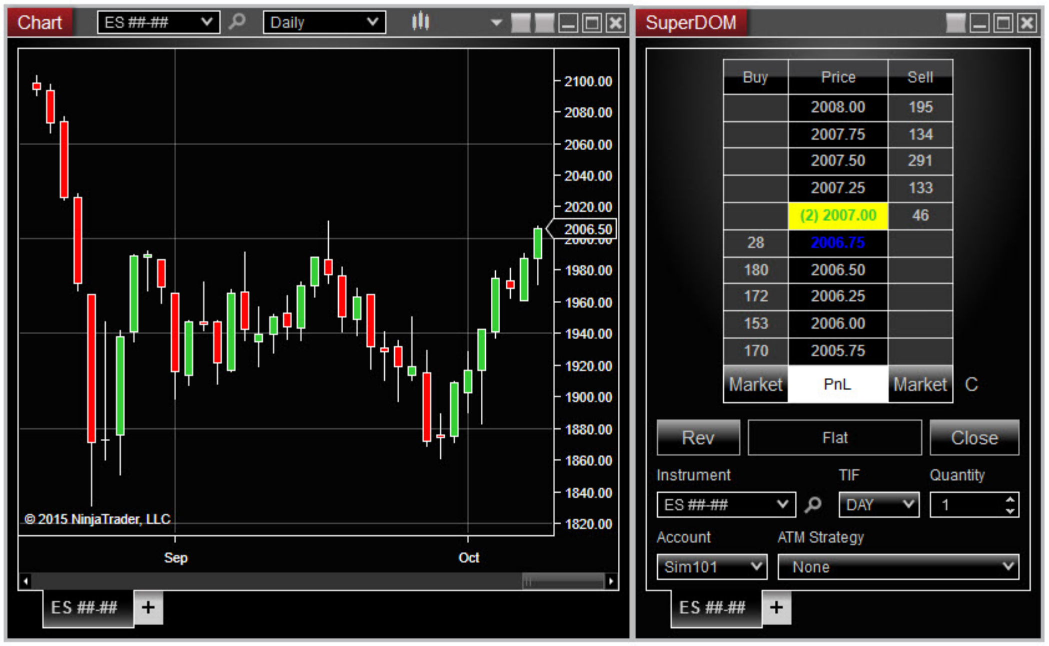
Light Skin



Buy	Price	Sell
	2008.25	105
	2008.00	198
	2007.75	379
	2007.50	286
	2007.25	135
15	(50) 200...	
101	2006.75	
181	2006.50	
169	2006.25	
151	2006.00	
Market	PnL	Market

Instrument: ES ##-##
TIF: DAY
Quantity: 1
Account: Sim101
ATM Strategy: None

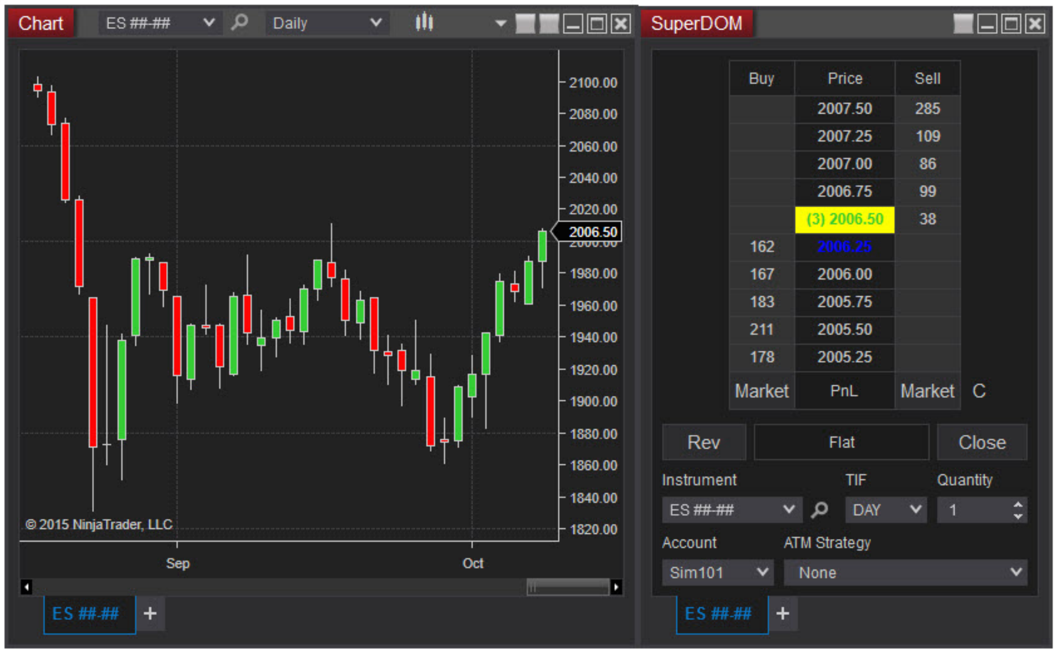
Dark Skin



Buy	Price	Sell
	2008.00	195
	2007.75	134
	2007.50	291
	2007.25	133
	(2) 2007.00	46
28	2006.75	
180	2006.50	
172	2006.25	
153	2006.00	
170	2005.75	
Market	PnL	Market

Instrument: ES ##-##
TIF: DAY
Quantity: 1
Account: Sim101
ATM Strategy: None

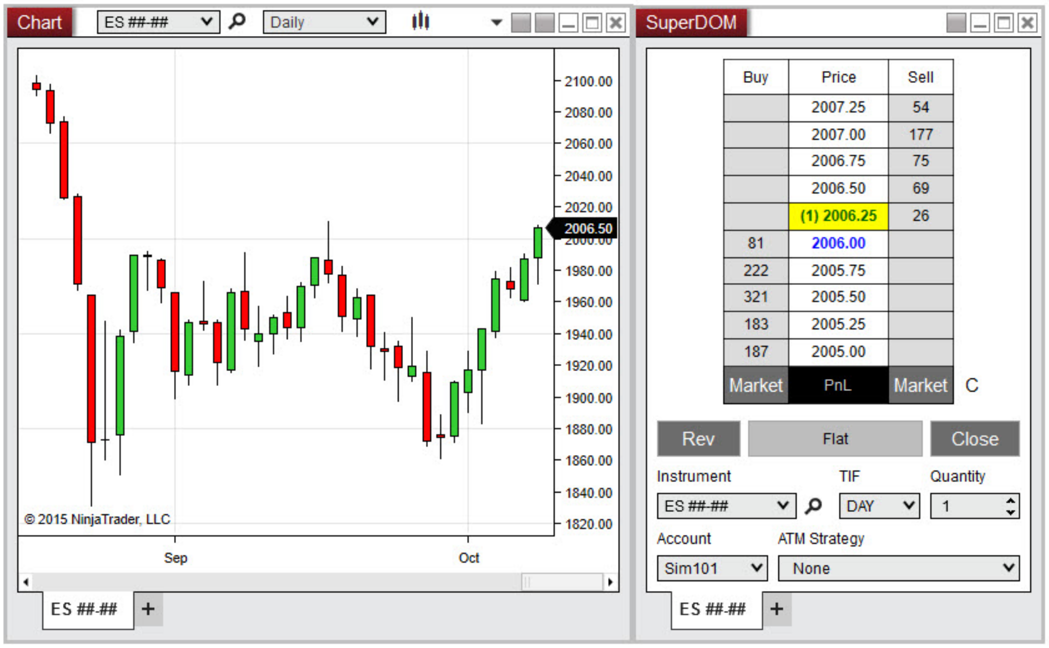
Slate Gray Skin



Buy	Price	Sell
	2007.50	285
	2007.25	109
	2007.00	86
	2006.75	99
	(3) 2006.50	38
162	2006.25	
167	2006.00	
183	2005.75	
211	2005.50	
178	2005.25	
Market	PnL	Market

Instrument: ES ##-##
TIF: DAY
Quantity: 1
Account: Sim101
ATM Strategy: None

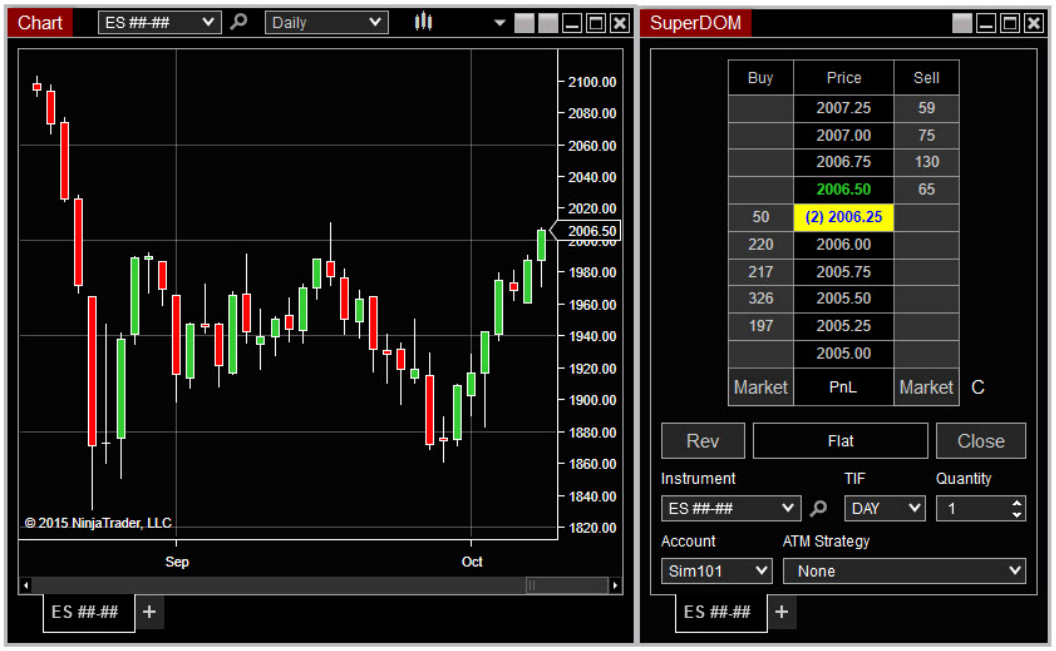
Slate Light Skin



Buy	Price	Sell
	2007.25	54
	2007.00	177
	2006.75	75
	2006.50	69
	(1) 2006.25	26
81	2006.00	
222	2005.75	
321	2005.50	
183	2005.25	
187	2005.00	
Market	PnL	Market

Instrument: ES ##-##
TIF: DAY
Quantity: 1
Account: Sim101
ATM Strategy: None

Slate Dark Skin

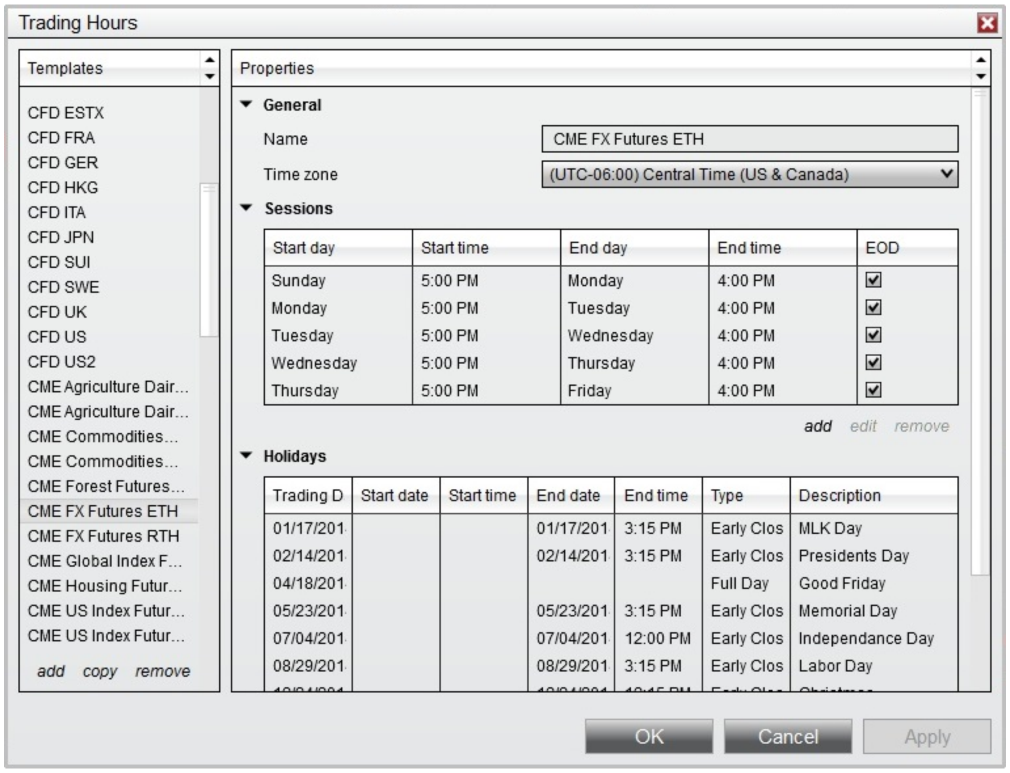


Buy	Price	Sell
	2007.25	59
	2007.00	75
	2006.75	130
	2006.50	65
50	(2) 2006.25	
220	2006.00	
217	2005.75	
326	2005.50	
197	2005.25	
	2005.00	
Market	PnL	Market

Instrument: ES ##-##
TIF: DAY
Quantity: 1
Account: Sim101
ATM Strategy: None

New Trading Hours

Formerly named the “Session Manager”, our new Trading Hours window was designed for easier management of the templates used for charting, indicator calculations, real-time strategy execution, and strategy backtesting periods. Each session now has an “EOD” (End of Day) option to support multiple trading session definitions within a single day. We also added Holiday support to handle various scheduled breaks in exchange hours (Early Close, Late Open, or Full Day Holiday).


Trading Hours

Templates

CFD ESTX
CFD FRA
CFD GER
CFD HKG
CFD ITA
CFD JPN
CFD SUI
CFD SWE
CFD UK
CFD US
CFD US2
CME Agriculture Dair...
CME Agriculture Dair...
CME Commodities...
CME Commodities...
CME Forest Futures...
CME FX Futures ETH
CME FX Futures RTH
CME Global Index F...
CME Housing Futur...
CME US Index Futur...
CME US Index Futur...

add copy remove

Properties

General

Name	CME FX Futures ETH
Time zone	(UTC-06:00) Central Time (US & Canada)

Sessions

Start day	Start time	End day	End time	EOD
Sunday	5:00 PM	Monday	4:00 PM	✓
Monday	5:00 PM	Tuesday	4:00 PM	✓
Tuesday	5:00 PM	Wednesday	4:00 PM	✓
Wednesday	5:00 PM	Thursday	4:00 PM	✓
Thursday	5:00 PM	Friday	4:00 PM	✓

add edit remove

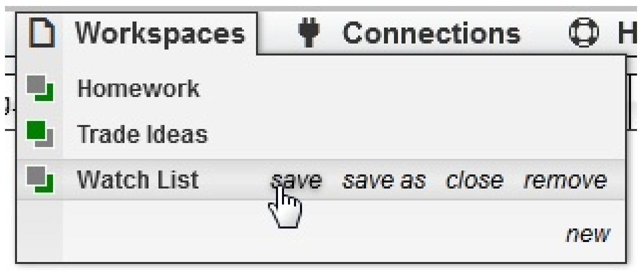
Holidays

Trading D	Start date	Start time	End date	End time	Type	Description
01/17/201			01/17/201	3:15 PM	Early Clos	MLK Day
02/14/201			02/14/201	3:15 PM	Early Clos	Presidents Day
04/18/201					Full Day	Good Friday
05/23/201			05/23/201	3:15 PM	Early Clos	Memorial Day
07/04/201			07/04/201	12:00 PM	Early Clos	Independance Day
08/29/201			08/29/201	3:15 PM	Early Clos	Labor Day

OK Cancel Apply

Sessions and Holidays can now be downloaded and updated from NinjaTrader servers, providing the ability for exchange mandated changes to be pushed to all users, rather than needing to reconfigure these updates individually.

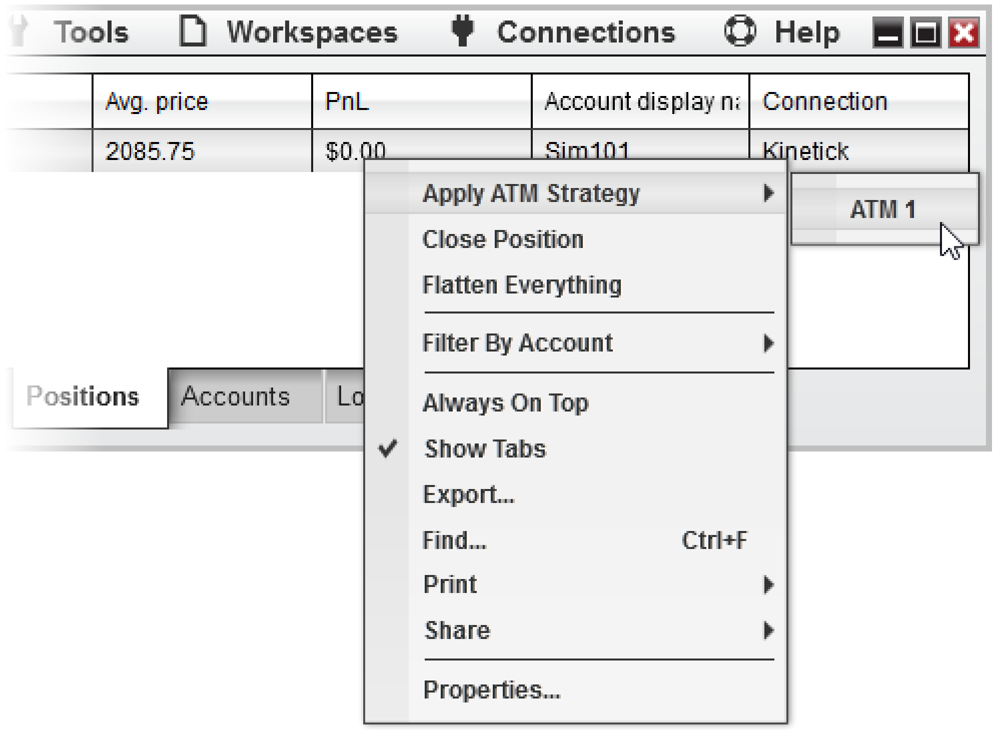
Changes to Managing Workspaces



The Workspaces menu has been integrated directly into Control Center toolbar for faster switching and workspace management. Inactive workspaces are visible from the Workspaces menu for a more functional approach to workspace organization. All windows opened outside of the viewable range of a monitor can be moved back in view of the primary screen with a single click.

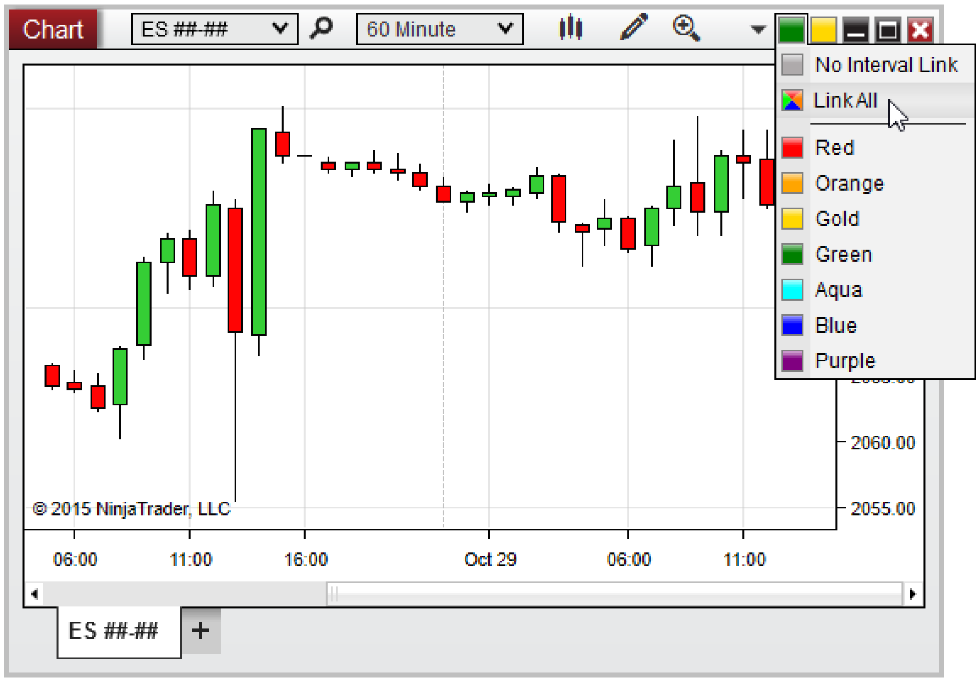
New Feature to Apply an ATM Strategy to an Unprotected Position

From the Positions grid, you can now apply an ATM Strategy to an open unprotected position. This allows you to add a layer of semi-automated risk management to a position after it has been filled.



Improved Instrument Linking

- Added “Link All Mode” to group changes to specific window
- New “Interval Link” which allows for simultaneous changing time frames on charts
- Added “Global Link button across workspaces” allowing users to keep instruments in separate workspaces unlinked if desired



Miscellaneous Enhancements

- Added support for CFD's as a new Asset type available for supported brokerage technologies (FXCM, Interactive Brokers as of this writing)
- Improvements regarding window sizing. Now all non-modal windows are resizable to user preferences
- Added support for Market-if-Touched (MIT) orders, which can be used as entries or as Profit Targets in ATM Strategies

Forex

General Enhancements

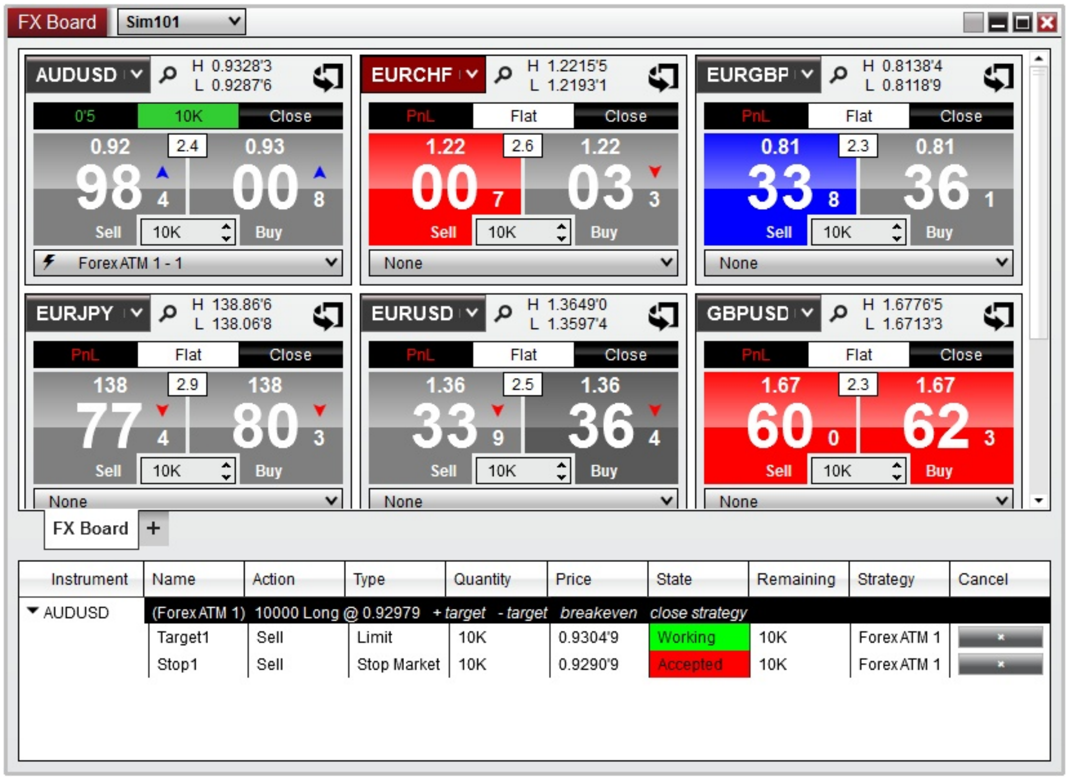
- Improved internal multi-currency rate conversion for accurate profit and loss reporting and added a configurable currency denomination setting per Forex and CFD account connection
- The Quantity Selector is aware of Forex Lot Sizes when scrolling up/down

with a Forex instrument selected

- Forex Lot Size is pulled from account automatically or manually selected per connection
- Strategy backtests for Forex instruments are now normalized by account lot size for more accurate reporting

New FX Board

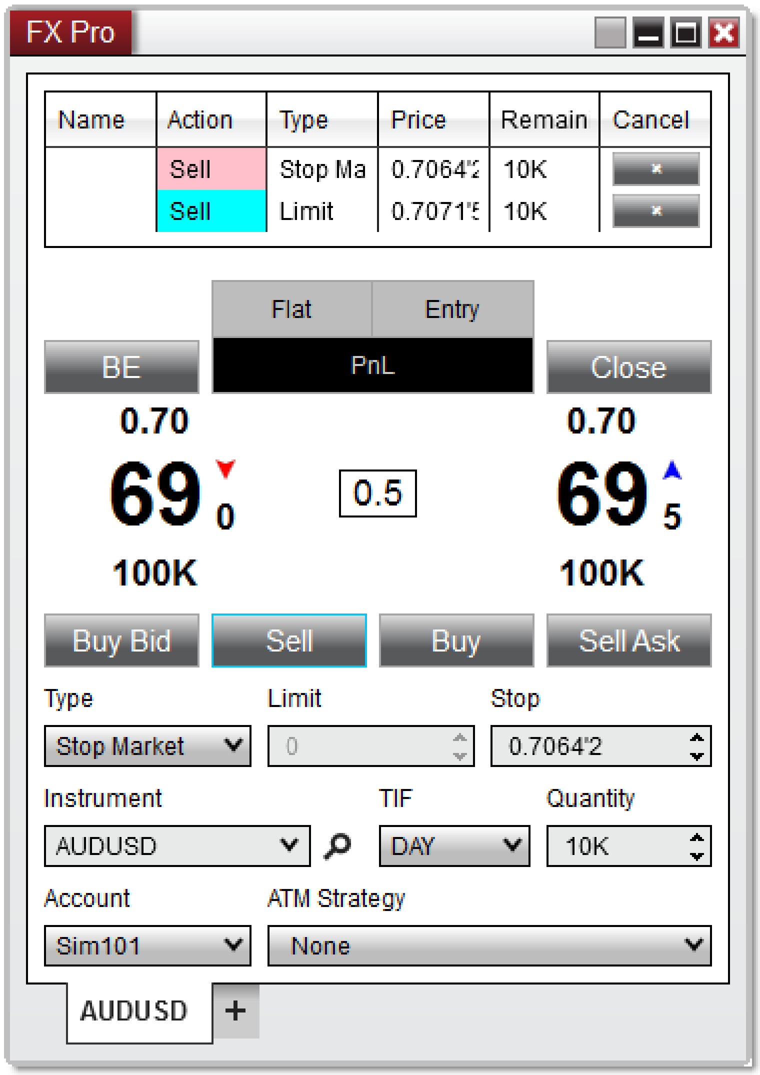
We fully conceptualized our take on a new market data and trading interface designed specifically for Forex and CFD products -- The FX Board. The market display components are laid out as a number of tiles, allowing you to view multiple instrument tiles at once. Each tile will be highlighted as the bid or ask price updates, to represent either an up-tick or down-tick in price. These instrument tiles double as an quick order entry interface, as well, allowing you to quickly place buy/ sell Market and Limit orders at current market prices. Flip the tile around to expose a more robust manual order entry feature that enables you to place orders at a specific price level. You will also view current account position, profit and loss, and open orders using the Orders Grid.



Instrument	Name	Action	Type	Quantity	Price	State	Remaining	Strategy	Cancel
AUDUSD	(Forex ATM 1)	10000 Long @ 0.92979	+target	-target	breakeven	close strategy			
	Target1	Sell	Limit	10K	0.9304'9	Working	10K	Forex ATM 1	×
	Stop1	Sell	Stop Market	10K	0.9290'9	Accepted	10K	Forex ATM 1	×

FX Pro Window

The FX Pro window has received a number of design updates for a more visible and readable Forex quote display that is consistent with FX Board. The order controls received a few tweaks, such adding the ability to manually edit the Limit and Stop price fields, and quick controls to bring in the current bid and ask price.



Name	Action	Type	Price	Remain	Cancel
	Sell	Stop Ma	0.7064'2	10K	×
	Sell	Limit	0.7071'5	10K	×

Type: Stop Market
Limit: 0
Stop: 0.7064'2
Instrument: AUDUSD
TIF: DAY
Quantity: 10K
Account: Sim101
ATM Strategy: None

Account Management

In addition to the new Account Data window, we have taken steps to improve the mechanics of the account data grids and displays.

Improved Order Grids

- Added GTD date display to TIF order column
- Increase/Decrease menu now located in the right-click menu by default (option to re-enable on the grid)
- Editing order price and quantity now uses a fully featured editor
- Active orders are now displayed by color, according to order types

Improved Strategies Tab

NinjaScript strategies now have the ability to synchronize the strategy position to adopt the real world account position. This improvement allows users to re-start their strategy completely in tune with their live account, without having to recalculate the strategy when enabling, after restarting, or when making changes to the strategy.

- Added a "Synchronize All Strategies" feature
- Added Account Position and “Sync” display Columns
- Added an option to edit the Instrument on which a strategy is running

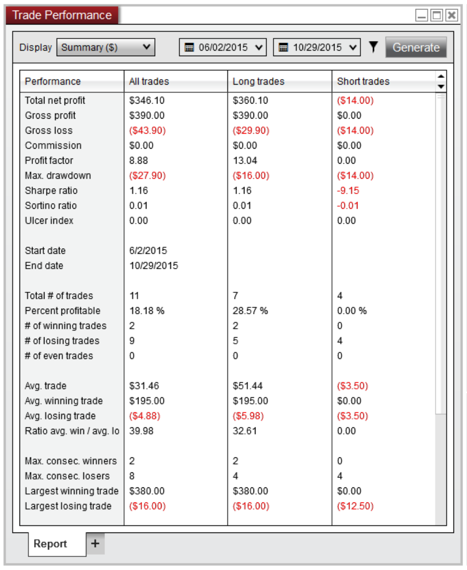
Trade Performance Window

The Account Performance tab was removed from the Control Center and redesigned as a dedicated window (Trade Performance) that can be saved to the workspace in order to recall previous settings, and which permits multiple window instances and multiple tabs within each window for a more thorough analysis and comparison of different reports. We added a number of additional features and enhancements and also various bug fixes regarding display and calculation of data.

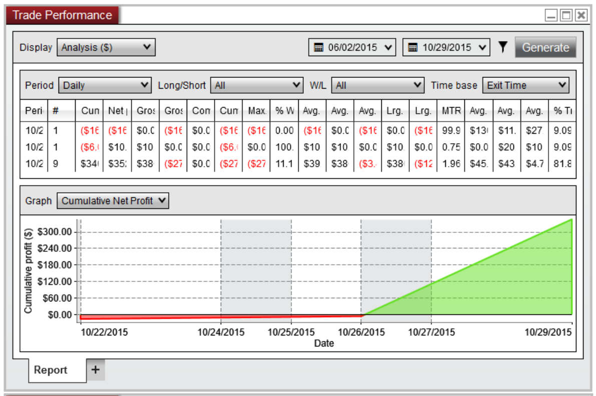
- Added Statistics:
  - Ulcer index
  - Sortino Ratio
  - Longest Flat Period
  - Number of Even Trades

   - Ability to add your own custom statistics

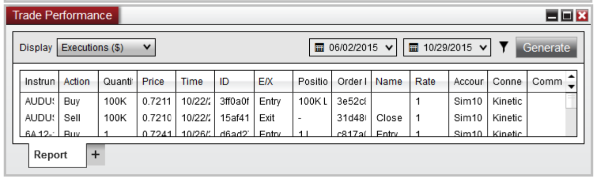
- Added new Pips and Ticks Display Modes in addition to the existing Currency, Percent, and Points modes
- Added the ability to add or remove executions directly from the Trade Performance window
- Added the ability to mark executions and trades with custom comments to be displayed in the Journal display



Performance	All trades	Long trades	Short trades
Total net profit	$346.10	$360.10	($14.00)
Gross profit	$390.00	$390.00	$0.00
Gross loss	($43.90)	($29.90)	($14.00)
Commission	$0.00	$0.00	$0.00
Profit factor	8.88	13.04	0.00
Max. drawdown	($27.90)	($16.00)	($14.00)
Sharpe ratio	1.16	1.16	-9.15
Sortino ratio	0.01	0.01	-0.01
Ulcer index	0.00	0.00	0.00
Start date	6/2/2015		
End date	10/29/2015		
Total # of trades	11	7	4
Percent profitable	18.18 %	28.57 %	0.00 %
# of winning trades	2	2	0
# of losing trades	9	5	4
# of even trades	0	0	0
Avg. trade	$31.46	$51.44	($3.50)
Avg. winning trade	$195.00	$195.00	$0.00
Avg. losing trade	($4.88)	($5.98)	($3.50)
Ratio avg. win / avg. lo	39.98	32.61	0.00
Max. consec. winners	2	2	0
Max. consec. losers	8	4	4
Largest winning trade	$380.00	$380.00	$0.00
Largest losing trade	($16.00)	($16.00)	($12.50)



Peri	#	Cun	Net	Gros	Gros	Con	Cun	Max.	% W	Avg.	Avg.	Avg.	Lrg.	Lrg.	MTR	Avg.	Avg.	Avg.	% Ti
10/2	1	($16	($16	$0.0	($16	$0.0	($16	($16	0.00	($16	$0.0	($16	$0.0	($16	99.9	$13	$11.	$27	9.09
10/2	1	($6.	$10.	$10	$0.0	$0.0	($6.	$0.0	100.	$10	$10	$0.0	$10	$0.0	0.75	$0.0	$20	$10	9.09
10/2	9	$34	$35	$38	($27	$0.0	($27	($27	11.1	$39	$38	($3.	$38	($12	1.96	$45.	$43	$4.7	81.8



Instrun	Action	Quanti	Price	Time	ID	E/X	Positio	Order I	Name	Rate	Accour	Conne	Comm
AUDU	Buy	100K	0.7211	10/22/	3ff0a0f	Entry	100K L	3e52c		1	Sim10	Kinetic	
AUDU	Sell	100K	0.7210	10/22/	15af41	Exit	-	31d48	Close	1	Sim10	Kinetic	
6A 12-	Buy	1	0.7241	10/26/	d6ad2	Entry	1 L	c817a	Entry	1	Sim10	Kinetic	

New Analysis Display

We stylized the existing graphs used in both the Account Performance window and the Strategy Analyzer to make it easier to analyze trade data and also added a new "Analysis" display which integrates Periods and Graph analysis which are interactive with the period selection.

- Added Cumulative Max Drawdown and Max Drawdown graphs
- Analysis can be based on Entry or Exit times
- Filter by long/short and winning/losing trades

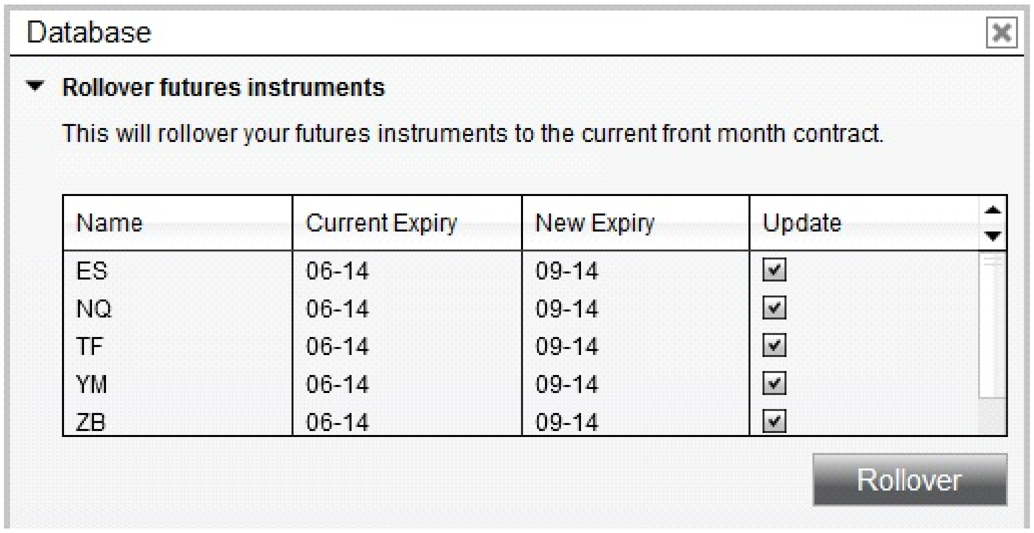
▽ Database Management

New Database Window

We added a new window to centralize the management of all database operations, along with the added capability to update Instruments, Instrument Lists, and Trading Hours directly from NinjaTrader 8 servers. Automatic updating of database items removes the need to manually edit or reset instruments, ensuring that database items can always be up-to-date with a single click.

New Automatic Rollover Feature

Automatically roll over future contracts to streamline the rollover process between contract months. This database feature goes through all instrument lists and finds the futures that are ready to roll over based on their rollover dates, and will update your existing lists when requested.

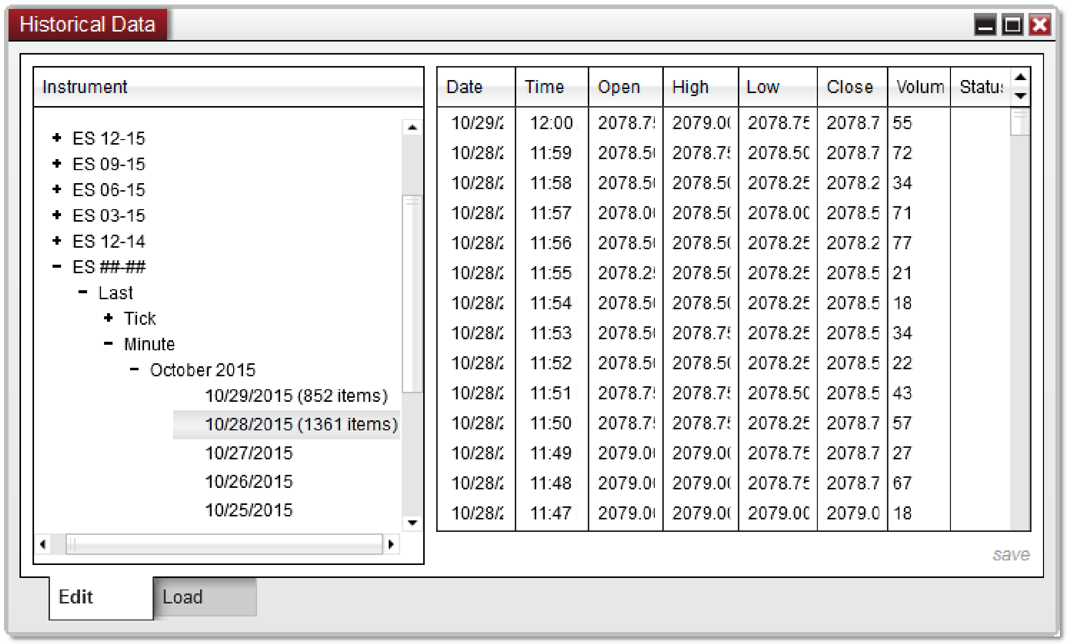


This will rollover your futures instruments to the current front month contract.

Name	Current Expiry	New Expiry	Update
ES	06-14	09-14	✓
NQ	06-14	09-14	✓
TF	06-14	09-14	✓
YM	06-14	09-14	✓
ZB	06-14	09-14	✓

New Historical Data Window

The Historical Data Manager has been redesigned, and renamed to the Historical Data window. This update centralizes the management of all data types, including Playback data, which was previously managed in a separate window. The Historical Data window will allow users to visualize and edit both their historical and Playback data, as well as import, export, and download new data, all from a single interface.



Historical Data

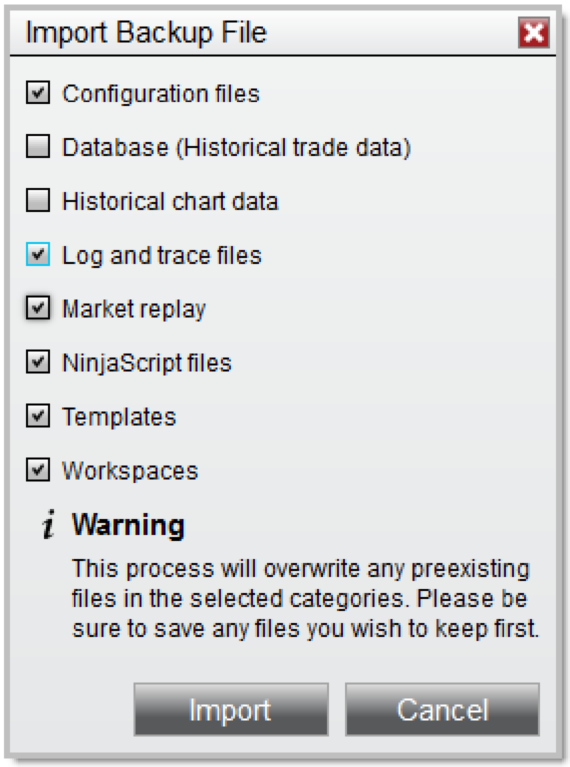
Instrument

- ES 12-15
- ES 09-15
- ES 06-15
- ES 03-15
- ES 12-14
- ES ##-##
  - Last
    - Tick
    - Minute
      - October 2015
        - 10/29/2015 (852 items)
        - 10/28/2015 (1361 items)
        - 10/27/2015
        - 10/26/2015
        - 10/25/2015

Date	Time	Open	High	Low	Close	Volum	Statu
10/29/	12:00	2078.7	2079.0	2078.75	2078.7	55	
10/28/	11:59	2078.5	2078.7	2078.50	2078.7	72	
10/28/	11:58	2078.5	2078.5	2078.25	2078.2	34	
10/28/	11:57	2078.0	2078.5	2078.00	2078.5	71	
10/28/	11:56	2078.5	2078.5	2078.25	2078.2	77	
10/28/	11:55	2078.2	2078.5	2078.25	2078.5	21	
10/28/	11:54	2078.5	2078.5	2078.25	2078.5	18	
10/28/	11:53	2078.5	2078.7	2078.25	2078.5	34	
10/28/	11:52	2078.5	2078.5	2078.25	2078.5	22	
10/28/	11:51	2078.7	2078.7	2078.50	2078.5	43	
10/28/	11:50	2078.7	2078.7	2078.25	2078.7	57	
10/28/	11:49	2079.0	2079.0	2078.75	2078.7	27	
10/28/	11:48	2079.0	2079.0	2078.75	2078.7	67	
10/28/	11:47	2079.0	2079.0	2079.00	2079.0	18	

Import/Export

- Added options to select which items to restore from an existing backup archive
- Now does a scheduled backup on its own without manual interaction


Import Backup File

- [✓] Configuration files
- [ ] Database (Historical trade data)
- [ ] Historical chart data
- [✓] Log and trace files
- [✓] Market replay
- [✓] NinjaScript files
- [✓] Templates
- [✓] Workspaces

Warning

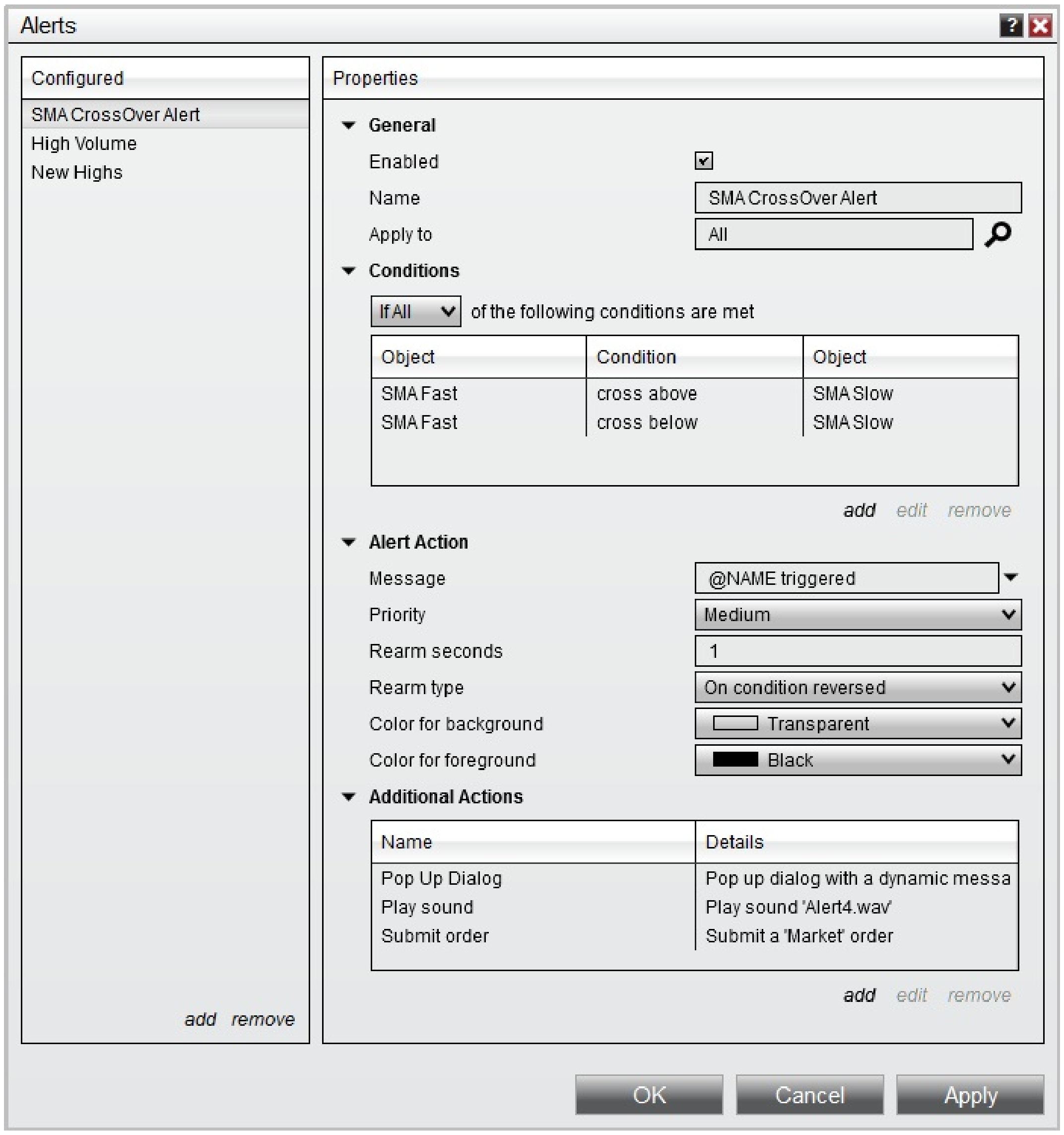
This process will overwrite any preexisting files in the selected categories. Please be sure to save any files you wish to keep first.

Market Research

New Advanced Alert Features

Our alerts functionality has been completely redesigned to allow the execution of complex conditions without programming of any kind. Multi-object alert conditions are now supported, allowing you to compare two data series or indicators. Alert conditions have been extended to work natively on charts, and now manually drawn Chart Objects can be used as an input condition for any alerts.

Alerts are also flexible, in that you have the option to apply alerts to all instruments, or even a specific typeset of instruments, as well as define “if all” or “if any” conditions would need to be satisfied in order for the alert to trigger.


Alerts

Configured
- SMA CrossOver Alert
- High Volume
- New Highs

add remove

Properties

General
- Enabled: ✓
- Name: SMA CrossOver Alert
- Apply to: All

Conditions

If All of the following conditions are met

Object	Condition	Object
SMA Fast	cross above	SMA Slow
SMA Fast	cross below	SMA Slow

add edit remove

Alert Action
- Message: @NAME triggered
- Priority: Medium
- Rearm seconds: 1
- Rearm type: On condition reversed
- Color for background: Transparent
- Color for foreground: Black

Additional Actions

Name	Details
Pop Up Dialog	Pop up dialog with a dynamic messa
Play sound	Play sound 'Alert4.wav'
Submit order	Submit a 'Market' order

add edit remove

Once an alert condition has been satisfied as true, the following actions can be automatically triggered:

- Play Sound
- Share (Twitter, Facebook, Email, etc.)
- Show custom message box
- Submit an Order

New “Re-arm” types have been added in order to reset an alert upon a few different events:

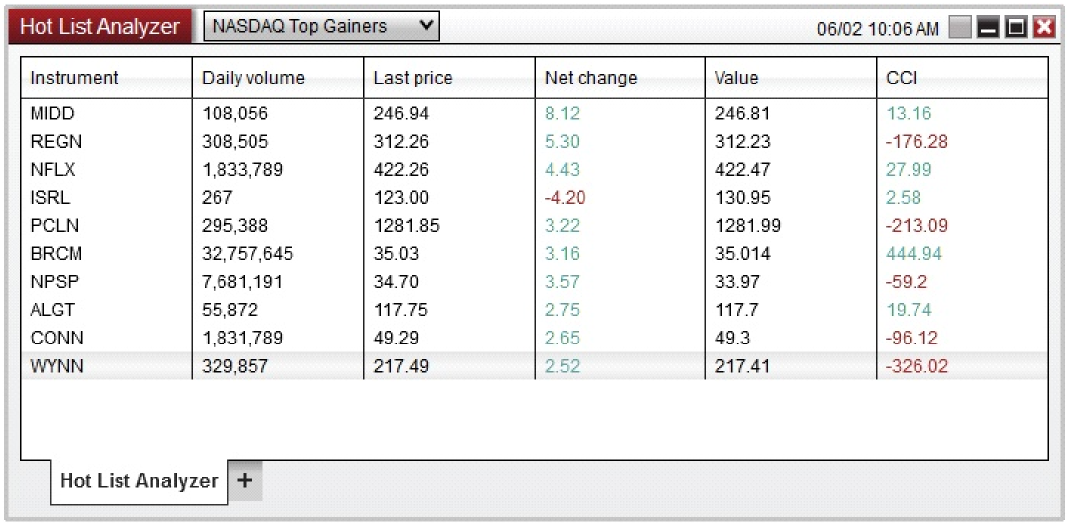
- On Timer – after so many seconds have elapsed
- On Bar Close – after the selected data series has generated a new bar
- On Condition Reversed – when the condition becomes false
- On Connect – after NinjaTrader 8 has been manually connected to a data feed

Improved Alerts Log

The Alerts Log has been improved to now show all alerts generated in the session, rather than alerts since the window was open. A new feature allows users to display all alerts from all open workspaces in a single Alerts Log window, or to suppress any alerts originating in inactive workspaces. The new "Go To Alert" feature will immediately bring the window which triggered an alert into focus in your workspace. We’ve also added options to filter and sort by Instrument Type and Source (Charts, Market Analyzer, NinjaScript, etc.).

New Hot List Analyzer

Similar to the Market Analyzer, the new Hot List Analyzer dynamically loads “hot lists” from market data providers who supply this information, such as Kinetick. This new window retains all the behavior of the Market Analyzer, allowing you to add columns and indicators for adaptive market analysis. We’ve also included the ability to create an instrument list directly from the Hot List Analyzer to easily access these lists from anywhere in the platform.



Instrument	Daily volume	Last price	Net change	Value	CCI
MIDD	108,056	246.94	8.12	246.81	13.16
REGN	308,505	312.26	5.30	312.23	-176.28
NFLX	1,833,789	422.26	4.43	422.47	27.99
ISRL	267	123.00	-4.20	130.95	2.58
PCLN	295,388	1281.85	3.22	1281.99	-213.09
BRCM	32,757,645	35.03	3.16	35.014	444.94
NPSP	7,681,191	34.70	3.57	33.97	-59.2
ALGT	55,872	117.75	2.75	117.7	19.74
CONN	1,831,789	49.29	2.65	49.3	-96.12
WYNN	329,857	217.49	2.52	217.41	-326.02

Numerous Hot Lists are available, depending upon what an specific data provider supports. Examples include:

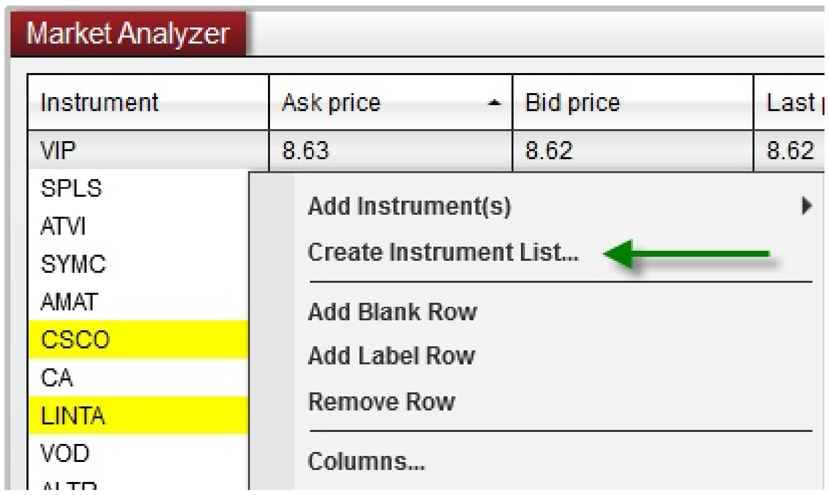
- Most Active
- Top Gainers
- Top Losers
- Top 52-week Highs or Lows
- Volume Increase
- Unusually High Volume

Market Analyzer

The Market Analyzer has been improved to allow for more robust management of different groups of instruments and columns. From the Instrument Search, you can now select multiple instruments to quickly add to or remove from the Market Analyzer display. You can also apply Cell or Filter conditions to a specific instrument name(s) to allow for more customizable conditions.

We now support custom Market Analyzer column development in NinjaScript, which means that you can develop your own custom columns to run directly within the Market Analyzer.

Instruments lists can now be created from an open Market Analyzer window. Simply select **Create Instrument List** in the Market Analyzer's right-click menu to add all selected instruments to a new Instrument List.



Instrument	Ask price	Bid price	Last
VIP	8.63	8.62	8.62
SPLS			
ATVI			
SYMC			
AMAT			
CSCO			
CA			
LINTA			
VOD			

Trading Hours templates can now be applied to each indicator column to help control data requirements for specific columns. Using Trading Hours templates allows you to restrict the data used in historical and real-time data processing in a specific column.

Performance Improvements:

- Indicator columns load at least 300% faster than NinjaTrader 7
- Indicator columns no longer reload when changing Market Analyzer properties
- "Days to load" now supported, improving bar loading times

New Data Columns:

- Ask/Bid/Last size
- Bid/Ask Spread
- Current Ratio
- Dividend Amount
- Dividend Pay Date
- Dividend Yield
- Earnings Per Share
- Market Capitalization

- Notes
- Open Interest
- Price/Earnings Ratio
- Revenue per share
- Settlement Price
- Short Interest
- Short Interest Ratio
- VWAP

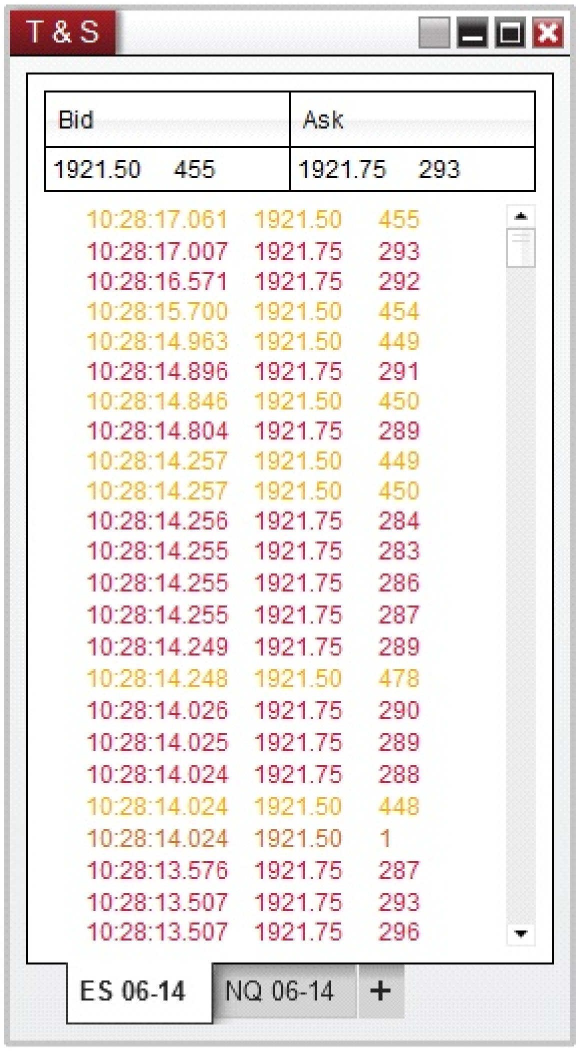
News Window

- Simplified filtering and alert interface
- News articles can be read in a separate window (double-click any news item)

Enhanced Time and Sales Window

The Time and Sales display has been improved to seamlessly display market data updates without flickering, which allows for a smoother and distraction free operation.

- Added real-time bid/ask price updates
- Added ability to scroll and analyze past records
- Added a new ‘Block’ alert sound option
- Added Configurable Time display format



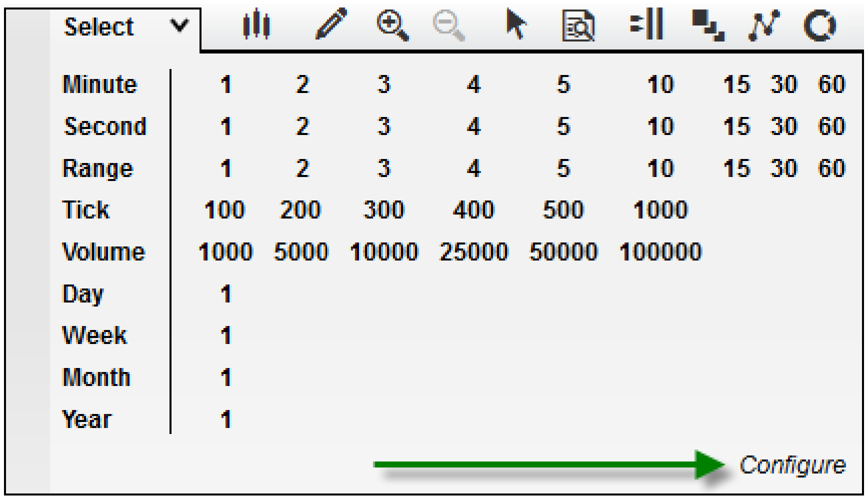
Bid	Ask
1921.50 455	1921.75 293

```
10:28:17.061  1921.50  455
10:28:17.007  1921.75  293
10:28:16.571  1921.75  292
10:28:15.700  1921.50  454
10:28:14.963  1921.50  449
10:28:14.896  1921.75  291
10:28:14.846  1921.50  450
10:28:14.804  1921.75  289
10:28:14.257  1921.50  449
10:28:14.257  1921.50  450
10:28:14.256  1921.75  284
10:28:14.255  1921.75  283
10:28:14.255  1921.75  286
10:28:14.255  1921.75  287
10:28:14.249  1921.75  289
10:28:14.248  1921.50  478
10:28:14.026  1921.75  290
10:28:14.025  1921.75  289
10:28:14.024  1921.75  288
10:28:14.024  1921.50  448
10:28:14.024  1921.50  1
10:28:13.576  1921.75  287
10:28:13.507  1921.75  293
10:28:13.507  1921.75  296
```

Chart

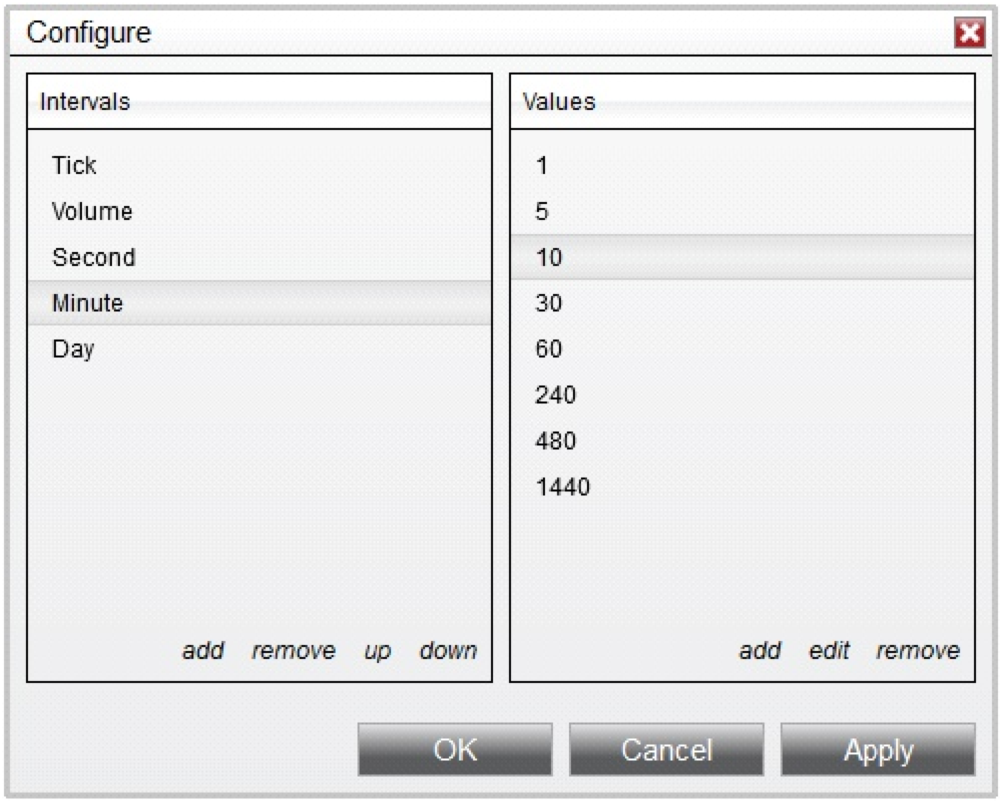
New Interval Selector

We've re-designed the chart's Interval Selector for more intuitive navigation. The new Interval Selector is now 100% customizable (factory defaults shown in the image below, but all values can be changed, and additional interval types can be added).



Select

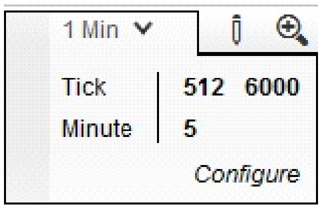
Minute	1	2	3	4	5	10	15	30	60
Second	1	2	3	4	5	10	15	30	60
Range	1	2	3	4	5	10	15	30	60
Tick	100	200	300	400	500	1000			
Volume	1000	5000	10000	25000	50000	100000			
Day	1								
Week	1								
Month	1								
Year	1								



Configure

Intervals	Values
Tick	1
Volume	5
Second	10
Minute	30
Day	60
	240
	480
	1440

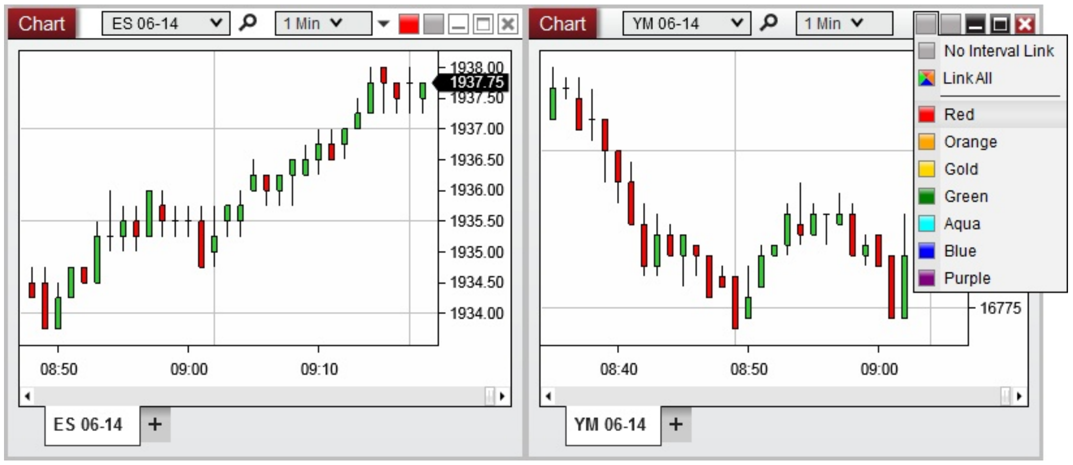
This new design allows for users to completely define the interval types as well as the values used for easy access from the charts.



You can also remove or add the interval types as well as change the order in which they are displayed in the selector.

New Interval Linking

Based on the familiar instrument link feature, we've added a new Interval Link to charts, which will allow you to duplicate interval changes from one chart to another. For example, if you have two different instruments set to a 1 minute chart, you can now link these two charts where switching one chart to a 15-minute interval will also update the interval on the linked chart.



New Free Mode Scrolling

Using the control key + click and dragging on the chart now scrolls the chart's x-axis and y-axis in any direction that the mouse is moved, giving a more adaptable display.

New centering of price on y-axis mode

Charts now have an option to automatically rescale the bars on a chart to ensure the last traded price is exactly centered on the price axis.

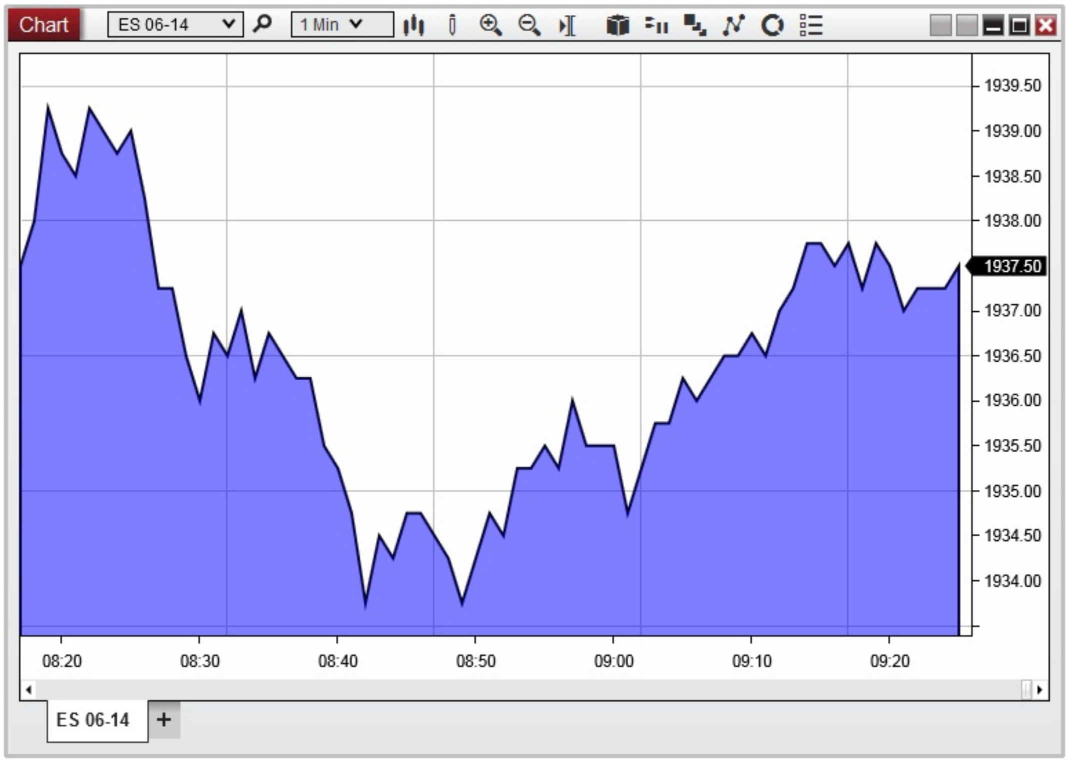
New Cross Hair Anchoring

You can now lock the crosshair to a specific point on the time axis. This allows you to freely move your cursor to other areas of your desktop without disturbing the placement of the crosshair in order to further analyze price at a specific time point.

New Chart Styles

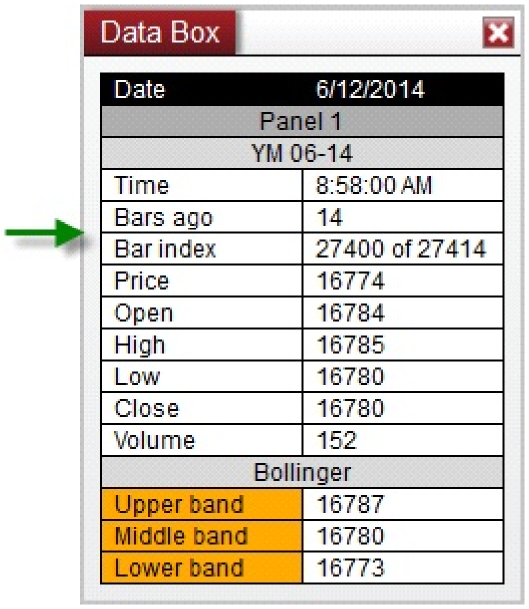
We've added presets to chart styles, allowing you to further customize different styles based on specific periods that are selected.

- Improved OHLC chart style with additional HLC and HiLo capability
- New Open/Close style
- New Mountain chart style



Data Box

The chart's data box has been updated to include multiple indicator plots, and individual plots can be displayed or suppressed via a property in the Indicators window. In addition, we have added the option to display “Bars ago” and “Bar index” values to aid in the process of debugging NinjaScript indicator and strategies.



Date	6/12/2014
Panel 1	
YM 06-14	
Time	8:58:00 AM
Bars ago	14
Bar index	27400 of 27414
Price	16774
Open	16784
High	16785
Low	16780
Close	16780
Volume	152
Bollinger	
Upper band	16787
Middle band	16780
Lower band	16773

Drawing Tool Enhancements

Drawing tools have been improved in a number of ways. In previous versions, only a handful of drawing objects had the ability to define and save multiple settings via templates. In NinjaTrader 8 we have enhanced all drawing objects to all include a template option. This allows you to define multiple different settings for a single Drawing Tool and apply these templates in a more efficient manner.

When using Global drawing objects, you can now set these to be global across all workspaces, or for only a single workspace, allowing you to eliminate certain drawing objects from specific workspaces if desired.

All Drawing Tools have been implemented as NinjaScript objects, allowing developers to build their own custom drawing tools.

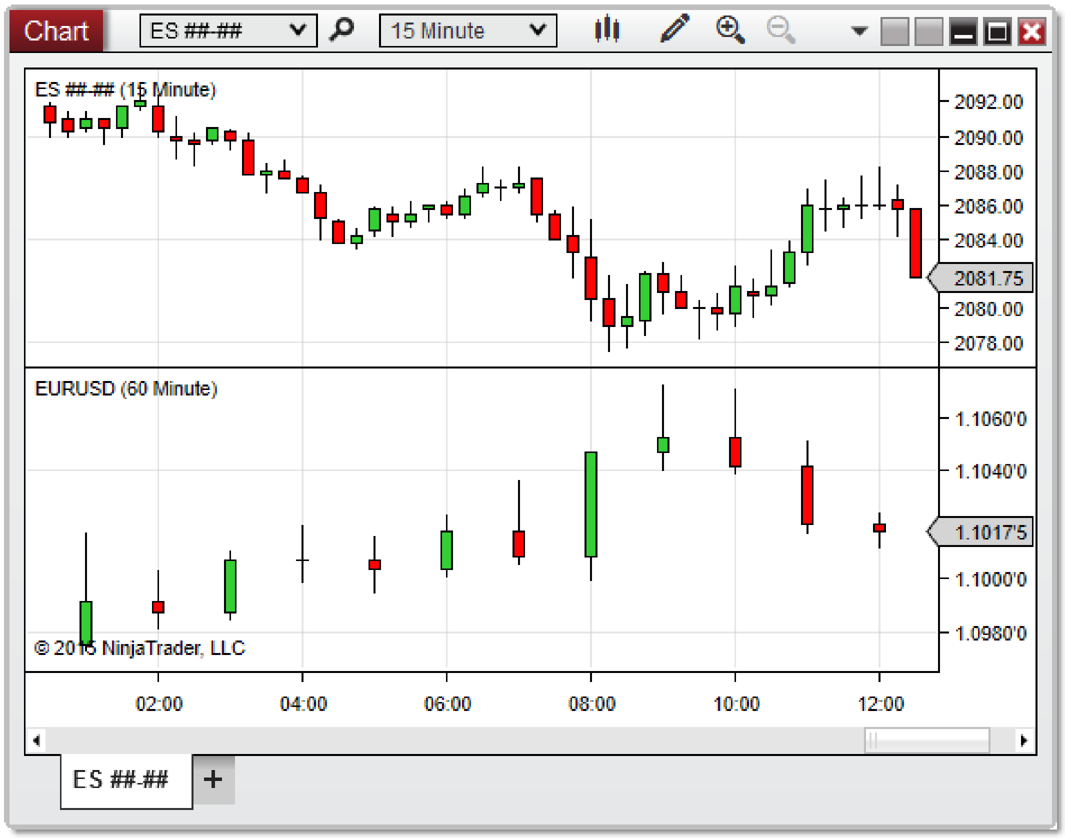
Miscellaneous Drawing Tools Improvements

- New Region Highlight tools
- New Risk-Reward tool
- New ability to hide all drawn drawing objects
- New ability to roll drawn objects over to new futures contract expiries
- Enhanced draw objects dialogue window with the ability to manage multiple drawing objects at once

- Improved Arc tool
- Improved Gann Fan tool

Multi-Series Equidistant Bar Spacing

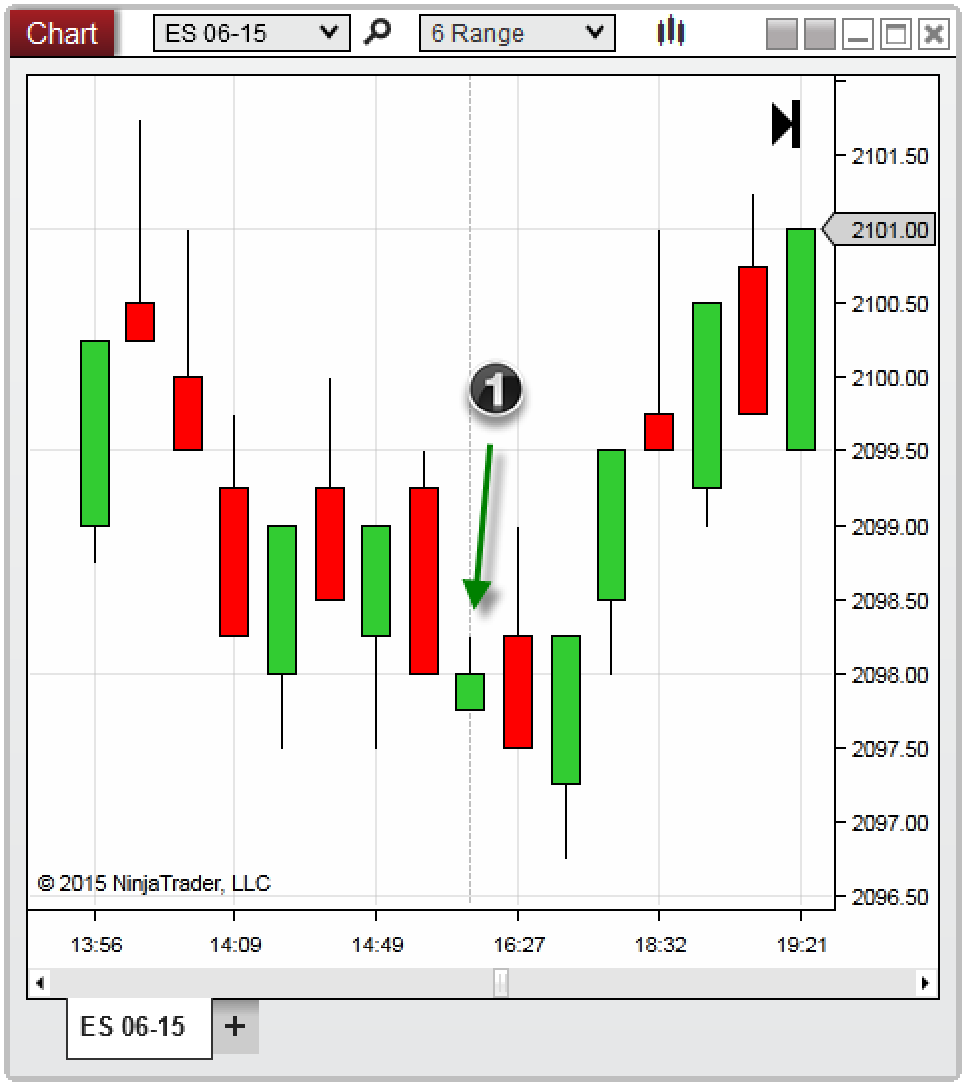
When adding more than one data series to a chart, NinjaTrader 7 used a non-equidistant bar spacing by default, in order to accurately align each bar series to the time axis.   This charting display mode has been improved in NinjaTrader 8 by giving you the option to mix the equidistant bar spacing display. This gives users the ability to select which period is used to space the bars evenly and ensure that additional bar series follow this sequence.



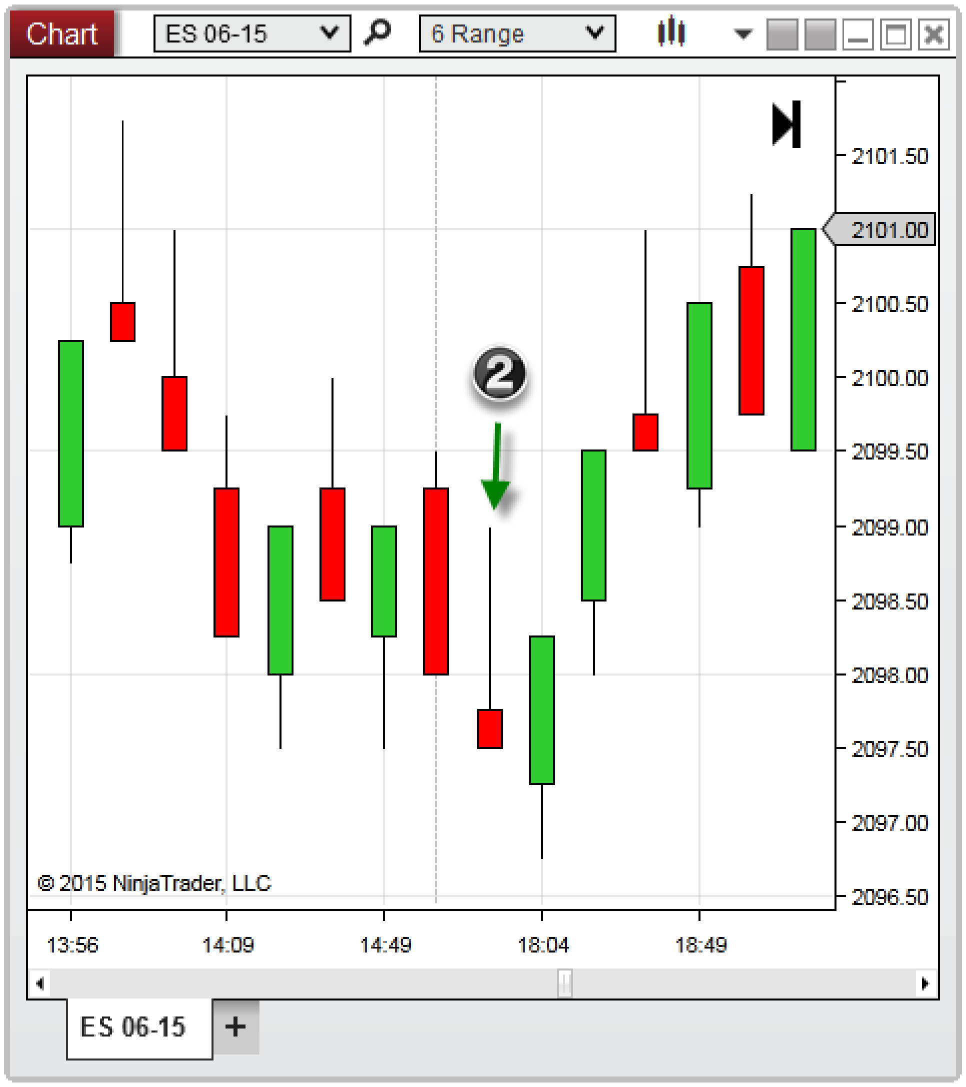
New Data Series Break at EOD

In NinjaTrader 7, Data Series set to a non-time-based interval, such as Tick or

Renko, could be cut at the end of the trading session, at which point a new bar would begin to be painted in the subsequent trading session. If the new property “Break at EOD” is set to false, a tick based bar would carry over from one session to the next, spilling over the end of the session defined in the Trading Hours template.



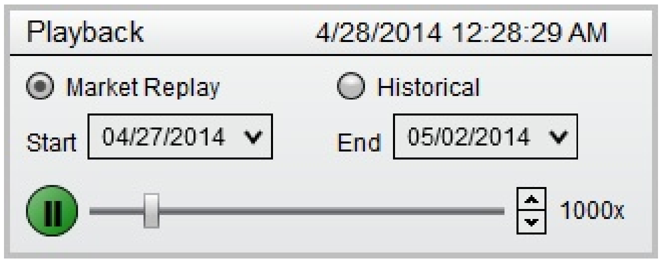
1. **Break EOD** *enabled* - a new bar was formed during the new trading session before the 6 range bar had completed



2. **Break EOD** *disabled* - a new bar was not formed until the criteria for the 6 point range was satisfied

Playback

New Playback



Market Replay was renamed to Playback and now has the option to play back historical tick data downloaded from a market data provider such as Kinetick in addition to the classic Market Replay data files, just like previous versions. We've also enhanced the Market Replay data structure by storing Level 1 and Level 2 data in a single file, which ensures 100% accurate replay sequencing.

Playback Performance Improvements

Playback now pre-loads the current day when connecting, which ensures that during fast forward operations the entire day is replayed. This ensure that every session is fully stable. We also added faster playback speeds (100/200/300/500/1000) and a new "Max" speed which will process as many ticks as your CPU can handle.

Strategy Analyzer

General Enhancements

The Strategy Analyzer has benefited from many of the general performance enhancements done to the NinjaTrader 8 codebase, which has improved the speed of backtesting and optimizing substantially.

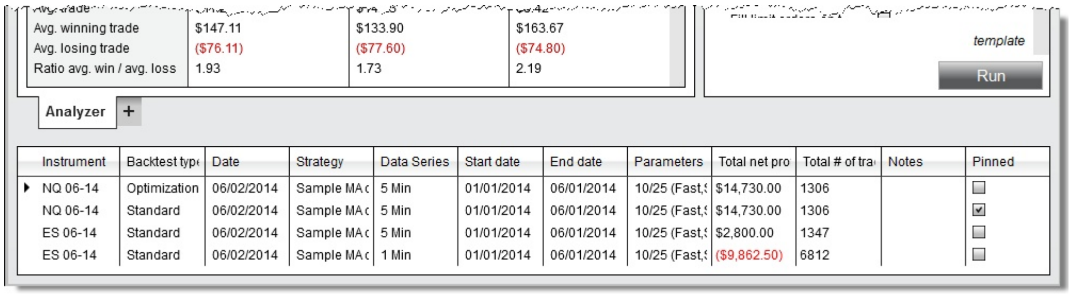
We've added the ability to save a Strategy Analyzer session in the workspace. When restored, a Strategy Analyzer saved in a workspace will restore the last tested result summary, allowing you to pick up where you left off after a restart. You can also now save multiple templates of individual strategy settings, permitting you to research and track many different scenarios using the same strategy.

- Added Strategy Analysis Statistics
    - RSquared

- Sortino Ratio
- Total Slippage
- Number of Even Trades
- User developed custom Statistics are now supported through NinjaScript®

New Strategy Analyzer Log

The Strategy Analyzer has a new “Log” feature which allows you to save results and keep records of each backtest and optimization in real-time. This will help you track your progress as you perform new backtests and optimizations where you can even pin/remove favorite results to review for later as you attempt to obtain better results.



Avg. winning trade	$147.11	$133.90	$163.67
Avg. losing trade	($76.11)	($77.60)	($74.80)
Ratio avg. win / avg. loss	1.93	1.73	2.19

Instrument	Backtest typ	Date	Strategy	Data Series	Start date	End date	Parameters	Total net pro	Total # of tra	Notes	Pinned
NQ 06-14	Optimization	06/02/2014	Sample MA c	5 Min	01/01/2014	06/01/2014	10/25 (Fast,$	$14,730.00	1306		☐
NQ 06-14	Standard	06/02/2014	Sample MA c	5 Min	01/01/2014	06/01/2014	10/25 (Fast,$	$14,730.00	1306		☑
ES 06-14	Standard	06/02/2014	Sample MA c	5 Min	01/01/2014	06/01/2014	10/25 (Fast,$	$2,800.00	1347		☐
ES 06-14	Standard	06/02/2014	Sample MA c	1 Min	01/01/2014	06/01/2014	10/25 (Fast,$	($9,862.50)	6812		☐

The new log will also allow you to filter by instrument, strategy, and date of backtest, as well as leave custom notes and remarks on the results themselves. Additionally, each result logged will save a snapshot of your strategy code so that you can compare your working copy to previous generations. This can allow you to keep track of changes in backtest performance resulting from changes made in code.

New Strategy Parameter Templates

After you’ve refined your strategy parameters through backtesting and optimization procedures, you can quickly save a series of parameters in an unlimited number of templates specific to that strategy. This allows you to quickly move your desired parameters from backtesting to live deployment without the need to re-input the optimized values when finally deploying your results to a live trading system.

Enhanced Backtesting Engine

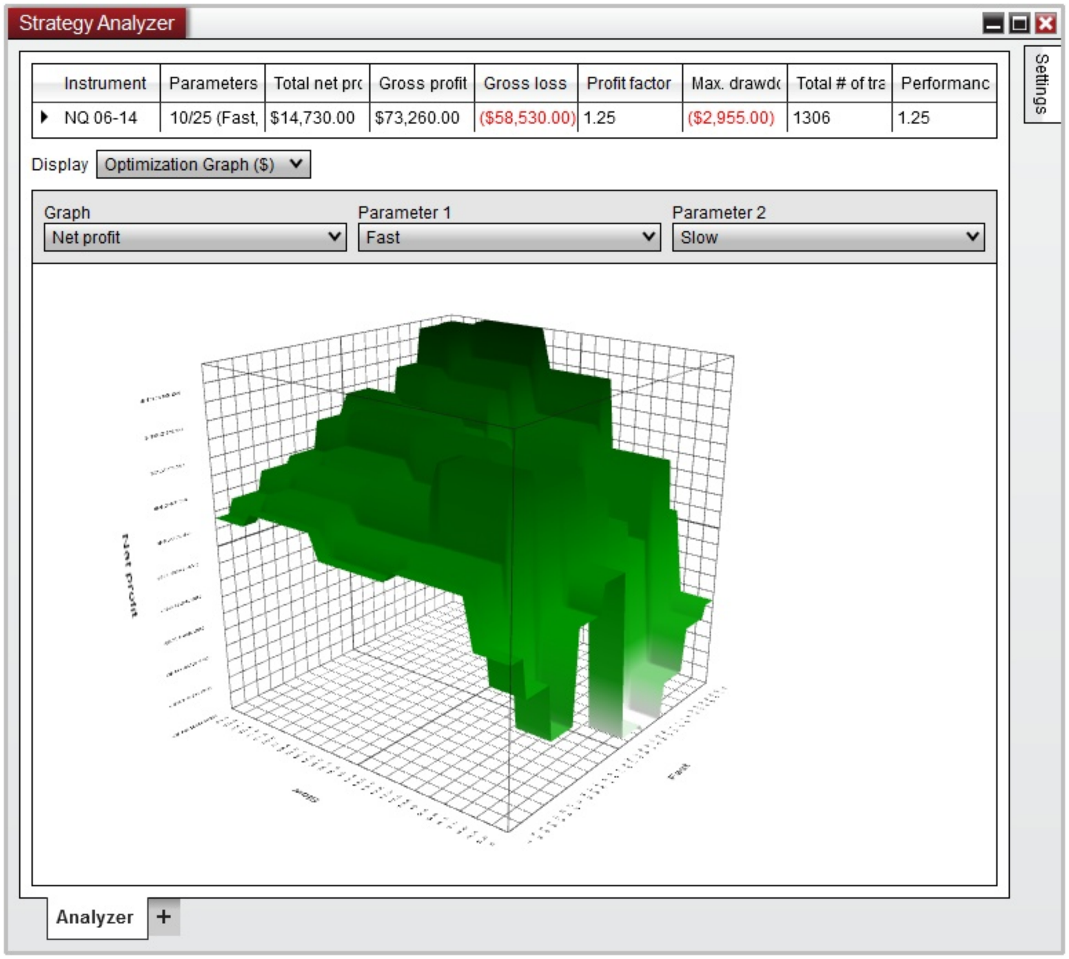
- Order Fill Resolution and Fill engine enhanced for greater precision and accuracy
- Standard Fill Resolution breaks bars in to three virtual bars to simulate the direction of the price which was used to form the bar
- High Resolution mode automatically adds a secondary data series as the additional resolution used for fills, without needing to custom program

Improved Optimization

We've addressed several performance limitations in terms of optimizing time values, boolean variables, and enumerated constants, to ensure that the optimizer takes full advantage of multi-threaded processors. Additionally, the 64-bit version of NinjaTrader will automatically store trade details for each backtest in memory, which allows for quicker analysis.

New 3D Optimization Graph

In addition to the general 2D graphs used to review strategy performance, we've introduced a new 3D graph for analysis when using two or more parameters in an optimization test, helping you visualize how each input parameter influences the results of your overall strategy performance.



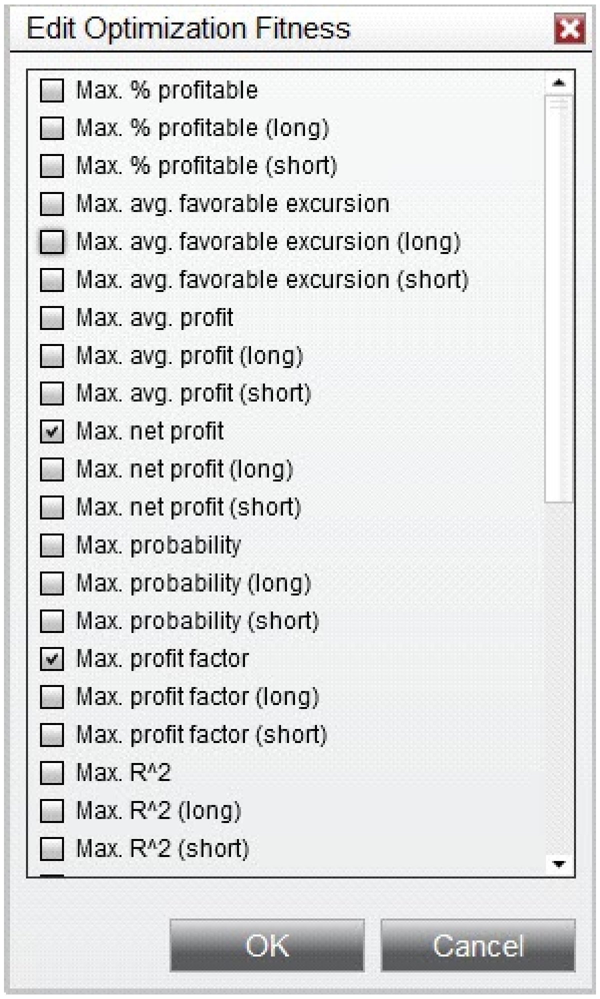
Instrument	Parameters	Total net prc	Gross profit	Gross loss	Profit factor	Max. drawd(	Total # of tra	Performanc
NQ 06-14	10/25 (Fast,	$14,730.00	$73,260.00	($58,530.00)	1.25	($2,955.00)	1306	1.25

New Multi-Objective Optimization

You can now select multiple optimization objectives to test the best tradeoff between the performance of different parameter combinations on individual fitness metrics. Once completed, you can display results in the form of a Pareto Graph which shows the set of parameter combinations for which there are no superior alternatives on all metrics tested.

- Graphing the Pareto Frontier reveals the optimal tradeoff between two statistics (out of any number of tested metrics)
- Test all fitness metrics, or any subset of available metrics
- Include custom-developed fitness metrics in multi-objective optimizations
- Combine different fitness combinations in real time

- Example: Find the results of the most profitable strategy with the least draw down risk, and every best combination thereof.



Edit Optimization Fitness

- [ ] Max. % profitable
- [ ] Max. % profitable (long)
- [ ] Max. % profitable (short)
- [ ] Max. avg. favorable excursion
- [ ] Max. avg. favorable excursion (long)
- [ ] Max. avg. favorable excursion (short)
- [ ] Max. avg. profit
- [ ] Max. avg. profit (long)
- [ ] Max. avg. profit (short)
- [✓] Max. net profit
- [ ] Max. net profit (long)
- [ ] Max. net profit (short)
- [ ] Max. probability
- [ ] Max. probability (long)
- [ ] Max. probability (short)
- [✓] Max. profit factor
- [ ] Max. profit factor (long)
- [ ] Max. profit factor (short)
- [ ] Max. R^2
- [ ] Max. R^2 (long)
- [ ] Max. R^2 (short)

Walk Forward Optimization

The new Trading Hours templates allow for more accurate optimization when performing walk forward analysis, especially while using trading hours which span multiple days.

NinjaScript

General Improvements

- Reload historical data programmatically via code
- Programmatically add custom bar types
- Drawing objects can be set as "global" via code
- Choose Drawing Tool templates via code
- Improved the new bar detection using FirstTickOfBar
- Access instruments from instrument lists
- Added “Break EOD” feature which optionally reset indicator values on a new trading day for stability

Supported NinjaScript® Object Types

- AddOns
- Bar Types
- Chart Styles
- Drawing Tools
- Import Types
- Indicators
- Market Analyzer Columns
- Optimization Fitness
- Optimizers
- Performance Metrics
- Share Services
- Strategies
- SuperDOM Columns

NinjaScript® Import/Export

- Only a single .DLL File provided for both 32-bit and 64-bit systems
- The export process has been improved to more clearly identify which references are required for an export
- Agile.net protection improved significantly
    - Requires a purchase of an Agile.net license to take advantage of more secure protection methods

NinjaScript® Code Wizard

Our NinjaScript® Code Wizard, which is used to generate minimum required code for new scripts, has been updated to include all supported NinjaScript object types. We've also enhanced the available configuration options to help generate the desired script base in a much more efficient manner.

- Use an unlimited number of Input Parameters
- Optionally select additional data series
- Select any additional event methods to use relevant to NinjaScript object type (OnConnectionLoss, OnMarketData, OnMarketDepth, etc.)

New NinjaScript Suspension Optimization

Newly added NinjaScript Suspension Optimization allows programmers to halt market data events from being processed when an indicator is not visible (such as Chart window minimized or another window is on top of the window hosting the indicator), saving CPU resources when not in use.

New Tick Replay Engine

We've included an optional NinjaScript feature which will replay OnMarketData on each tick stored in the database. This will ensure that your indicators and strategies receive the exact sequence of stored events for the most accurate calculations on historical data that include the historical bid/ask price, just as you would expect in real-time.

Expanded NinjaScript® Access

NinjaTrader 8 will introduce a new level of depth and breadth which will allow developers to build incredibly rich and integrated trading applications limited only by their imagination.

Developers will have access to:

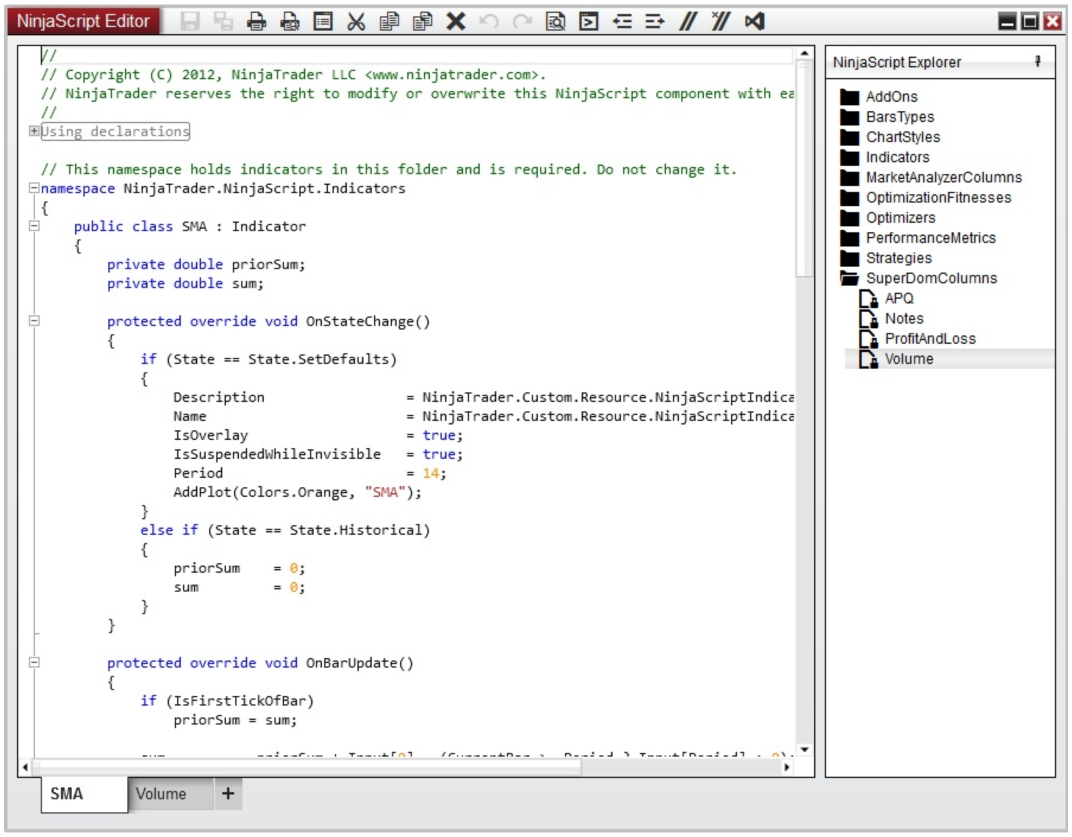
- Trading objects such as connections, accounts, orders and executions
- Market data objects including historical data
- Controls such as Instrument Selector, Account Selector and Quantity Selector, Instrument Link, Interval Link etc.
- Window and Workspace methods
- Access to control, modify and interact with UI elements via Windows Presentation Foundation (WPF)

NinjaScript is no longer limited to writing custom indicators and strategies. The possibilities are truly endless. Build what you can dream of and integrate it directly into the NinjaTrader 8 application.

NinjaScript Editor

The NinjaScript Editor has been redesigned to include a new NinjaScript Explorer menu which is pinned to the right side of the editor. This new explorer feature allows developers to nest and organize different NinjaScript files into custom folders, in order to easily locate and reference other scripts directly from the editor itself.

Additionally, we've included the ability to exclude scripts from compilation if they are still under development or contain code breaking changes that would have previously prevented developers from working on secondary scripts. This new feature gives developers the flexibility to keep their undeveloped scripts installed on their system and accessible from the Editor until the scripts are ready to be compiled.



```
//
// Copyright (C) 2012, NinjaTrader LLC <www.ninjatrader.com>.
// NinjaTrader reserves the right to modify or overwrite this NinjaScript component with ea
//
Using declarations

// This namespace holds indicators in this folder and is required. Do not change it.
namespace NinjaTrader.NinjaScript.Indicators
{
    public class SMA : Indicator
    {
        private double priorSum;
        private double sum;

        protected override void OnStateChange()
        {
            if (State == State.SetDefaults)
            {
                Description                 = NinjaTrader.Custom.Resource.NinjaScriptIndica
                Name                        = NinjaTrader.Custom.Resource.NinjaScriptIndica
                IsOverlay                   = true;
                IsSuspendedWhileInvisible   = true;
                Period                      = 14;
                AddPlot(Colors.Orange, "SMA");
            }
            else if (State == State.Historical)
            {
                priorSum    = 0;
                sum         = 0;
            }
        }

        protected override void OnBarUpdate()
        {
            if (IsFirstTickOfBar)
                priorSum = sum;
```

The NinjaScript Editor now supports direct Visual Studio Integration, which means you can open, edit, and debug your classes directly in Visual Studio. This allows the NinjaScript Editor to detect changes made outside of itself and automatically reload these changes in order to compile. This functionally applies to any other text editor of your choice, giving the ability to directly edit files outside of NinjaTrader 8 should you desire.

NinjaScript Strategies Working With Real-World Data

Strategies can now work with real-world order, execution, and account information. This greatly improves the startup behavior of the strategy, allowing it to adopt the real-world position and continue operation as if the strategy was running.

- Exposed Real World Order, Execution, and Account Access
- Improved Start Behavior, including adopting Real World Position
- Improved Real-time Error Handling to provide ability to filter for rejections

and handle terminal order states in code

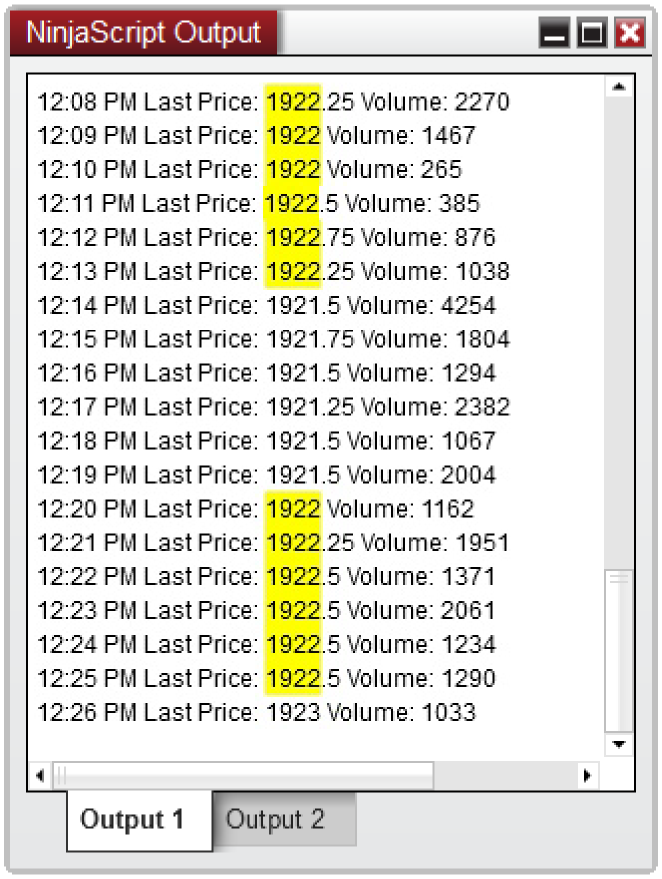
- Access commission rates

NinjaScript Output Window

The NinjaScript Output window has received a number of upgrades. Firstly, we've introduced a new smooth scrolling operation to help track and navigate data as it added to the output window.

The Output Window is now separated into two tabs, and you can choose which tab to use when outputting data via NinjaScript. This allows for a Dual View mode and synchronized scrolling operation to help compare data output from two scripts.

There is also a new facility to search for strings, and simply double clicking on a string token will quickly highlight any other strings that match that token.



```
12:08 PM Last Price: 1922.25 Volume: 2270
12:09 PM Last Price: 1922 Volume: 1467
12:10 PM Last Price: 1922 Volume: 265
12:11 PM Last Price: 1922.5 Volume: 385
12:12 PM Last Price: 1922.75 Volume: 876
12:13 PM Last Price: 1922.25 Volume: 1038
12:14 PM Last Price: 1921.5 Volume: 4254
12:15 PM Last Price: 1921.75 Volume: 1804
12:16 PM Last Price: 1921.5 Volume: 1294
12:17 PM Last Price: 1921.25 Volume: 2382
12:18 PM Last Price: 1921.5 Volume: 1067
12:19 PM Last Price: 1921.5 Volume: 2004
12:20 PM Last Price: 1922 Volume: 1162
12:21 PM Last Price: 1922.25 Volume: 1951
12:22 PM Last Price: 1922.5 Volume: 1371
12:23 PM Last Price: 1922.5 Volume: 2061
12:24 PM Last Price: 1922.5 Volume: 1234
12:25 PM Last Price: 1922.5 Volume: 1290
12:26 PM Last Price: 1923 Volume: 1033
```

▽ Connectivity

General Connections

- Forex Connections can now auto subscribe to required instruments for currency conversion (useful when trading non-USD-denominated pairs)
- All adapters are now supported on the 64-bit version of NinjaTrader 8
- Auto-connect handling improved when a connection cannot be established
- Account Denomination for PnL reporting can now be set per connection
- Preferred connections for real-time and historical market data, meaning you can now request historical data come from Provider A while your real-time data come from Provider B. This removes the dependency on the sequence in which you establish your connection to various brokers and data feeds

CQG/Continuum

- Always requests orders, executions, and positions when reconnecting
- Uses server time for all timestamps (market data order)
- Added settlement price as a fundamental data type

eSignal

- Updated to the latest API
- Added 64-bit support
- Supports all 10 levels of market depth on futures
- Improved pre/post market real-time data
- Added Hotlist Support

FXCM

- Native OCO support

Interactive Brokers

- Updated to the latest API
- Now supports TWS: Gateway

IQfeed

- Updated to the latest API

TDA

- Updated API
- 64-bit support

Rithmic

- Updated API

▽ Trading Interface

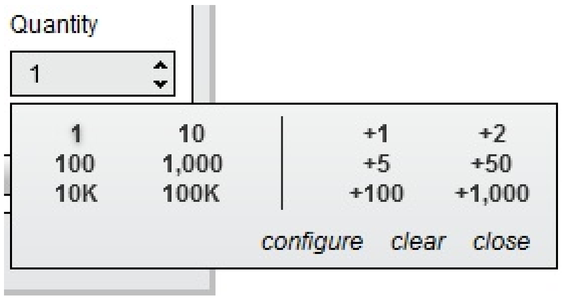
General Trading Improvements

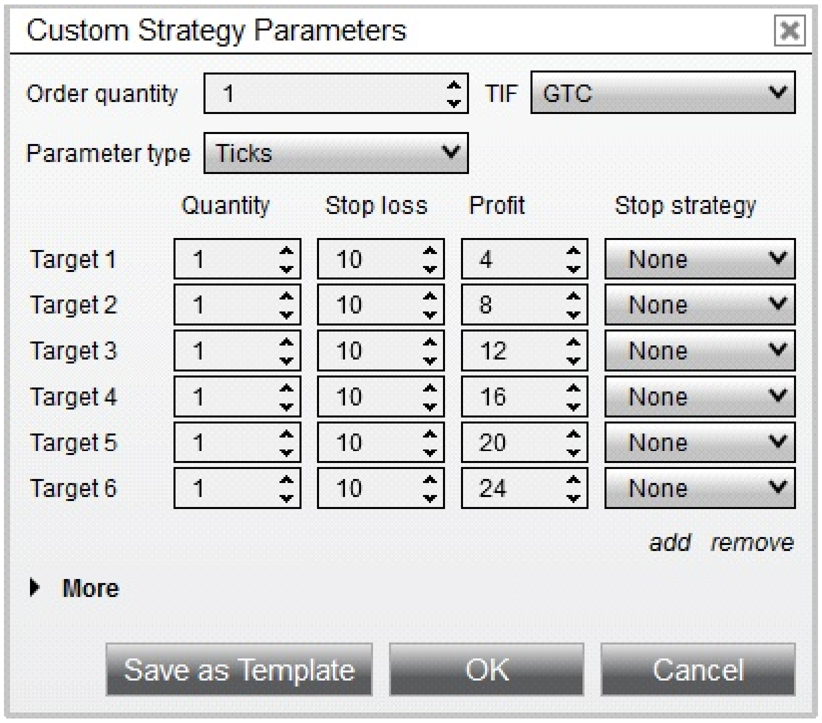
- You can now select secondary simulation accounts in the account selector while using global simulation mode (i.e., trade both Sim101 and a Sim102 account in global simulation mode)
- Added Pips and Ticks to PnL display
- Order Types and TIF selectors only load the supported order types and TIF settings supported by the selected account
- Account selector now attempts to auto-select the last selected account when connected
- Order Type and Order State colors are now integrated into one color system and standardized across entire application, where the color of the

order represents a specific order type unless order is pending/cancelled

Quantity Selector

We updated the quantity selector to provide users with quick access to preset quantity values, as well as increment values which can be customized ahead of time.




Custom Strategy Parameters

Order quantity: 1 TIF: GTC

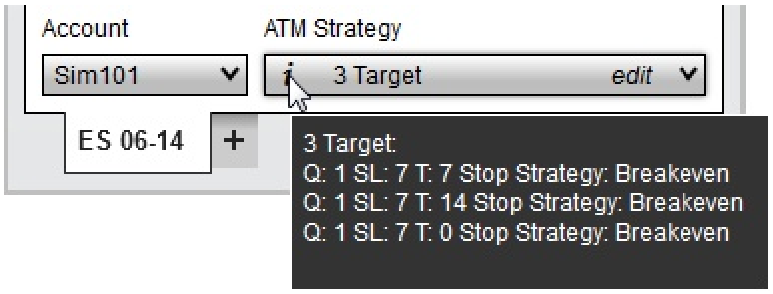
Parameter type: Ticks

	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	10	4	None
Target 2	1	10	8	None
Target 3	1	10	12	None
Target 4	1	10	16	None
Target 5	1	10	20	None
Target 6	1	10	24	None

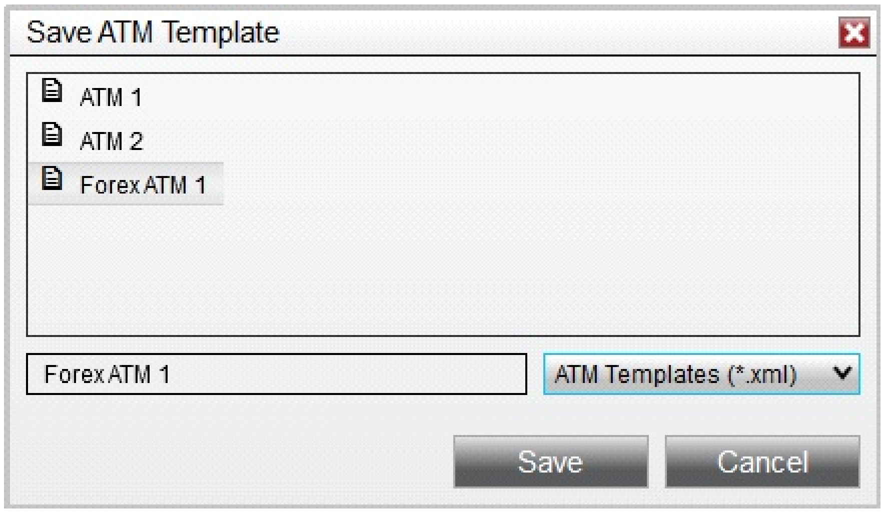
ATM Strategies

ATM Strategies have largely remained the same, with some slight improvements based on customer feedback over the years. Most significantly, you can now set an unlimited number of Stop Loss and Profit Target orders, resulting in more dynamic trade management. Profit target orders can optionally use the Market If Touched (MIT) order type rather than Limit orders.

Hovering your mouse cursor over a selected ATM Strategy will now display informative tooltips displaying details of the strategy parameters without having to open the ATM Strategy Parameters itself.



3 Target:
Q: 1 SL: 7 T: 7 Stop Strategy: Breakeven
Q: 1 SL: 7 T: 14 Stop Strategy: Breakeven
Q: 1 SL: 7 T: 0 Stop Strategy: Breakeven

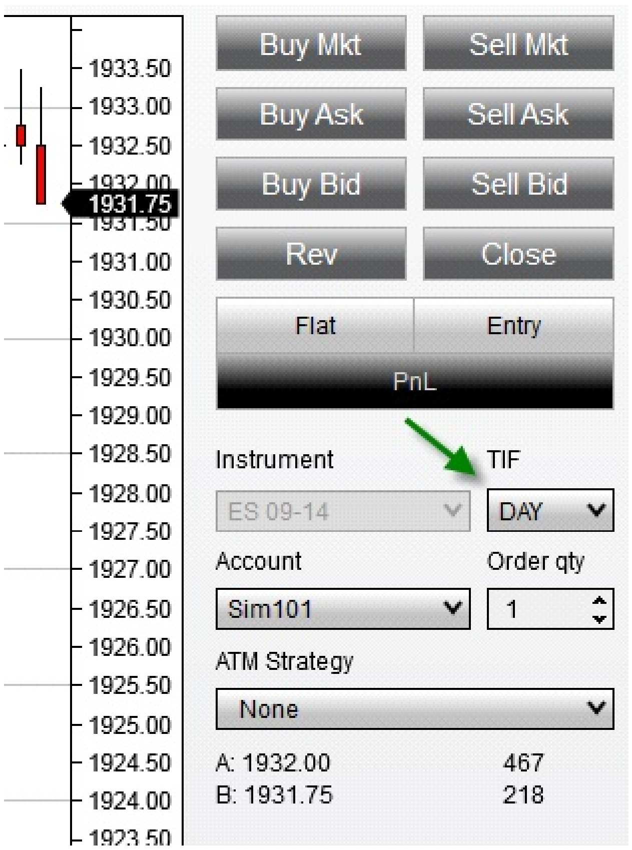


We removed the ATM Strategy template files from the database, and now store them in individual XML files for portability between computers.

Chart Trader

The Chart Trader interface has been improved, and can now be used on mulit-instrument charts. Previously, Chart Trader would only allow you to trade the primary instrument on the chart. Now we've given users the option to select which instrument on the chart they would like to trade.

The TIF option is now always visible, improving the control of the order types used on this interface.



Instrument: ES 09-14
TIF: DAY
Account: Sim101
Order qty: 1
ATM Strategy: None
A: 1932.00 467
B: 1931.75 218

The indicator tracking feature discussed in the General section of this document is also available from Chart Trader, allowing you to synchronize orders on Chart Trader with indicators which have been added to the chart.

Hot Keys

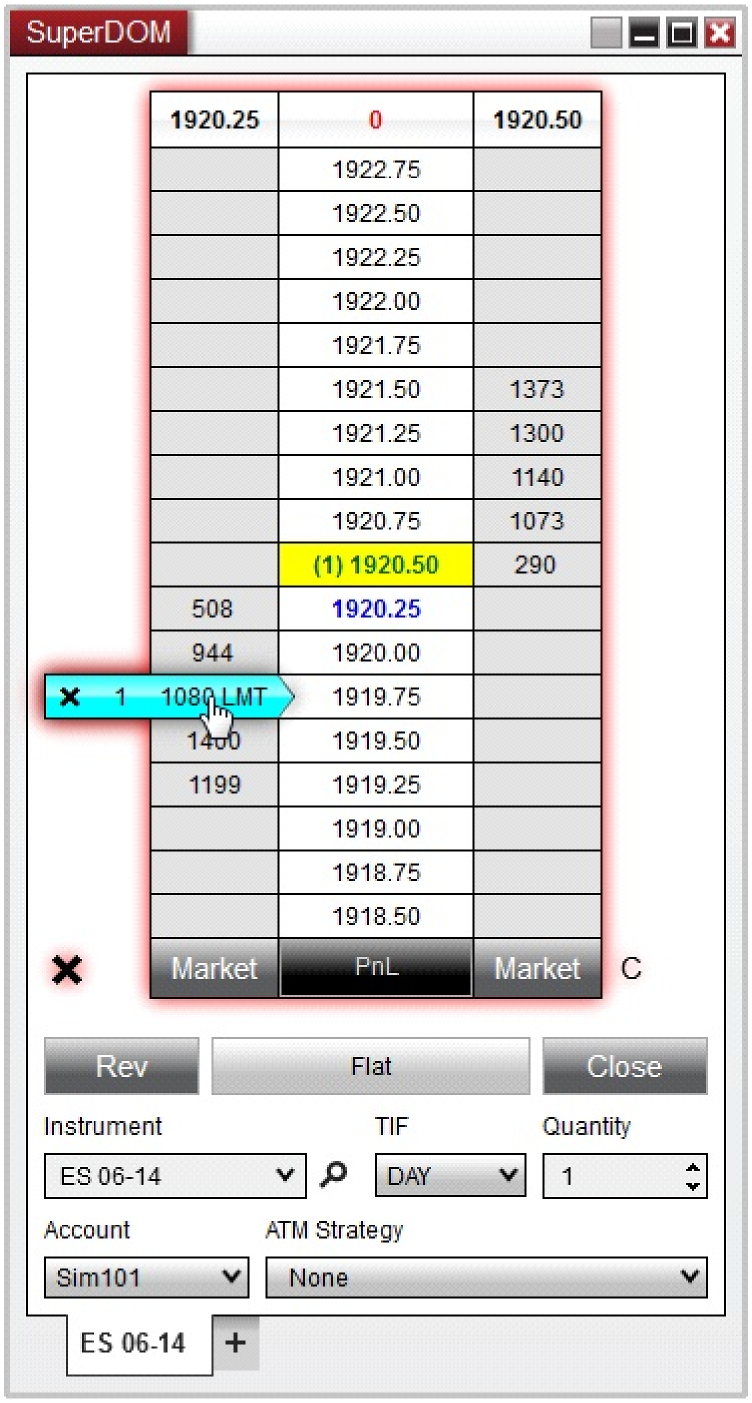
- Improved hot key setup process to allow for recording keystrokes
- Added various user-requested hot keys
- Added SuperDOM hot key category

SuperDOM

The SuperDOM has been redesigned to provide significant performance and usability improvements, and can now plot indicator values on the price ladder

© 2017 NinjaTrader, LLC

itself. The number of rows and size of the price ladder display will now dynamically update as you resize the window. The number of market depth levels is only limited by your data provider, meaning if your provider offers 10 levels of market depth, the SuperDOM can be configured to display all 10 of these levels.



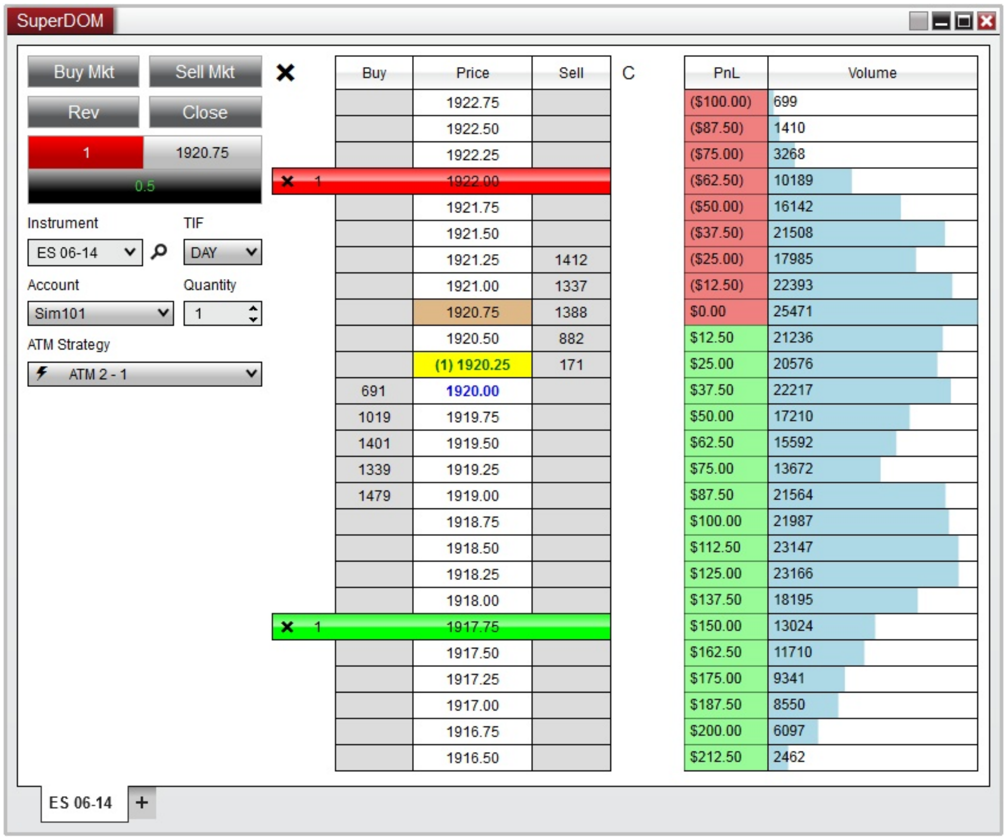
1920.25	0	1920.50
	1922.75	
	1922.50	
	1922.25	
	1922.00	
	1921.75	
	1921.50	1373
	1921.25	1300
	1921.00	1140
	1920.75	1073
	(1) 1920.50	290
508	1920.25	
944	1920.00	
✕ 1 1080 LMT	1919.75	
1400	1919.50	
1199	1919.25	
	1919.00	
	1918.75	
	1918.50	
Market	PnL	Market

Instrument: ES 06-14
TIF: DAY
Quantity: 1
Account: Sim101
ATM Strategy: None

The Dynamic SuperDOM's “Hold” button has been removed and replaced by a new “hover” mode which will temporarily freeze the price display when the mouse cursor hovers over the price display, allowing for a quicker and more intuitive order management process.

We've also introduced the ability to add Columns next to the Price Ladder, and a new “Trade Control on Left” mode. NinjaTrader 8 will install with several columns by default, and also allow users to create their own custom Columns via NinjaScript:

- PnL
- APQ (Approximate Position in Queue)
- Volume
- Notes
- Custom NinjaScript Columns supported



1 | 1920.75 | 0.5

Instrument: ES 06-14
TIF: DAY
Account: Sim101
Quantity: 1
ATM Strategy: ATM 2 - 1

	Buy	Price	Sell	C	PnL	Volume
		1922.75			($100.00)	699
		1922.50			($87.50)	1410
		1922.25			($75.00)	3268
× 1		1922.00			($62.50)	10189
		1921.75			($50.00)	16142
		1921.50			($37.50)	21508
		1921.25	1412		($25.00)	17985
		1921.00	1337		($12.50)	22393
		1920.75	1388		$0.00	25471
		1920.50	882		$12.50	21236
		(1) 1920.25	171		$25.00	20576
	691	1920.00			$37.50	22217
	1019	1919.75			$50.00	17210
	1401	1919.50			$62.50	15592
	1339	1919.25			$75.00	13672
	1479	1919.00			$87.50	21564
		1918.75			$100.00	21987
		1918.50			$112.50	23147
		1918.25			$125.00	23166
		1918.00			$137.50	18195
× 1		1917.75			$150.00	13024
		1917.50			$162.50	11710
		1917.25			$175.00	9341
		1917.00			$187.50	8550
		1916.75			$200.00	6097
		1916.50			$212.50	2462

3 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 pre-defined 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 pre-loaded 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.

© 2017 NinjaTrader, LLC

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 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.

4 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
8.0.3.0	January, 9 2017
8.0.2.0	December, 5 2016
8.0.1.0	November, 14 2016
Beta Releases	May 2015 - October 2016

4.1 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

109 07	Fixe d	Barchart	Updated DLL version to 1.1.0.9
108 14	Fixe d	Bars	UTC +2 time zone usage created erroneous bars on a chart
102 39	Fixe d	Bars	After editing workspace and restarting at times there was an unable to clear cache message
108 64	Fixe d	Chart	A crash occurred when creating daily charts in the evening
108 32	Fixe d	Chart	Removing indicator from multi-series chart removed unrelated data series as well
109 59	Fixe d	Chart	Center price on scale setting could cause crash on empty charts
109 57	Fixe d	Chart	AddOns and Caption Bar alignment was inconsistent
109 45	Fixe d	Chart	Changing axis line width caused incorrect margins
109 35	Fixe d	Chart	Chart was enlarged when restore from preview to secondary monitor
109 16	Fixe d	Chart	Bar spacing was not restoring correctly on multi-series chart
109 06	Fixe d	Chart	Indicator displacement did not auto scale correctly
108 93	Fixe d	Chart	Chart could scroll to the right unexpectedly
108 81	Fixe d	Chart	Chart rendering was failing after rollover of contracts
108 77	Fixe d	Chart	Trend Channel parallel line did not snap

108 58	Fixe d	Chart	Loading text did not stay in place for secondary series
108 15	Fixe d	Chart	Tab name variable tool tips were not defined
109 12	Fixe d	Chart Trader	Arrow keys moved chart instead of changing order quantity
109 69	Fixe d	Chart Trader	Right click order menu showed invalid order types for some connections
107 80	Fixe d	Chart Trader	Resolved a scenario where you could remove a DataSeries which has Scaled Justification Right and Chart Trader orders would no longer display
109 25	Fixe d	Chart, Bars	In some scenarios parts of Monday's sessions were not plotting
108 51	Fixe d	Chart, Drawing Tool	Drawing objects 'attached to' was lost when moving to overlay scale
107 20	Fixe d	Chart, Indicator	Editing indicators while chart is still loading at times cause a lock up
108 42	Fixe d	Chart, Window Linking	Incorrect chart tab was changing instruments with linked Market Analyzer
108 68	Fixe d	Connect ions, NinjaScr ipt	Disconnect Delay Seconds was not recognizing reconnect
109 26	Fixe d	Control Center	At times NinjaTrader could lock up on start up
108 57	Fixe d	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

109 46	Fixe d	FXCM, Historica l Data Window	Control Center locked up when closing Historical Data window while downloading data
108 30	Fixe d	Historica l Data Window	Historical data manager text was not readable when selected
108 85	Fixe d	Hot List Analyzer	@HOTLIST was not present in resources
108 34	Fixe d	Indicator	LinReg generated error with period of 1
108 83	Fixe d	Indicator , SuperD OM	Indicator's Input series Price type was not saving
109 39	Fixe d	Instrume nts	Stock list with unsupported characters could be imported but not edited
109 36	Fixe d	Instrume nts	EUREX quarterly rollover dates do not match other EUREX instruments
109 28	Fixe d	Instrume nts	Contract Month had invalid date format
109 20	Fixe d	Interacti ve Brokers	FA accounts could receive position for unknown symbol
108 97	Fixe d	Interacti ve Brokers	Requesting 1 day of historical data downloaded 2 days
108 53	Cha nge d	Interacti ve Brokers	Symbol mapping updated for ICE TF Point Value change
108 24	Cha nge	Interacti ve	Updated Traders Workstation to 960.2g

	d	Brokers	
107 92	Fixe d	Interacti ve Brokers	Bad order state occurred when order blocked due to TWS precautionary settings
108 46	Fixe d	Kinetick	Constant connect/disconnect could occur when PC clock was out of sync
106 04	Fixe d	Kinetick	Historical data was unexpectedly throttled
109 44	Fixe d	Market Analyzer	Template colors were not restored after closing and opening a workspace
109 09	Fixe d	Market Analyzer	Removed invalid conditions when using Alerts
108 49	Fixe d	Market Analyzer	Suspended indicators did not catch up until bar closes after restored
105 54	Cha nge d	Market Analyzer	Added log error when using indicators not compatible with end of day data
103 35	Fixe d	Market Analyzer	Typing label text which exceeded window length triggered instrument search box
109 18	Fixe d	NinjaScr ipt	LockRecursionException occurred on reloading NinjaScript after changing added series
109 29	Fixe d	NinjaScr ipt	IndexOutOfRange exception occurred when there was an added series in hosted indicator
109 02	Fixe d	NinjaScr ipt	AddDataSeries with specified template was not working as expecting in all scenarios
108 70	Fixe d	NinjaScr ipt	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

109 82	Fixe d	Strategy Analyzer	Assigned account could unexpectedly change on subsequent runs in certain scenarios.
109 52	Fixe d	Strategy Analyzer	Move to new window was available while running which resulted in an error
108 94	Fixe d	Strategy Analyzer	Null reference could occur when switching between backtest and optimization
108 86	Fixe d	Strategy Analyzer	Strategy was running two instances on single run
108 50	Fixe d	Strategy Analyzer	Repeatedly running a backtest caused memory to increase until there was a crash
108 43	Fixe d	Strategy Analyzer	There were incorrect tab names
108 26	Fixe d	Strategy Analyzer	Sorting pinned logs caused pins to be lost
109 67	Fixe d	Strategy Builder	IsFalling and IsRising was calling the bar index
109 49	Fixe d	Strategy Builder	MACD's plot name was Default rather than Macd
109 32	Fixe d	Strategy Builder	Unhandled Exception occurred when opening Strategy Building with compile errors
108 36	Fixe d	Strategy Builder	Removed invalid settings
108 16	Fixe d	Strategy Builder	An unhandled exception could occur when adding Actions in a certain scenario
109 13	Fixe d	SuperD OM	Dynamic SuperDOM was not always shown as a menu item
108 21	Fixe d	SuperD OM	Removed invalid order references for APQ column

10921	Fixed	TD AMERITRADE	External orders at times caused exceptions
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

4.2 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

107 51	Fixe d	Chart, Chart Trader	Selected account was not duplicating to new chart
108 18	Fixe d	Chart, Hot Key	Canceling a drawing tool caused drawing tool Hot Keys not to work until chart refocused
105 51	Fixe d	Chart, NinjaScr ipt	There was an render error on scripts at times when having multiple charts and switching the Period or Instrument
103 38	Fixe d	Chart, NinjaScr ipt	There was an exception in some cases when unchecking Equidistant Bar Spacing and adjusting the time scale
102 82	Fixe d	Chart, NinjaScr ipt	Box ChartStyle Bar would sometimes drawn to wrong date
106 85	Fixe d	Chart, NinjaTra der	Drawing Objects window caused external assembly script to stop updating
107 78	Fixe d	Chart, Worksp aces	Chart scale fixed range restored to automatic range for non primary panels on saved workspaces
107 38	Fixe d	Control Center	Strategies grid sync red flag was appearing incorrectly
107 58	Fixe d	Core	Exceptions were being generated by terminated/finalized NinjaScripts
107 86	Fixe d	Data Grids	Grid highlights were overriding underneath grid color
107 66	Fixe d	Databas e	Execution markers would multiply when secondary connection holding traded account was running through a reconnect cycle

106 96	Fixe d	Drawing	Draw objects were rendering on left side of chart during bar loading
107 90	Fixe d	Drawing Tool	Saving the Trend Channel's default template caused odd plots
107 88	Fixe d	Drawing Tool	Trend Channel template prevented manual plotting of the second line
107 84	Fixe d	Drawing Tool	Trend Channel was drawing the parallel line out of order on indicator panel
107 72	Fixe d	Drawing Tool	Could not move draw objects in indicator panel
107 44	Fixe d	Drawing Tool	Trend Channel additional lines option did not exist
106 23	Fixe d	Drawing Tool	Global drawing objects were disappearing on restart when not applied to the primary data series
107 52	Fixe d	FX Board	ATM was no longer selected after tile size change
107 47	Fixe d	FX Board	ATM strategy orders were not being grouped
107 43	Fixe d	FX Board	Selected ATM strategy was not removing
107 35	Fixe d	FX Board	Order grid state column displayed incorrect on order rejection
107 34	Fixe d	FX Board	Display value for TimeLastTick was not resetting when disconnected
107 21	Fixe d	FX Board	Editing ATM settings from a saved workspace was sometimes resulting in an error
107 22	Fixe d	FX Board,	FXCM default quantities were not matching across order entry windows

		FX Pro	
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

4.3 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

105 76	Fix ed	ATM Strategies	Keep selected ATM Strategy template on order submission selected active ATM instead of template
106 56	Fix ed	Attach Order To Indicator, UI	Indicator name missing for blank labels in Attach Order To Indicator dialog
106 94	Fix ed	Bars	On Fridays, currently building weekly bar only contained that date
104 92	Fix ed	Bars	HeikenAshi BarsType Chart label was incorrect
104 83	Fix ed	Bars	Historical data recording thread locking .ncd file causing various crash reports
105 99	Fix ed	Basic Entry, FX Board, FX Pro, UI	Some properties were not available when clicking orders grid
106 52	Fix ed	Chart	Empty panel remained after removing indicator while Chart was loading
106 45	Fix ed	Chart	Crosshair incorrectly enabled after clicking Chart
106 40	Fix ed	Chart	Maximize panel button group on Chart moves the historical bar button location on some DPIs
106 29	Fix ed	Chart	Chart would sometimes freeze while changing properties
106 19	Fix ed	Chart	Chart right side margin property allowed negative values
105 98	Fix ed	Chart	Crosshair rendered incorrectly after interval change
105 68	Fix ed	Chart	Scale justification fixed scale icon remained after scale was no longer used

105 66	Fix ed	Chart	Non-default bar width applied incorrectly when new tab created via direct type into Chart
105 46	Fix ed	Chart	Taskbar preview caused extreme memory usage as mouse held on task bar
105 40	Fix ed	Chart	Jump to execution was not working on time-based Charts
105 24	Fix ed	Chart	Chart did not auto-scroll on multi-series Chart with global Crosshair
105 19	Fix ed	Chart	Crosshair disappeared when using a Drawing Tool
105 09	Fix ed	Chart	Automated strategy indicators could not be dragged and dropped
105 07	Fix ed	Chart Trader	Chart Trader missing grids when used on another tab
104 85	Fix ed	Chart	Deleting Chart Panel sometimes caused ChartStyle OnRender error
103 92	Fix ed	Chart	ChartStyle was changing in advance of new series loading
104 87	Fix ed	Chart Trader	Simulated Stop-Limit order in Chart trader caused rejected order when triggered
106 84	Fix ed	Chart Trader, Orders	Chart trader sell stop limit order with positive offset in remained in trigger pending state
105 63	Fix ed	Chart, DrawingTo ol	Draw object on non-primary panel was removed on reload NinjaScript
106 07	Fix ed	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

106 23	Fix ed	DrawingTo ol	Global Drawing Objects disappeared on restart when not applied to the primary data series
105 50	Fix ed	DrawingTo ol	Making Fibonacci levels all invisible would dead lock the Chart
105 23	Fix ed	DrawingTo ol	Changing of Reward Anchor Y in Drawing Objects properties did not work
105 06	Fix ed	DrawingTo ol	AutoScale Draw.Text was causing incorrect AutoScale and hotkey to not work
105 15	Fix ed	DrawingTo ol	DrawingTool Slot Index changed when directly setting anchor to the same value
105 21	Fix ed	DrawingTo ol	Compressing the Chart changed angle of Gann fan
103 96	Fix ed	DrawingTo ol	Arc Drawing Tool line property affected the straight line instead of the Arc line
106 69	Fix ed	DrawingTo ol	Setting Arc drawing object to global resulted in exception
102 95	Fix ed	DrawingTo ol	Region Highlight did not recognize Z-Order change
102 69	Fix ed	DrawingTo ol	Bar and price did not snap correctly with Trend Channels Drawing Tool
102 50	Fix ed	DrawingTo ol	Draw object errors when changing multi series Chart series
104 67	Fix ed	DrawingTo ol, Indicator	Indicator using Rectangle type could not be used on two instance/panels of same Chart
105 35	Fix ed	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

106 38	Fix ed	Instruments	Offset values would reset to server offset regardless of rollover date change
106 25	Fix ed	Interactive Brokers	Real-time equity volume was divided by 100
105 77	Fix ed	Interactive Brokers	TWS live account was not obeying signal names in managed approach
105 13	Fix ed	Interactive Brokers	Position was not closed when selecting close under rare circumstances
104 79	Fix ed	Kinetick	USDCHN instrument was not displaying data
103 79	Fix ed	Kinetick	Kinetick remain connected when IP changed
106 95	Fix ed	Log	Missing "reversing..." log message on clicking REV button on order UI
106 61	Fix ed	Market Analyzer	Default parameters shown in columns window after loading a template
105 67	Fix ed	Market Analyzer	CurrentText value was not available to NinjaScript after reloading workspace
105 27	Fix ed	Market Analyzer	Position Avg Price column would display whole numbers only
106 44	Fix ed	News	Minor news filter UI issues at DPI 125%
106 83	Fix ed	NinjaScript	RemoveDrawObject(tag) did not remove NinjaScript drawn global objects
105 52	Ad de d	NinjaScript	Added ChartBars.GetBarIdxByX to locate center of bar
105 43	Fix ed	NinjaScript	AddOn stayed in UI after assembly was removed

105 41	Fix ed	NinjaScript	Multi-series chart with TickReplay and Calculate.OnEachTick was not triggering OnBarUpdate correctly
105 48	Fix ed	NinjaScript	RenderTarget sometimes had null properties in the middle of OnRenderTargetChanged
104 70	Fix ed	NinjaScript	Internal exception on closing chart with Multi-Series indicator with heavy load
104 62	Fix ed	NinjaScript	Provide additional information to client on import of NinjaScript
105 28	Fix ed	NinjaScript	GetSlotIndexByTime would cause crash on Time-based Bar Spacing
104 66	Fix ed	NinjaScript Editor	SetState was missing from IntelliPrompt
105 36	Fix ed	NinjaScript, Workspace s	Workspace persistence was not working for Add-ons immediately after import
104 88	Fix ed	Orders, Risks	Risk template Max Position Size was not working
105 14	Fix ed	Other	Updated mail to support to work with Yahoo and AOL domains
106 74	Fix ed	Playback	Right click in playback controller triggered a data reload
105 12	Fix ed	Playback	Playback reset caused exceptions in indicators
103 82	Fix ed	Playback	Playback chart was not always auto scrolling with playback
105 04	Fix ed	Property Grids	Some property grid combo box's were missing UI hover effect

107 18	Fix ed	Rithmic	Rithmic adapter for TopStepTrader did not set the correct routing information which resulted in rejected orders
106 43	Fix ed	Strategy	Live strategy could be visually "enabled" while global simulation mode was on
105 79	Ad de d	Strategy Analyzer	Added dialog when opening Strategy Analyzer running 32-bit NinjaTrader on a 64-bit machine
106 82	Fix ed	Strategy Analyzer	Optimization summary tab was not populating for some results
106 66	Fix ed	Strategy Analyzer	3D Optimization graph display view would reset when duplicating
105 94	Fix ed	Strategy Analyzer	Saving Strategy Analyzer preset resets non-default column widths
105 56	Fix ed	Strategy Analyzer	Resolved issues with drag/dropping tabs
105 17	Fix ed	Strategy Analyzer	Column presets not working as expected
105 90	Fix ed	Strategy Analyzer, Templates	Was not loading instrument from template
105 91	Fix ed	Strategy Analyzer, UI	Preview window with chart display sometimes did not display
106 89	Fix ed	Strategy Builder	Custom series allowed for values to be set of the wrong type
106 76	Fix ed	Strategy Builder	Condition Builder incorrectly uses close data series when other Data Series selected
106 10	Fix ed	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

4.4 Beta

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

Version	Released
8.0.0.14 (RC2)	October 6, 2016
8.0.0.13 (RC1)	August 31, 2016
8.0.0.12 (Beta)	July 11, 2016
8.0.0.11 (Beta)	May 12, 2016
8.0.0.10 (Beta)	March 21, 2016
8.0.0.9 (Beta)	February 16, 2016
8.0.0.8 (Beta)	January 8, 2016
8.0.0.7 (Beta)	December 2, 2015
8.0.0.6 (Beta)	October 26, 2015
8.0.0.5 (Beta)	September 21, 2015
8.0.0.4 (Beta)	September 10, 2015
8.0.0.3 (Beta)	July 14, 2015
8.0.0.2 (Beta)	May 28, 2015
8.0.0.1 (Beta)	May 4, 2015

4.4.1 8.0.0.14 (RC2)

Release Date
October 6, 2016

As this is a release candidate, NinjaTrader 8.0.0.14 is still considered a beta product and we will continue to focus on product quality. Please continue to report any issues you may encounter to our support staff.   We will monitor the status of this release to determine when we will announce and launch the production release of NinjaTrader 8.

There were no code breaking changes to documented code in this release.

Notes

Status	Issue #	Category	Comments
Fixed	10455	Account Data	Commissions were not subtracted from Sim101 cash value
Fixed	10448	Alerts	Copy function was inconsistent with other features
Changed	10442	Alerts	Rearm type "never" was not disabling the alert
Fixed	10333	Attach Order To Indicator	Indicator disappeared with Attached Order to it and then added a secondary data series
Fixed	10374	Attach Order To Indicator, Chart	Reloading NinjaScript with Attached Order did not notify that attached to would be disabled
Fixed	10370	Bars	Bars were not building correctly on gaping pre-market equities
Fixed	10403	Bars, NinjaScript	Bars were sometimes null when updating calling calculate min-max logic
Fixed	10469	Bars, NinjaScript	Multi-Series script sometimes failed to execute in real-time
Fixed	10249	Bars, Playback	Renko bars did not playback correctly in some situations

Fixed	10346	BarsType	Point and Figure bars were building differently in real-time compared to NT7
Fixed	10435	BarsType	OnDataPoint() did not always have bid/ask property set on custom bar type built from 1 tick series
Fixed	10430	Chart	Dragging and dropping indicator on a custom sized panel would revert to default size unexpectedly
Fixed	10429	Chart	Data Box was not showing "N/A" for indicator plot values which are not set yet
Fixed	10401	Chart	Data Box data series label was not shown in Data Box
Fixed	10371	Chart	Left-axis crosshair price marker outline did not match right-axis
Fixed	10387	Chart	Multi-Series crosshair time was sometimes incorrect
Fixed	10372	Chart	Exception could occur on dragging secondary series to primary panel with attached indicator
Fixed	10325	Chart	Chart rendering would fail after computer monitor turns off/resumes
Fixed	10419	Chart	Crosshair performance improvements and new "Draw Cursor only" crosshair chart property
Fixed	10408	Chart	Opening a new chart changed to open center parent
Fixed	10402	Chart	Performance issue when scrolling chart with data box
Change	10421	Chart	Show date range time formatting on charts was not matching chart format

d			
Fix ed	102 76	Chart, Indicator	Woodies CCI "CCI Panel" permanently would change chart right side margin after user action
Fix ed	104 52	Chart, Playback, Tick Replay	Playback controller was sometimes disabled while connecting
Fix ed	104 51	Chart, Property Grids	Bars Dialog did not apply edited settings after pressing OK
Fix ed	103 90	Chart, UI	Moving a tab to a new window required multiple clicks
Fix ed	103 29	Chart, UI	Restoring presets twice before applying sometimes resulted in unhandled exception
Fix ed	103 80	Commissio ns	Profit commission calculation issue on a trade was incorrect in scale in/out case
Fix ed	103 10	Connection s	Simulated Data Feed: enabling connect on start up caused issues
Fix ed	103 39	Continuum	Race condition caused orders to remain in cancel pending
Fix ed	102 81	Control Center, FX Board	Order Grid context menus were not consistent throughout the product
Fix ed	104 12	Control Center, UI	Control Center tab rename window retained information of the selected tab
Fix ed	104 11	Control Center, UI	Control Center Log tab could not be renamed
Fix ed	103 77	Core	Users received 'FATAL: there's no market data handler to unsubscribe' message in various scenarios.

Fixed	10373	CQG	Race condition caused orders for unknown order CQG.Adapter.Cancel
Fixed	10407	Database	Rollover logic should exclude ##-## continuous contracts
Fixed	10308	Database	Adding rollover could yield exception when on certain dates
Fixed	10427	Drawing	Non-Equidistant bar spacing causing unmovable draw objects and NinjaTrader crash
Fixed	10395	DrawingTool	Arc Drawing Tool line was 2px dot, should be 1px dash
Fixed	10389	DrawingTool	Manually adjusting global draw objects anchor did not work properly
Fixed	10350	DrawingTool	Draw.Text methods were not matching color of axis
Fixed	10295	DrawingTool	Region Highlight did not recognize Z-Order change
Added	10161	DrawingTool	ExtendedLine global overloads were missing
Fixed	10394	FX Pro	Errors could occur in real-time market data handling
Fixed	10426	FXCM, Orders	Was possible to move FX order to invalid price on Sim101
Fixed	10286	Historical Data Window	Exceptions could occur on accessing historical data for instrument which did not exist
Fixed	10259	Historical Data Window	Exceptions could occur when saving excluded data in Historical Data menu

Fixed	10312	Indicator	Pivots did not match between 7 and 8 when using tick based charts
Fixed	10277	Indicator	Woodies CCI Sidewinder Text Label were not visible on real-time data
Fixed	10302	Indicator	Pivot indicator weekly pivots did not match intra-day verses daily bars data.
Fixed	10368	Indicator	Indicator NinjaScriptInfo text was not working on individual panels due to draw on price panel defaults
Fixed	10251	Indicator	Changing the namespace to added indicator caused crash
Changed	10255	Indicator, NinjaScript	Adjusted default NinjaScript colors to be generically visible on contrasting white and dark skins
Fixed	10288	Indicator, SuperDOM	Restoring indicator preset generated unhandled exception
Fixed	10450	Instruments	Instrument Lists were not always sorted alphabetically
Fixed	10369	Instruments	Instrument type drop-downs were not sorted alphabetically under non-English language settings
Fixed	10324	Kinetick	Connection status not properly reflected under some error scenarios
Fixed	10432	Licensing	Changing to Direct Edition license key caused custom 3rd party code to crash
Fixed	10441	Licensing	Direct Edition incorrectly blocked add-ons
Fixed	10289	Licensing	Vendor License Add-on displayed "duplicate" row upon applying license edit

Fixed	10439	Localization, UI	Errors were seen when adding Share Services when set to another language
Fixed	10294	Market Analyzer	Sorting Market Analyzer rows were incorrectly including blank rows.
Fixed	10336	Market Analyzer	Market Analyzer total row was not always showing value
Fixed	10318	Market Analyzer, Templates	Template label rows persisted after loading new template
Fixed	10437	NinjaScript	Exceptions were seen on generating from Code Wizard with Brush Description
Fixed	10287	NinjaScript	Implemented ForceRefresh() method for Strategies and DrawingTools
Fixed	10446	NinjaScript	Adding series with AddDataSeries() causes OnBarUpdate to malfunction on lower primary time frames
Fixed	10445	NinjaScript	Improved logging when user creates scripts that generate more than 65535 brushes set to BarBrush, etc. properties
Fixed	10399	NinjaScript	Improved logging for DirectX D2DERR_WRONG_RESOURCE_DOMAIN/WrongResourceDomain to direct to help guide
Fixed	10391	NinjaScript Editor	Right-click context menu copy / cut / paste were sometimes grayed out in NinjaScript Editor
Fixed	10342	NinjaScript, Orders	Stops and targets automatically cancelled incorrectly on multi-series strategy
Fixed	10381	NinjaScript, Strategy	Incorrect execution time on strategies due to multi time frame processing

Fixed	10413	Options	"Compiled Successfully" option was missing from Tools > Options > Sounds
Fixed	10303	Playback	Playback controller could incorrectly display times in the future
Fixed	10291	Playback	Migration from NT7 playback data could yield in time offset
Fixed	10362	Skins	Styled NTMessageBox Hyperlink so it can be skinable
Fixed	10323	Skins	Up/Down arrow highlight looked disabled on some skins
Changed	10405	Skins	Updated all position and profit/loss brushes using existing skin resources
Changed	10360	Skins	Data Series long execution brush was hard to see on dark skins
Changed	10359	Skins	SFT-1516 - Slate Gray ItemHover blue color makes position PnL Red hard to read
Changed	10447	Strategy	Updated Fill Engine to fill limit orders clamped to the high/low of the bar on gap up/down bars
Fixed	10352	Strategy Analyzer	Incorrect instrument was loading with chart-saved strategy template
Fixed	10344	Strategy Analyzer	Booleans were not working correctly in Strategy Analyzer optimization
Fixed	10400	Strategy Analyzer	Performance improvements on Basket Backtest for futures contracts

Fixed	10361	Strategy Analyzer	Maximizing Strategy Analyzer chart was not positioned correctly
Fixed	10431	Strategy Builder	"If All" condition group using OR instead of AND logic
Fixed	10416	Strategy Builder	TimeSpan was incorrectly adding seconds which was not editable on UI
Fixed	10417	Strategy Builder	Offsets were incorrectly applied on Cross conditions
Fixed	10393	Strategy Builder	Unable to use indicators that used public enums
Fixed	10345	Strategy Builder	Set color action resulted in exception
Fixed	10343	Strategy Builder	Drawings Tool stroke widths were defaulted to 0
Fixed	10340	Strategy Builder	Setting Time input/variable was missing quotes
Fixed	10331	Strategy Builder	Creating a Time parameter resulted in only the date being selectable
Fixed	10328	Strategy Builder	Selected offset arithmetic were not applied
Fixed	10330	Strategy Builder	Exit methods were missing SignalName / FromEntrySignal
Fixed	10319	Strategy Builder	Editing an order price value was not persistence
Fixed	10301	Strategy Builder	Indicator input series display was inconsistent with other features
Fixed	10307	Strategy Builder	Minor UI improvements on string builder

Fixed	10309	Strategy Builder	Indicator input series was sometimes listed twice
Fixed	10273	Strategy Builder	Improved Convert.ToInt32 usage and visibility
Changed	10386	Strategy Builder	Removing rows from string builder did not work as expected
Fixed	10351	SuperDOM	Switching SuperDOM tabs could result in unhandled exception
Changed	10428	SuperDOM	Increased default levels of Market Depth to 10 on SuperDOM
Fixed	10320	TD AMERITRADE	Real-time Index data was not being processed
Fixed	10415	Tick Replay	Tick Replay was not correctly disabled for Renko or LineBreak
Fixed	10189	Tick Replay	Tick replay should no longer run on bid/ask data series
Fixed	10384	UI	Pressing apply would incorrectly some windows
Fixed	10349	UI	Add-on popup message minor grammar errors
Fixed	10300	UI	Order Confirmation Dialog on Strategies was displayed too frequently
Fixed	10290	UI	Chart Trader order confirmation window did not have focus after Windows 10 update

Fixed	10357	UI	Pressing "Enter" on a dialog did not default to the "OK"/"Cancel" buttons
Fixed	10385	Workspaces	Workspace with the same name opens after removing workspace
Fixed	10409	Workspaces	Workspace did not save/restore data series label

4.4.2 8.0.0.13 (RC1)

Release Date

August 31st, 2016

This release marks our first NinjaTrader 8 Release Candidate. Since the very first NinjaTrader 8 beta version, we have resolved over 4500 bugs thanks to the ongoing efforts of our beta community, and we feel confident we are closer than ever to a production NinjaTrader 8 release. If you have installed a NinjaTrader 8 beta version and reported feedback to our support team - thank you!

As this is a release candidate, NinjaTrader 8.0.0.13 is still considered a beta product and we will continue to focus on product quality. Please continue to report any issues you may encounter to our support staff. We will monitor the status of this release to determine when we will announce and launch the production release of NinjaTrader 8.

Attention MB Trading Users: Due to limited use and low user feedback during the beta period, we have removed the **MB Trading** adapter from NinjaTrader 8. You may continue to use NinjaTrader 7.

Code Breaking Changes

Compile Errors

- The Stroke object **.Dispose()** method was removed due to technical redundancy. To remove memory resources from any stroke objects, simply set the stroke to null.
- Removed property **Bars.IsTimebased** -> please use Bars.BarsType.IsTimeBased instead
- **Account.Accounts** was renamed to Account.All

Implementation changes

- The common signature "isInclude60" used in various SessionIterator methods was renamed to "includesEndTimeStamp" to be more specific
- Category display order values of standard NinjaTrader **Property Grid Categories** were updated to be more consistent application wide. These changes could impact any customization you were doing using the CategoryOrderAttribute, however, the

documentation was also updated to reflect the implementation more accurately and will allow you to use this attribute reliably.

- To assist with transitioning historical order objects to real-time order references, please use the new GetRealtimeOrder() method.

Notes

Status	Issue #	Category	Comments
Fixed	10267	Account Data	Sorting by Commission in Executions tab caused unhandled exception
Fixed	10126	Account Data, Database	Database caused accounts to show Realized PnL when not connected
Fixed	10123	Account Data, Rithmic,	Realized PnL only reflected 1 side of the commission
Fixed	10003	Alerts	Alert Condition Localization issue
Fixed	10164	Alerts	Exception on opening alerts log window when existing alerts log entry used custom brush
Fixed	10192	Alerts, DrawingTool	Alerts stop triggering after editing drawing tool anchor via drawing objects window
Fixed	10131	ATM Strategies	ATM field did not change when submitting ATM from another window on startup
Fixed	10253	ATM Strategies	ATM is terminated and ATM order cancelled upon order modification failure
Fixed	10279	ATM Strategies	Modifying ATM Order modified ATM template

Fixe d	102 17	Attach Order To Indicator, NinjaScript	Attaching order to indicator via CTRL key resulted in Unhandled Exception
Fixe d	101 37	Bars	Crashes could occur when restoring workspaces
Fixe d	102 20	Bars	Dates were not showing correctly in New Zealand on daily bars
Fixe d	994 0	Bars	Errors could occur when loading data from TD Ameritrade.
Fixe d	101 32	Bars	RequestBarSeries1 ERROR while removing workspace in the middle of bars request
Fixe d	101 05	Bars	Toggling the Break at EOD with DoNotMerge on large data sets caused blank charts or less bars
Fixe d	100 36	Bars, NinjaScript	BarsRequest MergePolicy did not match UseGlobalSettings when using the same policy
Cha nged	101 36	BarsType	Custom BarType time variable was equal to bars.LastBarTime on new data point
Fixe d	101 60	BarsType	Custom BarsType OnDataPoint sometimes had unexpected bid ask data
Fixe d	100 86	BarsType	Point and Figure Charts were not calling OnBarUpdate() for each tick or price change
Fixe d	100 95	Chart	Chart were allowed to remove last series incorrectly
Fixe d	102 80	Chart	Crosshair position showed old position when toggled via hotkey until mouse move

Fixe d	102 40	Chart	Drag and drop primary series in same panel caused indicator to jump to primary panel
Fixe d	102 84	Chart	Extraneous plot selection point displayed on multi-level indicator as input plot
Fixe d	102 18	Chart	Global crosshair render issue at chart boundaries
Fixe d	101 78	Chart	Global crosshair incorrect x axis flag time when locked via context menu
Fixe d	101 56	Chart	Global draw object was not removed with NinjaScript unless NinjaScript removed manually
Fixe d	100 97	Chart	Save Chart Image file name only included the primary data series
Fixe d	102 42	Chart	When primary series deleted on multi series chart, instrument selector not updated to new primary
Fixe d	102 36	Chart	Z-Order: reloading the historical data reset the z-order
Fixe d	102 45	Chart Trader	Errors using ChartTrader with Playback connection
Fixe d	102 48	Chart Trader, Templates	Chart Trader settings were ignored when chart template is loaded on an open chart
Fixe d	102 37	Chart, Drawing	Unhandled exceptions when moving series panels with GlobalDrawObjects
Fixe d	980 6	Chart, Drawing, NinjaScript	Chart could freeze using NinjaScript Draw method/Removing Drawing Tools
Fixe d	101 17	Chart, Indicator	System indicators did not load properly after connecting to live data

Fixed	10183	Chart, Templates	Template caused "An item of the same key has been added" when using two of the same indicator
Fixed	10254	Chart, Workspaces	InvalidOperationException on restoring chart with template/preset
Fixed	10002	Commissions	Forex Commissions were factoring Per-Unit instead of Per Lot Size
Fixed	10046	Control Center	Connection status indicator sometimes did not update
Fixed	10144	Control Center	Edit strategy dialog was throwing incorrect error
Fixed	10091	Control Center	Deleted account connection could be disconnected
Fixed	10023	Control Center, Playback, Strategy	Playback trades performance realized PnL did not match strategies tab of control center
Changed	10111	Control Center, Strategy	SystemPerformance object was not updating for control center enabled strategies
Fixed	10158	Control Center, Strategy	Strategy CurrentBars index repored incorrectly after changing parameter and enabling strategy
Fixed	10275	Control Center, Workspaces	Control Center was incorrectly restored to primary screen when saved maximized
Fixed	10139	CQG, Workspaces	Workspace charts did not load after connecting to account with no data

Fixed	9909	Drawing, Strategy Analyzer	Strategy analyzer was not releasing memory when adding indicator that draws objects
Fixed	10167	DrawingTool	"Remove all drawing objects" did not remove objects unless the originating tab is selected
Fixed	10184	DrawingTool	Draw object incorrect resize or incorrect anchor on attempting to move past start bar of chart
Fixed	10113	DrawingTool	Draw objects in future would move when days to load changed
Fixed	10265	DrawingTool	Draw.Text autoscale was not working correctly
Fixed	10264	DrawingTool	Draw.Text no alignment behavior did not match NT7
Fixed	10099	DrawingTool	Drawing tool template with attach to all charts was drawing on other charts even if cancelled
Fixed	10179	DrawingTool	Moving global draw object on multi series chart changed anchors incorrectly
Fixed	10061	DrawingTool	Ray selection points were not lined up on logarithmic y-axis scale
Fixed	10127	DrawingTool, Playback	Global draw anchors were not consistent in multi-series playback
Fixed	10224	DrawingTool, Templates	Unable to apply a template more than once to a drawing object
Fixed	10124	eSignal	ESignal historical tick data timestamps did not match NT7

Fixed	10214	FX Board, Hotlist Analyzer, Market Analyzer	Create instrument list context menu item did not work if no instruments loaded
Changed	10208	Indicator	Consistency updates to indicator error handling
Fixed	10157	Indicator	BuySellPressure when called from another indicator could cause errors
Fixed	10274	Indicator	Indicator label did not saved with preset
Fixed	9748	Instruments	@GER30 CFD Data does not show decimal values
Changed	10145	Instruments, Yahoo	Yahoo connection is not updating Splits and Dividends.
Changed	10170	Interactive Brokers	Removed IB Linked Account Support
Fixed	10212	Interactive Brokers	IB Paper Trading account would not connect on version 954 TWS
Fixed	9993	Interactive Brokers	Incorrect instrument update price on some instruments
Fixed	9923	Kinetick	Connection loss loop could occur in some situations
Fixed	10088	Licensing	FreeTrial Vendor License caused excess lines in Config.xml
Fixed	10209	Licensing	From and to dates in grid in Vendor Licensing window region formatting
Fixed	10060	Licensing	Vendor License Addon did not update license messages when changing between vendors

Fixed	10108	Market Analyzer	Market Analyzer Indicator settings were not recognized when applying template
Fixed	10201	Market Analyzer	Unable to remove expired instruments from market analyzer in some scenarios
Changed	10196	Market Analyzer, Workspaces	Custom Market Analyzer Column could not serialize CurrentText
Changed	10180	MBTrading	Removed MBT Adapter in NinjaTrader 8
Changed	10106	NinjaScript	Incorrect sound played when SetProfitTarget target is reached
Changed	10260	NinjaScript	Renamed a few properties to meet coding guidelines
Fixed	10071	NinjaScript	CancelOrder() would not cancel historical working orders in State.Realtime
Fixed	10222	NinjaScript	Errors could occur after deleting indicator and reloading chart
Fixed	9927	NinjaScript	Draw.Region displacement was from the left of the chart and not from bar 0
Fixed	10206	NinjaScript	Expandable properties were not reseting to defaults
Fixed	10177	NinjaScript	Indicator Error on calling 'SetState' method with tick replay
Fixed	10112	NinjaScript	IsSuspendedWhileInactive did not work if suspended prior to data feed connection
Fixed	10221	NinjaScript	Null Stroke object in AddPlot() cuased platform crash

Fixe d	101 53	NinjaScript	Unhandled exception if using PasswordBox in Addon
Fixe d	101 72	NinjaScript	Update() on multiseries indicator caused primary series OBU called twice
Cha nged	841 0	NinjaScript Editor	Added Additional Default Snippets
Fixe d	100 85	NinjaScript, Orders	Only first identical State.Historical order moved to State.Realtime
Fixe d	102 33	NinjaScript, Strategy	Enabling multiple AdoptAccountPosition strategies would result in incorrect popup message
Fixe d	101 40	NinjaScript, UI	Strategy Catagories/properties could get out of sequence in strategy and in strategy analyzer
Cha nged	101 62	Options, Strategy	ConnectionLossHandling was overwritten between State.SetDefaults and State.Configure
Fixe d	101 48	Orders, SuperDOM	Cancel all order icon was not visible for simulated orders on SuperDOM
Fixe d	102 05	Playback	Simulation accounts could be added unexpecitly while connected to playback
Fixe d	101 38	Playback	Exception was thrown unsubscribing data disconnecting playback
Fixe d	102 02	Playback	Errors existed around persisting errors to database
Fixe d	101 42	Playback	Playback controller date, time and speed sometimes was not visible
Fixe d	102 11	Playback	Playback controller end date did not update until platform restart

Fixe d	101 49	Playback	Playback manual trade MAE/MFE/ETD values were incorrect
Fixe d	997 4	Playback	Sustained playback could resultng in chart rendering error
Fixe d	102 58	Playback	Errors could occur connecting to playback on UTC time zone
Fixe d	102 00	Skins, UI	Chart Properties Tab Name drop down was barely visible
Adde d	100 87	Strategy	Strategy concept needed to help users manage historical to live transitioned orders
Fixe d	101 07	Strategy	Limit order did not fill if CancelOrder() used on protective order
Fixe d	100 64	Strategy	Realized PnL was not updating for strategy with secondary series on strategies tab
Fixe d	101 28	Strategy	Strategy template options are not available when editing from strategy tab
Fixe d	101 94	Strategy Analyzer	Backtest sometimes ran twice if display was not set to summary or settings
Fixe d	101 43	Strategy Analyzer	Platform crash during optimization when accessing null object after State.Terminated
Fixe d	100 10	Strategy Analyzer	Platform could crash when using duplicate to new window
Fixe d	101 81	Strategy Analyzer	Trades Display "Strategy Column" was blank after duplicated
Fixe d	102 10	Strategy Analyzer	Optimization results did not match displays

Fixed	10204	Strategy Analyzer	Tab context menu items could become disabled incorrectly
Fixed	10263	Strategy Analyzer	Strategy template was not saving changes after running a backtest
Changed	10118	Strategy Builder	Allows Strategy Builder Strategies to be manged from NinjaScript Editor
Changed	10101	Strategy Builder	Could not export Strategy Builder strategy
Changed	10168	Strategy Builder	Now freezing custom brushes in Strategy Builder to match best practices
Changed	10100	Strategy Builder	No historical trades taken due to indicators with barsAgo check
Fixed	10103	Strategy Builder	Strategy Builder could lose reference to candlestick pattern
Fixed	10090	Strategy Builder	Plot On Chart was calling AddChartIndicator() in wrong state
Fixed	10225	Strategy Builder	Drawing action category was using wrong name
Fixed	10272	Strategy Builder	Exception selecting indicator with NinjaScriptProperty value not set
Fixed	10163	Strategy Builder	Alert message defaulted to same color for foreground/background
Fixed	10262	Strategy Builder	Did not set indicator Brush properties
Fixed	10271	Strategy Builder	Strategies with errors could not be deleted
Fixed	10155	Strategy Builder	Time series could not be compared with Time input or variable in Strategy Builder conditions

Fixed	10082	Strategy, Trade Performance	Strategy performance calcs had been off
Fixed	10083	SuperDOM	Order action name localization issue
Fixed	10246	SuperDOM	SuperDOM > Columns "i" did not give information on the Columns
Fixed	10187	SuperDOM, Workspaces	SuperDOM Indicator Days to load property reverts to 2 on restore workspace
Fixed	10195	TD AMERITRADE	Historical daily bars could hold more decimal places than TickSize
Fixed	10285	Time and Sales	Columns unexpectedly resize when removing/adding columns
Fixed	10203	UI	Account Tab -> Typing or selecting in account selector did not send to other linked windows
Fixed	10223	UI	Control Center could sometimes be out of focus after using menus
Fixed	10238	UI	When selecting the same instrument in the data series window (in the most recent list), the instrument did not add.

4.4.3 8.0.0.12 (Beta)

Release Date
July 11, 2016

Code Breaking Changes
Compile Errors

- The NinjaScript Strategy "**AccountSize**" concept was removed due to limited functionality. This change would also impact strategies which were coded to use SetOrderQuantity.ByAccountSize. Please use your own variables to set quantities by account size. **Tip**: NinjaTrader 8 can read real-world account values for live trading purposes through the Account class, e.g., `Account.Get(AccountItem.CashValue, Currency.UsDollar)` provides a method for returning a double value representing the current cash value of the account.

Implementation changes

- The concept used to force plot series objects through **ForcePlotsMaximumBarsLookBackInfinite** was disabled and tagged as obsolete. You should set any indicator/strategy plots to MaximumBarsLookBack.Infinite during its construction.
- It is recommended that any strategies generated with the **'Strategy Builder'** tool via Control Center > New > Strategy Builder be removed and recreated after installing B12. If you had invested significant effort in a **Strategy Builder** generated strategy in B11 which you would like to continue to use, please contact us platformsupport@ninjatrader.com

Notes

Status	**Issue #**	**Category**	**Comments**
Added	9127	NinjaScript	Added XML Comments to reflect Supported NinjaScript and NinjaTrader Core Methods (ongoing)
Added	9897	NinjaScript Editor	Added Multi-Select to the NS Editor References Win File Dialog
Added	9908	Rithmic	Added additional Rithmic Adapter Account Items
Added	9933	Window Linking	SFT-162 - Added more link button colors (up to 11)
Changed	10066	Account Data	Change Position Display option to "Show gross realized PnL when flat."
Changed	10076	FXCM	Reverted NT8 FXCM Forex volume to NT7 implementation

Added	9839	Interactive Brokers	Added TWS File picker to allow users to the specific location of TWS/Gateway executable which can vary in some scenarios. See the NinjaTrader 8 Interactive Brokers connection guide.
Changed	9976	NinjaScript	Disabled the .ForcePlotsMaximumBarsLookBackInfinite concept
Fixed	10022	NinjaScript Editor	Delimiter matching was hard to see
Fixed	9946	Orders	Trace Order reported old price after order change
Changed	9875	Strategy	Removed of "ByAccountSize" Strategy Order Option
Fixed	9887	Strategy Builder	Strategy Builder additional data window did not show instrument
Fixed	9996	UI	Removed UI Formatting of Market Data Volume (real-time and historical bar data)
Fixed	9879	Account Data	Account Data Window default tab order did not match order of Add Tab Menu Sub Items
Fixed	9992	Account Data	Excess Position Margin Column width was not saved/restored correctly
Fixed	9441	Adapter	Kinetick adapter was not properly resolving market data settlement time updates
Fixed	9977	Alerts	Alerts Log Window "priority" was not working as expected with some localization
Fixed	9876	Attach Order To	IsSuspendedWhileInactive did not always re-suspend after manually attached to

		Indicator, Chart	orders
Fixed	9906	Bars	Range Bars were built differently using custom data range
Fixed	9889	Bars	Renko Bars produced undesired ticks bars at EOD
Fixed	9854	Bars	Break EOD could cause issues with bar pool series when requested after session EOD
Fixed	9803	Bars, Chart	Bar spacing issues identified when scrolled into future which resulted in the "Automatically choose time-based series for x-axis (non-equidistant bar spacing)" concept to be removed. We will revisit this feature in the future.
Fixed	9859	Bars, NinjaScript	BarsSinceNewTradingDay was not operating as expected
Fixed	9985	Bars, Playback	Exception in multi-series Playback
Fixed	9856	Chart	Data Box did not update values as you scroll
Fixed	9843	Chart	Drag and drop series to tab was not creating new series as expected
Fixed	9934	Chart	Unable to use Instrument Search from new Data Series window
Fixed	9868	Chart	Fixed Data Box volume formatting issues for FX
Fixed	9858	Chart	Gap in Chart and Chart Window at 150% DPI
Fixed	1002	Chart	Multi-Series chart bar width was incorrect

	5		
Fixed	10078	Chart	Crosshair time label rendering incorrect on left side
Fixed	10058	Chart	Price market incorrect in condition with days to load and session template
Fixed	9988	Chart Trader	Changing colors of the action, buy, and sell buttons did not save
Fixed	9884	Chart, Drawing	Draw objects in future would move when chart timeframe changed
Fixed	9943	Chart, DrawingTool	Mouse cursor was not reverting to normal when canceling the creation of a draw object by right-clicking
Fixed	9964	Chart, Indicator	Out of range exception was thrown using Horizontal Line plot on Multi-Series chart
Fixed	9837	Chart, NinjaScript, Tick Replay	Tick Replay sometimes caused SynchronizationLockException when loading indicator
Fixed	9831	Chart, Sessions	Rollover date was missing on Default 24/7 with Break at EOD unchecked
Fixed	9931	Chart, Strategy Analyzer	Strategy Analyzer Data Box opened from trade review chart doesn't show trade execution
Fixed	9929	Chart, Strategy Analyzer	Aborting in Strategy Analyzer leaves disabled script on chart
Fixed	10048	Chart, Strategy Analyzer	Unhandled exception when switching to chart in Strategy Analyzer

Fixed	9864	Chart, Templates	Drawing tool option selection "Stay in Draw Mode" was not included when saving chart template
Fixed	10069	Code Wizard, Strategy Builder	Additional data option missed week and year type as choice
Fixed	10012	Commissions	Some commission values displayed with trailing 0's
Fixed	9890	Control Center	Restoring live executions were sorted reverse
Fixed	9882	Control Center, Database	Restoring executions were working differently from NT7
Fixed	9695	Core	FXCM Adapter reported duplicate trades in a race condition
Fixed	9894	CQG	Invalid error occurred when trying to connect to CQG after failed CQG logon
Fixed	9910	Data Grids	Tri-State Sorting was not working in all grids
Fixed	10075	Data Grids	Account Tab Filter With Positions Was Not Updating as Account position changes
Fixed	9936	Database	Performance improvements during first ever application startup using default workspaces
Fixed	10042	Database, Instruments	Removed instrument in instrument manager would sometimes reappear
Fixed	9932	Database, Trade	TradesPerformance showing incorrect Bars value for some trades

		Performance	
Fixed	9939	DrawingTool	Ruler was incorrectly rounding pip sizes
Fixed	9945	DrawingTool	Stay In Draw mode did not apply to Text tool
Fixed	9970	DrawingTool	Additional settings displayed when selecting drawing tools in the Configured list.
Fixed	9963	DrawingTool	Changing drawing tool EndTime into future leaves drawing tool drawn as it was
Fixed	9895	DrawingTool	Drawing tools when switching contract months do not reappear on the chart.
Fixed	9983	DrawingTool	Performance improvements on default text drawing tool
Fixed	10035	DrawingTool	Ruler exhibited localization issue
Fixed	9738	DrawingTool	Setting DrawObject ChartAnchor.BarsAgo did not change rendered object
Fixed	10009	DrawingTool	Drawing Tools lost auto scale as chart scrolled into future
Fixed	10052	DrawingTool	TrendChannel Parallel Line Start Anchor did not snap when using bar and price mode
Fixed	9804	DrawingTool	Trend Lines second line did not respect Snap Mode.
Fixed	9925	DrawingTool,	DrawObjects collection was not updating with global draw object until NinjaScript

		NinjaScript	reloaded
Fixed	10038	DrawingTool, Strategy Analyzer	AddChartIndicator indicator was always rendering draw objects on primary panel in Strategy Analyzer
Fixed	10049	DrawingTool, Templates	PriceLevels reverted after re-compile in Fibonacci Drawing Tools
Fixed	9962	DrawingTool, Workspaces	Global lines inconsistently saved across workspaces using" workspace save as"
Fixed	10006	eSignal	Esignal was not processing real-time index data
Fixed	9935	FX Board	Unable to load ATM Strategy Template Error from FXBoard
Fixed	9950	FX Board	FXBoard Unrealized PnL field did not update until unrealized PnL changes
Fixed	10067	FXCM	FXCM order rejection scenario
Fixed	10008	Hot Key	Removing custom drawing tool code did not remove assigned hot key
Fixed	9912	Indicator	Bar timer did not function on Playback connection
Fixed	9953	Indicator	Indicator was plotting on the second bar on chart despite BarsRequiredToPlot = 0
Fixed	9965	Indicator	@VolumeProfile.cs results were skewed towards buy volume using Tick Replay

Fixed	10021	Indicator, SuperDOM	SuperDOM Indicator Data series input changes using multi-time frame indicators
Fixed	9869	Interactive Brokers	Unable to receive data from the instrument on the SEHK exchange.
Fixed	9961	Interactive Brokers	Unexpected behavior with chart "Price Based On" was set to Bid/Ask
Fixed	9893	Interactive Brokers	Was not processing Index data due to no volume reported
Fixed	9995	Interactive Brokers	Interactive Brokers MarketDataType.DailyHigh/Low was not coming in
Fixed	9994	Kinetick	Adapter was processing volume of "1" for indexes instead of "0"
Fixed	10034	Licensing	VendorLicense did not always show all configured free trials
Fixed	9820	Licensing, NinjaScript	VendorLicense in AddOn prevented OnWindowCreated from working as expected
Fixed	10040	Market Replay, Time and Sales	Playback connection T&S Window did not scroll precisely to user actions while using Scroll Wheel
Fixed	9898	NinjaScript	Multi-series indicator was running into deadlocks
Fixed	9857	NinjaScript	Drawing objects were still visible to NinjaScript after changing instrument on the chart
Fixed	9795	NinjaScript	Errors on calling 'EventHandlerBarsUpdate' method from Strategy

Fixed	9937	NinjaScript	Remove NinjaScript assembly window needed vertical scrollbar
Fixed	9955	NinjaScript	User configured indicator input series changed after exception
Fixed	9978	NinjaScript	PlotBrushes were not working as expected with AddChartIndicator()
Fixed	9823	NinjaScript	Indicator plot did not match when using a custom bar type
Fixed	10041	NinjaScript	Exception calling Draw.Triangle() from AddChartIndicator()
Fixed	10070	NinjaScript	Blank string used for Name property resulted in crash
Fixed	9928	NinjaScript Editor	NinjaScript Editor F3 did not work if Find window was closed
Fixed	9971	NinjaScript Editor	NinjaScript Editor invalid char allowed as name
Fixed	9865	NinjaScript, Orders	Stop price checks for strategies with OrderFillResolution High had been executed in the wrong bars series
Fixed	9956	NinjaScript, Strategy	Strategy disabled by SetState was considered as enabled in Strategies tab when attempting to edit
Fixed	10037	NinjaScript, Strategy Analyzer	Adding indicator to backtest chart with AddChartIndicators caused indicator plots to disappear
Fixed	9874	NinjaScript, Strategy Builder	Strategy Wizard did not utilize secondary series for input correctly. **Note**: This fixed resulted in change which preventing "Plot on chart" from working correctly, which is schedule to be fixed in the next

			beta release (# 10090)
Fixe d	99 42	Options	Changing Auto Close Position settings did not create warning if already connected
Fixe d	10 05 1	Orders	Exception on placing order to non-USD sim account created after connecting to provider
Fixe d	99 15	Playback	Unhanded exception on Playback disconnect
Fixe d	99 07	Playback	Tick Replay on Playback was frozen when using Go To
Fixe d	99 47	Playback	Empty warning pop-up when changing Playback start date
Fixe d	99 44	Playback	Playback did not "Go To" available date before current start date.
Fixe d	99 68	Playback	Execution markers were not plotting correctly in Playback Connection
Fixe d	99 75	Playback	Minor issues with Playback behavior if no data was available
Fixe d	98 99	Playback	Playback data auto-replaying on chart when created after Playback connected
Fixe d	99 89	Playback	Strategy exception with multiple range series during Playback
Fixe d	99 90	Playback	Did not disconnect all the way when disconnected before loaded
Fixe d	99 99	Playback	"Error in real time market data handling" when running Playback
Fixe d	99 86	Playback	Gaps on dragging slider on Playback data

Fixed	9979	Strategy	Strategies grid changes to respect region denomination
Fixed	9987	Strategy	Unexpected strategy accounts issue on Playback data
Fixed	9870	Strategy Analyzer	Data Box exception using chart display
Fixed	9855	Strategy Analyzer	Settings display was not updating include commissions param on subsequent backtests
Fixed	9836	Strategy Analyzer	Parameter tooltip displayed incorrectly
Fixed	9828	Strategy Analyzer	Control Center logo was sometimes being displayed in chart
Fixed	9980	Strategy Analyzer	Taskbar previews were not working
Fixed	9922	Strategy Analyzer	Analysis displayed incorrectly after running separate backtest on optimization result
Fixed	9973	Strategy Analyzer	Chart Exception: Cannot call DragMove or Activate before a Window is shown
Fixed	9842	Strategy Analyzer, Templates	Strategy template saved on chart did not restore correctly in Strategy Analyzer
Fixed	9917	Strategy Builder	Lookback period defaulted to 0
Fixed	9916	Strategy Builder	TimeSpan did not use user-applied settings
Fixed	9891	Strategy Builder	Various Indicator Syntax changes

Fixed	9888	Strategy Builder	Did not retain added data series in conditions
Fixed	9885	Strategy Builder	Unlocking code did not open NinjaScript Editor
Fixed	9883	Strategy Builder	Strategy opened in NinjaScript Editor sometimes produced unhandled exception
Fixed	9867	Strategy Builder	Did not prevent input of invalid Int value
Fixed	9861	Strategy Builder	Duplicate input series displayed in selector
Fixed	9938	Strategy Builder	Conditions window incorrectly scrolled to the top of the list of options when opening configured condition
Fixed	9984	Strategy Builder	Used incorrect Set methods OnStateChange()
Fixed	9991	Strategy Builder	Used incorrect value of 0 for selected order quantity
Fixed	9981	Strategy Builder	Consistency improvements regarding Condition/action copy/save
Fixed	10011	Strategy Builder	Did not confirm on removing strategy
Fixed	10015	Strategy Builder	Could not remove Strategy Builder Strategy when opened from NinjaScript Editor
Fixed	10043	Strategy Builder	Did not automatically compile strategy on clicking Finish
Fixed	1001	Strategy Builder	Save As' rejected a new name

	7		
Fixed	10050	Strategy Builder	Print() action was not accepting any arguments
Fixed	10055	Strategy Builder	File names were cut off in the strategy builder load/save strategy popup
Fixed	10039	Strategy Builder	Did not perform bars check on barsAgo and multi-series scripts
Fixed	10073	Strategy Builder	Sometimes lost active button highlight
Fixed	10045	Strategy Builder	String variable did not escape characters
Fixed	9951	Strategy Builder	Opening screen referred to "wizard" which caused confusion
Fixed	10077	Strategy Builder	German region decimal format caused compile errors
Fixed	9871	Strategy, Workspaces	Switching workspaces do not maintain realized PnL and randomly sorts rows in Strategies tab
Fixed	9924	TD AMERITRADE	Partial Position was not displaying while connected to TDA
Fixed	10056	Trade Performance	Changing start date/end date on trade performance prevented generating report for only current day
Fixed	1005	Trade Performanc	Trade Performance column sorting reset when regenerated

	7	e	
Fixed	10027	Data Grids	Performance Improvement on data grids (Changes require users to restore custom sorting/filtering manually)
Changed	10063	Indicator	Removed redundant logic of BuySellVolume and BuySellPressure indicators

4.4.4 8.0.0.11 (Beta)

Release Date

May 12, 2016

Code Breaking Changes

Compile Errors

- Draw.RegionHighlightY(), Draw.ArrowLine() – Added missing "isAutoScale" bool to some signatures

Implementation changes

- In some scenarios, the OnRender() method no longer guarantees the bars indexer to be up to date before it is called. Due to the multi-threaded nature of NinjaTrader, attempting to access a series barsAgo value could return seemingly unexpected results (E.g., Close[0], SMA(20)[0], etc). This change does **NOT** impact data driven events, such as OnBarUpdate(), OnMarketData(), OnOrderUpdate(), etc. For all OnRender() purposes, you should consider using an absolute index lockup through Bars.GetClose(barIndex), or <series>.GetValueAt(barIndex) which are generally more reliable in non-data driven events. Please also see the new IsValidDataPointAt() which was added to help with absolute index look up.

- Type Casting DrawObjects can fail from a compiled assembly. In situations where you need to cast a draw object in an assembly, it is recommended you use the dynamic type and compare the object type by its string. This is not required for non-protected scripts. Please see the help guide article on Considerations For Compiled Assemblies for more details and examples.

Notes

Status	Issue #	Category	Comments

Change	9829	Adapter	Gain is no longer supported with NinjaTrader 8.
Fixed	9390	Adapter	Unable to connect to TDA after migrating connection from NT7
Changed	9679	Alerts	Alerts Cross Above/below selector now more aware of current market dynamics
Fixed	9657	Alerts	Default lookback period for CrosssAbove/ Below condition in Alerts changed to 1
Fixed	9559	Alerts, Workspaces	Alerts Log column widths did not restore from workspace
Fixed	9618	ATM Strategies	Errors could be produced when modifying ATM order size of active ATM orders.
Fixed	9744	ATM Strategies, FXCM	ATM SL and PT were always not submitted for CFDs.
Fixed	9658	Attach Order To Indicator	Detaching Order from Indicator could result in errors
Fixed	9767	Backup & Restore	Restoring backup by double clicking file with NinjaTrader closed would not restore backup
Fixed	9586	Backup & Restore	Exceptions could be produced after importing a backup
Fixed	9567	Barchart	Barchart.com connection did not always report disconnection

Fixed	9794	Bars	Last bar of session showing high/low of entire session where bars are requested in gap between session definitions
Fixed	9070	Bars, Chart	Race conditions could cause render loop to freeze indefinitely !!! code breaking change !!!
Fixed	9611	BarsType, Chart	Default ChartStyle for Renko could change after a restart
Fixed	9681	Chart	X Axis Grid Lines could be slightly incorrect on non-equidistant charts
Fixed	9524	Chart	Race conditions after aborting pending bars requests could exhibit issues in real-time data subscriptions
Fixed	9792	Chart	Chart Label did not denote bid/ask market data types
Fixed	9703	Chart	Chart could be come unresponsive when using fully compressed time axis on some bars
Fixed	9589	Chart	Databox loses 'always on top' behavior after right clicking on chart
Fixed	9668	Chart	Intermittent issues with fixed horizontal grid lines not rendering
Fixed	9643	Chart	Crosshair time incorrect on multiseries tick-based chart
Fixed	9622	Chart	On 125%+ DPI global crosshair timestamp marker can show wrong timestamp
Fixed	9570	Chart	Dragging bar series from additional panel into panel 1 causes shared z-order
Fixed	9571	Chart	Chart does not update with additional bars when resized and equidistant bar spacing off
Fix	96	Chart	Multi-Series Chart Problems with Non-

ed	59		Equidistant Bar Spacing
Fixed	9561	Chart	Crosshair Label outline inconsistent with Indicator Price Labels
Fixed	9834	Chart	Databox uses local denomination for execution commission
Fixed	9678	Chart	ChartStyles GetBarPaintWidth logic inconsistent
Fixed	9671	Chart Trader	Chart Trader PnL flag resizes at random when scaling in/out of position
Fixed	9560	Chart, Drawing Tool	Fibonacci text size does not adjust
Fixed	9597	Chart, Indicator	Multi Series chart not using secondary input when configuring indicator from ui
Fixed	9796	Chart, NinjaScript	With multi-instrument strategy, plot executions drawing on both series instead of series executed against
Fixed	9772	Chart, NinjaScript	SetZOrder() method not setting chart object to desired ZOrder level
Fixed	9572	Chart, NinjaScript	Draw.Region in indicator draws on wrong panel using AddChartIndicator in strategy
Fixed	9551	Chart, NinjaScript	Empty leftover indicator panel after strategy added indicator removal
Fixed	9732	Chart, Strategy	"Reload NS" on charts will reload strategies on other charts using same instrument
Fixed	9742	Chart, Strategy Analyzer	SA execution chart draws executions incorrectly

Fixed	9665	Commissions	Commission "Per Instrument Type" is applied to incorrect row
Fixed	9711	Commissions	Minimum Commission not applied when no Per-Unit Commission on Historical Trades
Fixed	9736	Connections	Connect on startup did not always work correctly
Fixed	9821	Control Center	Error when exporting Strategies tab to Excel.
Fixed	9574	Control Center	Account connection status displays as yellow on subsequent connections
Fixed	9573	Control Center, Playback	Playback account connection status yellow on second connection attempt
Fixed	9569	Control Center, Strategy	Strategy restores on strategies tab even if not saved with workspace
Fixed	9558	Control Center, Strategy	Strategy not deleted from DB if it was attached to a chart
Fixed	9745	Core, Data Grids	Unrealized PnL not updating for all open positions
Fixed	9575	CQG	Crash if try to connect to two CQG connections
Feature	9677	Data Grids	Added tri-state grid sorting
Fixed	9751	Data Grids	Unknown' order state display should be treated as a terminal order
Fix	96	Data	Grid export default filename is inconsistent with

ed	73	Grids	other date-driven default filenames
Fixed	9582	Database	Database Window: Deselect contract does not prevent rollover from occurring
Fixed	9680	Database	ObjectDisposedException related to SqlCeConnection when shutting down NT
Fixed	9757	Database	Tools->Database->Update instrument/ instrument lists did not always work
Fixed	9797	Drawing	SnapMode "Disabled" is not respected on multi series charts
Fixed	9581	Drawing, NinjaScript	Draw objects not drawn initially when using overload that accepts DateTime
Fixed	9798	Drawing, Workspaces	Global objects do not fully delete from workspace they were not created/last saved in
Fixed	9849	Drawing Tool	DrawTools Text scaling issue on 4k displays
Fixed	9766	Drawing Tool	Some Draw methods missing "autoscale" argument !!! Code breaking !!!
Fixed	9802	Drawing Tool	Drawing object jumps away from cursor when selected
Fixed	9728	Drawing Tool	Global drawing object disappears after disabling/enabling 'Show global drawing objects'
Fixed	9576	Drawing Tool	Draw.TextFixed text centered on position instead of aligned
Fixed	9537	Drawing Tool	Drawing Tool Anchors in the Future on Tick Charts Moving

Fixed	9498	Drawing Tool	Draw objects change size when moved with mouse
Fixed	9827	Drawing Tool	RiskReward Anchor styles apply to all plots.
Fixed	9709	Drawing Tool	Fixed scaling issue with @RegressionChannel
Fixed	9583	Drawing Tool, NinjaScript	RemoveDrawObject does not update DrawObjects when called on historical data
Fixed	9614	Historical Data Window	Issues Importing Historical Tick data for minute bars
Fixed	9604	Indicator	Unexpected Autoscaling when using "Horizontal Line" plot
Fixed	9541	Indicator	Pivots can yield 'Error on calling OnBarUpdate method on bar 0'
Fixed	9840	Indicator	Exception thrown from multi-series chart with two different days to load configured
Fixed	9851	Indicator	InvalidOperationException on pulling up indicators dialog while compiling
Fixed	9634	Indicator, Market Analyzer	Cannot view all indicators in subfolder in indicators column of market analyzer
Fixed	9729	Indicator, SuperDOM	Indicator preset sets data series on SuperDOM indicator
Fixed	9793	Instruments	Orders placed on sim account(s) on a setup with multiple connections could cause an assert on shutdown

Fixed	9755	Interactive Brokers	Interactive Brokers stops plotting on chart using custom trading hours template
Fixed	9667	Interactive Brokers	IB volume for FX displays incorrect amounts
Fixed	9642	Interactive Brokers	Interactive Brokers does not connect when the client has multiple accounts/advisor accounts
Fixed	9556	Interactive Brokers	IB RTH Trading Hours skips Rollover Date
Fixed	9549	Interactive Brokers	IB secondary data series does not resume realtime data after reconnect
Fixed	9555	Interactive Brokers	Historical data manager and IB/TWS does not load last day requested and will remove if exists
Fixed	9593	Interactive Brokers, SuperDOM	SuperDOM Close button does not close IB ZW position
Fixed	9367	IQFeed	Invalid instrument error could beproduced only on weekends
Fixed	9848	Log	Improve Logged text indicating connection to HDS/IS
Fixed	9653	Market Analyzer	Exception when making a Market Analyzer column unvisible/visible.
Fixed	9446	Market Analyzer	NinjaTrader freeze/crash/errors upon switching instruments in MA with linked charts
Fix	95	Market	Cannot download market replay data

ed	98	Replay	
Fixed	9372	NinjaScript	Casting object from DrawObjects fails in compiled assembly but not in source code
Fixed	9784	NinjaScript	RemoveDrawObject(TAG) did not remove NS Drawn Global Objects
Fixed	9651	NinjaScript	Indicator could exhibit memory leak due to overwriting OnConnectionStatusUpdate
Fixed	9577	NinjaScript	Unable to import assembly which uses SimpleFont as NinjaScriptProperty
Fixed	9580	NinjaScript	Indicator removed from configured list did not get finalized
Fixed	9548	NinjaScript	Unable to import NS export containing reference to System.Drawing
Fixed	9789	NinjaScript	Inconsistent Resource String usage in System NinjaScript Types
Fixed	9631	NinjaScript	CandleOutline Brushes unset after a second indicator also sets CandleOutline Brushes
Fixed	9596	NinjaScript	Indicator calling Print() in State.Terminated causes platform to crash
Fixed	9578	NinjaScript	PPO Secondary series as input series returns 0
Feature	9730	NinjaScript	Added IsInstantiatedOnEachOptimizationIteration to State.Configure of Strategy Wizard generated code
Fixed	9008	NinjaScript	Native Pivots indicator plots different values between 7 and 8.
Fixed	9608	NinjaScript Editor	Duplicate file name causes unhandled exception

Fixed	9612	NinjaScript, Playback	Unhandled exception during playback when using AddDataSeries
Fixed	9605	NinjaScript, Strategy Analyzer	Backtest strategy which uses method in Draw namespace causes exception/failure
Fixed	9811	NinjaScript, SuperDOM	Volume Profile colors inconsistent on dom and chart
Fixed	9660	NinjaScript, Workspaces	AddPlot() in State.Configure causes error upon restoring workspace
Fixed	9562	Orders	Trigger pending OCO orders will not cancel on Server Side OCO connections
Fixed	9725	Playback	Market replay freezes when paused and will not resume
Fixed	9620	Playback	Playback Connection causes delay as charts cycle through data
Fixed	9635	Playback	Multiseries replay race condition
Fixed	9670	Playback	Playback plot executions do not show until the trade is completed
Fixed	9672	ShareAdapter	Default file name of images saved via Share dialogue inconsistent with file name preview in Share dialogue
Fixed	9765	Skins	Interval Selector "Configure" text cut off with Slate light skin and 125 DPI
Fix	96	Strategy	Incorrect error message for invalid strategy sell

ed	40		stop order
Fix ed	98 33	Strategy Analyzer	Memory Leak When Aborting an Optimization
Ch an ge	97 71	Strategy Analyzer	Spec Change: Move the Strategy Parameters section right below 'General'
Fix ed	96 84	Strategy Analyzer	SA Optimization Time Remaining inaccurate when over 24 hours
Fix ed	98 09	Strategy Analyzer	SA reporting grid rendering issue
Fix ed	97 81	Strategy Analyzer	When restoring workspace analysis graph does not display until graph changed
Fix ed	97 77	Strategy Analyzer	Strategy analyzer resize lag after backtest
Fix ed	97 78	Strategy Analyzer	SA Display selector showing PC currency instead of account currency
Fix ed	97 59	Strategy Analyzer	SA Genetic lists multiple duplicate results
Fix ed	97 43	Strategy Analyzer	Strategy Analyzer unhandled exception upon sorting column multiple times
Fix ed	97 22	Strategy Analyzer	Strategy Analyzer Tooltip localization issue
Fix ed	97 00	Strategy Analyzer	Strategy Analyzer default values for data series period always 1
Fix ed	97 15	Strategy Analyzer	Strategy Analyzer chart/executions/orders blank on optimization rerun
Fix ed	96 38	Strategy Analyzer	Separate denominations displayed in Strategy Performance.
Fix	96	Strategy	SA Logs restore order incorrect for instrument

ed	55	Analyzer	list backtests
Fixed	9543	Strategy Analyzer	Optimizer silently fails when min is greater than max
Fixed	9544	Strategy Analyzer	NullReferenceExeption in Chart when switching Strategy Analyzer Tabs
Fixed	9366	Strategy Analyzer	Failed Optimization when optimizing on data series
Feature	7634	Strategy Analyzer	Report time frame used for order fill resolution in strategy settings
Fixed	9825	Strategy Analyzer	Enums in Default strategy template not re-applied in SA after compiling in NS Editor
Feature	5171	Strategy Builder	Strategy Builder
Fixed	9826	Strategy Builder	Strategy Builder buttons hover highlight missing top and bottom
Fixed	9832	Strategy Builder	Strategy builder wizard region
Fixed	9852	Strategy Builder	Builder condition edit incorrect comparison combobox
Fixed	9566	SuperDOM	Error rendering real time market data in SuperDOM
Fixed	9718	SuperDOM	SuperDOM Order Flags triangle rendering
Fixed	9710	SuperDOM	Dynamic SuperDOM Can inadvertently suspend if Order Flag moves into mouse cursor
Fixed	9822	SuperDOM	SuperDOM Buy/Sell Text Stays Truncated After Resizing Window

Fixed	9691	SuperDOM	Indicator not synchronized between Superdom and Chart
Fixed	9790	TD AMERITRADE	TDA 2 Cancel Events
Fixed	9779	TD AMERITRADE	TDA - Stuck orders when modifying price
Fixed	9737	TD AMERITRADE	Orders placed from TDA Website do not show up in NT8
Fixed	9733	TD AMERITRADE	TDA Partial Fills Handled Incorrectly With Orders And Executions Displayed Improperly
Fixed	9786	TD AMERITRADE	Modifying TDA order temporarily shows as double the qty amount
Fixed	9764	TD AMERITRADE	Daily data not displaying for Mondays with TD Ameritrade
Fixed	9683	TD AMERITRADE	TD Ameritrade error for all instruments
Fixed	9701	TD AMERITRADE, Time and Sales	Large first print in T&S when connecting to TD Ameritrade
Fixed	9758	UI	Tab on instrument window
Fixed	9652	UI	Pre-populate data in Instrument Lists, Risk, Commissions, and Alerts to be consistent

Fixed	9731	UI	Inconsistent cursor type in Email Support window
Fixed	9702	UI	Account Properties selected template "view" text not aligned to the right
Fixed	7733	Workspaces	Viewable range message box counts when control center restored below other windows
Fixed	9666	Workspaces	Unhandled exception in New Workspace dialogue
Fixed	9554	Workspaces	Global Draw Object template not saved when select Save workspace

4.4.5 8.0.0.10 (Beta)

Release Date

March 21, 2016

This is a general maintenance release which includes various bug fixes due to feedback received during beta. Please note that as this is a maintenance release, there are still outstanding bugs which were reported in previous beta versions and this release does not guarantee all previously reported bugs have been resolved. If you cannot find the status of a bug you previously reported using the table below, please feel free to contact our support team with the tracking ID which was provided at the time the bug was reported (e.g., NTEIGHT-1234)

Code Breaking Changes

There are no code-breaking changes which would impact the compilation of documented and supported NinjaScript code; your custom NinjaScript scripts should upgrade without errors from B9 to B10. If you run into errors, please contact the original author, or reach out for assistance on the support forum.

Due to user feedback, we have changed the default Strategy behavior during optimizations which sometimes caused a strategy to return inconsistent results. To help reduce variance in each optimization run, we have changed the **IsInstantiatedOnEachOptimizationIteration** property to be to **true** by default. Please note, unless you have specifically coded your strategies to take advantage of re-using strategy data, **Strategy Optimizations** could potentially run slower. To take advantage of performance benefits during Strategy Optimization, please see our examples IsInstantiatedOnEachOptimizationIteration Help Guide entry.

Notes

Status	Issue #	Category	Comments
Fixed	8717	Chart	Issues could occur with rendering of the x-axis
Fixed	8896	Chart	With Equidistant Bar Spacing enabled, bars with identical time stamps sometimes used the same time slot
Fixed	9087	Playback	Switching instruments in Playback sometimes caused large spikes in volume due to all ticks posting to a single bar
Fixed	9182	Bars	Switching instruments in window-linked charts could cause the application to crash
Fixed	9295	ATM Strategies	Orders submitted in NinjaScript strategies via AtmStrategyCreate() sometimes left an unprotected position at MAX speed when running Playback
Fixed	9330	Indicator	Indicators with special handling for bars with IsRemoveLastBarSupported set to true returned incorrect values using a Calculation Mode other than Calculate.OnBarClose
Change	9332	NinjaScript	Changed positioning of Draw.Text objects to match NinjaTrader 7 implementation

Fixed	9339	Chart	Chart canvas was rendering incorrectly in Windows 10
Fixed	9368	Strategy Analyzer	Optimization results sometimes differed between iterations of the same test if the strategy hosted additional data series
Fixed	9369	ATI, Instruments	Emails from TradeStation 9.5 related to order submissions used incorrect instruments
Fixed	9370	Database, HDS	Rollover offsets on the NinjaTrader Info Server provided "0" value for 06-15 futures contracts and prior
Fixed	9371	NinjaScript	If an indicator called OnMarketData() with TickReplay enabled, the indicator did not produce historical results unless added directly to a chart.
Fixed	9374	NinjaScript Editor	The NinjaScript Editor ran compilations twice each time
Fixed	9376	NinjaScript Editor	The NinjaScript Editor error list did not jump to errors when clicked
Fixed	9380	Market Replay, Playback	Instruments added to a Market Analyzer during Playback did not appear until Playback was disconnected
Fixed	9381	NinjaScript Editor	A Custom.XML error was thrown when importing NinjaScript if a NinjaScript Editor using inline syntax checking was open
Fixed	9382	Drawing Tool	Newly defined Drawing Tool Hot Keys did not apply to charts which were already open

Fixed	9383	Drawing Tool	The "Remove" option in the Drawing Objects window's Configured section could be obscured when a large number of objects with long tags were listed
Fixed	9384	NinjaScript Editor	Files in the ExportNinjaScript directory were included in compilation
Task	9385	Data Grids	Increased performance in Control Center and Account Data grids
Fixed	9386	Chart	Bar Widths were sometimes inconsistent on multi-series charts
Added	9388	Playback	Added default starting and ending dates for historical tick mode on the Playback connection
Fixed	9392	Chart	Using the Range Bar Type with the Open/Close Chart Style, the "Down Bar Outline" property did not apply changes, and the "Up Bar Outline" property changed both up and down bars
Fixed	9393	Chart, Indicator	Indicator displacement plotted further than expected on higher time frame charts with Equidistant Bar Spacing enabled
Changed	9395	Chart	Updated the "Save Chart Image" feature
Fixed	9396	SuperDOM	SuperDOM price cells overlapped order flags on some DPI settings
Fixed	9397	Order Ticket	High/Low/Open values unexpectedly truncated after resizing the window

Fixed	9399	Level II Window	Level II window grid colors sometimes overlapped or left gaps
Changed	9401	Share Adapter	Updated the feature to save images of window or tab contents on all relevant windows
Fixed	9402	UI	Data Box was always on top of other windows, even when "Always on Top" was disabled
Fixed	9404	Drawing Tool	NinjaScript Drawing methods did not allow objects to be drawn beyond the current bar
Fixed	9407	Adapter	The IQFeed adapter's connection timeout was more brief than expected
Fixed	9408	Bars, Chart	The "Bar Width" property did not always restore with Data Series presets
Fixed	9410	Chart	Chart panel scaling did not maintain its size when dragging plots within the same panel
Fixed	9411	Workspaces	Windows did not maintain their display order when stacked in a workspace
Fixed	9412	Indicator	The Pivots indicator threw an unexpected error message when applied to a multi-series chart
Fixed	9413	Chart	The Mini Data Box sometimes showed "N/A" for indicator plot values if AddDataSeries was used in the indicator.
Fixed	9415	Chart	The Cross Hair locked to incorrect coordinates when locked via the

			right-click menu on a large chart
Fixed	9419	Bars, Chart	The first Daily bar was not displayed when Equidistant Bar Spacing was disabled.
Fixed	9420	Adapter	No pre-market or post-market data was displayed when connected to TD Ameritrade.
Fixed	9421	Chart	The Global Cross Hair consumed CPU resources when not moving.
Changed	9422	NinjaScript Editor	Added an "Export" function to the right-click context menu of the NinjaScript Editor compile errors grid
Fixed	9423	Indicator	The Woodies CCI "Sidewinder" plot did not display on charts with a Daily or higher interval
Fixed	9425	Trading Hours	The EOD check box could not be left unchecked in any session of a new Trading Hours template
Fixed	9426	Strategy Analyzer	Statistics in the Optimizer "Results" grid were not affected by the selected Calculation Mode
Fixed	9427	Chart	Chart Window thumbnails in the Windows task bar were scaled incorrectly.
Fixed	9428	Chart	Stop Limit orders could not be submitted via Chart Trader.
Fixed	9429	Chart Trader	Profit Target and Stop Loss quantity buttons did not function on Chart Trader
Fixed	9432	Bars, Trading Hours	Tick charts were not built correctly in non-EOD sessions

Fixed	9433	Drawing Tool	Extended Line anchor point calculations sometimes changed unexpectedly
Fixed	9434	Market Analyzer	An unexpected error was thrown when using the "Realized Profit Loss" Market Analyzer column while connected to multiple connections with their own accounts
Fixed	9437	Playback, UI	The Playback controller could not be set back to the start time of the selected date range
Fixed	9438	NinjaScript	Draw.Line caused an Object reference exception and froze charts
Fixed	9439	Chart, NinjaScript	The Z-Order was not setting correctly when one indicator was moved into the price panel, if the price panel contained another indicator
Fixed	9440	NinjaScript, Tick Replay	A deadlock occurred when accessing a Series in OnMarketData with Tick Replay enabled
Fixed	9444	NinjaScript Editor	An additional AddOns folder was created when creating a new Add On
Fixed	9459	Strategy Analyzer	Genetic optimizations failed to produce more than four results
Fixed	9460	Data Grids	Realized PnL was not updating for the Sim101 account

Fixed	946 1	Chart	Changing a Chart Style via the Data Series window did not affect the Chart Style icon on the chart toolbar
Fixed	946 4	Attach Order to Indicator	Attaching an order to a multi-plot indicator threw an unhandled object reference exception
Fixed	946 5	Chart	The "Remove" function in the Drawing Objects window did not always remove objects
Fixed	946 6	Strategy Analyzer	The Strategy Analyzer threw an error when trying to re-run a test already saved in the Log panel
Fixed	946 8	Strategy Analyzer	Memory usage could increase when changing values for an optimization
Fixed	947 0	NinjaScript Editor	The default text in the "Save As" dialogue was not consistent with NinjaScript Wizard naming conventions
Fixed	947 3	Strategy Analyzer	When optimizing over a large number of parameters, the application sometimes became unresponsive or threw "out of memory" exceptions
Fixed	947 4	Chart, Indicator	If an indicator threw a start-up error, a blank chart panel would be left when the indicator was removed
Fixed	947 5	Drawing Tool, Templates	The location saved in the Drawing Object window's "Save/Load Template" dialogue unexpectedly changed after loading a template

Fixed	9477	Chart, Templates	When an indicator used another instance of itself as the input series, the input series would be lost on multi-timeframe charts
Fixed	9478	Chart	Using weekly Pivots and switching to a one-week interval resulted in a deadlock
Fixed	9479	Strategy Analyzer	Two clicks were required to select another optimization parameter after changing a parameter
Fixed	9480	Stock Import	Importing stock symbol lists threw an unexpected exception
Fixed	9482	UI	NTMessageBox caused slower than expected performance
Fixed	9484	Control Center, Database	Unhandled exceptions were thrown when adding simulation accounts
Fixed	9485	Market Analyzer, Workspaces	Market Analyzer Column foreground and background colors did not visually restore with workspaces
Fixed	9487	Chart, Drawing Tool	Global drawing objects disappeared from all charts if one chart containing the object was closed
Fixed	9488	Chart	Chart scale and pan location were lost when right clicking in a chart's x-axis, then clicking away
Fixed	9490	Chart, Time and Sales	Chart tabs did not load properly when linked with a Time and Sales window

Changed	9491	NinjaScript, Strategy Analyzer	The IsInstantiatedOnEachOptimizationIteration property is now set to true by default. **Note**: Strategy Optimizations could run slower. Please see Help Guide entry IsInstantiatedOnEachOptimizationIteration to take advantage of performance benefits.
Fixed	9492	Chart, Drawing Tool	Exceptions were thrown when deleting data series with Global Drawing Objects attached
Fixed	9496	Chart	Unhandled exceptions were thrown when moving data series to new panels
Fixed	9497	Level II Window	Unhandled exceptions were thrown when switching instruments
Fixed	9499	Level II Window	An exception was thrown when double clicking the "Details" row (to add or remove tracked Markers Maker)
Fixed	9500	Trade Performance	Some trade statistics column headers in Trades grids used incorrect resource strings
Fixed	9504	Chart, SuperDOM	Indicator dialog info/arrows did not function as expected
Fixed	9505	SuperDOM	The SuperDOM price ladder sometimes rendered text incorrectly after an order was filled
Fixed	9509	Localization, Strategy Analyzer	The Strategy Analyzer sometimes restored incorrect performance metric values when using the German region format

Fixed	9510	Visual Studio Integration	NinjaScript auto-generated code did not update in Visual Studio
Fixed	9511	Playback	The Playback connection left unexpected executions on charts
Fixed	9514	SuperDOM	The SuperDOM 's "Cancel Order" button was displayed after a Simulation account was reset
Fixed	9515	Level II Window	Depth/Spread Columns' text did not populate after resizing the window
Fixed	9516	Chart, Drawing Tool	Performance was impacted when using an Extended Line with Equidistant Bar Spacing disabled
Fixed	9517	Strategy Analyzer, Tick Replay	Tick Replay was unexpectedly available for Renko bars
Fixed	9519	Drawing Tool	NinjaScript Draw objects became attached to the wrong bars when multiple bars shared the same time stamp
Fixed	9529	Chart	Charts zoomed when the border was clicked
Fixed	9530	NinjaScript Wizard	An incorrect OnConnectionStatusUpdate() method signature was added through the NinjaScript Wizard for new Strategies
Fixed	9531	SuperDOM	The "PnL" column calculated values incorrectly with the "Ticks" Calculation Mode when scaling into a position

Fixed	9532	Chart	An exception was thrown when dragging a data series after copying it
Fixed	9533	Drawing Tool	The Trend Channel did not plot parallel lines when moving in some situations
Fixed	9534	Drawing Tool	Draw.HorizontalLine() could not be used with Equidistant Bar Spacing disabled
Fixed	9535	Adapter	^TICK historical data values all plotted above 0 on TD Ameritrade connections
Fixed	9538	Database	After an order rejection, Tick Sizes or Point Values could not be changed, even when disconnected
Fixed	9539	Strategy Analyzer	CFD's could not be backtested or optimized when the "Include Commissions" property was enabled
Fixed	9540	Playback	NinjaScript strategies submitted orders when the Playback slider was updating to a future time
Fixed	9542	Chart	Charts sometimes froze after copying and pasting or dragging and dropping chart objects
Fixed	9546	Database	The UI could lock up when using the database rollover feature
Fixed	9547	Chart, NinjaScript	Plot widths with a value of 1 were the same as those with a value of 2, and were semi-transparent

Fixed	9563	Licensing	Incorrect list of available connection providers appeared with Simulated (@SIM) license keys
Fixed	9550, 9365	Alerts	Cross conditions triggered alerts even if a Line cross did not occur within the look back period

4.4.6 8.0.0.9 (Beta)

Release Date
February 16, 2016

This is a general maintenance release which includes various bug fixes due to feedback received during beta. Please note that as this is a maintenance release, there are still outstanding bugs which were reported in previous beta versions and this release does not guarantee all previously reported bugs have been resolved. If you cannot find the status of a bug you previously reported using the table below, please feel free to contact our support team with the tracking ID which was provided at the time the bug was reported (e.g., NTEIGHT-1234)

Attention Trading Technologies Users: Due to limited use and low user feedback during the beta period, we have removed the **Trading Technologies** adapter from NinjaTrader 8. Users can still connect to their brokerage by using NinjaTrader 7.

Attention FXCM Users: As a result of Issue #9270, related to FXCM CFD's and commissions, you may need to manually delete your database file if you were using an **FXCM CFD** account with a **Commission Template** configured, otherwise errors may be generated. Please contact platformsupport@ninjatrader.com should you receive errors on start-up after updating.

Code Breaking Changes

Tip: We have expanded information in our NinjaScript **Educational Resources** section regrading best practices for new muti-threading considerations in NinjaTrader 8

- **SetTrailStop()**, **SetStopLoss()**, **SetProfitTarget()** - removed redundant double type "***currency***" overloaded. Set **CalculationMode** enum instead e.g., *SetTrailStop(CalculationMode.Price, Low[0])*
- **DrawingTools.ChartAnchor** - removed **ChartAnchor** constructor with int type "***barsAgo"*** parameter as it was problematic. Use **DateTime *time*** values to build your drawing tools chart anchor objects instead.
- **Bars.SessionIterator** object was removed due to complications with some bar types. You can use your own custom SessionIterator built from a **Bars** object instead.
  a. For **Indicators,** please see the system Custom\Indicators *@CurrentDayOHL.cs, @Pivots.cs, PriorDayOHLC.cs*, or the examples in the SessionIterator Help guide section for examples of correct indicator usage.
  b. For custom **Bar Types**, please see the BarsType.SessionIterator Help Guide page, or any of the system Custom\BarTypes installed by default.
- There were several changes to **PerformanceMetrics**. These changes will **NOT** affect any strategy/system performance calculations, but if you develop own custom PerformanceMetric please be aware of the changes listed below. You may also review the **PerformanceMetric** *@SampleCumProfit.cs* for an example of correct usage.
  a. **PerformanceMetric.ValueArrayLength** was removed as it is no longer need
  b. Added new OnCopyTo() method
  c. OnMergePerformanceMetric() has a reversed the target/source logic to be in sync with new **OnCopyTo()** method

Notes

Status	Issue #	Category	Comments
Changed	5734	UI	Tooltips in property grids: Disabled full text tip as entire text is visible
Added	9319	UI	Add undocumented string PersistenceId GUID to NTTabControl/NTTabPage

Fixed	9373	Core, Playback	The simulator option 'Enforce partial fills' no longer produces random partial fills
Changed	8110	Property Grids, UI	Property Grid Background Color and Styling
Fixed	8773	TD AMERITRADE	TD Ameritrade sometimes lost connection immediately after connecting
Fixed	9044	Indicator, NinjaScript	BarTimer was not always updating every second
Fixed	9045	Interactive Brokers	IB Gateway was not always properly restoring connection after disconnect
Fixed	9082	SuperDOM	SuperDOM Volume column did not always include volume from the previous session.
Fixed	9178	TD AMERITRADE	Requesting Index instruments could result in errors
Fixed	9207	Indicator	Indicators should now behave identical independent of IsRemoveLastBarSupported
Fixed	9208	Adapter	Fixed issues with Kinetick EOD ConnectionLost Orange and Red Connection Colors
Fixed	9211	ShareAdapter	StockTwits share adapter did not successfully posting images
Fixed	9234	Chart, Strategy	Strategy would be disabled when second strategy on multiseries chart was enabled
Fixed	9235	Tick Replay	Tick Replay Was Not Providing Bid/Ask Properly Prior to Rollover with Merge Back Adjusted

Fixed	9236	Market Analyzer	Market Analyzer "Grid Foreground" was not saving to workspace
Added	9242	Trading Hours	Added Trading Date holiday check on applying trading hour definition
Fixed	9243	Database	Custom Futures instruments migrated to NT8 switched rollover date and contract month values
Fixed	9244	Indicator	CandlestickPattern text did not load correctly
Fixed	9247	Chart, Workspaces	Chart object Z-Order was not saving with workspaces
Fixed	9250	Chart, DrawingTool	Draw Markers Were Not Rendering in Second Panel
Fixed	9251	NinjaScript	Bars.PercentComplete was not matching NT7 expected output in Market Replay
Fixed	9252	NinjaScript	Indicator wrappers could inadvertently create multiple instances of the 'same' hosted indicator
Fixed	9253	Bars, Chart	Daily Bars were not applying days to load until reload
Fixed	9254	DrawingTool, NinjaScript	Draw.TextFixed "Template" overload was not loading saved template
Fixed	9255	Quantity Selector	Quantity Selector was incorrectly rounding some values
Fixed	9257	Control Center	Email Support... with attached image file was not working
Fixed	9258	Indicator	SampleCustomPlot rectangle was shrinking incorrectly

Fixed	9259	Position Display	Position Display Points Mode was not using FormatPrice()
Fixed	9260	Backup & Restore, NinjaScript	Could export file in physical sub-folder
Fixed	9261	NinjaScript Editor	Could not create folders if a file of the same name exists
Fixed	9262	NinjaScript	Drawing objects were not appearing at correct time when drawn by indicator
Fixed	9265	NinjaScript	SetTrailStop with CalculationMode.Currency was causing ignored orders
Fixed	9267	Chart, Workspaces	Restoring workspace was not applying Show Tabs chart property
Fixed	9268	Playback, UI	Replay Controller did not remember the last start date used
Fixed	9269	Market Analyzer	Total row was not adding total
Fixed	9270	Adapter, Commissions	FXCM CFD orders/executions/ positions not displaying when using commissions
Fixed	9271	Chart, Strategy Analyzer	AddChartIndicator() did not plot indicator in strategy analyzer chart
Fixed	9274	DrawingTool	Incorrect Text location of Fibonacci Retracement
Fixed	9276	DrawingTool, Playback	Draw.TextFixed() disappeared from chart after moving Playback slider

Fixed	9277	NinjaScript	DrawObjects.Count not updating correctly during State.Historical
Fixed	9278	Strategy Analyzer	Walk Forward Optimization Results Exported to Excel Were Poorly Formatted
Fixed	9281	Core	Simulator Cash Value incorrectly subtracted commissions
Fixed	9282	NinjaScript	Gui.CategoryOrder was not applying in Strategy Analyzer
Fixed	9283	NinjaScript	IsFirstBarOfSession not triggering in PNF charts
Fixed	9284	Chart	Y-axis unexpectedly compressed after moving the panel
Fixed	9285	Data Grids	Unstacked orders in Basic Entry grid remained displayed in the grid after being cancelled
Fixed	9286	Strategy Analyzer	Custom performance metrics not initially populating after optimization
Fixed	9287	Chart, Templates	Chart template were not restoring panel sizes on first load after chart created
Fixed	9288	NinjaScript, UI	Draw.Region() was not always rendering
Fixed	9289	Strategy Analyzer	Optimization graph results were offset by 1 when strategy utilizes a property of type string
Fixed	9290	Attach Order To Indicator	Space at beginning of indicator name prevented order's 'Attach to indicator' function from working

Fixed	9291	SuperDOM	SuperDOM was not using Indicator name defined in code
Fixed	9292	Alerts	Alerts within tabs were only triggering if tab is focused
Fixed	9293	Alerts, Property Grids	Alerts Config Window has sometimes had property line through entire grid
Fixed	9294	Alerts	Resolved threading conflict with alerts in different tabs when connecting to data provider
Fixed	9296	BarsType	PnF charts with break at EOD unchecked showed bars with all the same time
Fixed	9298	DrawingTool, Templates	Actual text in text drawing object saving/restoring with template
Fixed	9299	Alerts	Horizontal Line Alerts was not triggering when using Rearm: OnBarClose
Fixed	9300	DrawingTool	Draw.TextFixed() objects were drawn out of order
Fixed	9301	Bars	Bars.SessionIterator no longer accessible in NinjaScript
Fixed	9303	Playback	Playback controller freezing after twice connect/enable strategy/ disconnect
Fixed	9306	Skins	Left hand bar in Strategy Wizard on Slate Grey Skins not colored correctly
Fixed	9307	Backup & Restore	On exit, NT8 prompts to save Backup before prompting to save Workspace

Fixed	9309	Data Grids	German decimal missing in exported excel grid
Fixed	9310	Strategy	Strategy was not added to chart when hits exception on historical data
Fixed	9311	Backup & Restore	File Dialog should now maintain the directory where last navigate
Fixed	9314	Database, Playback	Wrong 'from' and 'to' date was triggered on locally updating rollovers while connected to playback
Fixed	9316	NinjaScript	Opening data box throws exception from backtest when strategy using AddPlot()
Fixed	9320	DrawingTool	Newly generated drawing tool would freeze chart
Fixed	9324	DrawingTool	PriceLevels not were working In Compiled Assemblies
Fixed	9326	DrawingTool	DrawingTool.IsSeperateZOrder was not working as intended
Fixed	9328	NinjaScript	Draw objects from AddChartIndicator() were not drawn on SA chart
Fixed	9329	NinjaScript	Space before namespace in script causes crash when exporting to compiled assembly
Fixed	9331	Interactive Brokers	IB Linked Account were not connecting due to race condition
Fixed	9333	NinjaScript	Exporting compiled assembly that contains a class in custom namespace failed to import

Fixed	9334	NinjaScript	Exporting compiled assembly containing DateTime property failed to import
Fixed	9337	Strategy Analyzer	Strategy Analyzer crashed on closing second tab
Fixed	9343	Indicator	NT8 Pivots indicator did not match NT7's values
Fixed	9344	Chart, DrawingTool	Indicator draw objects disappeared when attempt to move series
Fixed	9345	NinjaScript	Code Wizard for strategy setting incorrect double property ranges
Fixed	9346	Chart Trader	Pending order modification persists when switching windows
Fixed	9347	Chart	Day/Week/Month/Year chart lookback defaults were incorrect
Fixed	9350	NinjaScript	Export failed when excluded drawing tool has compilation errors
Fixed	9352	NinjaScript	Using a Strokes .BrushDX property caused the platform to crash
Fixed	9354	Strategy Analyzer	Strategy Analyzer "Chart" Drawing object showed extra z-order after second run
Fixed	9355	NinjaScript Editor, UI	NinjaScript incorrect 'auto focused' when maximized
Fixed	9356	Playback	Playback start date generated error too soon when trying to type in a new year

Fixed	9357	Data Grids	Sorting Partial Fills in Executions Tab did not sort as expected
Fixed	9358	Strategy Analyzer	Errors were generated when running SampleCumProfit on 64-bit
Fixed	9359	DrawingTool	Strategy Global Drawing Object Were Not Removing After Strategy Removed
Fixed	9360	Playback	Chart "Days to load" was changing after connecting to Playback
Fixed	9361	Backup & Restore, NinjaScript	Unable to import NinjaScript when image file exists
Fixed	9362	Chart, DrawingTool	Drawing objects scaled inconsistently on multi series charts
Fixed	9364	Strategy	Exception on startup when strategy on chart
Fixed	9378	DrawingTool	Draw.RegionHighlight..() not using templates "AreaColor"
Fixed	9400	Adapter	Changed "Run time pop up handling" to default on new connection to prevent TWS stealing focus when Global Config window open.
Fixed	6135	Alerts	Performance - Alerts / Alert Log
Changed	9340	Trading Technologies	Removed TT Adapter

4.4.7 8.0.0.8 (Beta)

Release Date

January 8, 2016

This is a general maintenance release which includes various bug fixes and minor feature enhancements due to feedback received during beta. Please note that as this is a maintenance release, there are still outstanding bugs which were reported in previous beta versions and this release does not guarantee all previously reported bugs have been resolved. If you cannot find the status of a bug you previously reported using the table below, please feel free to contact our support team with the tracking ID which was provided at the time the bug was reported (e.g., NTEIGHT-1234)

Attention Interactive Brokers Users: This release is up-to-date with the most recent stable Interactive Broker's software version 954 for both **TWS** and **Gateway**. These versions included changes to the way the IB interfaces are executed and thus the way NinjaTrader uses the "**Auto login**" feature. As a result, you **MUST** update in order for NinjaTrader to automatically start and connect to your account; otherwise you will need to manually start your **TWS** or **Gateway** interfaces before attempting to connect from NinjaTrader. Please see our Connection Guide here.

Code Breaking Changes

The following NinjaScript changes have taken effect between B7 and B8:

PriceLevel.GetPrice(): Removed redundant chartScale overload

NS Editor Folders:

There were changes to the way folders are used in NS editor due to client feedback per issue # 9164. Users who had been using folders will need to restructure their NinjaScript files manually. For example:

- With NinjaTrader 8 shut down, navigate to bin\Custom\Indicators and remove the namespaces from any applicable files. (e.g., rename SpecialFolder.TheIndicatorName.cs -> TheIndicatorName.cs)
- Open NinjaTrader 8 and the NinjaScript Editor
- Right-click on the Indicators folder and add a new folder with the old namespace (e.g., SpecialFolder)
- Drag the relevant files for that namespace into the new folder to create the proper folder/file structure

Repeat these steps for any other NinjaScript types (Strategies, DrawingTools, etc) which may had been using the old folder structure. Doing so will now allow the NS Editor folders to be replicated on the physical file system.

You can read more about the NinjaScript Folders in the "Managing scripts and folders" section of the NinjaScript Explorer help guide article.

Notes

Status	Issue #	Category	Comments
Improved	8821	UI	Replaced Our Custom Loading dialog With The WinForms Save/Load File Dialogs (SFT-41)
Added	8880	UI	Added "Save Chart Image" Function Directly To Chart's Right Click Menu (SFT-336)
Improved	9011	UI	Allow Custom Color Types In Standard Color Picker (SFT-359)
Changed	9033	Skins, UI	Reduced Margins And Window Border Sizes To Improve Usable Space
Changed	9192	Chart, DrawingTool	Draw Text On Chart Entry Point
Changed	9209	Interactive Brokers	Updated to support for IB 954, adds High Precision Forex support - Important: IB Users must update
Changed	9218	Chart, NinjaScript	Expose ChartControl.PresentationSource To Handle Converting Device Pixels
Fixed	6143	FX Board	Various Performance Improvements

Fixed	6144	FX Pro	Various Performance Improvements
Fixed	8731	Ninja Script	CCI(ISeries<double> Input, int Period)[int barsAgo] Was Using The Primary Data Series When A Secondary Data Series Was Specified In Arguments
Fixed	8772	Chart, Performance	Drawing Objects Window Performance Was Slow When Using Large Number Of Objects
Fixed	8813	FXCM	FXCM CFD Instruments Request Could Be Invalid
Fixed	8868	DrawingTool, Ninja Script	Drawing Tools Shrunk When Moved To Right Side Margin
Fixed	8967	Chart, UI	Windows Task Switcher Did Not Render Whole Chart Window In Preview
Fixed	8997	SuperDOM, Tick Replay	OnMarketData Was Not Updating Always When Using Tick Replay On SuperDOM
Fixed	9008	Indicator	Pivots Indicator Plots Could Be Different Between 7 And 8.
Fixed	9017	Market Analyzer	Market Analyzer Label Did Not Automatically Adjust When Values Changed
Fixed	9020	Strategy Analy	Strategy Analyzer Optimizations Were Not Populating Chart Display On 64-Bit Version

		zer	
Fixed	9044	Indicator, NinjaScript	Bar Timer Was Not Updating Every Second
Fixed	9072	Chart Style	Kagi Style Settings Could Not Be Changed
Fixed	9075	Indicator	RVI Indicator Could Calculate NaN value
Fixed	9092	Market Data Archives	Historical Data Window Did Not Show Updated Historical Data On Subsequent Requests
Fixed	9099	Chart Style	Open And Close Tick Markers On OHLC Bars Were Too Long On When Using A Bar Width 1
Fixed	9100	DrawingTool	Default Templates Did Not Save The 'Attach To:' Setting.
Fixed	9101	Market Data Archives	Fixed Issue Where Historical Data Was Incorrectly Being Removed When User Edits Futures Rollover Date
Fixed	9115	NinjaScript, Tick Replay	Unexpected Loading Times With TickReplay And VendorLicense Checks
Fixed	9120	DrawingTool	Ruler Tool Was Not Updating Time Value When Entire Object Was Moved Around On Non-Time-Based Charts
Fixed	9123	Interactive	Unexpected Errors In Log Tab When Connecting To IB With Options Positions.

		Brokers	
Fixed	9125	Ninja Script	Errors And Crashing When Closing A Workspace While Indicators Are Attempting To Load Data
Fixed	9134	Playback	Strategy Values Were Not Being Reset When Playback Resets Playback101 Account
Fixed	9135	DrawingTool, Ninja Script	Regression Channel Anchors Were Not Attached To The Regression Line.
Fixed	9141	FXCM	NT7 And NT8 FXCM "NinjaTrader Historical Data" Configuration Was Not Matching
Fixed	9142	Ninja Script	With Break EOD Disabled, New Bar Was Plotted Without A New Low Or High
Fixed	9144	TD AMERITRADE	TDA Was Dropping Real-Time Ticks
Fixed	9149	Backup & Restore	.Resx Files Were Throwing Errors When Attempting To Export Them From Nt8
Fixed	9152	Tick Replay	Errors Could Be Generated When Loading Tick Replay On Some Bar Types
Fixed	9153	DrawingTool	Setting .IsGlobalDrawingTool Property To True Did Not Add A Drawing Object To Other Charts Until The Drawing Objects Window Was Opened/Closed

Fixed	9155	Market Data Archives	Historical Data Unhandled Exception When Modifying Contract Month
Fixed	9156	Ninja Script Editor	NinjaScript Generated Code Ignored NinjaScriptProperty Tag On Series Properties
Fixed	9157	DrawingTool	Draw.Region Did Not Redraw Correctly When Data series Adjusted
Fixed	9158	Bars	Range Charts Were Adding Large And Increasing Time Gap With 2+ Series In 2+ Panels, With German Locale Set On PC
Fixed	9159	Ninja Script	Deadlock Enabling NS Script After Chart Symbol Change
Fixed	9163	Chart, Ninja Script	Dynamically Changing PriceMarker.Background Caused Unhandled Exception
Changed	9164	Ninja Script Editor	NinjaScripts Located In Physical Subfolders Were Not Showing Up As Available In Appropriate Uis
Fixed	9165	Strategy Analyzer, Ninja Script	SetState() Could Cause Crash When Used In A Backtest
Fixed	9166	Chart, UI	Maximizing a Chart On Secondary Monitor Chart Inner Chart Did Not Fill The Needed Area Leaving A Gap Between WFP Window And Chart
Fixed	9167	Chart,	Enabling Strategy On A Chart Then Removing On The Grid Left Drawing

		Strategy	Objects On The Chart From Added Chart Indicator
Fixed	9169	UI, Workspaces	Saving And Re-Opening Workspace Caused Height Of First Panel In Second Chart Tab To Change To 0
Changed	9170	Rithmic	There Could Be Situations Where Orders Were Not Properly Managed If 2 PCs Had Been Connected To The Same Account
Fixed	9171	Chart	Bar spacing Was Off With Real-time Multi-Series Chart
Fixed	9172	Bars	GetDayBar() Was Returning Null While "Break At EOD"
Fixed	9173	Chart	Global Cross Hair Sometimes Moved At Unexpected Times
Fixed	9175	DrawingTool	Trend Channel Was Not Working Correctly When Snap Mode Was "Bar And Price"
Fixed	9176	Market Analyzer	Errors When Reordering Columns via Drag Methods
Fixed	9177	Chart, DrawingTool	NS Drawing Objects Lost Auto Scale When Chart Was Moved
Fixed	9179	ATM Strategies	ATM Parameters Stop Loss Value Could Be Zero
Fixed	9181	Chart	Maximize Chart Panel Was Not Saving With Workspace Or Chart Template

Fixed	9183	DrawingTool	Fibonacci Template Was Causing Fib Retracements To Be Removed From Chart When Set To "All Charts"
Fixed	9188	Core, ATI	DLL Interface Market Data Was Not Subscribing
Changed	9189	Skins, UI	Replaced Tab Control Legacy Hard Coded Values With Blueprint Values
Fixed	9193	Chart	Session Break Line was Not Plotting When Last Bar Was First Visible Bar On Chart
Fixed	9194	DrawingTool	Drawing Object Templates Did Not Save 'Lock' Setting
Fixed	9195	Market Data Archives	Unhandled Exception When Removing Historical Data Sub Folder.
Fixed	9196	Bars, Trading Hours	No Bars Loaded When Using A Custom 24/7 Trading Hour Template
Fixed	9197	Chart	Chart Toolbar Overflow Button Was Not Visible After Maximized Then Restored.
Changed	9199	Trading Hours	Incorrect Trading Hours Ose/Tocom Templates
Fixed	9201	Chart	Errors Could be Generated When Changing Data Series On The Fly
Fixed	9203	Properties, UI	Duplicated SuperDOM Was Incorrectly Created Without Tabs When Preset To No Tabs
Fixed	9204	Chart,	Plots Disappeared When Using Calculate On Each Tick/On Price Change If Plotting A

		Ninja Script	Different Color
Fixed	9205	Licen sing	Resolved Various Vendor License Bugs
Fixed	9206	Strate gy	Second Strategy Instance Created Duplicate Executions
Fixed	9210	IQFe ed, Kineti ck	Settlement Data Was Not Properly Supported
Fixed	9212	Chart Trade r	Stop Limit On Chart Trader Went Away After External Click After Selecting 'No' On Confirm Order Window
Fixed	9214	Time and Sales	When Resizing Columns Once The Space For A Column Got So Small It Would Snap Into The Next Column
Fixed	9215	Adapt er, Datab ase	Some IB CFD Prices Did Not Match Tw's With Correct Tick size Granularity
Fixed	9216	Strate gy Analy zer	Strategy Names Were Not Displaying When Region Set To German
Fixed	9220	Ninja Script	Market Analyzer Column With Nested Indicator(s) Crashed On Configuration
Fixed	9221	Share Adapt er	Share Service Names Weren't Reflected In The List Of Configured Services
Fixed	9222	Chart	Color Of Zoom Field Line Did Not Change.
Fixed	9223	Data Grids,	Error Could Be Generated When Right-Clicking FF Board Orders Grid

		FX Board	
Fixed	9226	Chart	On Non-Time Based Charts, X-Axis Values Did Not Always Line Up With Bars/Drawing Objects Timestamps
Fixed	9228	Core	Errors Could Occur When A Trading Hours Holiday Was Defined Where No Session Day Existed
Fixed	9229	Time and Sales , UI	Columns Widths Reset After Enabling/ Disabling Additional Columns
Fixed	9230	Order s	Order Windows Were Incorrectly Allowing Unsupported IOC/OPG Orders With B Connection
Fixed	9231	Ninja Script	Remove Unused Chart scale Overload
Fixed	9233	Adapt er, Core	Bugged Backup Connection Handling When Data Connection Goes 'Connection Lost'
Fixed	9237	Drawi ng, Temp lates	Drawing Object Did Not Displayed Globally Even Though Set Global In Default Template

4.4.8 8.0.0.7 (Beta)

Release Date

December 2, 2015

This is a general maintenance release which includes over 140 bug fixes. Please note that as this is a maintenance release, there are still outstanding bugs which were reported in previous beta versions and this release does not guarantee all previously reported bugs have been resolved. If you cannot find the status of a bug you previously reported using the table below, please feel free to contact our support team with the tracking ID which was provided at the time the bug was reported (e.g., NTEIGHT-1234)

Code Breaking Changes

The following NinjaScript changes have taken effect between B6 and B7:

OnConnectionStatusUpdate: The signature for OnConnectionStatusUpdate() for all Types inheriting from NinjaScriptBase has been updated to `OnConnectionStatusUpdate(ConnectionStatusEventArgs connectionStatusUpdate)`. This was the previous implementation for Indicators only.

Notes

Status	Issue #	Category	Comments
Improved	8519	UI	Enhanced features for new-user orientation in the platform
Fixed	8826	Share Services	Facebook accounts could not always be configured in Share Services
Fixed	8834	Chart	Tick Replay only functioned within the current contract month for merged, back-adjusted futures contracts
Fixed	8837	MBTrading	Accounts were not always loaded when connected to MBTrading
Fixed	8845	NinjaScript	Draw.Rectangle could cause errors in some cases
Fixed	8847	Charts	Unexpected gaps between chart bars sometimes formed on tick charts
Fixed	8858	Chart, Drawing Tool, NinjaScript	Reloading NinjaScript on charts resulted in Drawing Objects' C# Types to differ from expected Types defined in the newly compiled Custom project

Fixed	8867	Chart, Drawing Tool	NinjaScript Draw.Region slowed chart performance when scrolling through bars
Fixed	8870	Chart, Playback	Playback charts sometimes only plotted real-time bars when using Historical Data as the data source
Chan ged	8873	Alert, Chart, Drawing Tool	Removed the option to use an offset in chart alerts based on Drawing Objects
Fixed	8906	TD Ameritrade	Platform sometimes froze during a partial order fill on a TD Ameritrade account
Fixed	8907	TD Ameritrade	Issue # 8906 could result in different PnL than expected
Fixed	8908	Chart	Tick Replay sometimes merged historical data incorrectly on futures contracts
Fixed	8933	Commission	Commissions sometimes applied incorrectly for positions which were scaled into
Fixed	8938	NinjaScript	Calling BarsRequest prevented future calls to OnBarUpdate()
Fixed	8961	UI	Control Center menus sometimes appeared to the side of menu headings
Fixed	8978	Core	Brent Crude futures contract sometimes rolled over to a prior contract expiry

Fixed	8981	Chart, Bars	The Right Side Margin property of the Box chart style was applied to the right and left side of the bars
Fixed	8982	Chart, Drawing Tool	Drawing Objects would not snap to bars with identical timestamps
Fixed	8990	Market Analyzer, NinjaScript	Adding more than one Cell Condition in a custom Market Analyzer Column sometimes resulted in errors
Fixed	8999	Strategy Analyzer	Strategy Analyzer did not recognize commas to denote decimal places on PCs with European region settings
Fixed	9000	SuperDOM, Indicator	When using an indicator as the input series to another indicator in the SuperDOM, the input series sometimes switched back to the instrument configured on the SuperDOM
Fixed	9001	Chart, SuperDOM, Indicator	Related to Issue # 9000, when an indicator used another indicator as an input series, and that second indicator used a third indicator as its input series, errors could be thrown
Fixed	9003	Strategy Analyzer	Strategy Analyzer grids sometimes sorted dates in unexpected ways
Fixed	9006	Drawing Tool	Drawing tool "Attach To" property could not be changed in some circumstances

Fixed	9007	NinjaScript	The following properties returned values different from those shown in the UI: • AccountItem.ExcessInitialMargin • AccountItem.ExcessIntradayMargin • AccountItem.ExcessMaintenanceMargin • AccountItem.ExcessPositionMargin
Changed	9012	NinjaScript	Expanded support for the NTWindow class for AddOns
Fixed	9014	Instrument Lists	Attempting to remove a custom Instrument List with the same name as a pre-loaded Instrument List resulted in an error
Fixed	9015	Chart	Time labels in the x-axis of charts could be only partially displayed at the left edge of the chart canvas
Fixed	9016	NinjaScript	OnBarUpdate() was no longer called after connecting to a second data provider
Fixed	9018	Playback	Errors could result in some cases when adding two data series to a chart in Playback
Improved	9019	Chart	Chart performance was improved for cases in which a large number of charts were open simultaneously

Fixed	9021	SuperDOM, Indicator	SuperDOM indicators reloaded when connecting to a second data provider
Fixed	9022	SuperDOM, Market Analyzer, Indicator, NinjaScript	Indicators could reach State.Realtime while processing historical bars in some cases
Fixed	9023	Market Analyzer, Alerts	When using @MESSAGE for the text in Market Analyzer Alerts set to display a popup dialogue, the text "@MESSAGE" was displayed, rather than the text held by the @MESSAGE variable
Fixed	9025	Chart	Chart sometimes would not pan up when Ctrl+Click +Dragging up in the y-axis
Fixed	9026	Chart	Panning left or right on charts could result in errors in some cases
Fixed	9027	ATM Strategies	ATM Strategies could throw an error when multiple Targets were placed for a single entry
Fixed	9029	Interactive Brokers	Interactive Brokers connections were not always established successfully when using a live license key
Fixed	9031	Playback	Playback could sometimes freeze when set to Max Speed
Fixed	9032	Core	Attempting to launch NinjaTrader without an active internet connection could

			result in errors
Fixed	9034	ATM Strategies	Custom ATM Strategy template order quantities reverted to the Scale Quantity value after placing an order
Fixed	9035	Chart	After clicking and dragging in a chart axis, the Cross Hair could become stuck rendering in the axis
Fixed	9036	ATM Strategies	Realized PnL of ATM Strategies showed unexpected results with the AT Interface option enabled
Fixed	9040	UI, NinjaScript	The "Break at EOD" property of strategies configured on the Strategies Grid reverted to enabled after being manually disabled
Fixed	9042	Chart	The Cross Hair x-axis price label did not update when using the keyboard to pan the chart on the x-axis
Fixed	9043	Chart	Non-active chart tabs did not always automatically begin updating when made active
Fixed	9046	Core	Some German text was displayed in English installations
Fixed	9047	Trade Performance	Trade Performance window could produce duplicate executions in some cases
Fixed	9050	Chart	Cross Hair x-axis marker did not update automatically when new bars formed on a

			chart
Fixed	9051	NinjaScript	Disabling and re-enabling strategies could sometimes produce errors
Fixed	9054	Alerts Log	Alerts could re-appear in the Alerts Log after being cleared in some cases
Fixed	9055	Alerts	Only message text displayed in popup dialogues triggered by Alerts
Fixed	9056	Market Analyzer	Market Analyzer columns sorted rows in unexpected ways in some cases
Fixed	9057	NinjaScript	Working on NinjaScript files in Visual Studio could cause the file being worked on to be removed from the Custom project
Chan ged	9058	Alerts Log	Enhanced the way that the "Clear" function works in the Alerts Log when alerts are present in two tabs
Fixed	9060	Strategy Analyzer	Using the Genetic Optimizer sometimes resulted in errors
Fixed	9061	TD Ameritrade	Connections to TD Ameritrade could be unsuccessful in some cases
Fixed	9063	TD Ameritrade	Working orders from TD Ameritrade connections did not always display
Fixed	9064	Chart	Charts remained on top of other windows with "Always On Top" disabled

Changed	9066	Alerts	The "Show a Popup Dialogue" Alert Action exposed more properties than needed
Fixed	9067	UI	The Strategies window on charts could display the Strategies property grid for the Control Center
Fixed	9068	Chart	Strategy execution plots remained on charts after a strategy was disabled via the Control Center Strategies Grid
Added	9069	NinjaScript	Added ShowTransparentPlotsInDataBox property, accessible from IndicatorBase
Fixed	9071	UI	Strategies could not be re-enabled in the Strategies Grid in some cases
Fixed	9073	Chart, Drawing Tool	Drawing Tools could disappear from charts when panning on the x-axis
Fixed	9074	Workspace, Chart	The Data Box could throw errors in workspaces that included a parenthesis in their names
Fixed	9076	Chart, UI	The "Reload NinjaScript" context menu item was not available for Drawing Tools by themselves
Added	9077	UI	Added a "Training Webinars" item to the Help menu
Fixed	9078	NinjaScript	Chart toolbar buttons were not findable by Automation ID

			in some cases
Fixed	9079	Chart, Interactive Brokers	On Interactive Brokers, charts could remain in "Loading" state after an unsuccessful bars request
Fixed	9081	Chart, Workspace	Chart panels could be resized when reloading workspaces
Fixed	9084	Chart, Drawing Tool, Hot Key	Drawing Tool Hot Keys could draw a different Drawing Object than intended in some cases
Fixed	9085	Interactive Brokers	Interactive Brokers Financial Advisor account connections could be unsuccessful
Fixed	9086	Chart Trader	Chart Trader could be seen at the edge of charts with "Hidden" enabled
Fixed	9088	SuperDOM	SuperDOM columns could prematurely attempt to process a one-click order modification
Fixed	9089	Interactive Brokers	Position updates for CFDs were not always processed when connected to Interactive Brokers
Fixed	9090	NinjaScript	Placing a BarsType C# file in a subfolder of the BarsType folder could result in errors
Fixed	9091	NinjaScript	Attempting to export an indicator which referenced an AddOn could cause errors
Fixed	9093	Core	Downloading Playback data could freeze when requested

			an non-existent instrument
Fixed	9094	Interactive Brokers	FDXM trades were displaying as FDAX trades when connected to Interactive Brokers
Chan ged	9096	NinjaScript Wizard	Removed "Input Parameters" from the NinjaScript Wizard for custom BarsTypes
Chan ged	9097	UI, Skin	Changed the default Skin to "Slate Gray"
Fixed	9098	Drawing Tool, Workspace	Global Drawing Tools in workspaces could cause errors when loading chart templates
Fixed	9100	Drawing Tool	Drawing Tool templates did not save the "Attach To" property
Fixed	9101	Core	Historical Data was deleted when changing rollover date offsets for futures instruments
Fixed	9104	NinjaScript	AreaBrush property of Region Drawing Tools was not updating as expected
Fixed	9105	Chart	Viewing the Mini Data Box could cause errors in some cases
Fixed	9106	Logs	Kinetick Non-Pro fees verification did not add a message to Log files as intended
Fixed	9107	NinjaScript, Strategy Analyzer	Optimizing Data Series in the Strategy Analyzer could cause errors when

			AddDataSeries() was used in a strategy
Fixed	9108	NinjaScript	ChartControl could be null in State.Terminated unexpectedly
Fixed	9109	Workspace	Loading workspaces could cause errors in some cases
Fixed	9110	Hot Key	Pre-configured Hot Keys for some Drawing Tools were identical
Fixed	9112	NinjaScript Editor	Creating a new script inside a custom folder could cause errors
Fixed	9114	Instruments	Some futures instruments were set up with incorrect tick sizes
Fixed	9115	NinjaScript	Tick Replay caused Vendor License checks to load slower than usual
Fixed	9116	NinjaScript, Playback	IsExitOnSessionCloseStrategy could cause errors in some cases when connected to Playback
Changed	9121	Static SuperDOM	Changed TT transaction credit tracking
Fixed	9122	UI	Commission template names could be blank when viewed from Account Properties
Fixed	9124	NinjaScript, Chart	Plots drawn by strategies did not display in the Data Box
Fixed	9129	UI	Instrument Type lists were not sorted alphabetically as

			expected
Fixed	9130	Hot Key	F10 could not be set as a Hot Key
Fixed	9131	UI	No error messages were displayed when setting a Drawing Object to "All Charts" on a data series with "Show Global Drawing Objects" disabled
Fixed	9132	NinjaScript	Bars.GetDayBar() could throw errors on Daily charts
Fixed	9138	NinjaScript	Errors could result from re-enabling a strategy after applying a new Trading Hours template
Fixed	9140	NinjaScript, Chart	Indicators which painted bars could overwrite bars painted by other indicators
Fixed	9143	NinjaScript, Indicator	Setting a custom Stroke property in an indicator could change the same property in all instances of that indicator on a chart
Fixed	9145	FXCM	Orders sometimes could not be closed on FXCM connections
Fixed	9146	NinjaScript	AdoptAccountPosition did not allow more than one strategy to be enabled per account
Changed	9147	Trade Performance	Trade Performance "Template" filter now only filters by ATM Strategy, not NinjaScript strategy

Fixed	9148	UI	Switching through tabs via Hot Key did not always show the active tab
Fixed	9151	Share Service	Using email Share Services could cause errors
Fixed	9154	Drawing Tool, NinjaScript	Draw.Rectangle could generate incorrect starting anchor point times in some cases

4.4.9 8.0.0.6 (Beta)

Release Date

October 26, 2015

This is a general maintenance release which includes over 130 bug fixes. Please note that as this is a maintenance release, there are still outstanding bugs which were reported in previous beta versions and this release does not guarantee all previously reported bugs have been resolved. If you cannot find the status of a bug you previously reported using the table below, please feel free to contact our support team with the tracking ID which was provided at the time the bug was reported (e.g., NTEIGHT-1234)

Code Breaking Changes

There were no code breaking changes between B5 and B6

Notes

Status	Issue #	Category	Comments
Fixed	8584	UI, Chart	Chart canvas was improperly displayed on a Windows 10 virtual desktop
Fixed	8630	Bars	Last 1 minute bar for session could hold entire days range
Fixed	8720	Database, Interactive Brokers	VX futures was no longer resolving to monthly expiry since IB added weekly expiries

Fixed	8734	Indicator	Multiple indicator plots were not available under input series of indicators
Fixed	8741	DrawingTool, NinjaScript	Lock icon was showing up on drawing tools without moving them
Fixed	8762	DrawingTool	Draw.Text genearted error when reloading historical data
Fixed	8782	Bars	Bars out of sync from UI and core processing
Fixed	8790	eSignal	Resolved errors when requesting historical tick data
Fixed	8797	SuperDOM	Market depth subscriptions were sometimes being subscribed to twice resulting in uneven depth display
Fixed	8798	Backup & Restore	Importing backup file caused unhandled exception and platform freeze
Fixed	8802	DrawingTool, NinjaScript	Regression Channel not being drawn on secondary instrument
Fixed	8808	Chart	Moving data series to new panel reverts to "right" when moved
Fixed	8809	Chart	Holiday defininition can throw off some session calculation on chart
Fixed	8810	Rithmic	A race condition could prevent successful connect
Fixed	8815	Chart, Database	Rollover feature could exhibit issue displaying today's data
Fixed	8816	Strategy	Calling a second AddPlot() with same name did not

			render
Fixed	8817	NinjaScript	OHLC chart style right-facing line extends further than left-facing line
Fixed	8818	Strategy	Strategies count doubling when applied in strategies grid on playback connection
Fixed	8819	Control Center, Strategy	Strategies not updating trading hours template when applied in strategies grid
Fixed	8822	TD AMERITRADE	Error message when requesting some stocks
Fixed	8823	News	Error when opening the News Window on FXCM connection
Fixed	8828	Chart	Exception could occur when drag and dropping data series with strategy attached
Fixed	8829	NinjaScript	Visual Studio custom project not updating when file is pasted into an appropriate NinjaScript folder in file system
Fixed	8830	DrawingTool	Drawn objects duplicated when template is applied
Fixed	8831	Alerts, Market Analyzer	Market Analyzer alerts was only pulling condition columns from first tab
Fixed	8832	Chart	Right click time axis switches crosshair in daily charts
Fixed	8833	DrawingTool, NinjaScript	Andrew's Pitchfork anchor line being redrawn when calculation mode changed
Fixed	8834	Tick Replay	Tick Replay not functioning prior to rollover date on

			current futures contract with "merge back adjusted" set
Fixed	8835	MBTrading	Executions were sometimes duplicated
Fixed	8836	Chart, Window Linking	Interval link button did not always work after a restart
Fixed	8838	Chart, Indicator, NinjaScript	Plot values was not updating within the same millisecond in the data box on tick charts
Fixed	8839	NinjaScript	Draw methods did not render when compiled to DLL
Fixed	8840	Trade Performance	Trades display "chart" option was combining executions from multiple instruments in a single chart
Fixed	8841	Interactive Brokers	Adapter not consistently pulling external executions upon connecting
Fixed	8843	Market Analyzer	Text in note column of Market Analyzer reappears after deleting
Fixed	8844	Object Dialog	Print feature throwing unhandled null reference exception on all windows
Fixed	8846	Commissions	Sim101 applied commission template was not being replicated on Playback101 account
Fixed	8848	UI	Unhandled exception could occur after deleting last workspace
Fixed	8849	Chart	Drag and drop dataseries was incorrectly cloning previous charts bar spacing

Fixed	8850	Backup & Restore	Machine id error/attempts to launch second NT8 instance when double-clicking on NT8 backup archive
Fixed	8851	Skins	Font color on all skins was black in the "share" window
Fixed	8853	Chart, Indicator	DirectX error when using "horizontal line" plot style in CurrentDayOLC indicator
Fixed	8857	UI	Updated show help to always be language specific
Fixed	8860	Localization, Regionalization, SuperDOM	Russian localization not allowing stop limit orders on SuperDom and russian words are left over when switching to English
Fixed	8861	Tick Replay	No calls to OnMarketData with multiple indicators when using tick replay
Chan ged	8862	Chart	Removed "Visible" property from data series due to lack of applicable use cases
Fixed	8863	Market Analyzer	Public indicator properties were not showing unless using NinjaScriptProperty tag
Fixed	8864	Installer	A folder named GlobalDrawObject' was being created by the installer
Fixed	8866	Templates	Added error when attempting to load NT7 chart templates or corrupt NT8 templates
Fixed	8869	Chart	"price based on" property of data series was reverting to "last" when switching interval on a chart

Fixed	8875	Strategy	Strategy was unable to change account after restarting the platform
Fixed	8876	Strategy	Strategy columns did not update when strategy is enabled in a particular way
Adde d	8878	Commissions	Allow sub-cent commissions for stocks
Fixed	8881	Market Analyzer	Exception when sorting by indicator column
Fixed	8882	Chart, Playback	Global cross hair issues while connected to playback connection
Fixed	8884	Indicator, UI	Some features indicator "data series" does not show pnf bar type property name
Fixed	8885	Playback	Playback controller still showed "play" button when no data is available
Fixed	8886	DrawingTool	Border around the data area of ruler drawing objects was not selectable
Fixed	8887	Playback	Submitting an order as there is not playback data available, e.g. right after connecting and before starting playback, could crash ninjatrader
Fixed	8888	Chart	Chart panels re-sizing when data series properties are changed
Fixed	8889	Chart	Text in cross hair price marker extending outside of the marker itself when the value includes more than 4 decimal places

Fixed	8890	Interactive Brokers	Tws slow to respond when connected thru nt.
Fixed	8891	Interactive Brokers	Tws does not launch automatically when "auto logon" is disabled.
Fixed	8892	SuperDOM	Confirm order dialogue unfreezing dynamic dom ladder, resulting in orders being placed at incorrect prices
Fixed	8893	Chart	Draw.ray generates direct x error - d2derr wrong state
Fixed	8897	Installer	Installed to custom directory does not completely remove on uninstall
Fixed	8899	NinjaScript Editor	When installed to non standard directory can cause errors with NinjaScript
Fixed	8901	Strategy Analyzer	Genetic optimizer not incrementing per user defined values
Fixed	8902	SuperDOM	SuperDOM controls moving on hover
Fixed	8903	Chart	Y-axis performance slow on fractional tick sizes
Fixed	8904	Playback	Playback gets stuck when enabling multiple strategies.
Fixed	8910	News	Focus lost in news window
Fixed	8911	Chart	Indicator input series switches back to data series when chart interval is changed.
Fixed	8912	Backup & Restore	No file selection, progress or description window displayed

			during restore.
Fixed	8914	NinjaScript	Plotting a line and drawing a line at 1 pixel size produce different results.
Fixed	8915	DrawingTool	StartAnchor.Price and EndAnchor.Price returning 0 for regression channels
Fixed	8916	Chart, Indicator	Indicator information does not display in the data box if the color is set to 'transparent'
Fixed	8918	ATM Strategies	Atm strategy, trail stop will not trigger until after breakeven
Fixed	8919	Chart	Price marker flag bounds issue
Fixed	8920	Chart	Point and figure chart style's bar width property not changing when set in the data series window
Chan ged	8922	Backup & Restore	Migration now prompts users to check log for errors after completed
Fixed	8923	Market Analyzer	Changing the input series of an indicator in the Market Analyzer gave an error
Fixed	8924	Market Analyzer	Ma indicator column spamming tick replay warning message
Fixed	8926	Chart	Exception when disconnecting while data is loaded
Fixed	8927	Chart	Unable to decompress x-axis on multi-series non-equidistant chart

Fixed	8928	Chart	Chart cross hair jumps when using global setting.
Fixed	8929	Chart	Chart did not scroll with global enabled while disconnected.
Fixed	8931	Chart	Using on-the-fly method to change timeframe getting marketdatatype_unknown
Fixed	8932	NinjaScript	Redundancy in condition in @ChartMarker.cs
Fixed	8934	Chart	Vertical bar on crosshair not being rendered when first loaded
Fixed	8935	Connections	Message displays when no changes have been made with the connection
Fixed	8936	UI	Strategy/indicator parameters for double variables do not accept a dot for regions that use commas to denote a decimal
Fixed	8937	Chart	Bars too wide after compressing time axis and switching intervals
Fixed	8940	NinjaScript	Importing with custom dll causes error
Fixed	8941	Chart, DrawingTool	Drawing objects not selectable within the right side margin.
Fixed	8942	Chart	Indicator panel settings are not saved as part of a workspace
Fixed	8943	Hot Key, SuperDOM	SuperDOM hotkey causes quick bid/ask actions to get stuck on display

Fixed	8944	Database	Unable to enable strategies from the menu with replay.
Fixed	8945	Chart	Delete or esc would not take you out of drawing mode.
Fixed	8947	Chart	Local cross hair does sometimes did not move
Changed	8949	Kinetick	Improved connection loss handling
Fixed	8950	SuperDOM	SuperDOM Notes columns entries were not restoring in some cases
Fixed	8953	Chart	Chart panel maximum range not able to set < 0 in ui
Fixed	8954	Chart	X-axis label was sometimes cut off
Fixed	8957	Strategy Analyzer	Manually added instruments cause crash in strategy analyzer
Fixed	8958	DrawingTool, NinjaScript	Fibonacci extensions' fourth anchor point did not appear when plotting
Fixed	8959	DrawingTool, NinjaScript	Alignment was off for the text inside the outline of text drawing tool
Fixed	8960	Optimization Fitness	Ulcer ratio optimization fitness scripts were incorrectly using sortino ratio optimization fitness techniques
Fixed	8964	Strategy Analyzer	When optimizing bools were moved from their group to the parameters group.
Fixed	8965	DrawingTool, NinjaScript	Drawing methods not functioning from hosted indicator when hosting

			strategy also uses drawing methods
Changed	8966	Interactive Brokers	Trader Workstation not starting from NT using 952.1 or above
Fixed	8968	Simulator	Incorrect simulation execution quantity calculated on forex instruments
Fixed	8969	NinjaScript Editor	Platform incorrectly asked to authorize third party addons on every start up
Fixed	8971	DrawingTool, Strategy	Previous instance of a draw method remains when reloading NinjaScript on a chart
Fixed	8972	DrawingTool, NinjaScript	DrawTextFixed not rendering in correct order on reload
Fixed	8973	Chart	Crosshair still enabled after switching to regular cursor with hot keys
Fixed	8976	Instruments	Instrument search not working in some languages
Fixed	8977	Market Analyzer	Chart opens with no template when using the 'send to > chart' feature of the Market Analyzer
Fixed	8979	Chart Trader	Chart Trader order modifies to erroneous price when trading from multiple tabs
Fixed	8980	Drawing	Selecting all drawing objects cannot uncheck properties
Fixed	8983	Interactive Brokers	Different bid/ask sizes were being calculated

Fixed	8984	Market Data Archives	German localization - importable historical data files not appearing in load dialog
Fixed	8985	Market Analyzer	Exception with newly created ma column
Changed	8988	Database	Now all 'update' options are disabled as there already is a request on its way
Fixed	8991	Market Analyzer	Null exception when "add blank row" then entering instrument
Fixed	8992	Chart	Logic error in indicator crashes NT8B5 and latest build
Fixed	8993	UI	Instrument exchange picker misaligned
Fixed	8996	SuperDOM	SuperDOM indicators incorrectly displaying "Misc - InputPlot"
Fixed	9004	DrawingTool, NinjaScript	System.AccessViolationException could be thrown using some draw methods

4.4.10 8.0.0.5 (Beta)

Release Date

September 21, 2015

This is a milestone "Open Beta" release marking the first NinjaTrader 8 version available to the general public!

Code Breaking Changes

There were no code breaking changes between B4 and B5

Notes

Status	Issue #	Category	Comments

Improve d	8140	Data Grids	Performance on grid scrolling
Fixed	8743	SuperDOM	SuperDOM Headers/ Footer rows could be off as you resize
Fixed	8748	Trade Performance	Trade Performance Analysis Grid needed to format red if number is negative for consistency
Fixed	8753	Alerts	No message to switch workspaces when an alert is selected from another workspace.
Fixed	8754	Market Analyzer	Market Analyzer Font was not changing with subsequent "apply"
Fixed	8758	Alerts	Selecting a Market Analyzer alert from a different workspace the window was not brought to the front.
Fixed	8759	Data Grids, Strategy	Strategies Grid AvgPrice was not updating on new position
Fixed	8761	Alerts	Chart strategy alerts did not fire when not in active workspace
Fixed	8763	Alerts	Chart alert continued to trigger after disconnecting
Fixed	8767	NinjaScript	NS methods would not be triggered if only defined in a base class.

Fixed	8771	Chart	Exception was thrown when opening saved chart with fixed + center on price scale
Fixed	8774	Indicator, SuperDOM	Indicators on SuperDOM which are finalized (e.g.due coding bugs) could cause exceptions
Fixed	8776	Strategy Analyzer	WFO View optimization results could not have Executions/Orders/ Trades the first time you choose them from Display drop down
Fixed	8777	NinjaScript Editor	DLL Loading was less resilient then NT7
Fixed	8781	Installer	NT would not recompile Custom.dll after reinstall when UserDataDir are mapped to different drive
Fixed	8783	Interactive Brokers	Incorrect execution update could occur
Improve d	8784	Database, Instruments	Auto Roll Over now updates the last used in instrument list
Change d	8785	NinjaScript, Strategy	OnPositionUpdate will now only trigger for when the strategies position changes
Fixed	8787	ATI	OIFs were not always being read from the incoming folder

Fixed	8788	Chart, FXCM	CFD Instruments could lock up charts due to incorrect session templates
Fixed	8789	NinjaScript	Strategy analyzer could crash when using AddChartIndicator()
Fixed	8791	NinjaScript	CancelOrder() was only running once when using Unmanaged order-entry method SubmitOrder()
Fixed	8792	Chart, Database	Instrument rollover was only updating the first instrument configured on a chart
Fixed	8793	Installer	Skins in My Document Folder were not reloading with installer
Fixed	8795	Strategy Analyzer	WFO did not always use Strategy Parameters
Fixed	8796	NinjaScript Editor	Exception could be thrown when opening NinjaScript Editor when properties are preset to have inline syntax checking off
Fixed	8799	SuperDOM	Changing font on SuperDOM was not applied to the price ladder
Fixed	8800	Alerts	Using Alert() with the Alert window closed did not add alerts to the list
Changed	8801	UI	Updated NT8 icon

Fixed	8803	NinjaScript Editor, Skins	NS Editor Disabled Text was hard to read when highlighted
Fixed	8805	Workspaces	Workspace sometimes did not load after switching
Fixed	8806	Chart	Data Box was not updating indicator values until mouse cursor is moved
Fixed	8807	Market Data Archives	Historical Data requests were not limiting number of requests
Fixed	8810	Rithmic	A race condition could prevent successful connect
Fixed	8811	Installer	B4 Installation ends prematurely on some computer regions
Fixed	8812	Core	EventHandlerMarketData/ Depth could throw index out of bounds error
Fixed	8814	Strategy Analyzer	Some displays did not update while switching
Fixed	8820	MB Trading	Fixed error related to SSL/ TLS secure channel

4.4.11 8.0.0.4 (Beta)

Release Date

September 10, 2015

This release holds nearly 200 bug fixes based off of feedback from the last beta release in addition to a number of performance improvements among several UI features.

Code Breaking Changes

The following NinjaScript changes have taken effect between B3 and B4:

- **Strategy IsExitOnCloseStrategy** - renamed to IsExitOnSessionCloseStrategy
- **Strategy ExitOnCloseSeconds** - renamed to ExitOnSessionCloseSeconds
- **Order signal name "Exit on close"** - string renamed to "*Exit on session close*"
- **BarsType AddDataPoint()** - renamed to AddBar() (for consistency)
- **BarsType IsTickReplaySupported** - renamed to IsRemoveLastBarSupported (note: the logic changed from default TRUE to FALSE)
- **BarsType FirstBarAmended** concept removed. Now NinjaTrader internally keeps track of bars being removed and added and compiles a proper range of .MinIndex and .MaxIndex for the barsUpdateEvents
- **SuperDOMColumn OnRestoreValues()** - removed restoredValues object parameter

Notes

Status	Issue #	Category	Comments
Improved	6139	Basic Entry	Various performance improvements
Improved	6140	Chart	Various performance improvements
Improved	6159	Level II Window	Various performance improvements
Improved	6171	Order Ticket	Various performance improvements
Improved	6176	SuperDOM (Dynamic)	Various performance improvements
Improved	6177	SuperDOM (Static)	Various performance improvements
Improved	6405	SuperDOM	Volume column performance improvements
Fixed	7997	Chart, Indicator	No message was displayed when Pivots indicator was applied without sufficient

			historical data
Improve d	8052	Market Analyzer, Hot List Analyzer	Various performance improvements
Fixed	8128	Alerts	Alert set to "never" rearm would trigger after restart
Fixed	8306	Indicator	No message was displayed when WoodiesPivots indicator was applied without sufficient historical data
Fixed	8363	Bars	NinjaScript objects were sometimes not reloaded when bars removed
Enhanc ed	8410	NinjaScript Editor	Added additional default snippets for virtual methods
Added	8414	Trading Hours	Updated holidays for 2015
Fixed	8419	Chart, Strategy	Chart's strategy performance would sometimes incorrectly pair executions
Fixed	8430	Chart	Deadlocks could occur when switching between large workspaces
Fixed	8443	Strategy Analyzer	Backtest engine: stop limit orders could fill at stop price insetead of limit price

Change d	8471	Strategy	IsExitOnCloseStrategy & IsExitOnCloseSeconds renamed to match behavior
Fixed	8472	Market Analyzer	Columns incorrectly showed input series when IsDataSeriesRequired was false
Fixed	8482	Tick Replay	Some data could be missing when using tick replay during certain times of the day
Added	8500	Chart, NinjaScript	Added "UserControlCollection" to allow WPF content to be added in the chart canvas area without overriding our own
Fixed	8505	Drawing, NinjaScript	Draw Ray could plot in wrong direction when switching tabs on chart
Fixed	8510	Drawing	Drawing tools were calculated in scale when isvisible is false
Fixed	8527	Strategy Analyzer	Drawing objects were not being added from strategy in strategy analyzer
Fixed	8535	Trade Performance	Realized pnl was doubled when compared to trader performance
Fixed	8536	Chart	Zoom region on chart continued to appear after right clicking

Fixed	8538	Strategy Analyzer	Re-running strategy with added indicator removed added indicator panel
Fixed	8540	Strategy	Strategies grid: strategies which were disabled did not subscribe to realtime data when re-enabled
Fixed	8541	Data Grids, Strategy	Strategy grid (un)realized pnl columns did not have correct number of decimal places or account denomination
Fixed	8542	Market Replay	Unable to replay when start and end date were the same
Fixed	8543	Alerts	Chart alerts with time value condition triggered immediately instead of at the specified time
Fixed	8544	Interactive Brokers	Interactive brokers advisor account groups did not show up in ninjatrader 8
Fixed	8546	Alerts	Chart alert with action to display pop up dialog did not display the specified text
Fixed	8547	Chart Trader	Number of ticks away from the last traded price not updating dynamically while pending order is selected

Fixed	8548	NinjaScript	CurrentDayOHL "plot current values" throws exception
Fixed	8549	TD AMERITRADE	Unable to connect to TD Ameritrade with simulation license key
Fixed	8551	Database	Unable to update more than 3 database options at once
Fixed	8552	eSignal	Possible unhandled exception using latest version and nt8 64bit
Fixed	8553	DrawingTool	DrawText and DrawTextFixed font and area colors were not defined
Fixed	8554	Interactive Brokers	Interactive Brokers "trade after hours" connection option was not implemented
Fixed	8555	Chart Trader	Order marker remained visible on the chart after couldcelled
Fixed	8556	Account Data	Account realized pnl was not resetting each day
Fixed	8557	Trading Hours	Roll forward holidays function was not working as documented
Fixed	8558	SuperDOM	SuperDOM column trading hours did not take effect until data is reloaded

Fixed	8559	Chart	Global cross hair did not function when not connected
Fixed	8560	Strategy Analyzer	Strategy analyzer strategies would sometimes show up in strategy tab
Fixed	8562	Strategy	Strategy could display as disabled after adding another strategy
Fixed	8563	Hot Key, Chart	Drawing Snap mode hot keys were not working
Fixed	8564	Drawing	Andrew pitchfork did not apply changes to PriceLevels "dashstyle"
Fixed	8565	Time and Sales	T&S rendered margin issues when show quotes was disabled
Fixed	8567	Drawing, NinjaScript	System indicators using OnRender did not correctly Render all user defined plot properites
Fixed	8568	Strategy	Enable strategy on with multiple workspaces would not enabled properly
Fixed	8569	Indicator, NinjaScript	Added Pivots signature overloads
Fixed	8570	Code Wizard	Indicator wizard add plot used incorrect syntax

Fixed	8571	Hot Key	Hot keys could throw exceptions and also would not be persisted after restarting ninjatrader
Fixed	8572	Drawing	Unable to select drawing objects on daily chart to the right of the most recent bar
Fixed	8573	Tick Replay	Tick replay and databox causing issues reloading any indicator
Fixed	8575	Installer	Having <install> folder or <my documents> folder NT right in the root folder of a drive could cause problems with the 'references' in configxml needed for NinjaScript
Fixed	8576	Drawing	Multiseries chart snap mode was not snapping to each bar
Fixed	8577	Hot Key	Unable to set hot keys for opening commission or risk template windows
Fixed	8578	Rithmic	Native ticker symbols were not properly resolved but could result in unexpected error messages
Fixed	8579	Market Data Archives	Generate minute bars from tick data' did not generate minute bars when importing tick data

Fixed	8580	Account Performance	Commissions per instrument type not allowing sub-cent setting for forex
Fixed	8581	Commissions	Commissions window would discard instrument settings after a certain point
Fixed	8582	Interactive Brokers	CFD instruments were incorrectly mapped to "cmdy" and not "cfd"
Fixed	8583	Hot Key	Workspace windows disappeared when using the hot key to toggle next/ previous active workspace
Fixed	8585	Strategy, Data Grids	Oders tab did not show OCO IDs for NS strategy orders
Fixed	8586	Strategy	Unable to enable strategy after a restart via context menu
Change d	8587	Drawing, NinjaScript	Ruler tool did not display values until modified
Fixed	8588	Chart, Indicator	Difference in beginning plot points for indicators when comparing NT7 to NT8
Fixed	8589	SuperDOM	Superdom column width was not being saved correctly
Change d	8591	Installer	Installer now fails if proper NET version is not installed

Fixed	8592	Chart, Drawing, NinjaScript	Ruler tool immediately drew second point line off visible chart
Fixed	8593	Indicator, NinjaScript	VolumeZones opacity property did not work
Fixed	8594	DrawingTool	Text drawing tool alignment was incorrect
Fixed	8595	Chart	Unhandled exception when adding an indicator with the data box open and tick size set to 0001
Fixed	8596	Commissions	Commissions per instrument using TickSize incorrectly
Fixed	8597	Merge Policy	Merge policy was not working when chart start date is day before rollover
Fixed	8598	Strategy Analyzer	Walkforward optimization was excluding some results
Fixed	8599	Commissions, Risks	Various issues when copying commission and risk templates
Fixed	8601	Skins	Using the dark skin chart template names were not visible
Fixed	8603	Chart Trader	Buttons and context menu orders were not correctly implemented
Fixed	8604	Market Analyzer	Market Analyzer export to excel omitted instrument column entries

Change d	8605	NinjaScript	Vendor licensing failied to recognize license
Fixed	8606	Market Analyzer	Market Analyzer indicator column had additional text in label field
Fixed	8607	UI	Commission / Risk windows not was not closing with the use of the ESC key
Fixed	8608	Chart	Switching chart data series incorrectly changed the chart style
Fixed	8609	NinjaScript	Indicators in an assembly reported an error when called from a strategy during optimization
Fixed	8610	Chart	Multi-series strategy executions were not displaying
Fixed	8611	Strategy	Multi-series strategy was showing executions from another strategy performance chart
Fixed	8613	ATM Strategies	Atm "- target" feature were not disabled when last bracket was active
Fixed	8614	Chart, Drawing	Drawing objects disappeared after loading chart template
Fixed	8616	Market Analyzer	Unhandled exception when reloading a workspace with Dow30 included in the market analyzer

Fixed	8618	Interactive Brokers	Index instruments were not receiving last price
Fixed	8619	Chart	Drawn objects were not showing changes without real-time data
Fixed	8620	Orders	Filled orders still could have state PartFilled which then threw off the simulator, spammed the traces and caused performance issues
Fixed	8621	SuperDOM	Adding indicators across multiple DOM's could cause deadlocks
Fixed	8623	Chart	Incorrect indicator timestamp relative to cursor
Fixed	8624	NinjaScript Editor	NinjaScript editor uncomment selection hot key did not fire
Fixed	8625	Performance	Improved shutdown performance
Fixed	8626	Chart	Mini databox did not use chart font
Fixed	8627	Market Analyzer	Existing columns were too small as new columns are added
Fixed	8628	Licensing	Vendor licensing showed "regular" for "free trial"

Fixed	8629	Basic Entry	Order buttons on the basic entry window now maintain readability in other languages otherwise the user is warned
Fixed	8631	Regionalization	Date/time pickers was not translated correctly
Fixed	8632	ATI	B3 failed to load NTDirectdll
Fixed	8633	NinjaScript Editor	NS editor did not allow file names beginning with lower case letters when using "save as" in right click menu
Fixed	8634	Workspaces	Windows were sometimes showing through workspaces
Improve d	8636	Hot Key	Mulikey gesture hot keys
Fixed	8637	Workspaces	Deadlock could occur when opening/closing workspaces
Fixed	8638	Chart, Hot Key	Reload all historical data was possible to be triggered via hot key when disabled
Fixed	8639	SuperDOM	SuperDOM column was distorted with multiple tabs
Fixed	8640	Chart	Using interval selector, chart style would not load from saved preset

Fixed	8641	Connections	Option to disconnect from an active connection disappeared after renaming the account connection
Fixed	8642	Bars	MergeBackAdjusted bars did not correctly begin at the first expiration in the database
Change d	8643	UI	AutomationID was not being seen on trivialtextblock
Fixed	8645	Strategy Analyzer	String variable irregularities during optimization
Change d	8646	BarsType	Various changes in BarTypes
Fixed	8647	UI	Maximize window in portrait mode of monitor was not rendering correctly
Fixed	8648	Visual Studio Integration	Reimporting scripts cause duplicate compile include tags in NinjaTraderCustomCspr oj
Fixed	8650	Chart	Crosshair icon was not displaying correctly after adding a tab and switching back
Fixed	8651	NinjaScript	IsFirstTickOfBar now is true in case a BarsType has removed the last bar

Fixed	8652	Indicator	Change implementation on those indicators which may not be aware of bars types with IsRemoveLastBarSupported=true
Changed	8653	NinjaScript	NinjaScript: Series<t> now could be synced to any bars series
Fixed	8655	Alerts	Alert log source filtering was not working
Improved	8656	NinjaScript	Cross thread exception hint was not specific enough and covered up user exception
Fixed	8657	Installer	64-bit start menu shortcuts were not being created by installer
Fixed	8659	TD AMERITRADE	Unable to connect to TD Ameritrade 'linked' account
Fixed	8660	Chart	Chart toolbar indicator strategy icon was showing disabled until chart has focus
Fixed	8661	Kinetick	Kinetick stock data sometimes received large bar on inital load
Fixed	8662	Tick Replay	Orders were stuck "working" in strategy with calculate OnEachTick and TickReplay enabled

Fixed	8663	Chart	Deadlock could occur when closing NinjaTrader 8 with certain chart setups
Fixed	8664	Strategy Analyzer	Incorrect "value" at display-->settings in Strategy Analyzer
Fixed	8665	Bars	High memory usage could occur when requesting data while "Break at EOD" is disabled
Fixed	8668	NinjaScript	Market analyzer dividend yield column calculation was incorrect
Fixed	8669	Code Wizard	Indicator wizard "plots and lines" window should allow line values of 0 or negative numbers
Fixed	8671	Installer	NT did not recompile customdll after reinstall
Fixed	8673	Indicator	"Attach to Indicator" with parabolic sar indicator did not move order right away
Fixed	8674	Market Data Archives	Downloading historical tick data for 6J 09-15 from HDS could cause crash
Fixed	8675	Instruments	Exception when adding duplicate instrument multiple times

Fixed	8676	Chart Trader	Partial fills caused incorrect quantity displays with chart trader
Fixed	8678	NinjaScript Editor	Regions incorrectly expanded on editing
Fixed	8679	Drawing, NinjaScript	DrawingTool text moving was not smooth
Fixed	8682	NinjaScript Editor	Wizard generated lines incorrectly added to values index
Fixed	8684	NinjaScript	Various issues on exporting NS files
Fixed	8685	Yahoo	Fundamental data was not updating correctly
Fixed	8687	Orders	Incorrect order quantity in execution update could occur
Fixed	8688	Chart	Chart sometimes was not rendering bars when not connected
Added	8689	NinjaScript Editor	Various fixes related to 3rd party update
Fixed	8690	Installer	Failed to process install file: root element is missing could occur
Fixed	8692	NinjaScript Editor	Tabbed NinjaScript editor had issue retaining cursor location
Fixed	8693	Chart	Indicator input series not using "name" when set on custom bar type

Fixed	8694	Yahoo	Dividend yield was incorrectly formatted
Fixed	8697	ATM Strategies	Profit target and stop loss were not combined when partially filled
Fixed	8698	Chart, NinjaScript	Box style bars did not plot the first bar
Fixed	8699	Chart, Drawing	Drawing objects disappeared when moving the object when attached to all charts
Fixed	8700	Chart, Drawing	Drawing objects did not move as smoothly when attached to all charts
Fixed	8701	Market Analyzer	TradedContracts column used incorrect dispatcher
Fixed	8702	NinjaScript Editor	Insert code snippet was not showing hot key in context menu
Fixed	8703	NinjaScript Editor	NinjaScript wizard was not adding description to brush parameter
Fixed	8704	SuperDOM	Removed SuperDOM OnRestoreValues() restoredvalues parameter
Fixed	8705	Quam	Limit orders could get stuck in "PendingSubmit" when they immediately fill
Fixed	8707	Data Grids	Stacked orders on orders grid did not have increase/decrease menu items

Fixed	8709	Indicator	WoodiesCCI constructor overloads did not populate in intelliprompt, and were not accessible
Fixed	8710	Indicator	WoodiesPivots constructor overloads did not populate in intelliprompt were not accessible
Fixed	8712	Chart	DrawText would not change color when using two DrawText statements and same tag name
Fixed	8713	Drawing, NinjaScript	Ruler drawing incorrectly rending text on 2nd point when using non-equidistant bar spacing chart
Fixed	8714	Strategy Analyzer	Enum parameter with [NinjaScriptproperty] tag in optimizer field was blank and could cause crash
Fixed	8715	Chart, Drawing	Drawing tools shifted away from the cursor when attached to all charts
Fixed	8716	Chart	MTF strategy backtesting: if same series is added execution gets displayed twice in chart but only once in executions grid

Fixed	8718	Kinetick	Deadlock could occur when connecting to Kinetick - end of day
Fixed	8719	NinjaScript	Support of enums in indicator wrapper generators
Fixed	8721	NinjaScript	Pivots object reference exception when call as hosted indicator
Fixed	8722	Bars	Extending the 'lookback' period for historical data (BreakOnEOD=false or TickReplay) could result in data gaps
Fixed	8724	SuperDOM	Modifying an existing SuperDOM column could create a duplicate column
Fixed	8725	SuperDOM	SuperDOM indicators were reordering when properties change
Fixed	8726	Drawing, NinjaScript	Missing outline parameters in DrawText and DrawTextFixed
Fixed	8728	NinjaScript	High order fill resolution on strategy causing OnMarketDepth() and OnMarketData() not to trigger in realtime
Fixed	8732	Licensing	TT credits were still being applied with brokers who already apply TT credits

Fixed	8733	Database	Instrument manager and templates could result in errors
Added	8735	NinjaScript Editor	Added "Show Indentation Lines" property feature to NS Editor
Fixed	8736	Chart	Horizontal line plot style could result in exception and prevent selection points from working correctly
Fixed	8737	Chart	Indicators were not keeping y-axis fixed range after indicator dialog is open
Fixed	8738	SuperDOM	Exception opening indicators dialog from SuperDOM could occur
Fixed	8739	Drawing, NinjaScript	Trend channel parallel line could not be visible
Fixed	8740	Drawing, NinjaScript	RenderTarget was not set in @Text
Fixed	8742	Chart	No warning message when trying to remove drawing object drawn by indicator/strategy
Fixed	8744	CQG	CQG API order submission/amendment methods were called with incorrect timestamps
Fixed	8745	Alerts	Selecting an alert in the alerts log window did not switch chart to appropriate tab

Fixed	8746	Alerts	No error if an alert was selected after an instrument had been changed
Fixed	8747	Trade Performance	MAE/MFE for points has wrong calculations applied
Added	8749	Visual Studio Integration	Added Support for VS2015
Fixed	8751	Alerts	No error when selecting a Market Analyzer alert after the window had been closed
Fixed	8752	Alerts	NinjaScript alert could not be found in some cases
Fixed	8757	Indicator	CandleStickPattern indicator textcolor was incorrectly hard coded to black
Change d	8767	NinjaScript	NS methods would not be triggered if only defined in a base class
Fixed	8770	Alerts	Window properties were not listed in the Alerts Log properties

4.4.12 8.0.0.3 (Beta)

Release Date

July 14, 2015

This release holds over 200 bug fixes, with major effort going into stabilizing the Playback connection which had several bugs reported that needed resolution.

We added 2 new skins "Slate Dark" and "Slate Light" which are non-gradient versions of the already existing light and dark skins. We also added the ability for charts to center on price with a fixed scale and improved text input to allow for multiple line input.

Code Breaking Changes

The following NinjaScript changes have taken effect between B2 and B3.

- **AddPlot()** – Pen signature removed. Use Stroke instead
- **Stroke object** – Pen constructor removed.
- **Brush2String()** – Renamed to BrushToString()
- **Pen2String()** - Renamed to PenToString()
- **String2Brush()** - Renamed to StringToBrush()
- **String2Pen()** - Renamed to StringToPen()
- **DrawingTools ChartAnchor.BarIndex** – Renamed to ChartAnchor.SlotIndex to be more specific

Notes

Status	**Issue #**	**Category**	**Comments**
Fixed	8533	Chart	Chart Zoom Feature missing Hot Keys
Fixed	8521	Chart	Chart Trader Stop Loss could show a value of 0 if it had a custom order attribute.
Fixed	8516	Trade Performance	Pips were not correctly formatted in the analysis grid.
Fixed	8515	Window	Window would not restore to the correct size if saved to the workspace in a maximized state.
Fixed	8514	Trade Performance	MAE, MFE was not correctly calculated for pips display mode
Fixed	8513	Strategy Analyzer	Strategy Analyzer analysis tab was blank after being restored from workspace.

Fixed	8511	Kinetick	Other trades are consistently filtered and pre/post market trades are not filtered.
Fixed	8508	Tick Replay	BarsSinceNewTradingDay is -1 on LastBarOfSession when using TickReplay
Fixed	8502	Drawing Tools	Left clicking to the left of the Regression Channel's mid point anchor did not work
Fixed	8501	Drawing Tools	Drawing tool anchors cannot exceed current bar when using multiple instruments and equidistant bar spacing
Fixed	8499	Chart	ASX Interest Rate Session Template could throw an exception
Fixed	8498	NinjaScript	ChartPanel.MouseDown was not firing events on left mouse click
Fixed	8492	NinjaScript	CalcMinMax could error when adding new plots to an existing indicator
Fixed	8491	IQ Feed Adapter	Could yield unexpected data on non-English PC's
Fixed	8489	Charts	Enabling and disabling a strategy could cause an exception
Fixed	8488	Ninjascript	Unmanaged orders could be processed twice
Fixed	8485	NinjaScript	Pivots Indicator would not work with Tick Replay
Fixed	8480	Strategies	Strategy template saved from an optimization was not working correctly when applied to a live strategy
Fixed	8479	Charts	Large interval charts such a yearly charts going back 1974 would cause an exception

Fixed	8478	Export	Export of compiled assembly was not possible to properly select supporting references needed
Fixed	8477	Market Analyzer	Custom market analyzer column name was not persisting after restart
Fixed	8476	Chart	Second bars would not render correctly
Fixed	8475	Drawing Tools	Anchors could move on drawing tools when adding or removing a secondary data series
Fixed	8474	Drawing Tools	Removing a secondary data series would also unexpectedly delete drawing objects.
Fixed	8470	NinjaScript	Exception could occur if you attempt to pragmatically remove a drawing object that does not exist
Fixed	8469	Window	Ctrl+Tab does not switch between tabs if that tabs are not visible
Added	8469	Skins	Added "Slate Dark" and "Slate Light" skins
Fixed	8463	Import/ Export	Import NinjaScript fails after selecting 'Do not show this message'
Fixed	8455	Charts	Chart Properties where not applied when applying a template
Fixed	8454	Trade Performan ce	Strategy Performance chart Data Series does not match the strategy Data Series
Fixed	8453	NinjaScript	SetStopLoss did not pair correctly in real time
Fixed	8446	Indicators	Vertical connecting line missing on Prior day OHLC indicator

Fixed	8445	Drawing Tools	Stay In Draw Mode would only drawing in one panel
Fixed	8442	Interactive Brokers	Interactive brokers incorrectly had support for tick data enabled. However Interactive Brokers doesn't actually support tick data therefor historical data requests would fail.
Fixed	8441	NinjaScript	BarBrushes[0] coloring all series instead of only the primary series
Fixed	8440	Strategies	Editing strategy instance would incorrectly populate the instrument field
Added	8439	Instrument	Added 0.0000005 ticksize and assigned that ticksize to the 6J instrument.
Fixed	8437	NinjaScript	State.Transition barsAgo would incorrectly reference the first bar on chart
Fixed	8435	Drawing Tools	Unable to select drawing tool anchor after setting anchor to transparent
Fixed	8432	Chart	Exception changing equidistant settings with existing multi series chart
Fixed	8431	NinjaScript	NTMenuItem is null when a menu item was collapsed
Fixed	8429	Strategy Analyzer	Monte Carlo Simulation Graph formatted incorrectly for some graph types
Fixed	8425	Charts	Exception on disconnecting a connection with a chart open
Fixed	8424	NinjaScript Editor	NS Editor "save as" function incorrectly renamed namespaces in using declarations
Changed	8422	T & S	T&S default time format now defaults to the CurrentCulture's default long time

			format
Fixed	8421	Historical Data	Tick data timestamp was formatted incorrectly
Fixed	8416	Charts	Chart would be restored with no visible chart when using another language
Fixed	8415	Charts	Font size would not be consistent with multiple chart tabs
Fixed	8409	Strategy Analyzer	Optimizer would switch to Default from Genetic after compiling
Fixed	8405	NinjaScript	Improved resiliency of the Indicator/ Strategy dialog to show other indicators if one indicator has an exception
Fixed	8403	NinjaScript Editor	NS Indicator Wizard incorrectly generated public double Values series for lines
Fixed	8401	NinjaScript	NinjaScript folders were not Alphabetized and floated to the top
Fixed	8400	Backup	Resolved an Out of memory exception that could occur during backup
Fixed	8399	Charts	Minimize a Maximized chart results bars missing when they are out of scale
Fixed	8397	SuperDOM	Dynamic SuperDOM would have a static price with auto center disabled
Fixed	8396	SuperDOM	"Show realized PnL when flat" color did not immediately change
Fixed	8395	Chart	Chart presets did not save 'Days to Load' settings
Fixed	8392	Chart	Exception could occur when removing and adding an indicator on a multi series chart

Added	8390	Chart	Chart Trader support for brokers that require exact order action (Buy To Cover vs Buy) (Sell Short vs Sell)
Fixed	8389	Chart	'PnL display unit' property would switch to Currency when 'Show realized PnL when flat' is enabled
Fixed	8388	Drawing Tools	Horizontal Line would not participate in auto scale as expected
Fixed	8385	Indicators/ Strategies	AddDataSeries with an invalid custom bar type was not handled properly
Fixed	8384	FXCM	Positions opened in FXCM platform would not display within NinjaTrader
Fixed	8380	Market Analyzer	Properties did not save properly with template
Fixed	8378	Chart	Plot which uses transparent pen as default did not autoscale
Fixed	8377	Instrument	Instrument Type filter did not work as expected
Fixed	8373	Strategy Analyzer	Moving a strategy into new folder in the NS Editor removes strategy analyzer settings panel
Fixed	8372	Charts	Chart would stops updating realtime on second connection
Fixed	8371	Market Analyzer	NetChange column would not work properly when multiple accounts were available
Fixed	8370	Drawing Tools	Andrews Pitchfork negative percentage did not stay consistent
Fixed	8369	NinjaScript	MarketAnalyzerColumn label property was inconsistent with indicators in behavior

Added	8368	Charts	Center On price now works with Fixed Scale
Fixed	8367	Charts	Tabbed chart could display the wrong instrument in instrument selector
Fixed	8366	IQ Feed	No longer will try to request L2 data from an account that does not have L2 data authorized
Improved	8365	Chart	Draw Text will now accept multiple line text input by pressing enter in the text field.
Fixed	8364	Chart	Reload Historical Data with strategies enabled would move indicators to a wrong panel
Fixed	8362	Chart	Chart scrolling now occurs 3 bars at a time with scroll wheel and 9 bars at a time holding down the CTRL key.
Fixed	8361	SuperDOM	SuperDOM market depth did not clear correctly when auto center is turned off and price moves off of ladder
Improved	8360	Skins	Improved log colors on all skins for better readability.
Fixed	8356	NinjaScript	Improved error handling to help point NinjaScript Developers when brushes are not frozen resulting in callee cannot access exception due to multi threaded nature of NinjaTrader. We also improved documentation.
Fixed	8353	NinjaScript Editor	Moving Files To New Folder In NS Editor Would Not Be Reflected in Visual Studio
Added	8352	General	Remote Support tool is now pre-loaded
Fixed	8351	Drawing Tools	Hot Keys would not show on Drawing Tools menus as expected

Fixed	8348	Chart	Removing bars series could cause exception in some scenarios
Fixed	8346	General	Loading dialog incorrectly navigates to install directory when wanting to go to the root directory
Fixed	8345	General	Removed apply button on secondary dialog boxes since the changes were not permanently applied.
Fixed	8344	General	Install Directory path now can hold both single or double quotes.
Fixed	8342	NinjaScript Editor	Improved messaging with the Final All feature with NinjaScript Editor
Fixed	8341	General	NinjaTrader crash on startup if PC was using timezone not known to NinjaTrader
Fixed	8339	Strategy Analyzer	Exception running a custom bar type in the Strategy Analzer
Fixed	8329	Migration	Imported Instrument Lists from NT 7 are not sorted alphabetically
Fixed	8328	General	Exception in Commissions window selecting the up/down arrow rapidly
Improved	8327	NinjaScript Editor	Improved Multi-Select behaivior in the NinjaScript Explorer
Fixed	8325	Strategies	Strategies would be disabled when an unrelated provider would be disconnected
Fixed	8323	Strategy Analyzer	Strategy parameters ignored Order property of DisplayAttribute
Fixed	8322	Chart	Chart Trader Simulated Stop Limit Order incorrectly shows limit price

Fixed	8321	Chart	Exception is thrown when trying to save preset for custom BarType
Fixed	8319	Kinetick	Kinetick could be stuck loading when no symbol map is defined
Fixed	8318	NinjaScript	Indicator wrappers were not generated if indicator name held an underscore
Fixed	8317	Migration	TT Order and Price server passwords are not migrated from NT7.
Fixed	8315	NinjaScript	Indicator wrappers did not support nullable types.
Fixed	8313	Strategy Analyzer	Strategy Analyzer did not use the Sim101 denomination
Fixed	8312	Drawing Tools	Draw.TextFixed did not draw historically as expected
Added	8310	General	Installer no longer allows uninstall while NinjaTrader is running
Added	8308	NinjaScript Editor	Added symbol drop downs on the top of the NinjaScript Editor
Fixed	8300	CQG Adapter	Scenario where order shows Rejected however actually was filled
Fixed	8288	Strategy Analyzer	Backtest strategy was not terminated at the end of a backtest or optimization run. It is now terminated.
Fixed	8278	Charts	Chart Panels where not added in the correct size ratio
Fixed	8276	General	Non-tick bars that used tick data where not cached properly
Fixed	8272	Tick Replay	Tick Replay values not properly rounded to ticksize

Fixed	8257	NinjaScript	RemoveDrawObject was not working in historical and removing even manually drawn objects.
Fixed	6179	T & S	Performance enhancements on the T & S window have been completed

4.4.13 8.0.0.2 (Beta)

Release Date
May 28, 2015

This release has over 100+ bug fixes and enhancements thanks to our closed beta participants (Thank You!). With this release we have also expanded our closed beta group further to include additional users and ecosystem partners.

Code Breaking Changes
The following NinjaScript changes have taken effect between B1 and B2.

- **ClearOutputWindow()** - no longer static
- **Draw.Region()** - Draw.Region method has been improved to accept a new `int` displacement overload
- **System.Drawing.Drawing2D.DashStyle** - changed to **NinjaTrader.Gui.DashStyleHelper,** any Draw method that used DashStyle will need to be updated
- **StringAlignment** - changed to **TextAlignment**, any Draw.Text and Draw.TextFixed will need to be updated
- **MasterInstrument.RoundDown2TickSize()** - renamed to **MasterInstrument.RoundDownToTickSize()**
- **MasterInstrument.Round2TickSize()** - renamed to **MasterInstrument.RoundToTickSize()**
- **TradesPerformance.MaxTime2Recover** - renamed to **TradesPerformance.MaxTimeToRecover**
- **TradesPerformance.R2** - renamed to **TradesPerformance.RSquared**

Notes

Status	Issue #	Category	Comments

Improved	8133	CQG	Updated CQG API (6.02.100)
Improved	8198	NinjaScript	AddPlot()/AddLine() now can be called in .SetDefaults and .Configure
Improved	8209	NinjaScript	Wrapper generator now tolerates the 'new' keyword as part of property definitions.
Improved	8210	NinjaScript Editor	NS editor, VS integration: wrappers only are re-generated (and forcing a file reload in VS) as the wrapper content actually would change
Improved	8212	NinjaScript	Back-testing: Added 1 Tick Historical Fill Granularity for all bar types
Added	8239	Chart	Charts: CTRL + Left Arrow/Right Arrow now moves between executions
Added	8178	Chart	Persist last opened chart type for faster creation of next chart
Added	8183	NinjaScript	Added displacement parameter for Draw Region
Fixed	7655	NinjaScript	Back-testing: Stop limit orders could be filled outside of fill-able range
Fixed	7992	Indicators	Woodies Pivots: Was not rendering plots when connected to data provider
Fixed	7996	Indicators	No value for the "Ease of movement" indicator when applied to a Stock Seconds or Range chart
Fixed	8039	SuperDOM	SuperDOM gets stuck "Loading" on playback when no data exists
Fixed	8065	Playback	Playback 'historical' mode exceptions on FF/RR

Fixed	8082	Instrument Link	Duplicate error messages with Static/ Dynamic DOM
Fixed	8083	Attach to Indicator	Attach to indicator re-enable did not not keep correct settings
Fixed	8087	Playback	Playback does not continue after reloading historical data.
Fixed	8095	Chart	Horizontal scrolling not working; Ctrl + X-axis
Fixed	8105	General	Task bar preview image crash when NT window was behind other windows
Fixed	8108	SuperDOM	Single Click Order Modification after Add/Remove Target
Fixed	8113	Chart	Index out of range error when switching instruments with linked chart, market analyzer and SuperDOM
Fixed	8118	Attach To Indicator	Indicator is still allowed to suspend when and order is attached.
Fixed	8126	Chart	Chart deadlock when using SampleAtmStrategy
Fixed	8127	Indicator	Indicator not plotted correctly on Break EOD Range charts after session break
Fixed	8129	SuperDOM	Market depth display issue at very top/ bottom of the ladder
Fixed	8131	Account Data	Sort incorrect on Account Window > Positions Tab
Fixed	8134	Market Analyzer	Bars exception on closing workspace with NASDAQ 100 and indicator columns in a Market Analyzer
Fixed	8135	Bar Type	Point and Figure last price label not updating in real-time

Fixed	8143	Drawing Tool	Global Drawing Tool disappears after set back to normal
Fixed	8144	Yahoo	Error on requesting bar series
Fixed	8145	Chart	Chart area is transparent when re-sizing chart over separate monitor
Fixed	8146	Chart	Changing indicator panel from strategy requires refresh
Fixed	8147	Chart	Scroll bar area transparent after switching chart tab
Fixed	8148	SuperDOM	Order cancel button not visible after changing accounts
Fixed	8152	SuperDOM	Template with Column saved caused an unhandled exception
Fixed	8154	Bar Type	Point and Figure Chart still "Breaks at EOD" when disabled
Fixed	8155	Import	Importing Tick Data Would Generate Errors In Log
Fixed	8156	Chart	Plot Marker color ignores set color
Fixed	8157	NinjaScript Editor	Multi-selecting items in NS Editor is inconsistent
Fixed	8158	Chart	Input Series removed when applying a strategy template on a chart.
Fixed	8160	FX Board	Unhandled exception ATM strategy in FX Board
Fixed	8161	Share	Chart's "Share" option is disabled
Fixed	8162	Chart	Chart Panel scaling is off with multiple tabs
Fixed	8163	IQFeed	IQFeed crashes on connecting

Fixed	8164	Import	RAM consumption on tick data import was too high
Fixed	8165	Drawing Tool	Fib Extensions retracement values do not update
Fixed	8166	FX Board	Order Grid Memory Leak After Closing FX Board
Fixed	8167	NinjaScript Editor	Visual studio integration did not update excluded files
Fixed	8168	Strategy Analyzer	Formatting is incorrect for on "performance" for 'Profit Factor'
Fixed	8169	Strategy Analyzer	Remove unexpected context menu items from Optimization Grid
Fixed	8170	Strategy Analyzer	Optimization chart does not show executions
Fixed	8171	NinjaScript	Folders in object dialogs sorted by internal contents and not folder name
Fixed	8172	Instrument Link	Chart does not switch via Instrument Link if you don't have market data
Fixed	8175	Tick Replay	Changing instrument while loading tick replay data caused exception
Fixed	8176	Chart	NullReferenceException when adding a Indicator to a chart while "Loading"
Fixed	8177	Chart	Drawing Tool Cursor not displayed when set to Crosshair
Fixed	8179	NinjaScript Editor	NS intelliprompt did not work in various scenarios
Fixed	8184	Playback	Chart does not reset when toggling between Historical and Market Replay

Fixed	8185	Strategy Analyzer	Chart "snaps" out of Strategy Analyzer window when using Windows 7 native snapping
Fixed	8186	Share	Email Share Service missing test setup button
Improved	8187	NinjaScript Editor	Drawing snippets missing from NS Editor
Fixed	8189	SuperDOM	SuperDOM Columns caused unhandled exception
Fixed	8190	NinjaScript	Draw Methods use incorrect DashStyle enum
Improved	8191	General	Reduce number of error popups on bar requests
Fixed	8192	Playback	Unhandled exception when moving replay slider using the Go To feature
Fixed	8194	Strategy Analyzer	Optimization Graph exception in some scenarios
Fixed	8196	Playback	Crashes when Reload All Historical Data is selected when the chart is currently Loading when using Playback.
Fixed	8199	Strategy Analyzer	Basket test erroneously showed optimization graph for combined results
Fixed	8200	Chart	Interval Selector does not display Custom Bar Type name
Fixed	8201	General	DashStyle property grid not working as expected
Fixed	8202	Strategy Analyzer	Optimization on Instrument without data getting results in a negative number results being returned

Fixed	8203	General	Deleting an indicator results in exception
Fixed	8205	Chart	Unhandled exception when enabling a strategy within the Strategies tab then attempting to move the chart.
Fixed	8207	NinjaScript Wizard	NS Wizard did not tab through field correctly
Improved	8213	NinjaScript Wizard	Add On's now include additional using statements for ease of development
Fixed	8214	NinjaScript Editor	Moving NS into a folder repeatedly quickly causes exception/error
Fixed	8217	Trade Performan ce	Its possible to cut off the start / end date fields when making the window smaller
Fixed	8222	NinjaScript Editor	After deleting a indicator, if you compile a error is generated opening indicators menu
Fixed	8224	Drawing Tools	SharpDX errors using various Drawing Tools
Fixed	8228	Playback	Market Replay Plackback only allowing for today
Fixed	8231	Chart	Disabling Visible for an Indicator Panel leaves an empty space
Fixed	8233	Account Data	Strategy is not removed from the Strategies tab when the chart that started the strategy is closed.
Fixed	8234	Strategy Analyzer	Switching Period With High Resolution Selected Can Result in Blank Resolution Type
Fixed	8235	NinjaScript Editor	Cut only takes a single line in NS editor when triggered from context menu

Fixed	8236	Strategy Analyzer	Inconsistent currency formatting
Fixed	8237	Strategy Analyzer	Inconsistent display of optimization values
Fixed	8238	NinjaScript Editor	NinjaScript "New Folder" behavior was not standard
Fixed	8244	NinjaScript Editor	Compile does not add user defined references to csproj file used in Visual Studio integration
Fixed	8246	NinjaScript Output	NinjaScript Output Synchronized Scrolling gets out of sync when scrolling with mouse wheel
Improved	8247	Instrument s	Improved warning dialog on changing instrument settings with live orders
Fixed	8249	Strategy Analyzer	Object reference not set error when starting a walk forward optimization
Fixed	8250	Strategy Analyzer	Strategy WFO causes crash when closed during test.
Fixed	8251	NinjaScript Editor	Unable to open .cs file in NinjaScript Editor after right-clicking the file name and opening it in a new NinjaScript Editor
Fixed	8252	Strategy Analyzer	Assertion Failed error upon backtesting an instrument list
Fixed	8254	NinjaScript	Unable to clear OutputTab2 from NS
Fixed	8258	SuperDOM	SuperDOM Column null reference when using Instrument link
Fixed	8260	Skins	Custom Skin Selection Combobox is not displaying custom skins

Fixed	8261	Playback	Multi series chart gets stuck Loading with Playback
Fixed	8262	FX Board	FX Board price selectors are not working as expected
Fixed	8264	News	News window not reflecting local PC time/date settings
Fixed	8267	Strategy Analyzer	Incorrect strategy indicator plots in Strategy Analyzer >Chart display
Fixed	8268	Chart	Drag and drop not working as expected between panels
Fixed	8269	Strategy Analyzer	Strategy Account.Name returns "Sim101" instead of "Backtest"
Fixed	8274	Strategy Analyzer	Crash when using invalid range value in Optimizer
Fixed	8278	Chart	Chart Panels not added by correct size ratio in all scenarios
Fixed	8279	Chart	Remove Strategy on multi-series chart not restoring chart executions
Fixed	8280	Chart	Changing chart Data Series caused strategy settings to not be updated
Fixed	8281	Chart	Indicators on a multi-series chart did not default to correct series in the input series dialog.
Fixed	8282	Chart	Chart with multiple tabs did not respect changing the font size correctly
Fixed	8283	Chart Trader	Chart trader order did not follow mouse position correctly
Fixed	8286	NinjaScript Editor	Intelliprompt box width jumps when scrolling through content

Fixed	8287	Strategy Analyzer	Exception occurred when using aggregate feature
Fixed	8290	Chart	Chart property presets does not save vertical grid line visibility properly
Fixed	8294	Chart	Strategies saved as part of the chart template
Fixed	8295	Chart	Interval selector does not show the correctly selected interval in some scenarios
Fixed	8298	Strategy Analyzer	Property defaults were not being restored as expected in Strategy Analyzer
Fixed	8299	Strategy Analyzer	Strategy that failed to initialize resulted in blank property grid.
Fixed	8301	NinjaScript	Remove "2" in public NS method names
Fixed	8303	NinjaScript	No wrapper code generated for indicators with no parameters.
Fixed	8304	Indicator	RSI Indicator did not auto scale properly
Fixed	8305	NinjaScript	Using "New" in public property broke NS Wrappers

5 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.

6 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.

7 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 Software and/or completes the registration process to open and maintain an account with the Company's NinjaTrader Software ("Software/Service").

BY CLICKING THE ACCEPTANCE BUTTON OR ACCESSING, USING OR INSTALLING ANY PART OF THE SOFTWARE/SERVICE, 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 SOFTWARE/ SERVICE. THIS AGREEMENT IS APPLICABLE FOR ALL RELEASED VERSIONS OF THE SOFTWARE/SERVICE 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 SOFTWARE/SERVICE.

1. Software/Service Terms

a. ***Description.*** The Software/Service is proprietary to Company and is protected by intellectual property laws and international intellectual property treaties. User's access to the Software/Service is licensed and not sold. Software/Service is an order entry and trading application for transactions involving, but not limited to, stocks, futures, exchange traded funds, mutual funds, single stock futures, options, and currency orders (collectively "Orders") that interfaces through an Application Protocol Interface ("API") or Software Development Kit ("SDK"). These systems may be based on software platforms developed by various other third party brokers and/or software developers (collectively "Broker Platforms"). Orders may be executed by brokers via Broker Platforms.

b. ***Use of Third Party Vendors/Brokers.*** User is aware that the Software/Service implements various third party software, platforms, and equipment (collectively "Vendors") and Broker Platforms. Company warrants that use of third party software and/or services is fully licensed for use by Vendors/Brokers to Company and in-turn to licensed Users of Software/Service. User shall abide by all Vendors'/Brokers' individual terms of service agreements, if applicable. **COMPANY MAKES ABSOLUTELY NO WARRANTIES WITH REFERENCE TO THIRD PARTY VENDOR/BROKER SOFTWARE AND/OR SERVICES**.

c. ***Use of Third Party Plugins.*** User is aware that third parties may develop utilities, indicators or strategies ("Plugins") that interact and/or work within the Software/Service.

Installation and use of Plugins 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 Plugins individually or with respect to the Software/Service. **COMPANY MAKES ABSOLUTELY NO WARRANTIES WITH REFERENCE TO PLUGINS**

d. *Accessibility and Function*. User agrees that from time to time, the Software/ Service 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 Broker Platforms, 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 Software/Service, all hardware, software, electrical and other physical requirements for User’s use of the Software/Service including, without limitation, telecommunications and Internet connection(s), ISP, web browsers and/or other equipment, programs and services required to access and use the Software/ Service.

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

g*. Remote Access Services*: NinjaTrader 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 NinjaTrader personnel to remotely access and control User’s computer. By requesting and permitting remote access, User acknowledges that User may be providing NinjaTrader 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. NinjaTrader 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 NinjaTrader’s then-existing privacy policy.

NINJATRADER 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 NINJATRADER 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 Software/Service or on any computer or related equipment that is used to access the Software/Service. 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 Software/Service 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 Software/Service. 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 Software/Service 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 Software/ Service, 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 Software/Service in Company's sole discretion, and that Company may choose to discontinue support of Software/Service 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 Software/ Service upgrades as provided in the sole discretion of Company. User's entitlement to upgrades shall be limited to the specific edition of the Software/Service for which the User is licensed. For instance, if User subscribes to Edition A of the Software/Service, User shall be entitled only to Edition A upgrades and so forth. Software/Service editions relate to the service level of Software/Service 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 Software/Service 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 Software/Service, 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 Software/Service which unacceptable use includes, without limitation, use of the Software/Service 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 Software/Service or any other computer network, (vii) disseminate, store or transmit viruses or any other malicious code or program, (viii) develop an interface between Software/Service to Broker Platforms 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 Software/Service 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 Software/Service, (b) suspend User's access to or use of all or any portion of the Software/Service, and (c) terminate this Agreement.

7. Disclaimer of Warranties

THE SOFTWARE/SERVICE IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USE OF THE SOFTWARE/SERVICE IS AT USER'S SOLE RISK. COMPANY DOES NOT WARRANT THAT THE SOFTWARE/ SERVICE WILL BE UNINTERRUPTED OR ERROR FREE, NOR DOES COMPANY MAKE ANY WARRANTY AS TO ANY RESULTS THAT MAY BE OBTAINED BY USE OF THE SOFTWARE/SERVICE. 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 SOFTWARE/ SERVICE. 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 SOFTWARE/SERVICE OR THE INTERNET IN GENERAL, INCLUDING, WITHOUT LIMITATION, USER'S USE OR INABILITY TO USE THE SOFTWARE/SERVICE, ANY CHANGES TO OR INACCESSIBILITY OF THE SOFTWARE/SERVICE, 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 SOFTWARE/SERVICE, ANY DATA LOSS, OR ANY DATA OR MATERIAL FROM A THIRD PARTY ACCESSED ON OR THROUGH THE SOFTWARE/SERVICE, 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 SOFTWARE/SERVICE, USER'S SOLE AND EXCLUSIVE REMEDY SHALL BE FOR USER TO DISCONTINUE USE OF THE SOFTWARE/ SERVICE AND TERMINATE THIS AGREEMENT IN ACCORDANCE WITH SECTION 6. COMPANY IS NOT LIABLE FOR ANY ITEMS VIEWED OR TRANSMITTED VIA THE SOFTWARE/SERVICE. COMPANY IS NOT LIABLE, DIRECTLY OR INDIRECTLY, FOR ANY ACTS TAKING PLACE WHICH ARE NOT VIEWED OR TRANSMITTED VIA THE SOFTWARE/SERVICE. 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 SOFTWARE/ SERVICE FOR FUTURE REFERENCE. COMPANY IS NOT LIABLE, DIRECTLY OR INDIRECTLY, FOR ANY ACTION OR INACTION WITH RESPECT TO ANY CONTENT ON THE SOFTWARE/SERVICE. 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 SOFTWARE/SERVICE 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 Software/Service, including any data or work transmitted or received by User, and (c) any unacceptable use of the Software/Service, 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 Software/Service, 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 Software/Service 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 Software/Service as authorized by law. User has no expectation of privacy with respect to the Internet in general. User’s IP address and Software/Service generated GUID is transmitted and recorded with each User session.

b. *Cookies.* The Software/Service may use cookies. A cookie is a small data file that a website stores on the User’s computer when the Software/Service is accessed via the Internet. A cookie allows Company to monitor activity on its website and remember user preferences. Company uses information contained in cookies to improve the User’s experience, as well as track usage and tailor service options and content to usage patterns. Company does not use cookies to retrieve information from the User’s computer not related to the Software/Service.

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 Software/Service 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 Software/Service 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 delivering such amended terms to User by e-mail at the address provided to Company by User or by requiring the User to accept an updated Agreement upon accessing the Software/Service. User’s access to or use of the Software/Service 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 Software/Service and supersedes any and all prior or contemporaneous communications, representations, statements and understandings, whether oral or written, between the parties concerning the Software/Service.

USER HAS READ, UNDERSTANDS AND AGREES TO THE TERMS & CONDITIONS OF THIS AGREEMENT AND APPENDIX A AS INCORPORATED HEREIN.

8 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
Copyright (c) 1998-2004 The OpenSSL Project. All rights reserved.

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.

9 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.

› Getting Help & Support
› Purchase NinjaTrader
› Learning to Use NinjaTrader

9.1 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 to a data provider and using the pre-built connections
- 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 Educational Resources Forums feature sample code and tips covering real-world scenarios and commonly used snippets
- 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

9.2 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)
- License code (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

Current subscribers (evident by providing a license code) will receive priority support.

9.3 Purchase NinjaTrader

NinjaTrader is a free application for advanced charting, market analytics, system development and trade simulation. Should you wish to trade live, please see our available purchase options at our website.

You can also contact us at:

NinjaTrader, LLC
1422 Delgany Street
Suite 400
Denver, CO 80202 USA

FAX: 801.806.2591

NinjaTrader utilizes internet-based communications for all inbound sales and support inquiries and a combination of email, phone and remote assistance for an expeditious resolution. Please submit a request and we will respond quickly to your inquiry.

9.4 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

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 Training Events

NinjaTrader Live Training Events

- NinjaTrader offers Free Training Events on various features throughout the day every Monday through Friday.
- These events allow users to see live product demonstrations and ask questions live within the web room.

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.

9.5 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 & extendability 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 on our forum 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 8 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
- 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.

10 Configuration

Configuration Overview

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

› Installation
› Connection
› Options
› Performance Tips

10.1 Connecting

All connection management is done via the **Connection** menu in the Control Center. Details step by step instructions for setting up your connection can be found in the Connection Guide. NinjaTrader comes with the following default installed connections: External Data Feed, Kinetick - End Of Day (Free), Playback Connection and the Simulated Data Feed .

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.

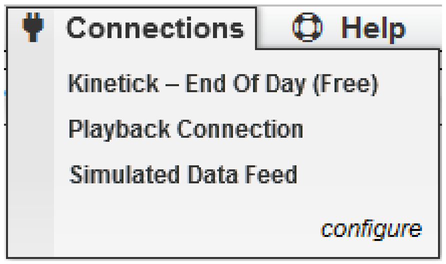
› Connecting to your Account
› Multiple Connections
› External Data Feed Connection
› Playback Connection
› Simulated Data Feed Connection
› Connecting to Kinetick

10.1.1 Connecting to your Account

▿ Understanding account connections

Account Connections

Within the NinjaTrader Control Center window, select the **Connections** menu, all defined connections will be displayed in this menu. 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. Selecting the connection will cause you to connect to it.



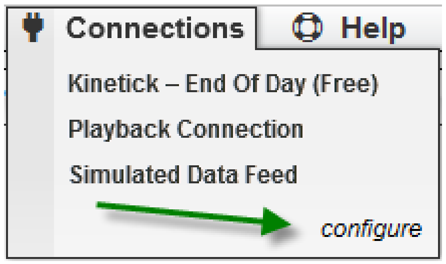
How to create an account connection

Creating an Account Connection

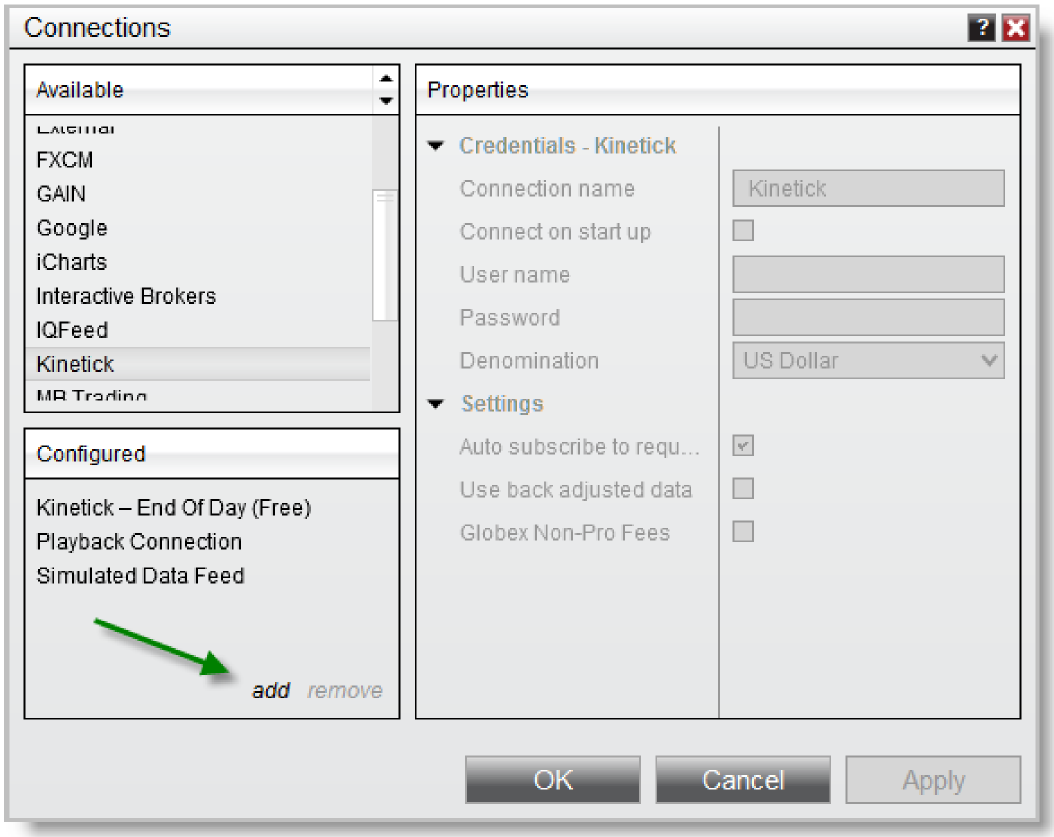
Within the **Connections** menu you can add a connection, change a connection or remove a connection. The following steps use Kinetick as the connectivity provider. This provider is used for demonstration purposes. You can access broker/technology specific connection help information via the NinjaTrader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

To create an account connection:

1. Open the **Connections** window by going to the **Tools** 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".


Connections

Available
External
FXCM
GAIN
Google
iCharts
Interactive Brokers
IQFeed
Kinetick
MB Trading

Configured
Kinetick – End Of Day (Free)
Playback Connection
Simulated Data Feed

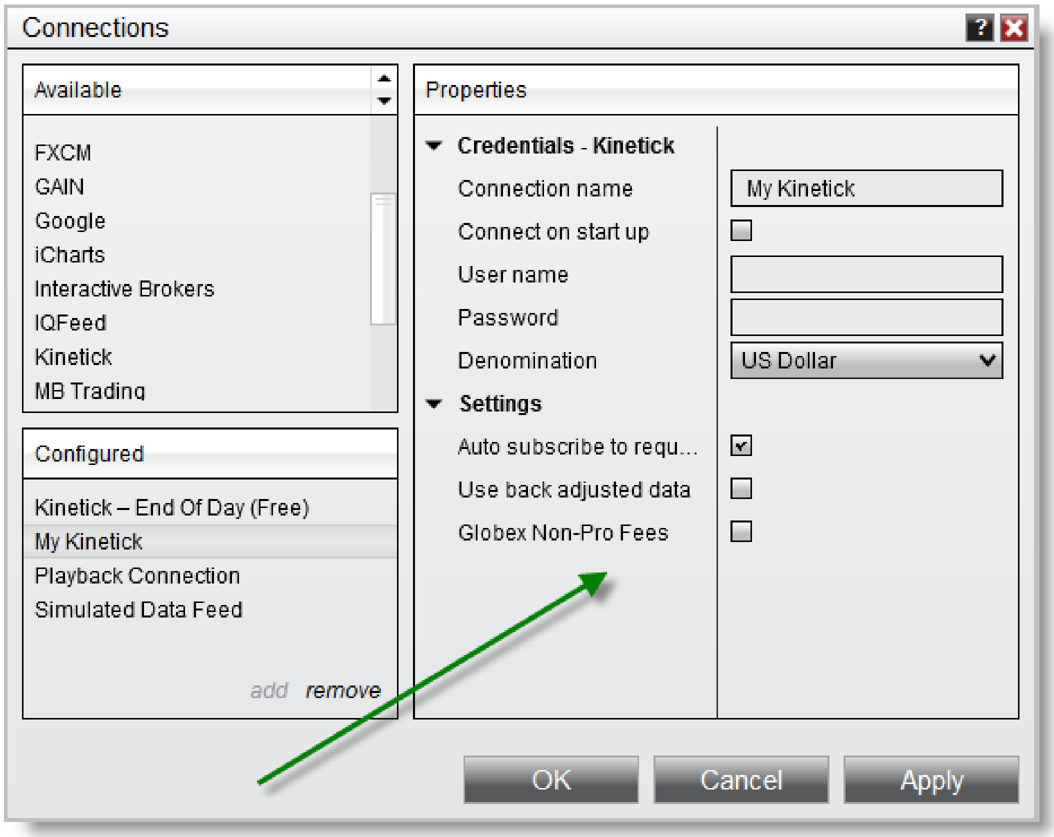
Properties
Credentials - Kinetick
Connection name: Kinetick
Connect on start up: ☐
User name
Password
Denomination: US Dollar
Settings
Auto subscribe to requ...: ✓
Use back adjusted data: ☐
Globex Non-Pro Fees: ☐

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.


Connections

Available
- FXCM
- GAIN
- Google
- iCharts
- Interactive Brokers
- IQFeed
- Kinetick
- MB Trading

Configured
- Kinetick – End Of Day (Free)
- My Kinetick
- Playback Connection
- Simulated Data Feed

Properties	
Credentials - Kinetick	
Connection name	My Kinetick
Connect on start up	☐
User name	
Password	
Denomination	US Dollar
Settings	
Auto subscribe to requ...	✓
Use back adjusted data	☐
Globex Non-Pro Fees	☐

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.

10.1.2 Connecting to Kinetick

What is Kinetick?

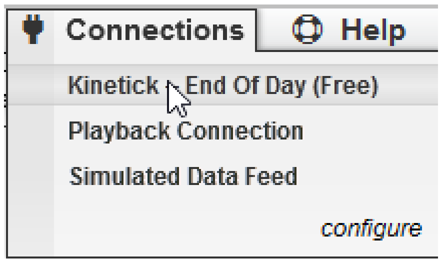
Kinetick is the preferred market data service for NinjaTrader 8. Kinetick provides FREE end of day data for stocks, futures and forex. Real-time data service starts as low as $55 per month and you can qualify to have your CME Globex Futures exchange fees reduced with a qualified brokerage account and a live NinjaTrader license key. 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.

10.1.3 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.

10.1.4 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.

10.1.5 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:

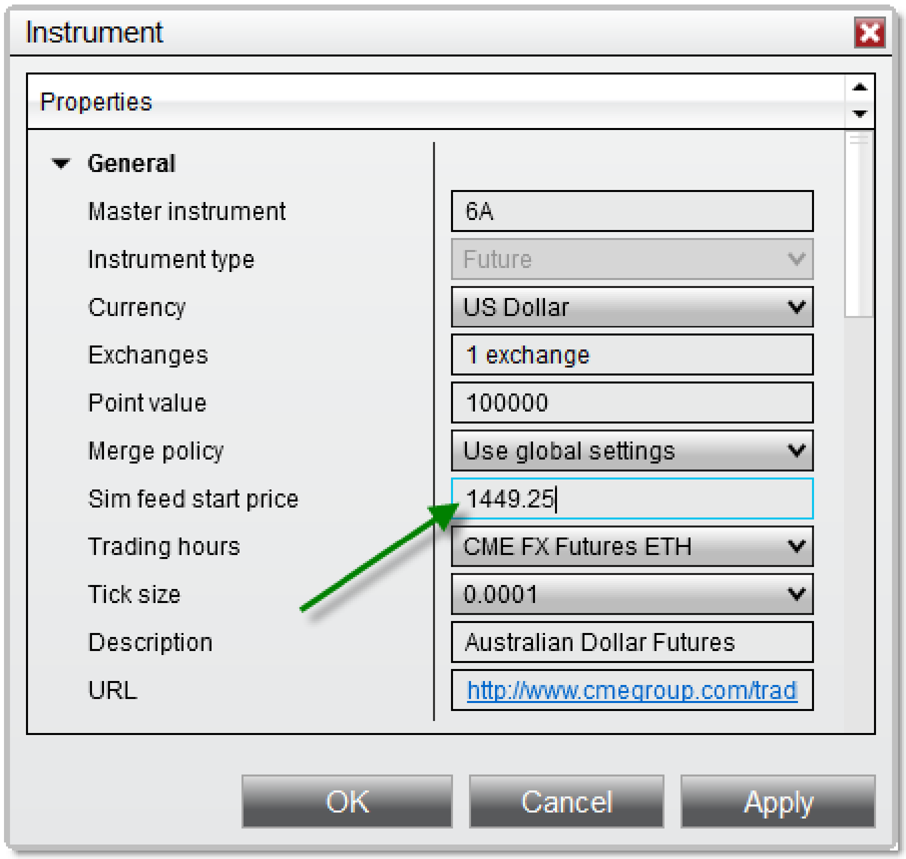
- Offline simulated training and practice of NinjaTrader
- Offline testing of strategies
- Offline testing of trade automation using NinjaScript strategies or the NinjaTrader Automated Trading Interface

Note: This connection is a random internally generated market and has **NO** correlation to real market 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.



General

Master instrument	6A
Instrument type	Future
Currency	US Dollar
Exchanges	1 exchange
Point value	100000
Merge policy	Use global settings
Sim feed start price	1449.25
Trading hours	CME FX Futures ETH
Tick size	0.0001
Description	Australian Dollar Futures
URL	http://www.cmegroup.com/trad

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.

10.2 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.

› Installation Guide
› Minimum System Requirements
› Clear Browser Cache

10.2.1 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 4.5 installed on your PC please download and install it from here.
2. Download and install NinjaTrader.
3. Firewall Software – NinjaTrader contacts our license server on application start up for license key validation. If you have a firewall, spyware or other such software running on your PC, please ensure that you grant NinjaTrader permission to access the internet or you may receive an invalid or license expired message.
4. Platform Activation for Live Trading – If you registered to use the free simulation version you can skip this step.
    - A license key to activate live trading is sent via email within minutes from the time you complete your purchase (check your junk/spam folder if you have not received it)

  - Start NinjaTrader, select the menu Help > License Key within the Control Center and enter your license key
5. Once the installation is complete, please review the appropriate Connection Guide to establish a connection to your broker or market data feed service provider

10.2.2 Minimum System Requirements

Minimum PC Requirements

Your PC must meet the minimum requirements listed below to run NinjaTrader 8
Windows Vista (SP2) w/ Platform Update, Windows 7, Windows 8, Windows 10, Windows Server 2008 w/ Platform Update, Windows Server 2008 R2 or later

- 1 gigahertz (GHz) or faster 32-bit or 64-bit processor
- 2GB RAM
- Microsoft .NET Framework 4.5
- (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 8 was designed to take full advantage of modern PC architecture. To achieve the highest possible level of performance, NinjaTrader 8 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

10.2.3 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

10.3 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.

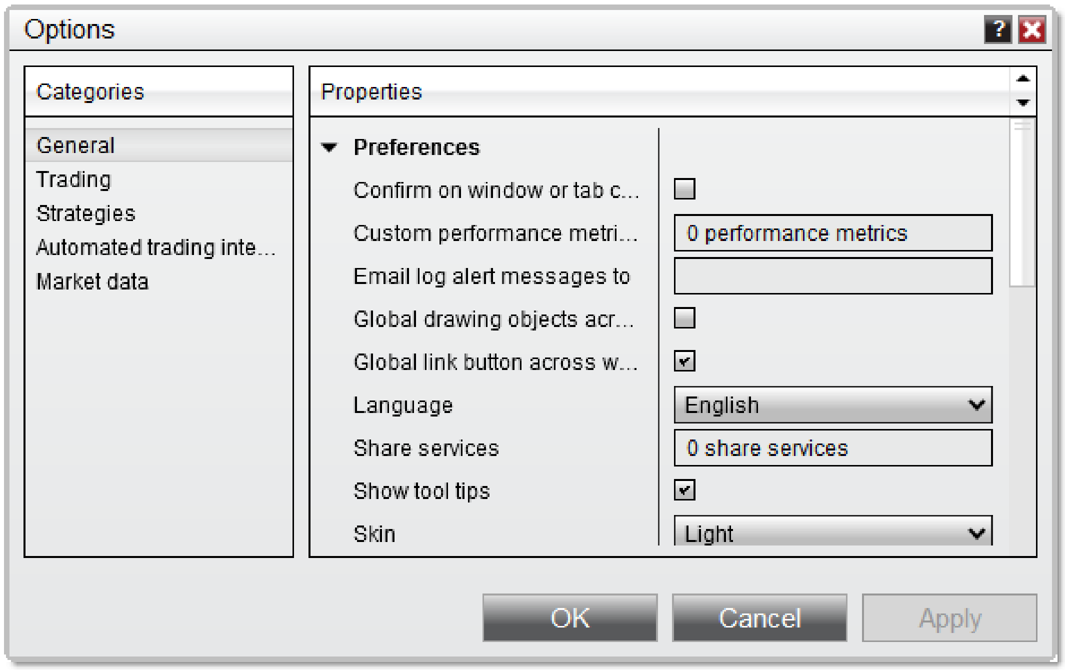
› General
› Trading
› Strategies
› Automated trading interface
› Market data

10.3.1 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:



Confirm on window or tab c... ☐
Custom performance metri... 0 performance metrics
Email log alert messages to
Global drawing objects acr... ☐
Global link button across w... ✓
Language English
Share services 0 share services
Show tool tips ✓
Skin Light

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 messages to	Sets the email address that you would like any alert messages from the log tab in the **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.

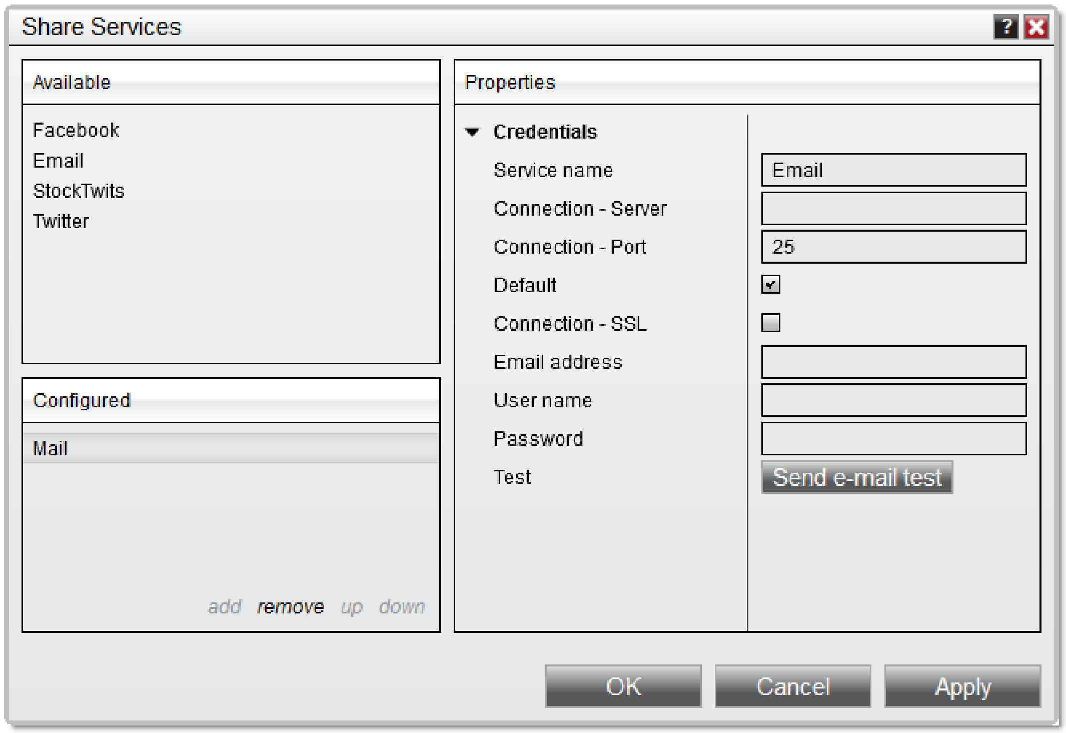
Global Drawing Objects Across Workspaces	Sets whether **Global Drawing Objects** should be applied across all open workspaces.
Global link button across workspaces	Sets whether the global link button will work across all open workspaces or only the current workspace.
Language	Sets the language you would like NinjaTrader to use. Changing this property requires a restart.
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 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.
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.
Sounds	
Play consecutivel y	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, Facebook, StockTwits, and Twitter, and it is possible for developers to create their own ShareService in NinjaScript to access other social media outlets.


Share Services

Available: Facebook, Email, StockTwits, Twitter

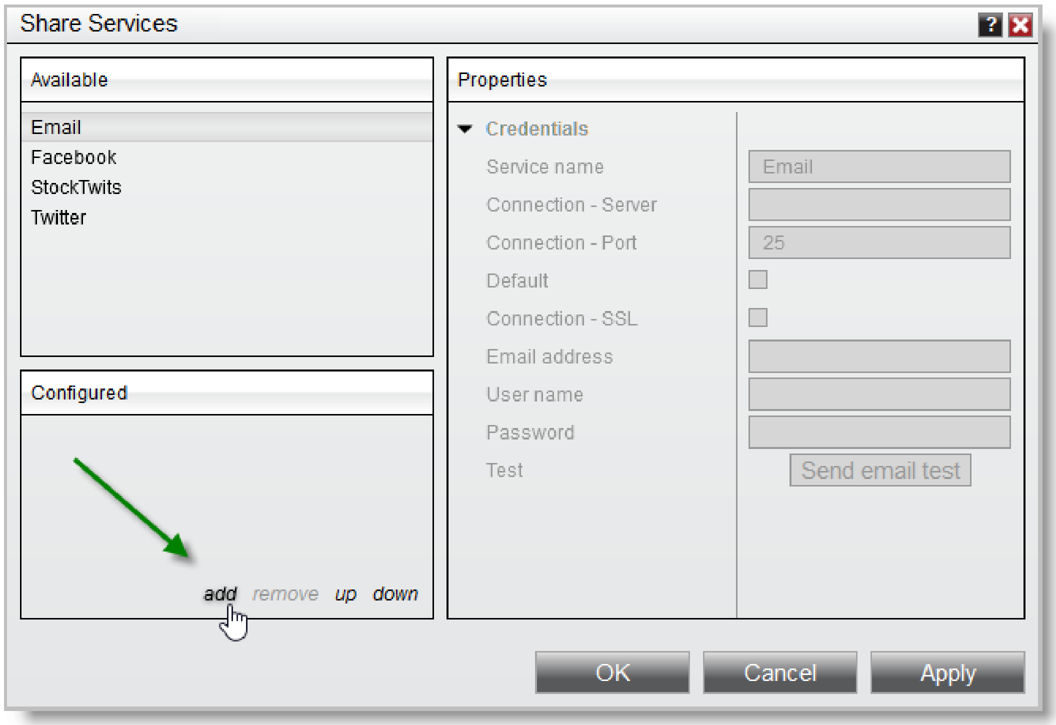
Configured: Mail

Properties	
Credentials	
Service name	Email
Connection - Server	
Connection - Port	25
Default	☑
Connection - SSL	☐
Email address	
User name	
Password	
Test	Send e-mail test

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.

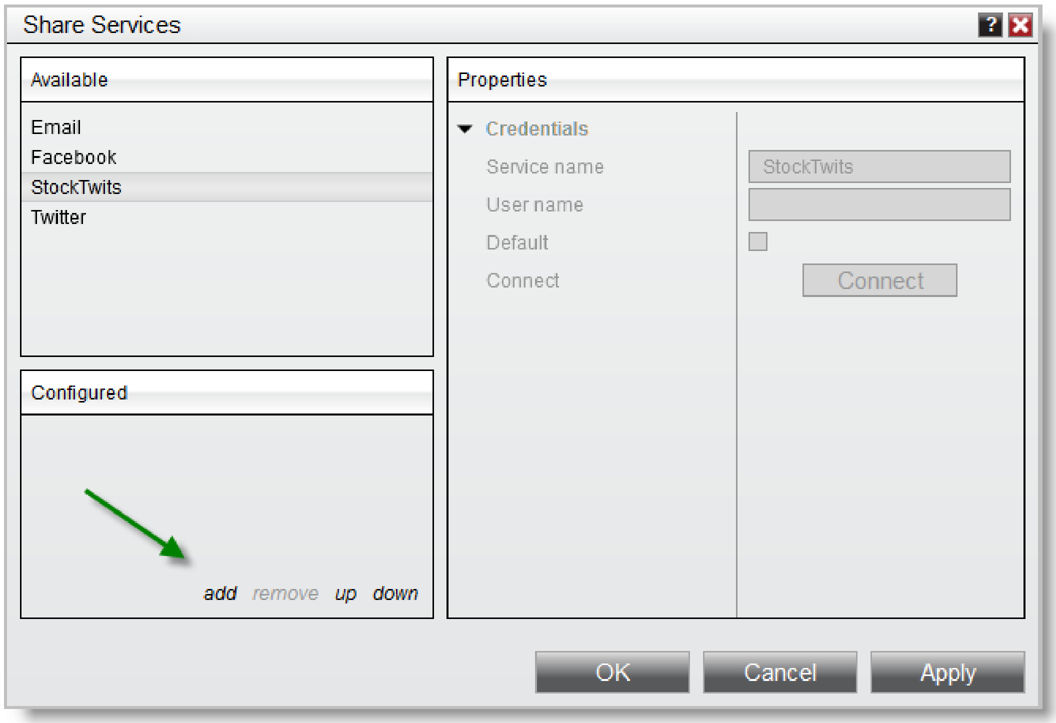


Credentials	
Service name	Sets the name of the **Share Service** that will be used to identify this account when selecting a service to which to share content
Connection - Server	Sets the server address used for the SMTP connection
Connection - Port	Sets the server port used for the SMTP connection
Default	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

Facebook, StockTwits, and Twitter Service Setup

To set up an account that can be used to post messages and images from NinjaTrader, select a provider from the available sectionn then click **`add`**. After this, enter the required data to set up the account in the properties section.



Credentials	
Service name	Sets the name of the **Share Service** that will be use to identify this account when sharing content
User name	Displays the username of the connected account. Note: This field will automatically populate after an account is connected.
Default	Sets if this is the default sharing service to be used for automated sharing from NinjaScript. Note: You can only have one default per service type.
Connect	Press "**Connect**" to launch a web browser to log in to your account and authorize NinjaTrader to post on your behalf. Once authorized, the browser window will be closed and your User name will be displayed in the field above.

10.3.1.1 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

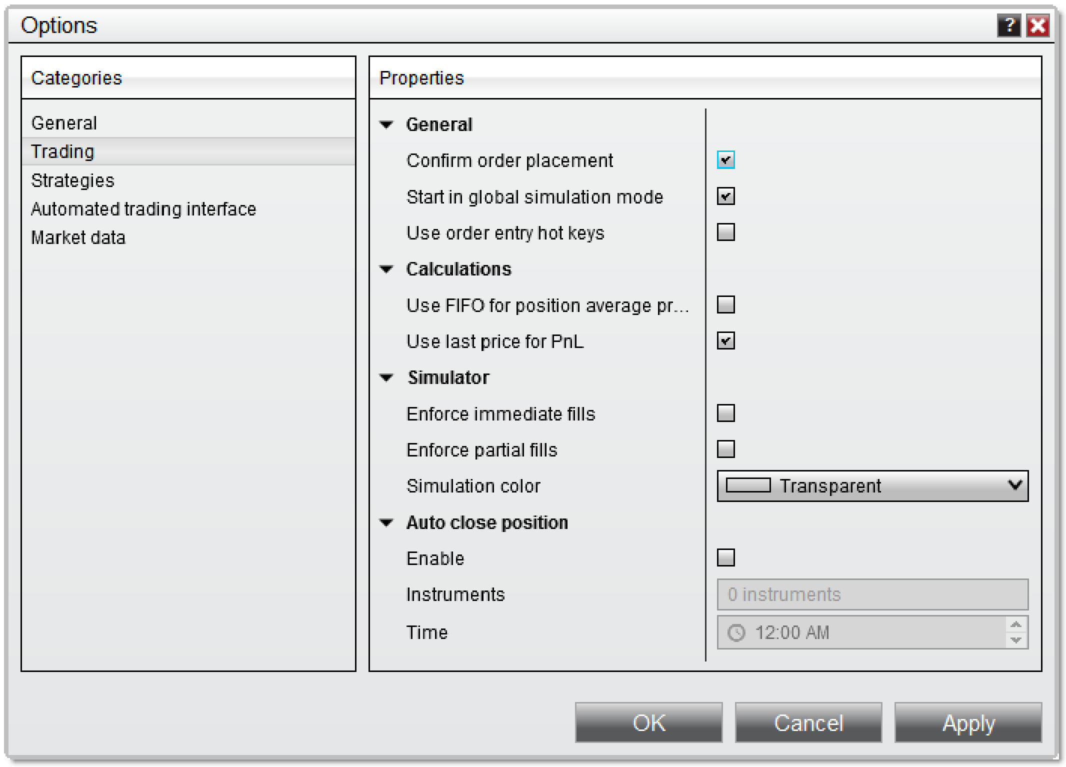
10.3.2 Trading

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 ✓
- Start in global simulation mode ✓
- Use order entry hot keys ☐

Calculations
- Use FIFO for position average pr... ☐
- Use last price for PnL ✓

Simulator
- Enforce immediate fills ☐
- Enforce partial fills ☐
- Simulation color: Transparent

Auto close position
- Enable ☐
- Instruments: 0 instruments
- Time: 12:00 AM

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**.
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.

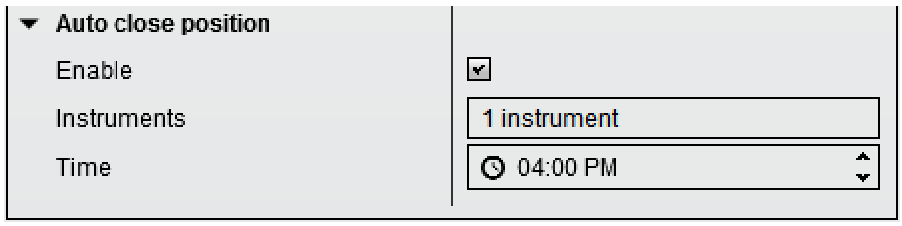
Calculations	
Use FIFO for position average price calculations	Sets if the position average entry price will be based on the FIFO (First in First Out) method or LIFO (Last in First Out) method.
Use last price for PnL	Sets if the last trade price is used to calculate profit and loss, or if the Bid will 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.
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	✓
Instruments	1 instrument
Time	04:00 PM

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

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 sales@ninjatrader.com for upgrade options.

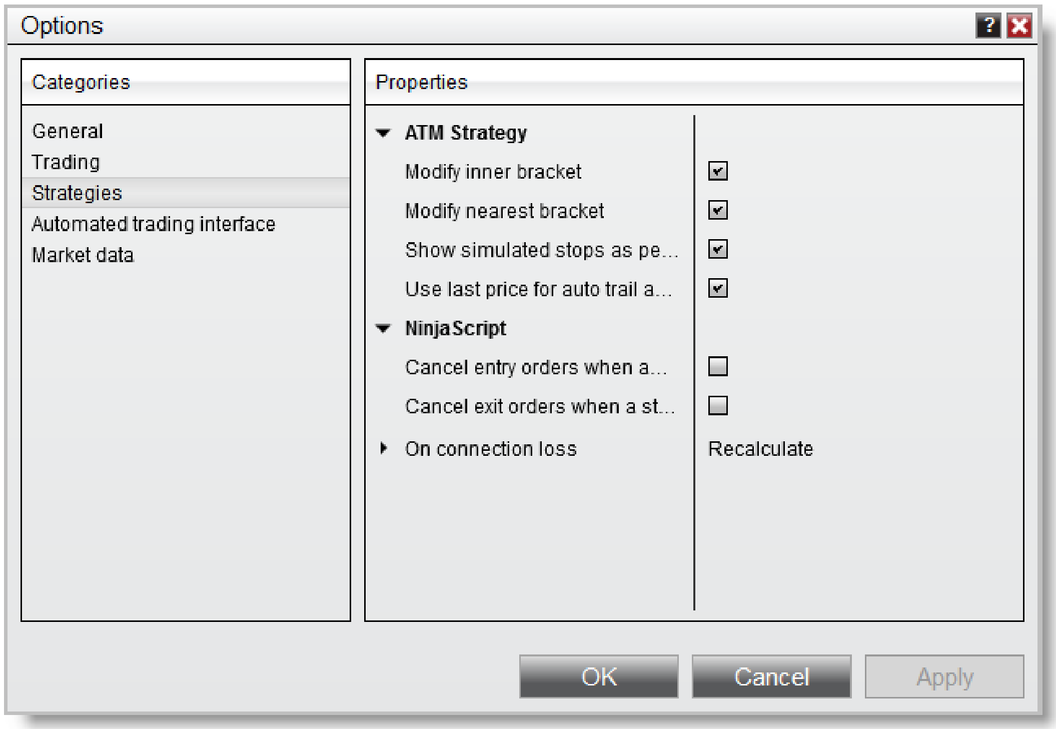
10.3.3 Strategies

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.



Options

Categories
General
Trading
Strategies
Automated trading interface
Market data

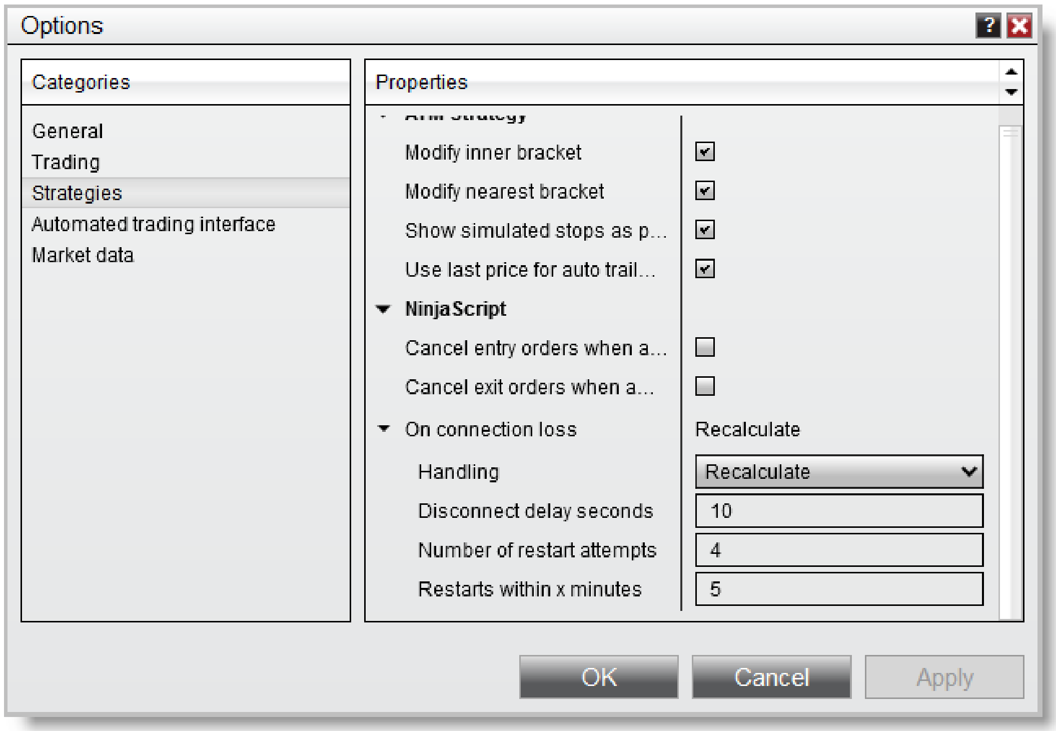
Properties	
ATM Strategy	
Modify inner bracket	✓
Modify nearest bracket	✓
Show simulated stops as pe...	✓
Use last price for auto trail a...	✓
NinjaScript	
Cancel entry orders when a...	☐
Cancel exit orders when a st...	☐
On connection loss	Recalculate

ATM Strategy	
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
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.
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.

Understanding NinjaScript properties

NinjaScript properties

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



Modify inner bracket	✓
Modify nearest bracket	✓
Show simulated stops as p...	✓
Use last price for auto trail...	✓
NinjaScript	
Cancel entry orders when a...	☐
Cancel exit orders when a...	☐
On connection loss	Recalculate
Handling	Recalculate
Disconnect delay seconds	10
Number of restart attempts	4
Restarts within x minutes	5

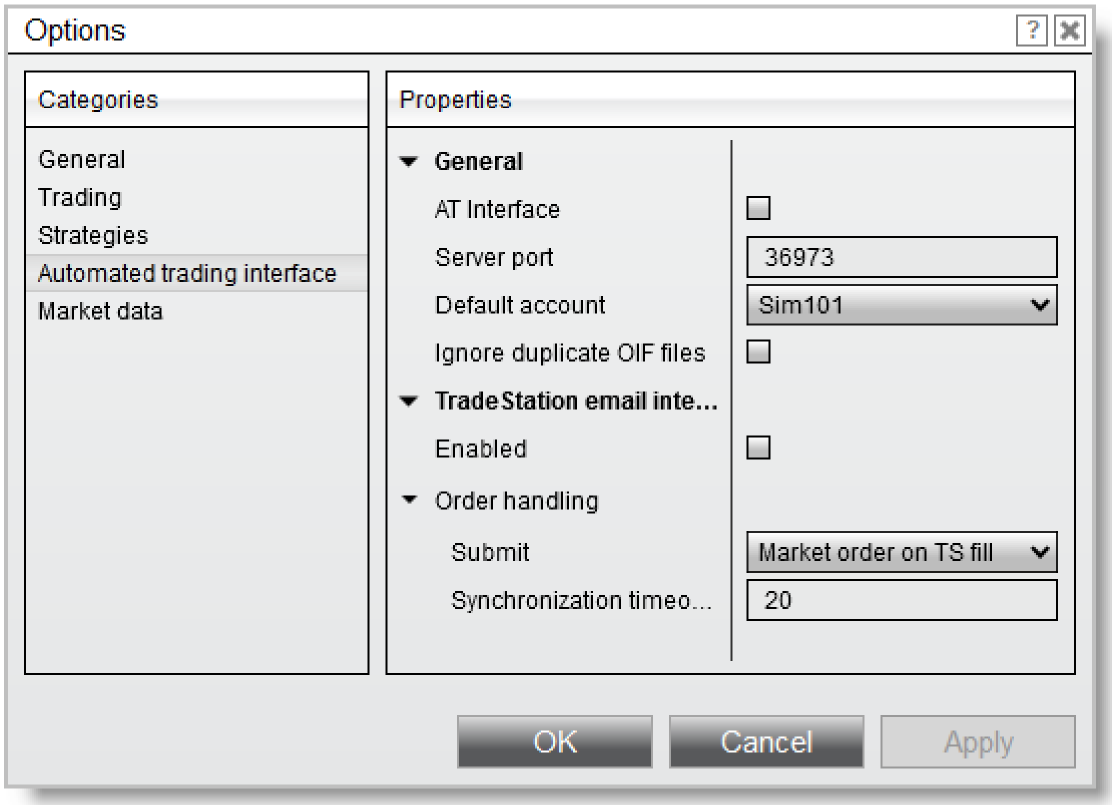
Cancel entry orders when a strategy is disabled	Enables or disables automatic cancellation of a NinjaScript strategy's entry orders when the strategy is disabled.
Cancel exit orders when a strategy is disabled	Enables or disables automatic cancellation of a NinjaScript strategy's exit orders when the 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	Sets the number of times NinjaTrader will

restart attempts	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".

10.3.4 Automated trading interface

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


Options

Categories: General, Trading, Strategies, Automated trading interface, Market data

Properties

General
AT Interface ☐
Server port 36973
Default account Sim101
Ignore duplicate OIF files ☐

TradeStation email inte...
Enabled ☐

Order handling
Submit Market order on TS fill
Synchronization timeo... 20

General Properties

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

General	
AT Interface	Sets if the **automated trading interface** is enabled.
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.

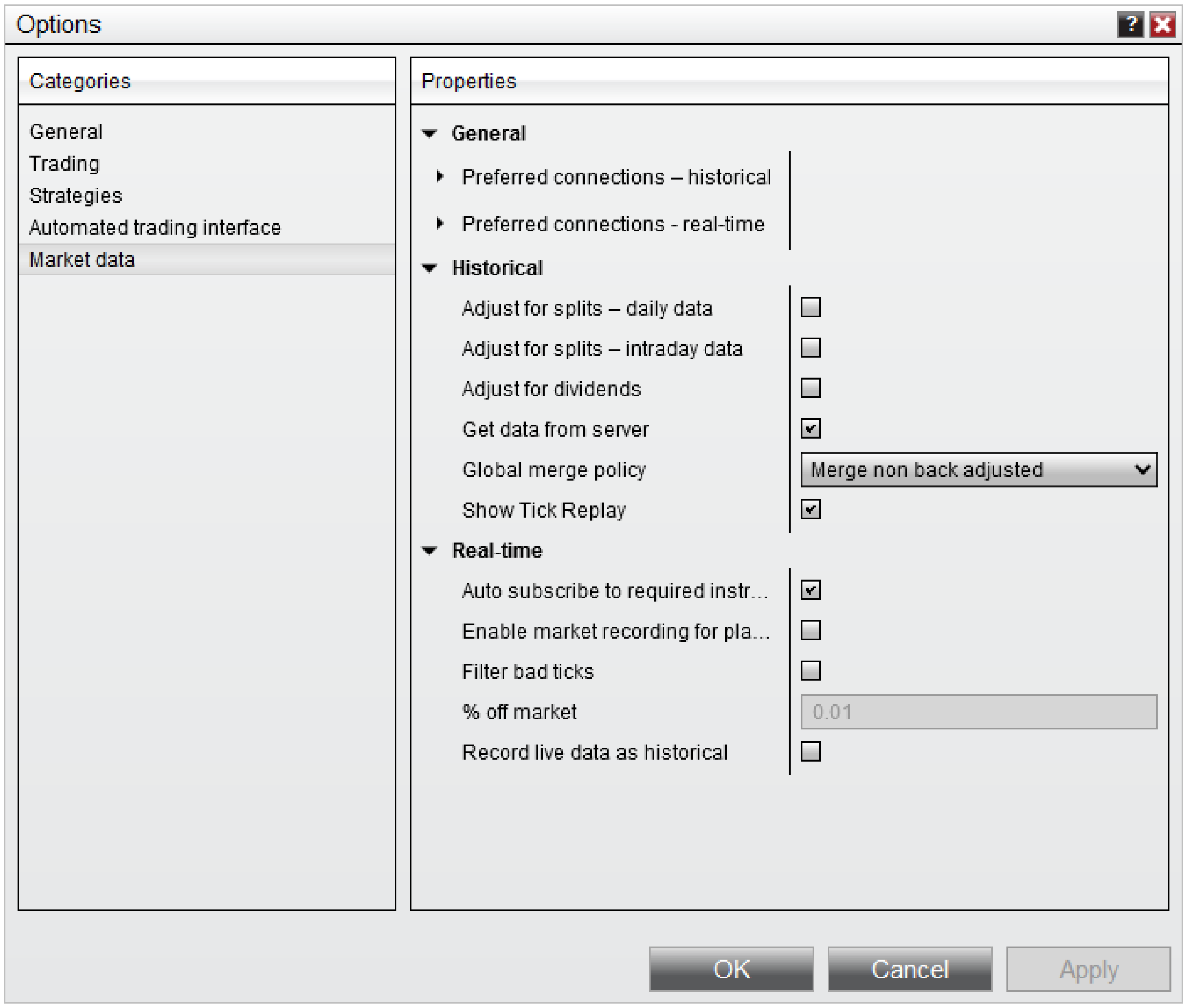
Enabled	Sets if the Tradestation email interface is enabled.
Order handling	
Submit	Sets how NinjaTrader will handle orders submitted from the email interface. **Market orders on TS fill:** NinjaTrader will submit market orders when NinjaTrader receives a strategy order filled email notification from 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.

10.3.5 Market data

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



General	
Preferred connections – historical	
Preferred connections - real-time	
Historical	
Adjust for splits – daily data	☐
Adjust for splits – intraday data	☐
Adjust for dividends	☐
Get data from server	✓
Global merge policy	Merge non back adjusted
Show Tick Replay	✓
Real-time	
Auto subscribe to required instr...	✓
Enable market recording for pla...	☐
Filter bad ticks	☐
% off market	0.01
Record live data as historical	☐

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
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)
Save chart 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.

10.3.5.1 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

Google	YES	YES	NO	NO
Interactive Brokers	----	----	----	----
IQFeed	NO	YES	NO	NO
TD Ameritrade	----	----	----	----
YAHOO	NO	NO	NO	NO

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

10.3.5.2 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.

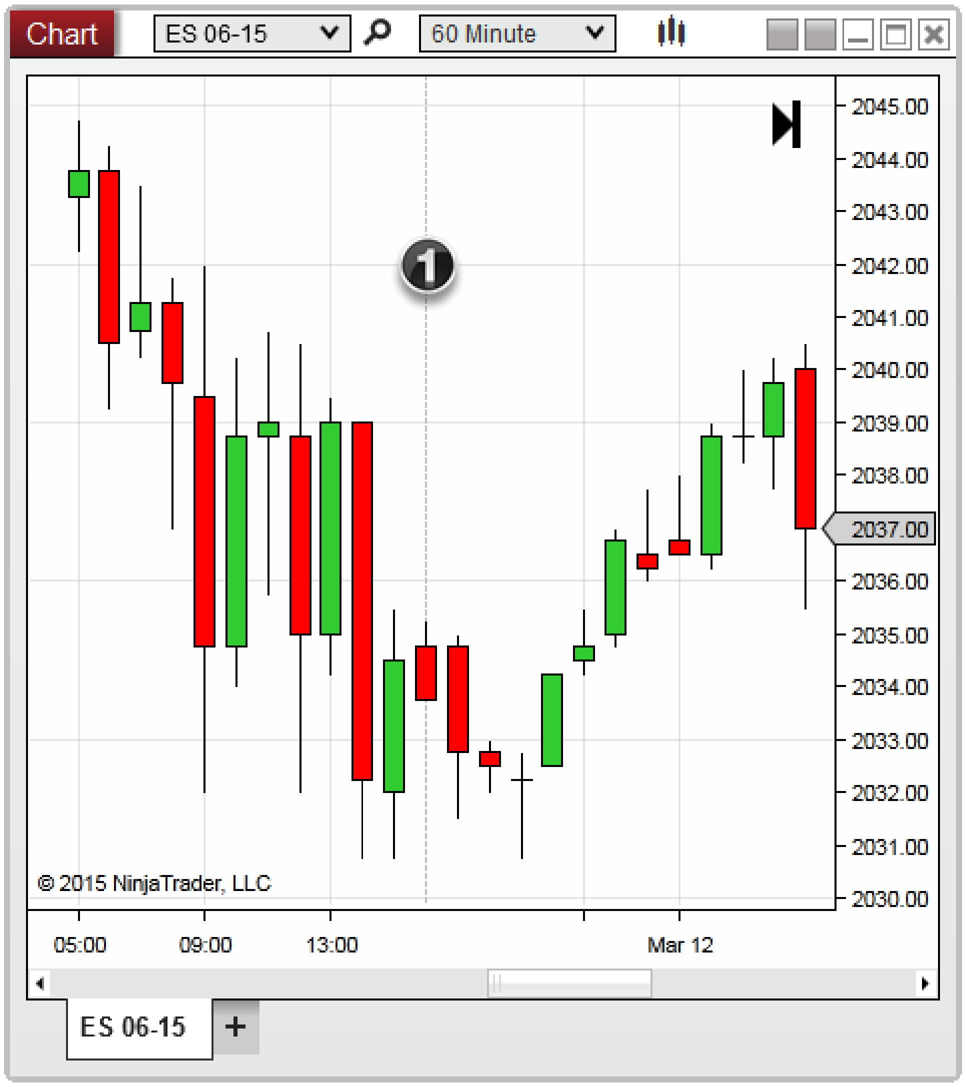
Note: More information on Configuring Rollover Dates and Offsets can be found in the Editing Instruments section of the Help Guide.

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.

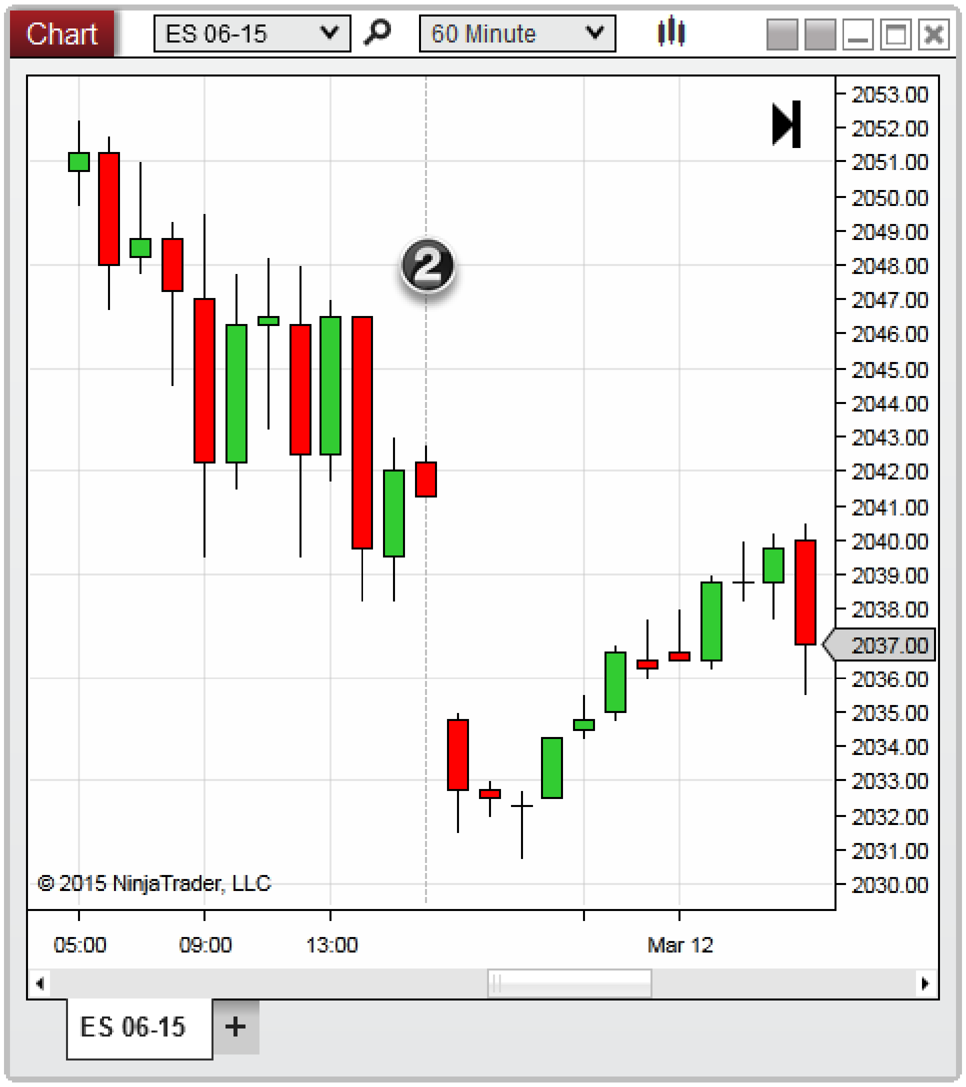


MergeNonBackAdjusted

- 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.

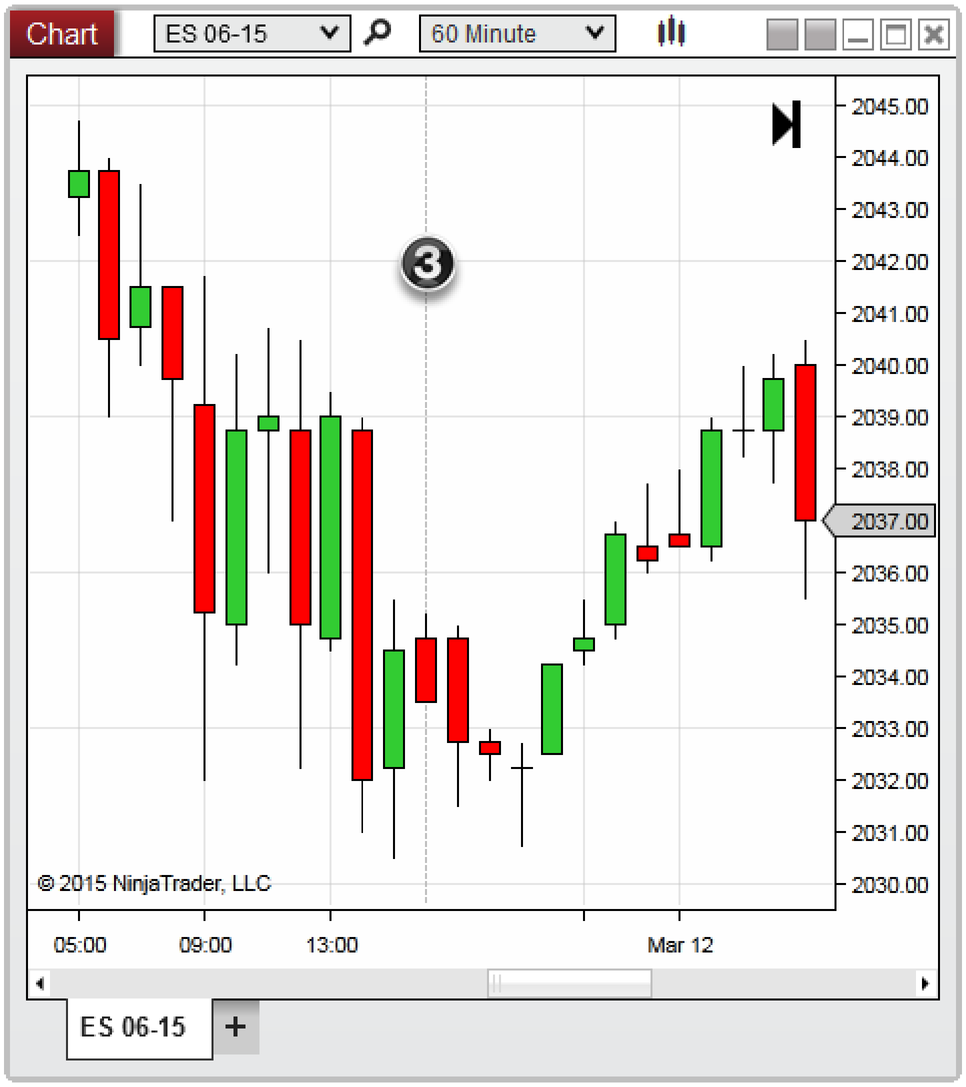


DoNotMerge

- 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.



10.3.5.3 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

10.3.5.4 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 3 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.

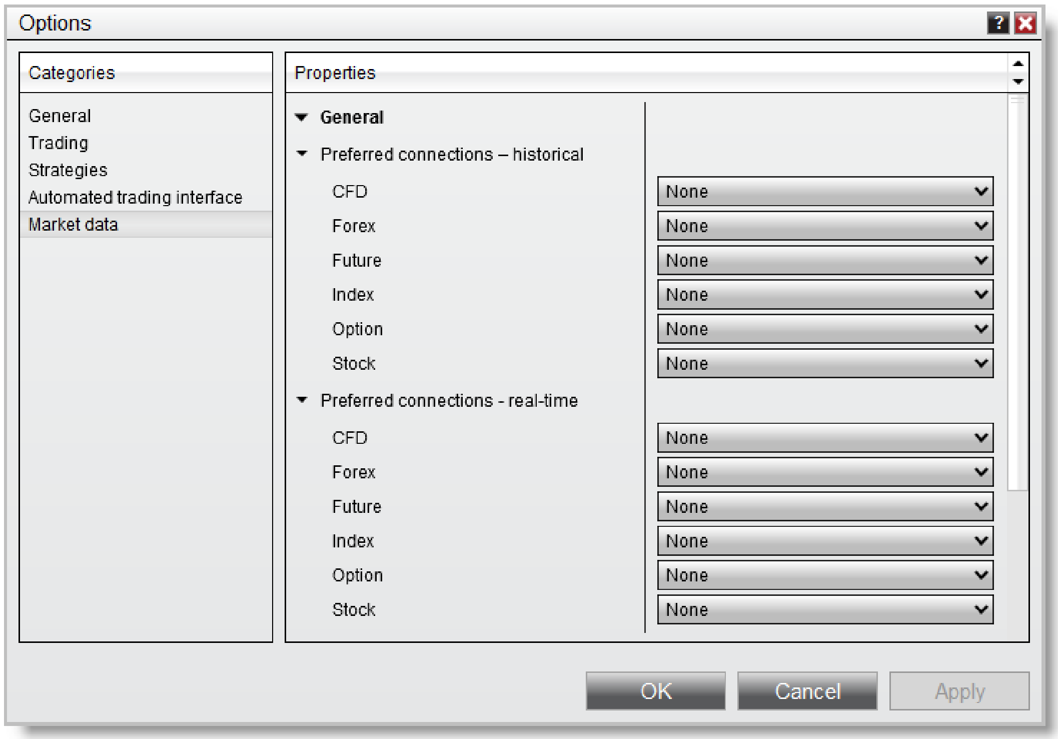
Setting preferred data connections

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.



General

Preferred connections – historical

CFD	None
Forex	None
Future	None
Index	None
Option	None
Stock	None

Preferred connections - real-time

CFD	None
Forex	None
Future	None
Index	None
Option	None
Stock	None

10.4 Performance Tips

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)

The following are some suggestions that can help you fine tune your NinjaTrader installation to run optimally.

Charting

Chart Performance Tips

- Set chart indicators "Calculate" property to a value of 'On Bar Close' or 'On Price Change'
- Reduce the number of bars on your chart (days back or bars back settings under "Data Series" dialog window
- Reduce the number of running indicators
- If you are running a custom or 3rd party indicator or strategy, remove them to isolate if these custom NinjaScript objects may be a CPU or memory drain. It only takes one poorly coded NinjaScript object to bring down performance of the entire application.
- For setups operating with a lesser powerful GPU, set the crosshair mode to 'Draw cursor only' - it will then only draw a mini cross hair without the full lines but including the price / time axis labels. This helps to decrease the load on the rendering process.
- 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. If you need assistance with this configuration, please email platformsupport@ninjatrader.com

Lagging Market Data

Variables that can impact market data latency

- Connecting wirelessly can drastically impact performance during heavy volume periods. Depending on signal strength, your modem/router is unable to push all the data through. This can be resolved by connecting directly into the modem/ router via a hard line.
- Serious abnormal increases in market volume
- Data provider servers could be lagging
- Limited bandwidth internet service (56K dial up modem is not acceptable for example)
- Inadequate PC hardware or running too many applications on your PC

Market Analyzer

Market Analyzer Days To Load property

- "Days To Load" property set this number to the minimum number of bars required to properly initialize any indicator columns. 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.

Miscellaneous

- Within the Control Center window, select **Tools** > **Options** > **Market Data** and make sure "Enable market recording for playback" is unchecked.
- Within the Control Center window, select **Tools** > **Options** > **Data** and make sure "Record live data as historical" is unchecked.
- Reduce the number of applications running on your PC

Note: "Save chart data as historical" should only be checked if are using a connectivity provider that does **NOT** provide historical data

Workspaces

Hidden Workspaces

- Hidden workspaces consume CPU cycles so check under **Control Center** > **Workspaces** to see all of the workspaces that are open and close any that you seldom or never use.

SuperDOM

Show Volume Text

- 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.

Scanning and Search Indexing

Anti-Virus or Backup Software Scanning

- Anti-virus or backup software that actively scans files for infection or changes can impede PC performance when using NinjaTrader, as anti-virus scanners can access historical data, the database, and other files which could take time to scan. To eliminate this impact, it is recommended to add exclusions in your anti-virus or backup software for the following two directories:

  - C:\Users\User\Documents\NinjaTrader 8
  - C:\Program Files (x86)\NinjaTrader 8

Microsoft Windows Indexing

Windows search indexing can also place an extra burden on your processor when running NinjaTrader, negatively impacting performance. It is recommend to exclude the folders listed above from indexing, as well, which can be done via the Windows Control Panel.

Playback

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 MarketAnalyzer but then only replaying MSFT data is expected to have lower performance in contrast to having only this one MSFT instrument listed in the MarketAnalyer as well.

11 Operations

Operations Overview

11.1 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.

Play Video

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

11.1.1 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

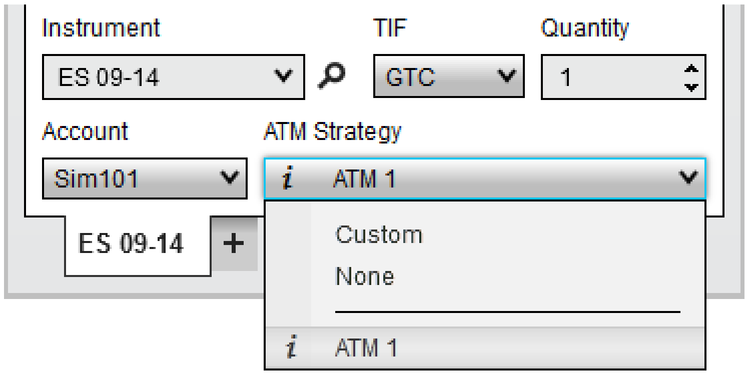
Now that we understand what an ATM Strategy is, what exactly is the value of it? When trading, one develops ideas and methods for entry and further management of their position. The management of this position can be simple to complex and everything in between. The ATM Strategy allows the trader to define 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.

11.1.1.1 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.

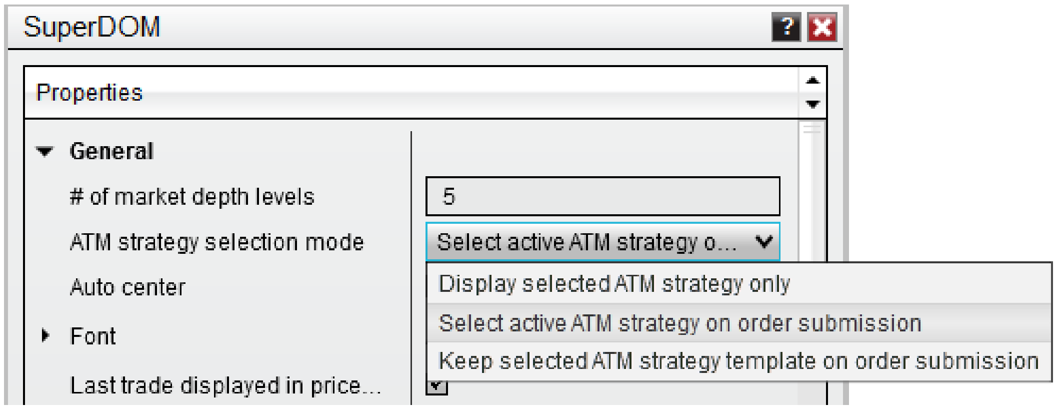
Custom or ATM Strategy Template Names
When an ATM Strategy template name is selected, all of the parameters will update to reflect your pre-defined ATM Strategy, or when **Custom** is selected, you have the ability to define a new ATM Strategy on the fly. Once an order is submitted, the ATM Strategy 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.


SuperDOM

Properties

General

# of market depth levels	5
ATM strategy selection mode	Select active ATM strategy o...
Auto center	
Font	
Last trade displayed in price...	

Display selected ATM strategy only
Select active ATM strategy on order submission
Keep selected ATM strategy template on order submission

When it comes to the automatic submission of Stop Loss and Profit Targets and how subsequent order fills are handled, there are two approaches:

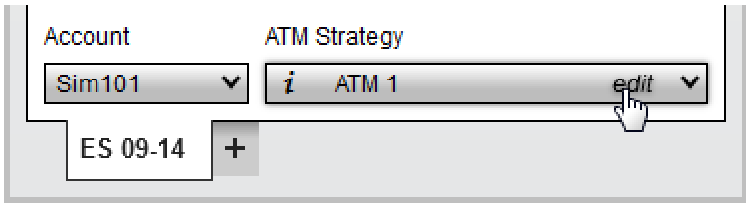
1. Scaling into a position or out of a position should automatically update the order sizes of existing stop and target brackets
2. Scaling into a position should create a new set of stop and target brackets based on the new order fill price

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.

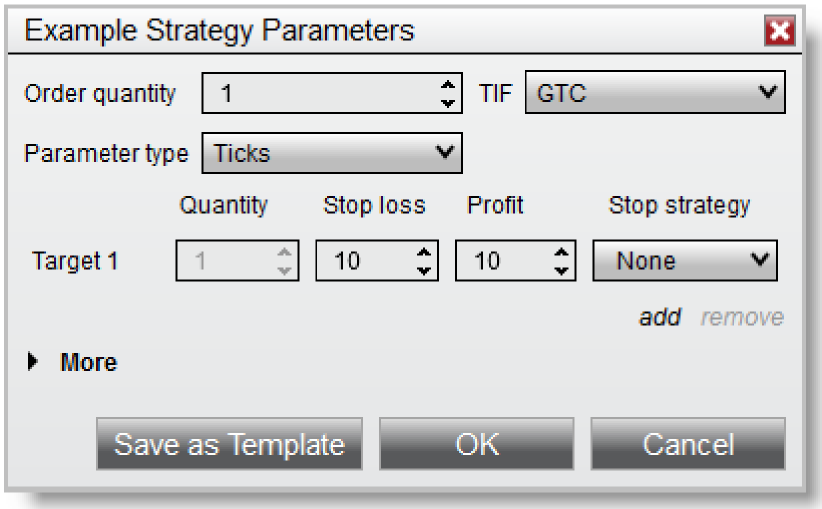
Understanding Stop Loss and Profit Target parameters (how to set your stop and target value

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.



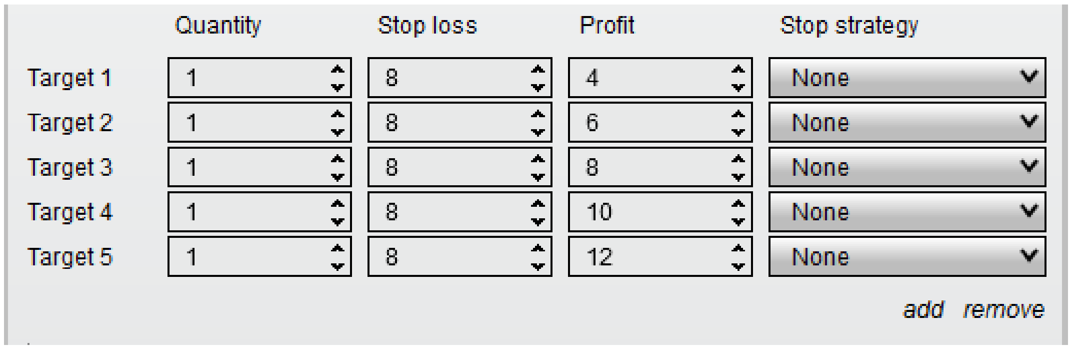
In the image below there are parameters that define the **ATM Strategy**. This strategy is a single quantity strategy that will automatically place its target at 10 ticks above the average entry price and stop loss 10 ticks below.



Order quantity: 1
TIF: GTC
Parameter type: Ticks

	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	10	10	None

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.



	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	8	4	None
Target 2	1	8	6	None
Target 3	1	8	8	None
Target 4	1	8	10	None
Target 5	1	8	12	None

Order	Replicated from the order entry display and sets

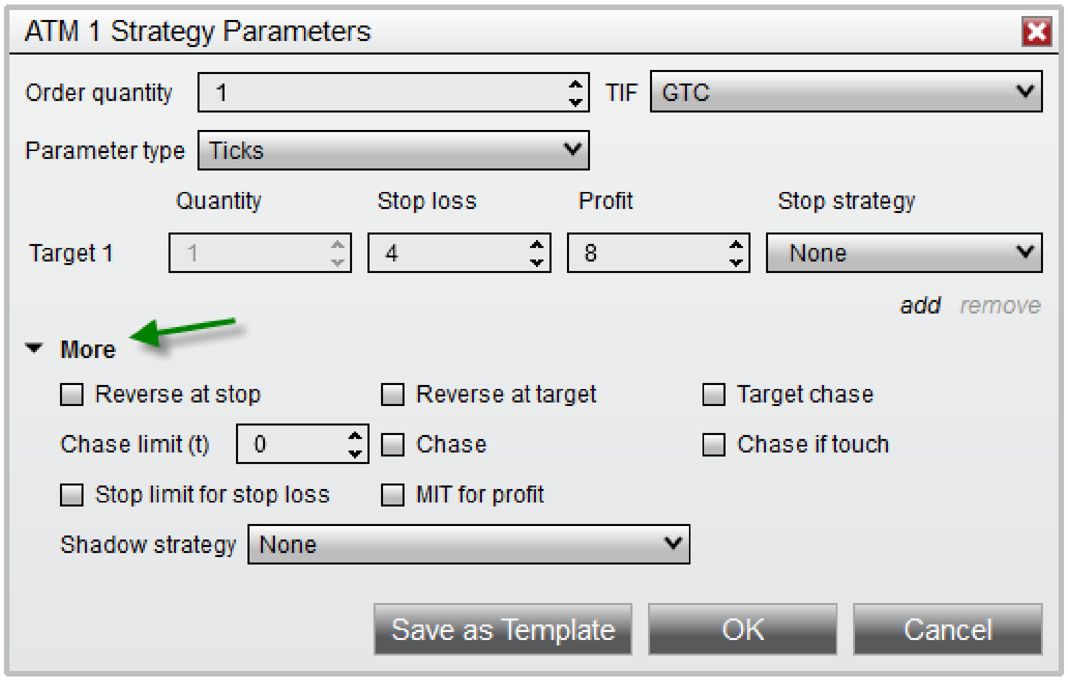
quantity	the initial quantity used for the entry order.
TIF (Time In Force)	Replicated from the order entry display and sets the TIF used for entry, profit target, and stop loss orders.
Paramete r type	Sets the type of parameter used for defining where the stop loss and profit target will be placed. Parameters can be entered as ticks, pips, currency, percent or absolute price
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.



ATM 1 Strategy Parameters

Order quantity	1	TIF	GTC	
Parameter type	Ticks			
	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	4	8	None

☐ Reverse at stop	☐ Reverse at target	☐ Target chase
Chase limit (t) 0	☐ Chase	☐ Chase if touch
☐ Stop limit for stop loss	☐ MIT for profit	

Shadow strategy: None

From the Advanced Options section you can enable the Shadow Strategy, Auto Reverse, or Auto Chase features.

11.1.1.2 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.

Play Video

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

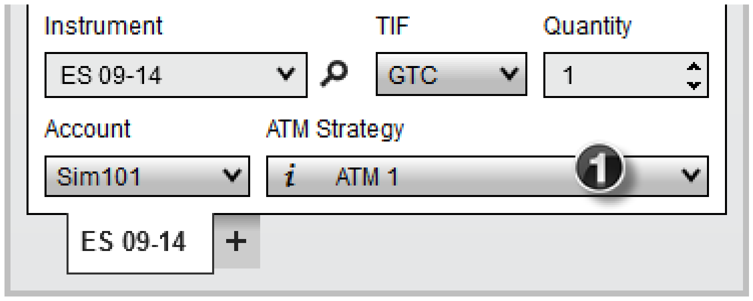
This mode will automatically select the newly created active ATM Strategy on entry order submission in the ATM Strategy control list. This is the default setting upon initial NinjaTrader installation.

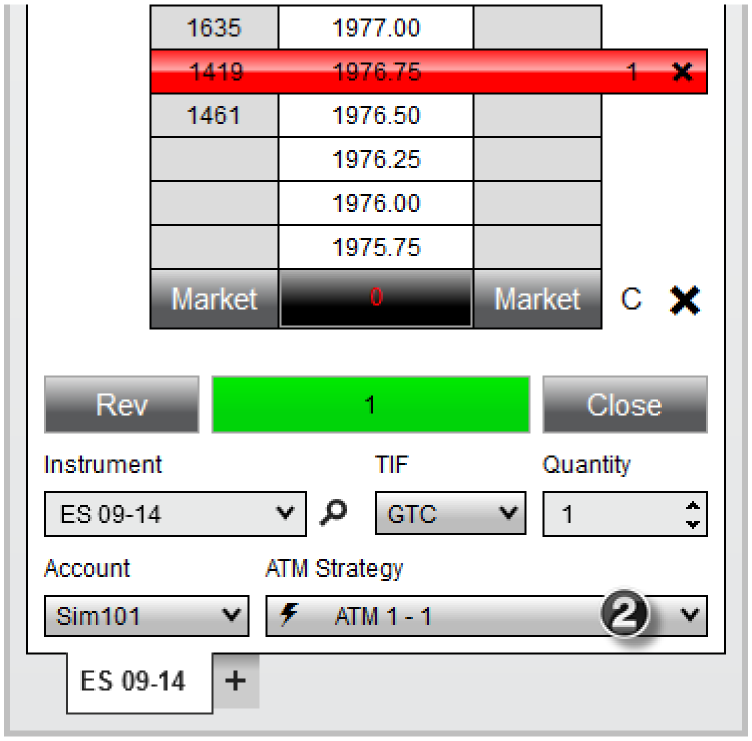
Who is this mode designed for?

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>**).





▽ Understanding the "Keep Selected ATM Strategy Template on Order Submission" mode

Keep Selected ATM Strategy Template on Order Submission

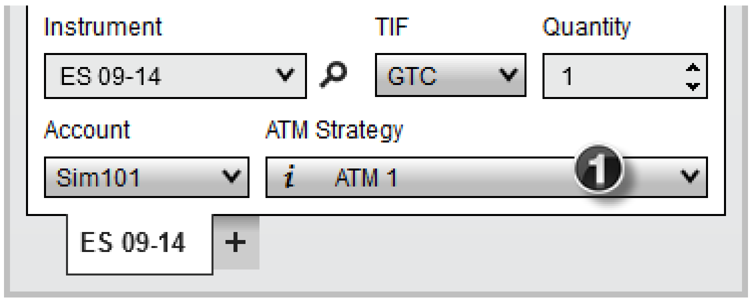
This mode will keep the currently selected ATM Strategy template selected in the strategy control list upon order submission.

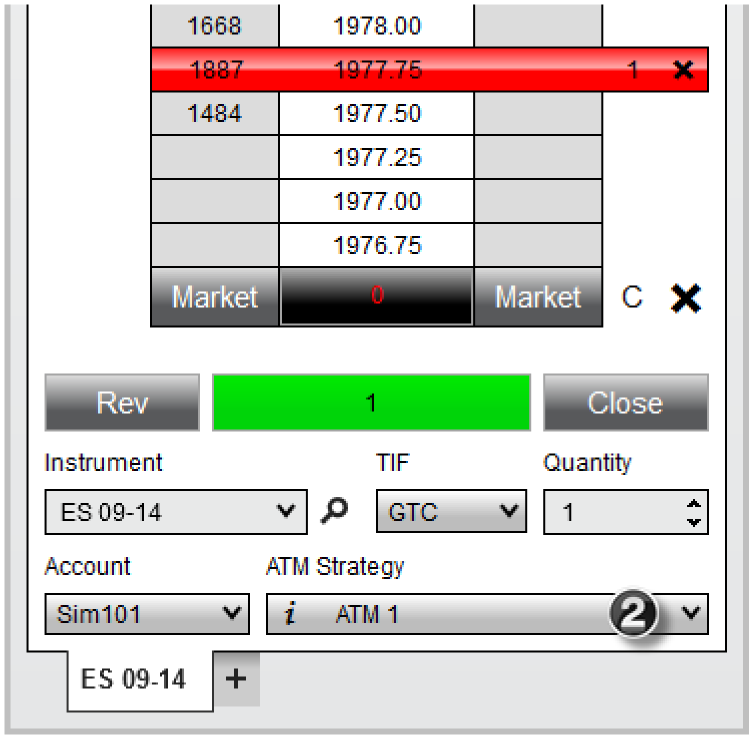
Who is this mode designed for?

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.


Instrument: ES 09-14
TIF: GTC
Quantity: 1
Account: Sim101
ATM Strategy: ATM 1



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.

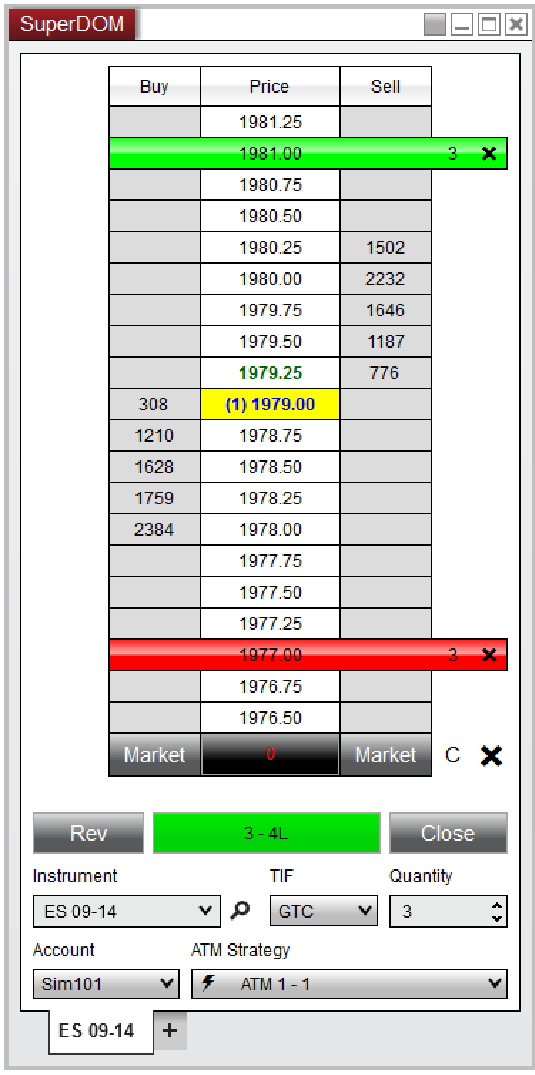
Who is this mode designed for?

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.


SuperDOM

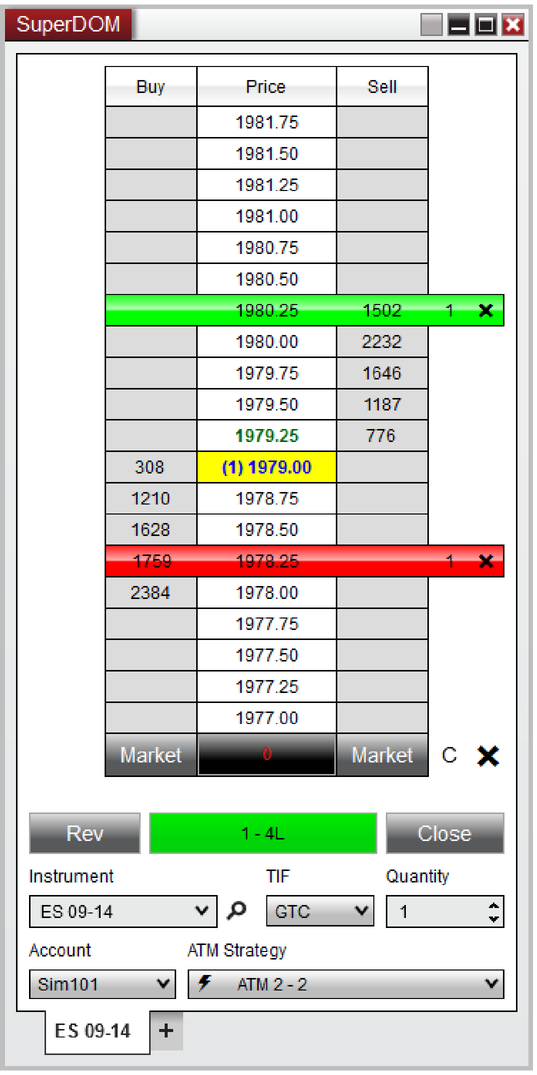
Buy	Price	Sell	
	1981.25		
	1981.00		3 ✕
	1980.75		
	1980.50		
	1980.25	1502	
	1980.00	2232	
	1979.75	1646	
	1979.50	1187	
	1979.25	776	
308	(1) 1979.00		
1210	1978.75		
1628	1978.50		
1759	1978.25		
2384	1978.00		
	1977.75		
	1977.50		
	1977.25		
	1977.00		3 ✕
	1976.75		
	1976.50		
Market	0	Market	C ✕

Rev | 3 - 4L | Close

Instrument: ES 09-14 | TIF: GTC | Quantity: 3

Account: Sim101 | ATM Strategy: ATM 1 - 1

ES 09-14 +


SuperDOM

Buy	Price	Sell	
	1981.75		
	1981.50		
	1981.25		
	1981.00		
	1980.75		
	1980.50		
	1980.25	1502	1 ✕
	1980.00	2232	
	1979.75	1646	
	1979.50	1187	
	1979.25	776	
308	(1) 1979.00		
1210	1978.75		
1628	1978.50		
1759	1978.25		1 ✕
2384	1978.00		
	1977.75		
	1977.50		
	1977.25		
	1977.00		
Market	0	Market	C ✕

Rev | 1 - 4L | Close

Instrument: ES 09-14 | TIF: GTC | Quantity: 1

Account: Sim101 | ATM Strategy: ATM 2 - 2

ES 09-14 +

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.

11.1.1.3 Stop Strategy

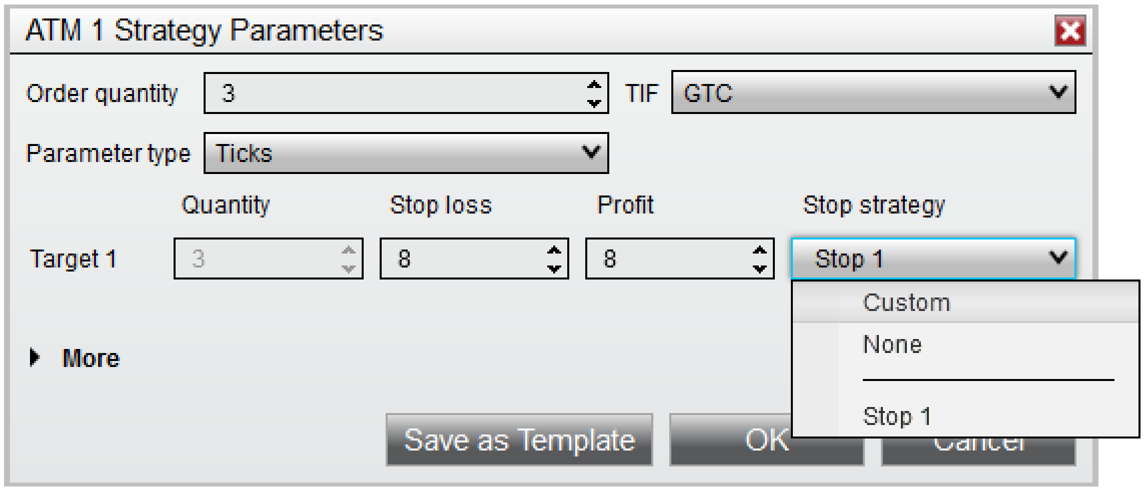
ATM Stop Strategies

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

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.


ATM 1 Strategy Parameters

Order quantity 3 TIF GTC

Parameter type Ticks

	Quantity	Stop loss	Profit	Stop strategy
Target 1	3	8	8	Stop 1

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.

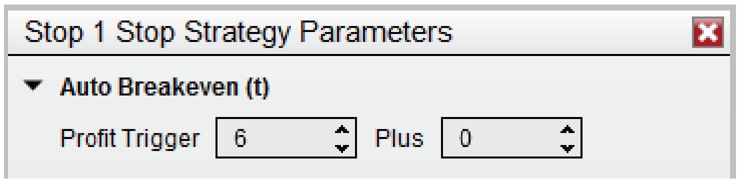
11.1.1.3.1 Auto Breakeven

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


Stop 1 Stop Strategy Parameters

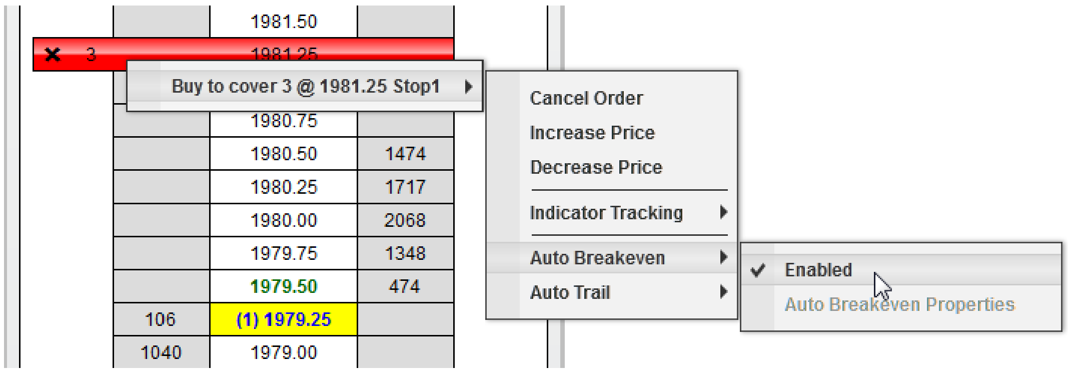
Auto Breakeven (t)

Profit Trigger 6 Plus 0

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.

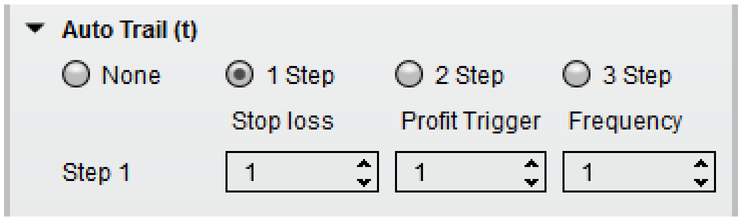
11.1.1.3.2 Auto Trail

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
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

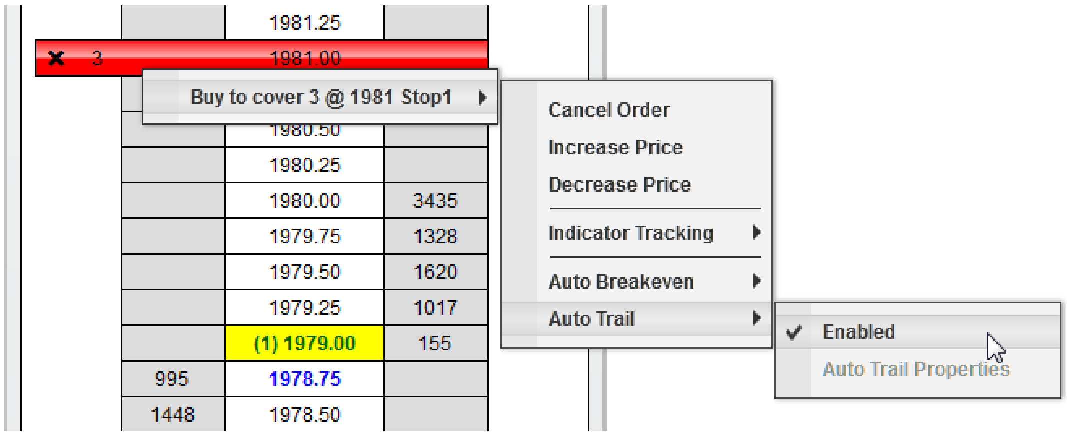

Auto Trail (t)

None [✓ 1 Step] 2 Step 3 Step

	Stop loss	Profit Trigger	Frequency
Step 1	1	1	1

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.

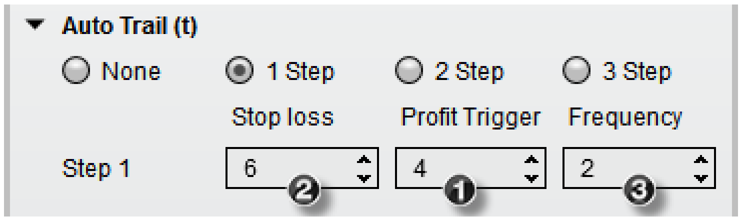


Auto Trail Examples

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."*


Auto Trail (t)

( ) None [✓ 1 Step] ( ) 2 Step ( ) 3 Step

	Stop loss	Profit Trigger	Frequency
Step 1	6	4	2

Average Entry - 1000 Long (SP Emini contract)

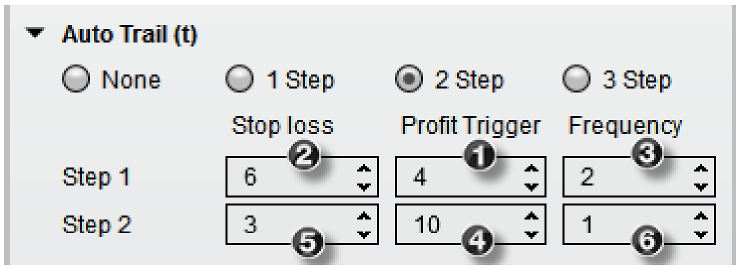
The market moves up to 1001 and the Auto Trail is triggered **(Average Entry + Profit Trigger = 1000 + 4 ticks = 1001)** and the Stop Loss is adjusted to 999.50 **(1001 - Stop Loss = 1001 - 6 ticks = 999.50)**. For every additional 2 ticks **(Frequency of 2 ticks)** the Stop Loss will be adjusted by 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."



	Stop loss	Profit Trigger	Frequency
Step 1	6	4	2
Step 2	3	10	1

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.

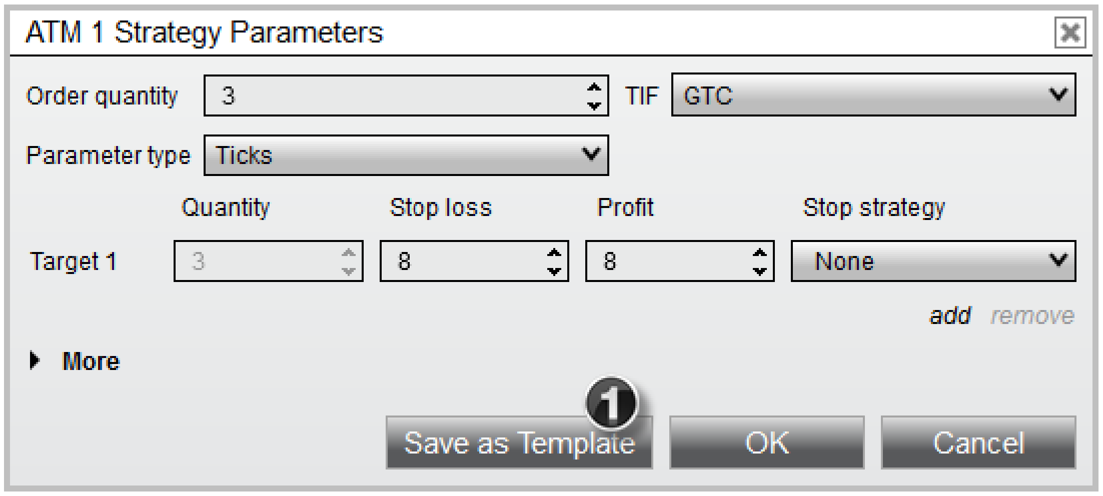
11.1.1.4 Manage ATM Strategy Templates

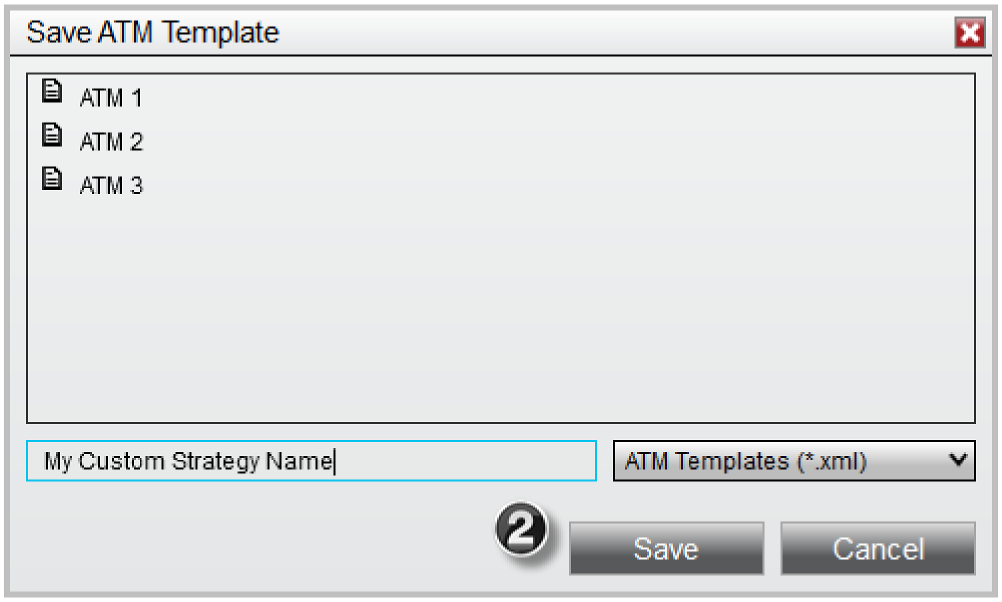
An ATM Strategy is defined by the parameters you enter into the ATM Strategy parameters section on any of the order entry screens. The collection of parameters that make up a strategy can be saved as a template that you can recall at a later date to automatically populate all of the ATM Strategy parameters.

Saving ATM Strategy Templates

To save your current **ATM Strategy parameters** in a template:

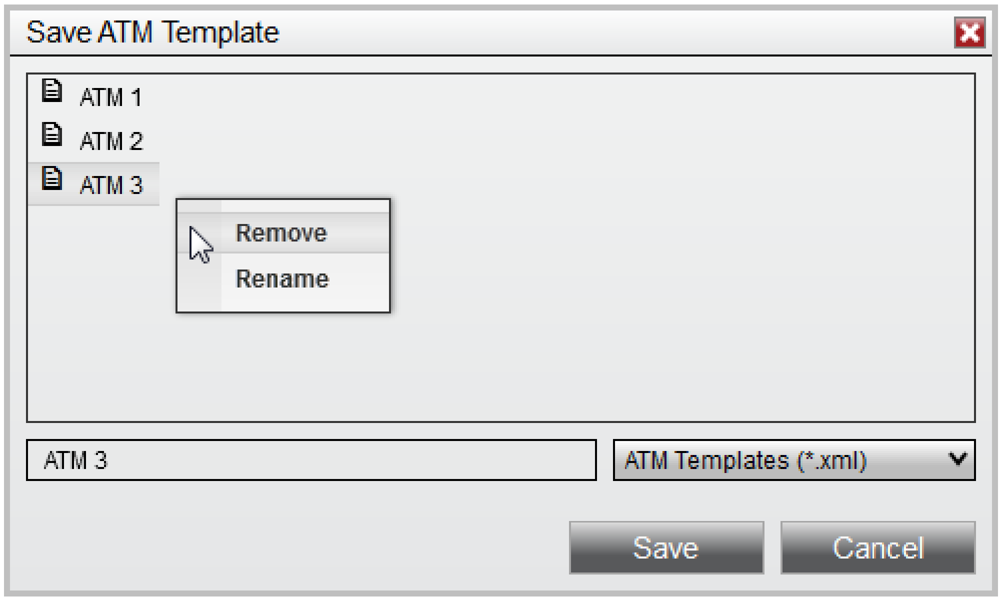
1. Select the **Save as Template** button
2. From the presented file dialog give the template a custom name





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.

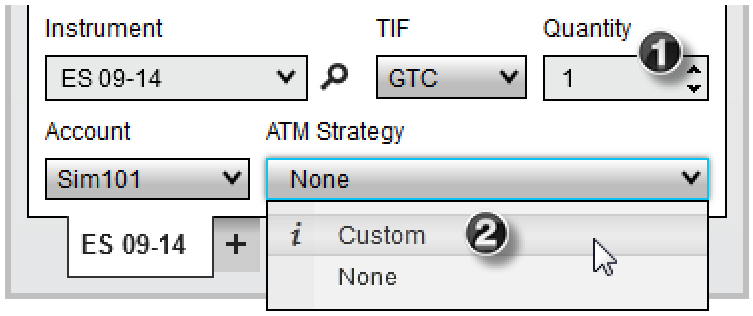


See ATM Strategy Example #1 and ATM Strategy Example #2 for further reference on how to create and save an ATM Strategy template.

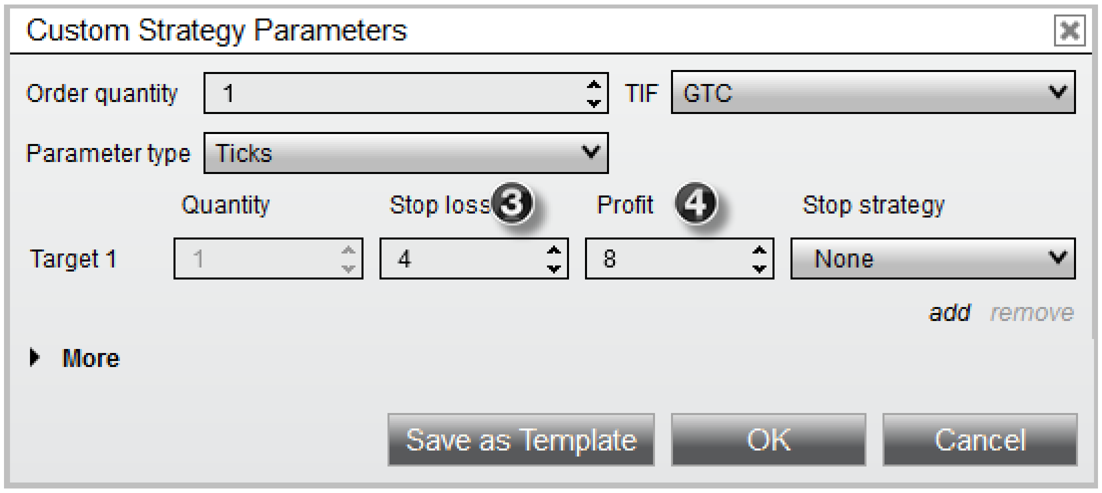
11.1.1.5 Tutorial: ATM Strategy Example #1

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)



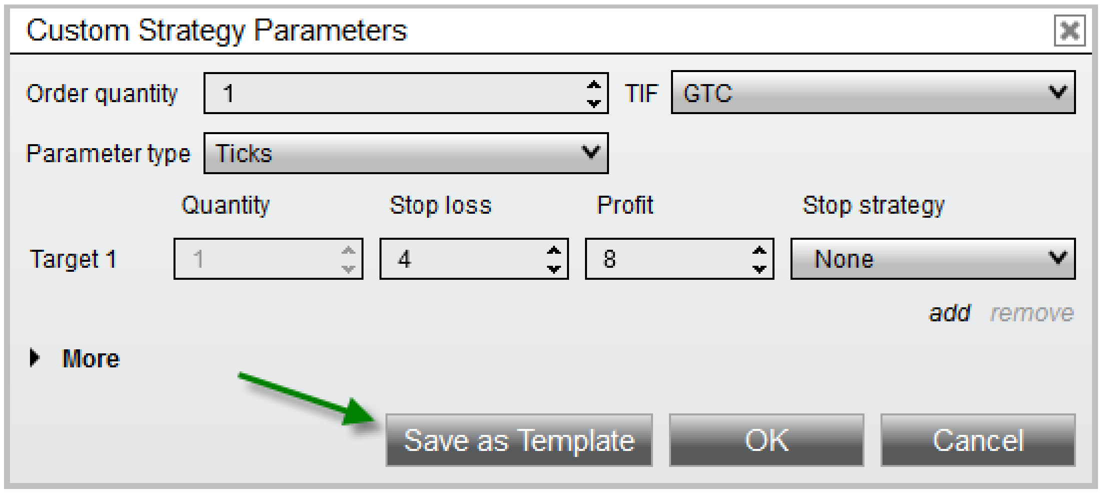
1. Set the order quantity to 1 contract
2. From the ATM Strategy control list select **<Custom>** which will open the **Custom Strategy Parameters** window

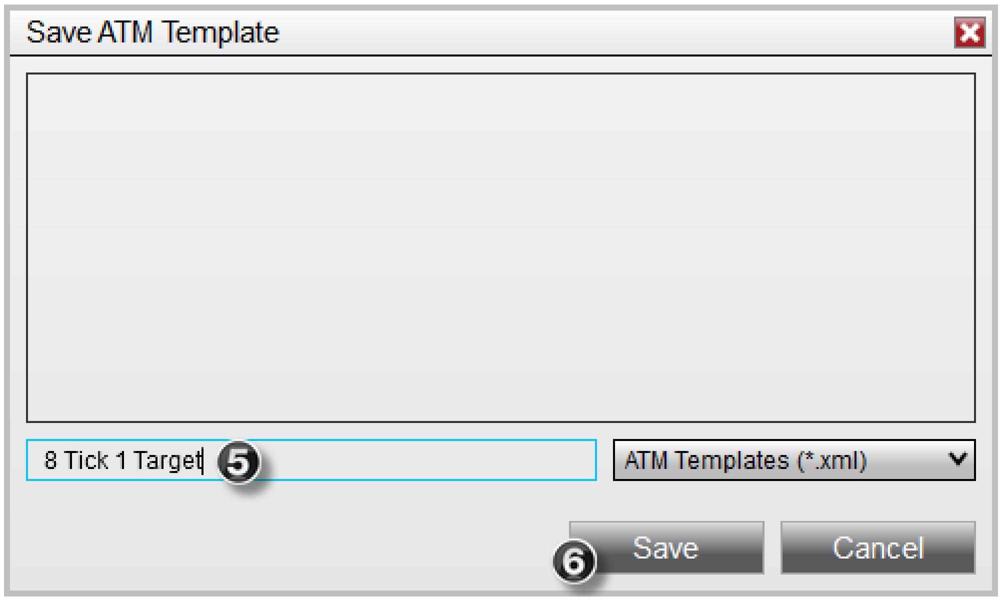


3. Set the Stop Loss value to 4 ticks
4. Set the Profit Target value to 8 ticks

This simple ATM Strategy will automatically submit a Stop Loss order 4 ticks from entry and a Profit Target order 8 ticks from entry.

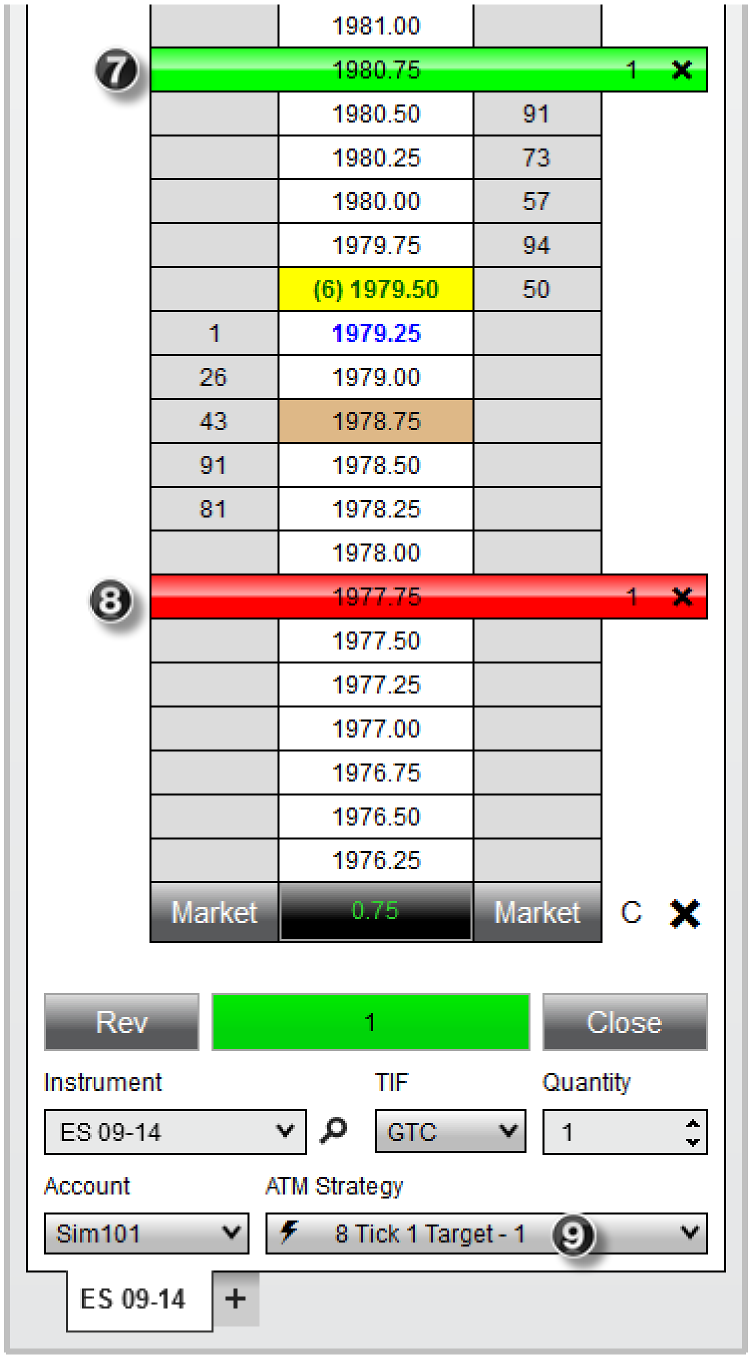
You can save this ATM Strategy as a template by clicking the **Save as Template** button from the **Custom Strategy Parameters** window





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.



	1981.00	
	1980.75	1
	1980.50	91
	1980.25	73
	1980.00	57
	1979.75	94
	(6) 1979.50	50
1	1979.25	
26	1979.00	
43	1978.75	
91	1978.50	
81	1978.25	
	1978.00	
	1977.75	1
	1977.50	
	1977.25	
	1977.00	
	1976.75	
	1976.50	
	1976.25	

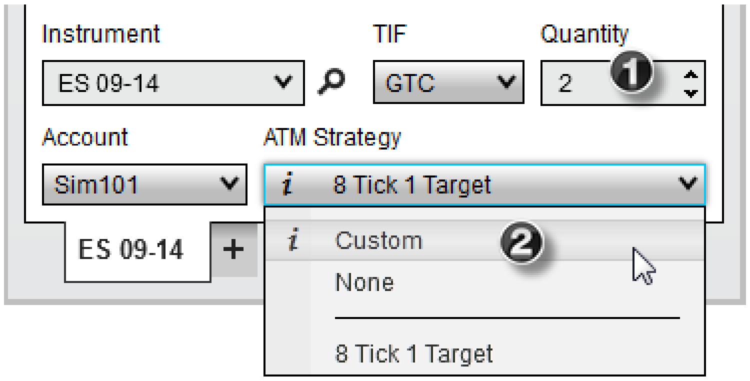
7. A Profit Target was submitted at 1980.75 which is 8 ticks from our entry price of 1978.75
8. A Stop Loss was submitted at 1977.75 which is 4 ticks from our entry price of 1978.75
9. An active strategy named "⚡ 8 Tick 1 Target - 1" is created and listed under the ATM Strategy control list.

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.

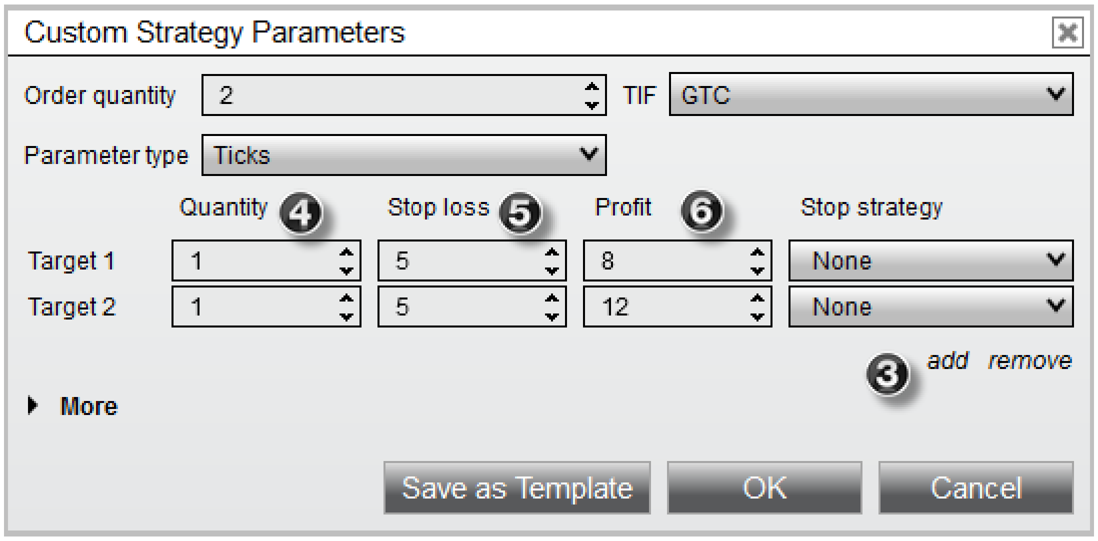
11.1.1.6 Tutorial: ATM Strategy Example #2

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.



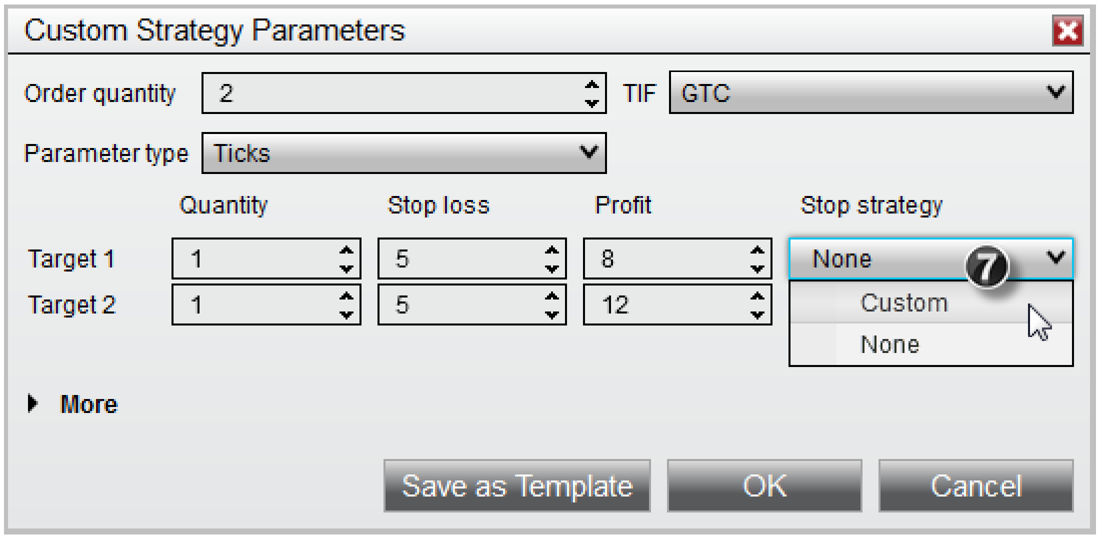
1. Set the order quantity to 2 contracts
2. From the ATM Strategy control list select **<Custom>**


Custom Strategy Parameters

Order quantity: 2
TIF: GTC
Parameter type: Ticks

	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	5	8	None
Target 2	1	5	12	None

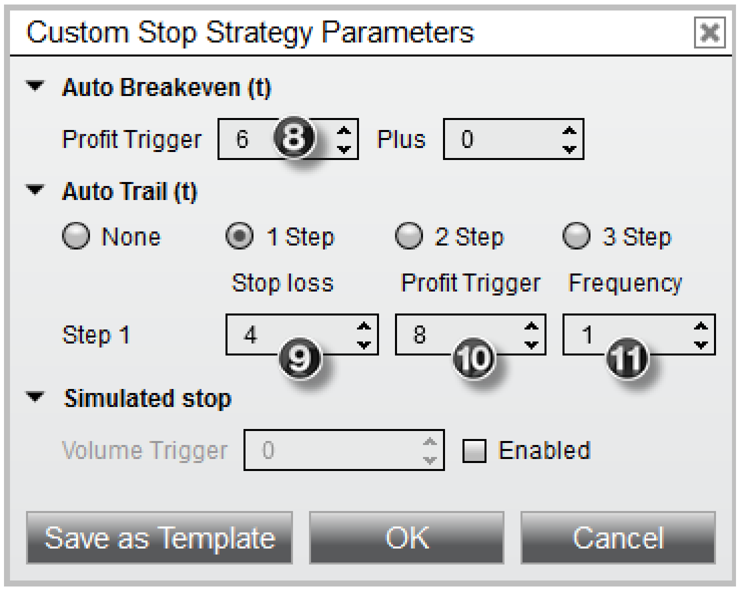
3. Select "*add*" once to enable a 2nd target
4. Set "**Quantity**" fields to 1 contract each (that represents 1 contract for the first Stop Loss/ Profit Target bracket and 1 for the 2nd)
5. Set the Stop Loss values to 5 ticks (you can set the 2nd Stop Loss to a wider value)
6. Set the first Profit Target to 8 ticks and the 2nd Profit Target to 12 ticks



Custom Strategy Parameters

Order quantity	2	TIF	GTC	
Parameter type	Ticks			
	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	5	8	None
Target 2	1	5	12	

7. Select **<Custom>** from the Stop Strategy control list under the first target.



Custom Stop Strategy Parameters

Auto Breakeven (t)

Profit Trigger 6 Plus 0

Auto Trail (t)

None [✓ 1 Step] 2 Step 3 Step

	Stop loss	Profit Trigger	Frequency
Step 1	4	8	1

Simulated stop

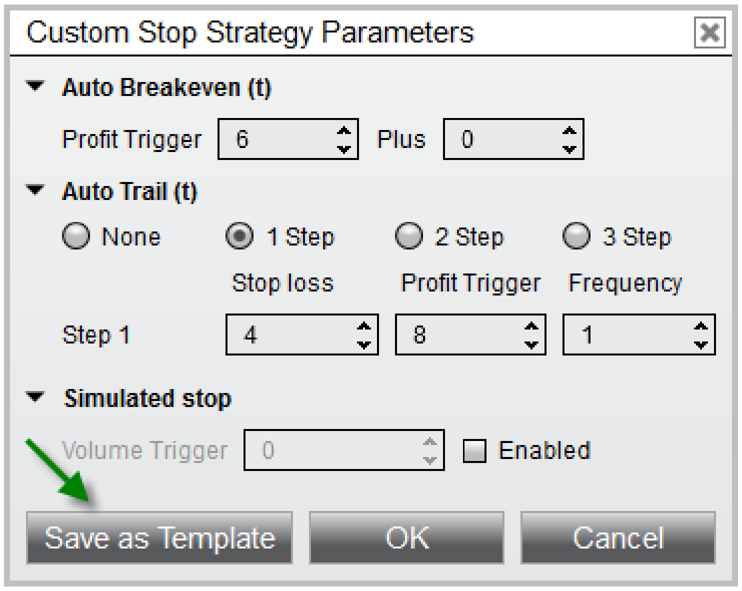
Volume Trigger 0 Enabled

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.


Custom Stop Strategy Parameters

Auto Breakeven (t)

Profit Trigger 6 Plus 0

Auto Trail (t)

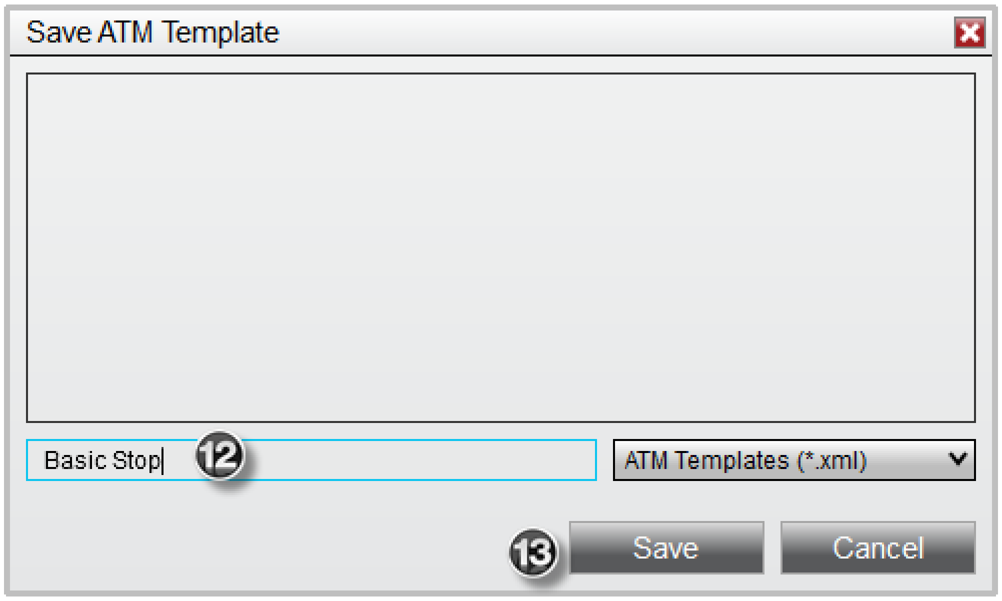
None [✓ 1 Step] 2 Step 3 Step

	Stop loss	Profit Trigger	Frequency
Step 1	4	8	1

Simulated stop

Volume Trigger 0 Enabled

You can save the Stop Strategy as a template by clicking the **Save as Template** button

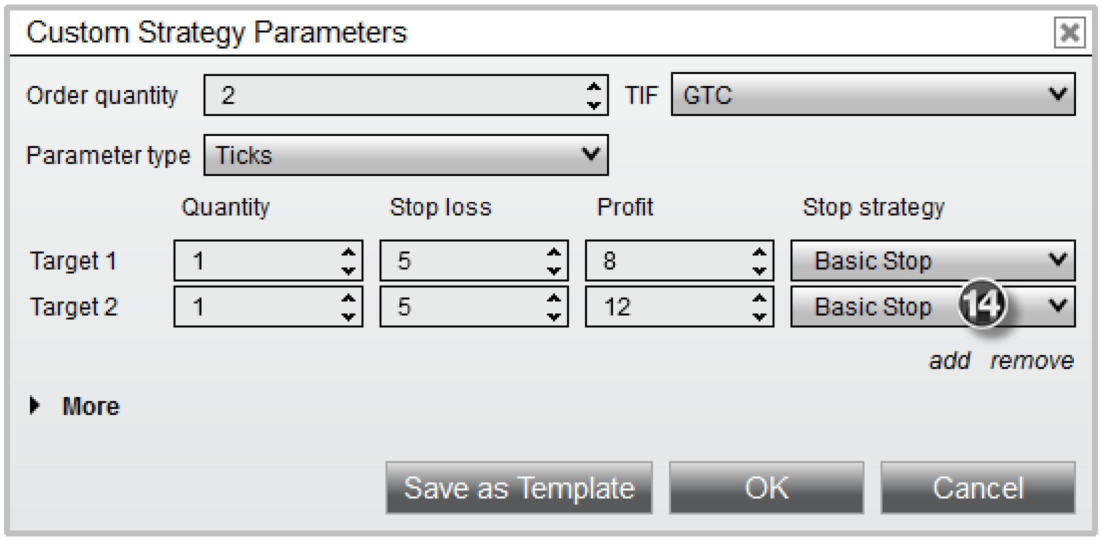

Save ATM Template

Basic Stop

ATM Templates (*.xml)

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.


Custom Strategy Parameters

Order quantity 2 TIF GTC

Parameter type Ticks

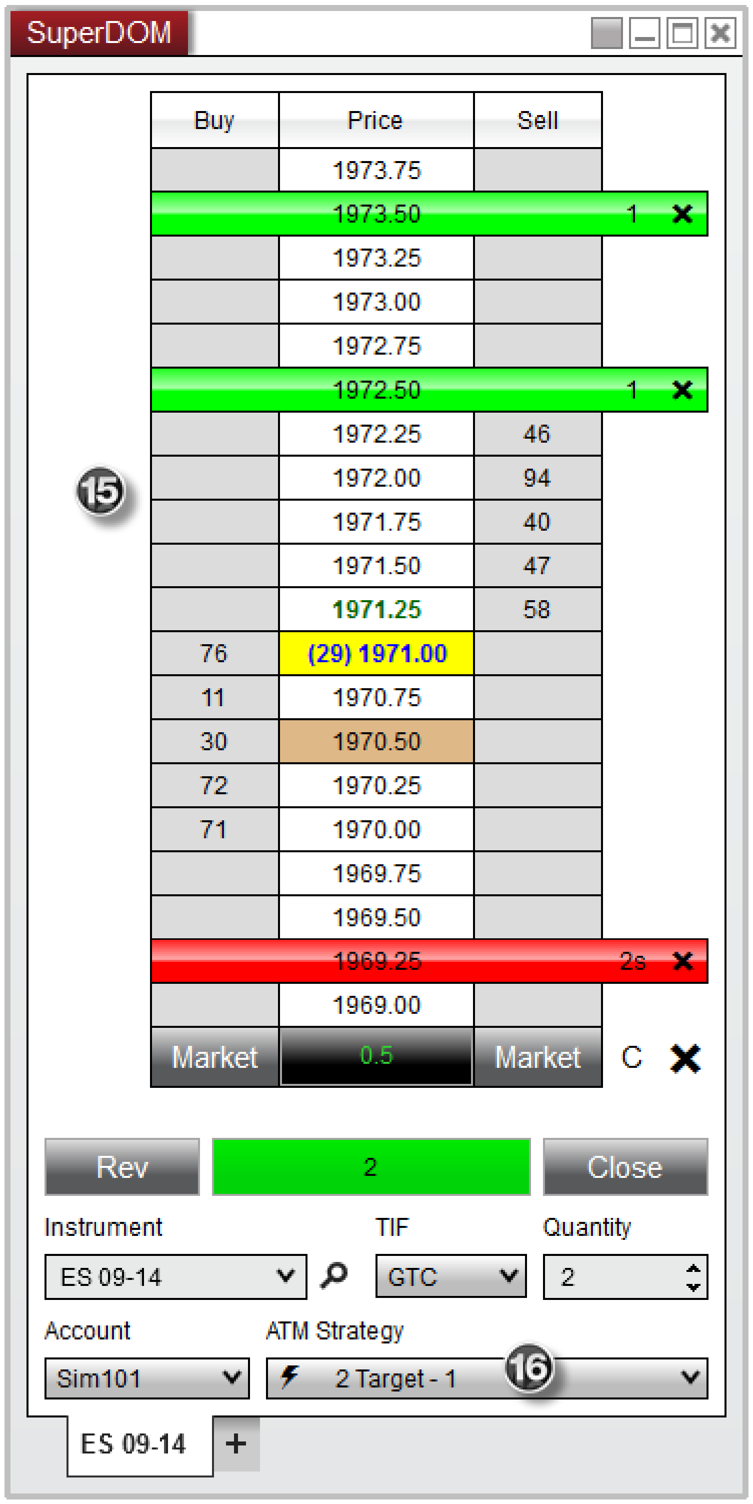
	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	5	8	Basic Stop
Target 2	1	5	12	Basic Stop

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.



Buy	Price	Sell	
	1973.75		
	1973.50		1 ✕
	1973.25		
	1973.00		
	1972.75		
	1972.50		1 ✕
	1972.25	46	
	1972.00	94	
	1971.75	40	
	1971.50	47	
	1971.25	58	
76	(29) 1971.00		
11	1970.75		
30	1970.50		
72	1970.25		
71	1970.00		
	1969.75		
	1969.50		
	1969.25		2s ✕
	1969.00		
Market	0.5	Market	C ✕

Rev | 2 | Close

Instrument: ES 09-14
TIF: GTC
Quantity: 2
Account: Sim101
ATM Strategy: 2 Target - 1

ES 09-14 +

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

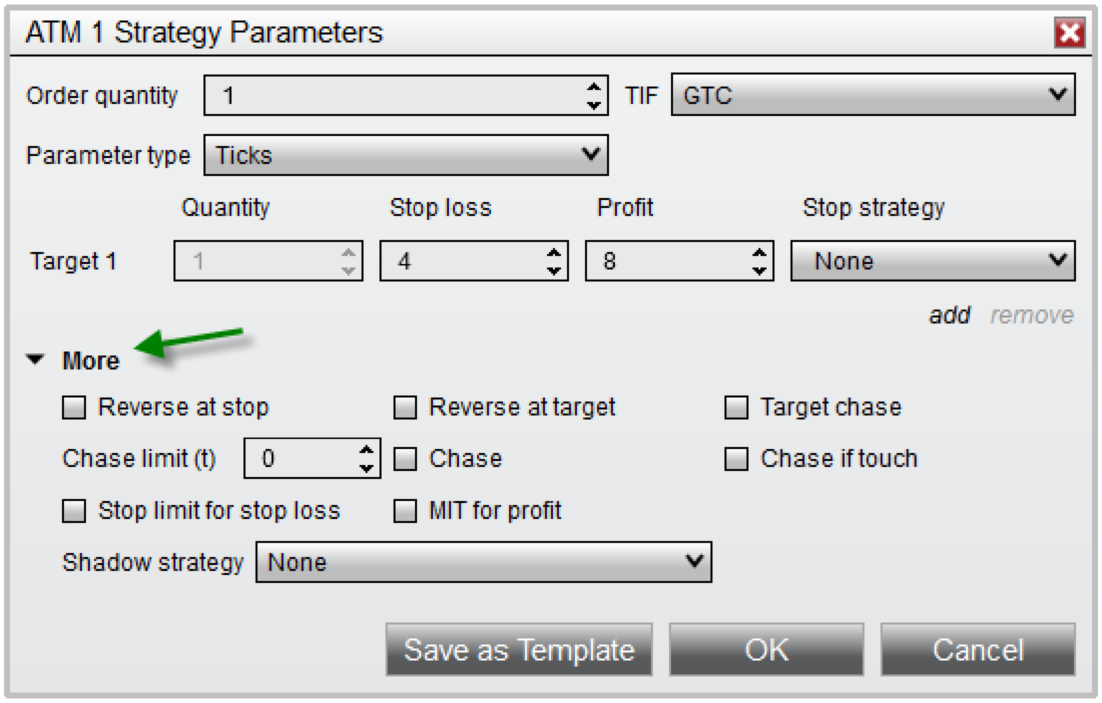
we have multiple orders consolidated at the price 1969.25.

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.

11.1.2 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.


ATM 1 Strategy Parameters

Order quantity: 1 | TIF: GTC

Parameter type: Ticks

	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	4	8	None

add remove

More

- ☐ Reverse at stop
- ☐ Reverse at target
- ☐ Target chase
- Chase limit (t): 0
- ☐ Chase
- ☐ Chase if touch
- ☐ Stop limit for stop loss
- ☐ MIT for profit
- Shadow strategy: None

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, as these markets do not support Stop Limit orders. If this property is enabled when trading a stock or forex instrument, Stop Market orders will be used, instead.

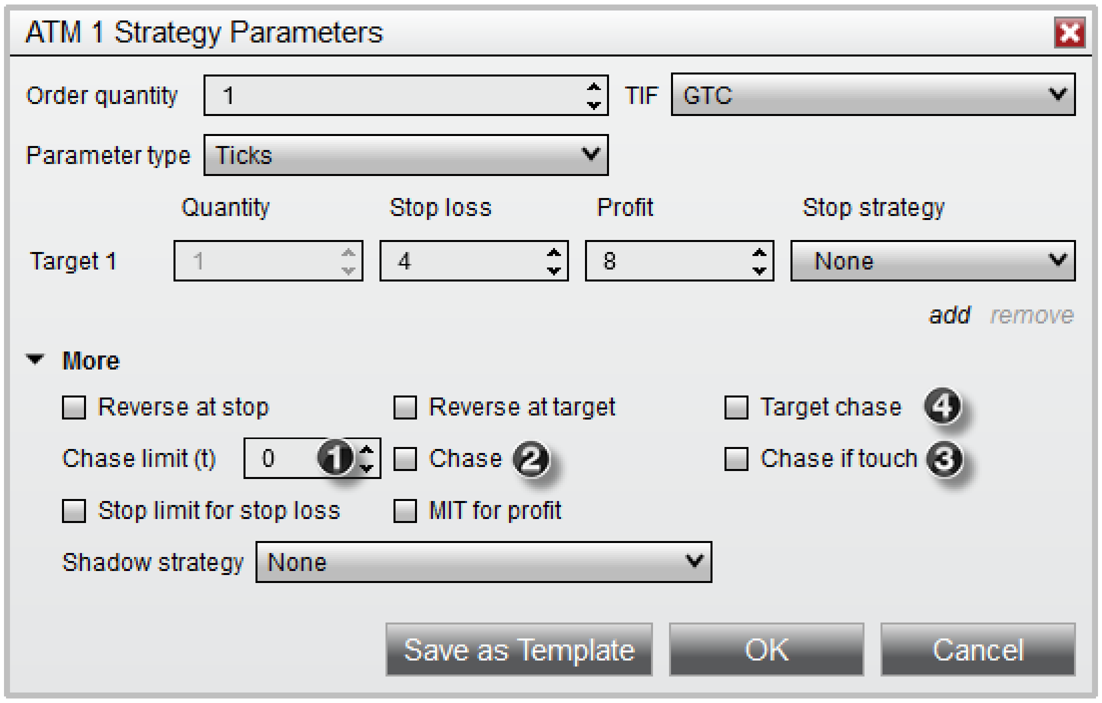
11.1.2.1 Auto Chase

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


ATM 1 Strategy Parameters

Order quantity: 1 | TIF: GTC

Parameter type: Ticks

	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	4	8	None

More

- [ ] Reverse at stop
- [ ] Reverse at target
- [ ] Target chase 4
- Chase limit (t): 0 1
- [ ] Chase 2
- [ ] Chase if touch 3
- [ ] Stop limit for stop loss
- [ ] MIT for profit

Shadow strategy: None

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.

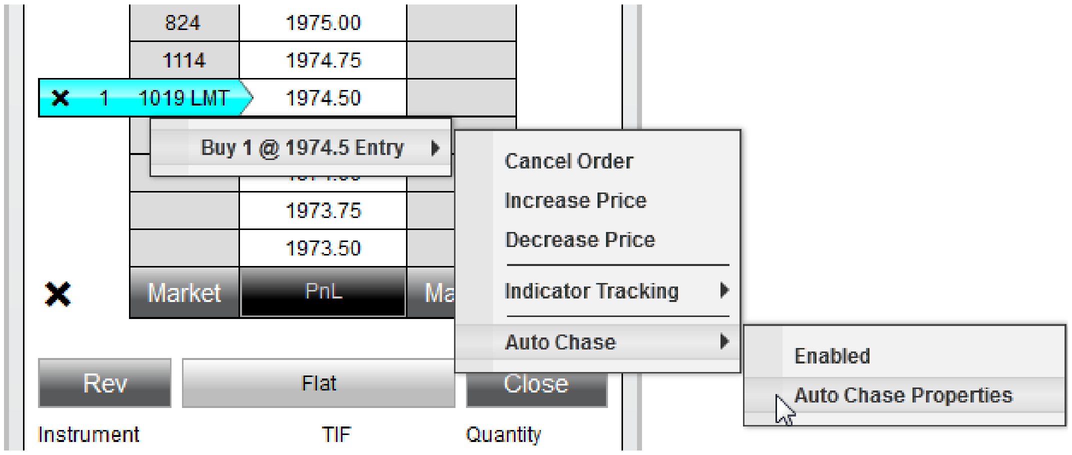
How does Chase if touched work?

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.

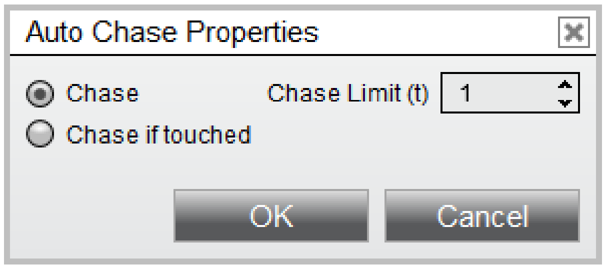
How to enable the Auto Chase features

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.



[✓ Chase] Chase if touched

Chase Limit (t) 1

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.

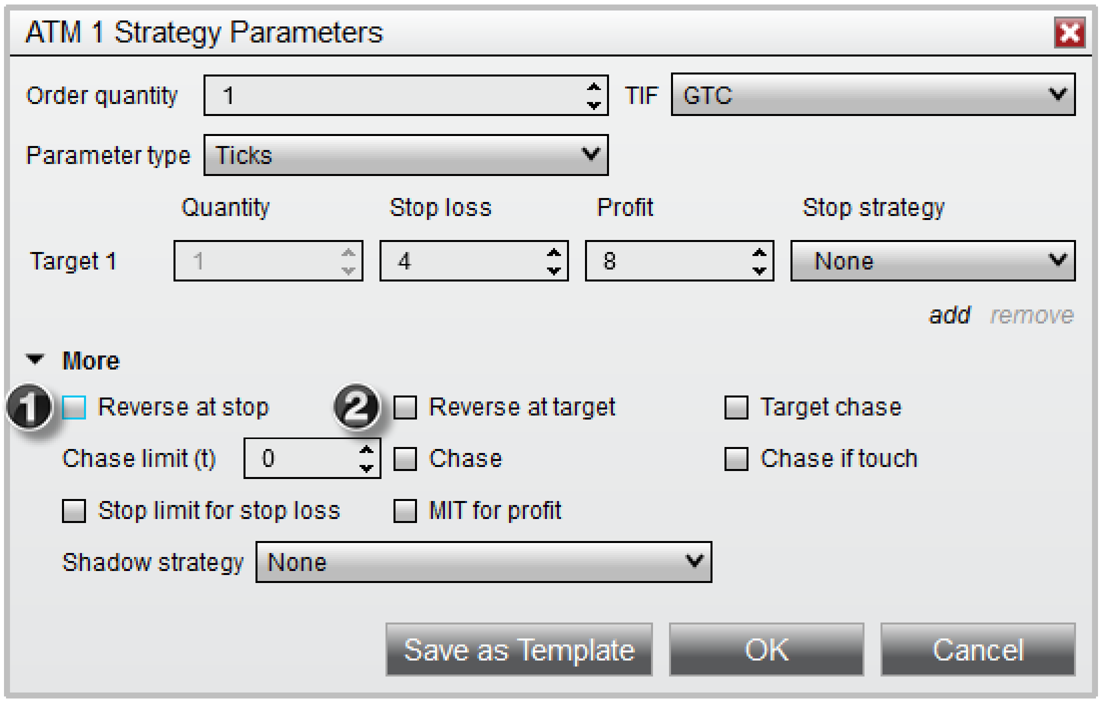
Auto Chase Example #2

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.

11.1.2.2 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.


ATM 1 Strategy Parameters

Order quantity: 1 — TIF: GTC

Parameter type: Ticks

	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	4	8	None

add remove

More

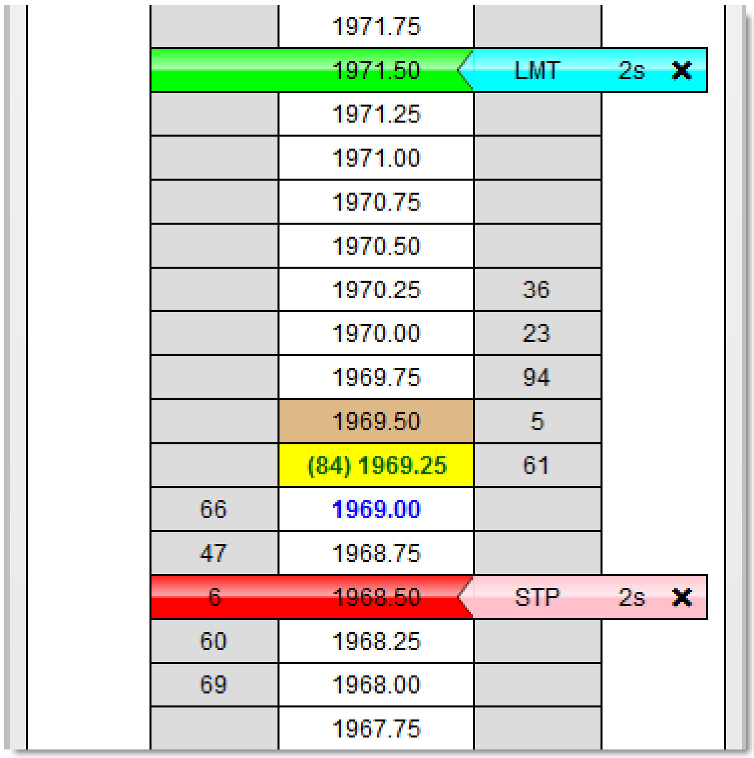
1 ☐ Reverse at stop — 2 ☐ Reverse at target — ☐ Target chase

Chase limit (t): 0 — ☐ Chase — ☐ Chase if touch

☐ Stop limit for stop loss — ☐ MIT for profit

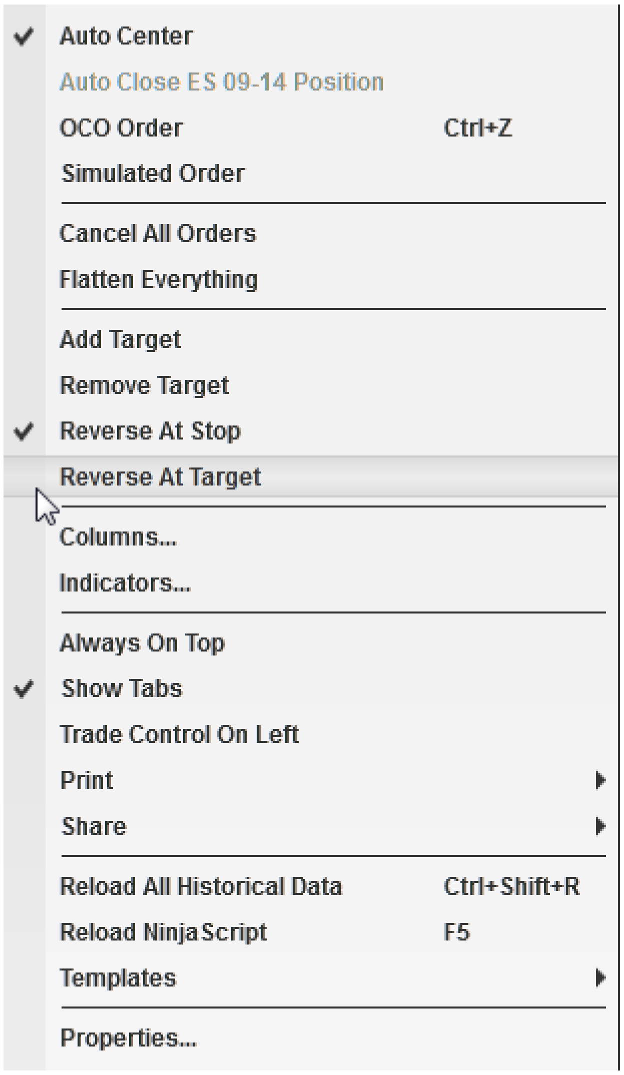
Shadow strategy: None

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.



	1971.75		
	1971.50	LMT 2s ✕	
	1971.25		
	1971.00		
	1970.75		
	1970.50		
	1970.25	36	
	1970.00	23	
	1969.75	94	
	1969.50	5	
	(84) 1969.25	61	
66	1969.00		
47	1968.75		
6	1968.50	STP 2s ✕	
60	1968.25		
69	1968.00		
	1967.75		

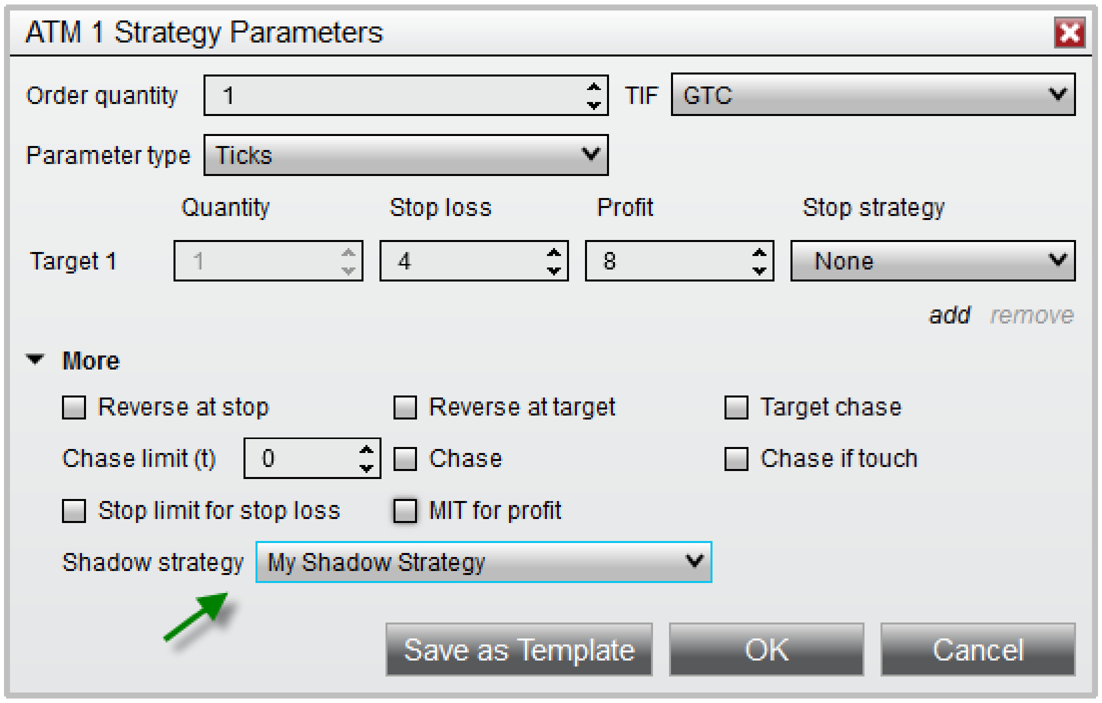
Modifying the price of either your Stop Loss or Profit Target will result in the modification of the reverse order as well. You can also enable or disable Auto Reverse of an active ATM Strategy at any time by selecting the "**Reverse At Stop**" or "**Reverse At Target**" menus via the right mouse click context menu in either the SuperDOM or Basic Entry windows.



11.1.2.3 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.


ATM 1 Strategy Parameters

Order quantity: 1
TIF: GTC
Parameter type: Ticks

	Quantity	Stop loss	Profit	Stop strategy
Target 1	1	4	8	None

add remove

More

- [ ] Reverse at stop
- [ ] Reverse at target
- [ ] Target chase
- Chase limit (t): 0
- [ ] Chase
- [ ] Chase if touch
- [ ] Stop limit for stop loss
- [ ] MIT for profit

Shadow strategy: My Shadow Strategy

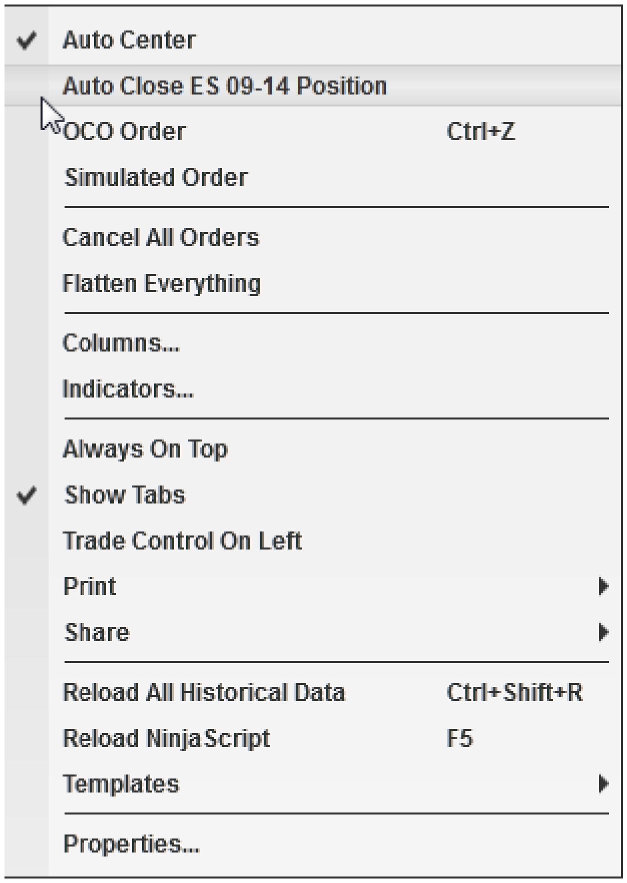
Tips

- 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

11.1.3 Auto Close Position

Automatically Closing Positions at a Specific Time

Auto Close Position is a strategy that will automatically close your position at a 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.



Note: This feature not available to Direct Edition license users and will be disabled. Please contact sales@ninjatrader.com for upgrade options.

11.1.4 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.

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.

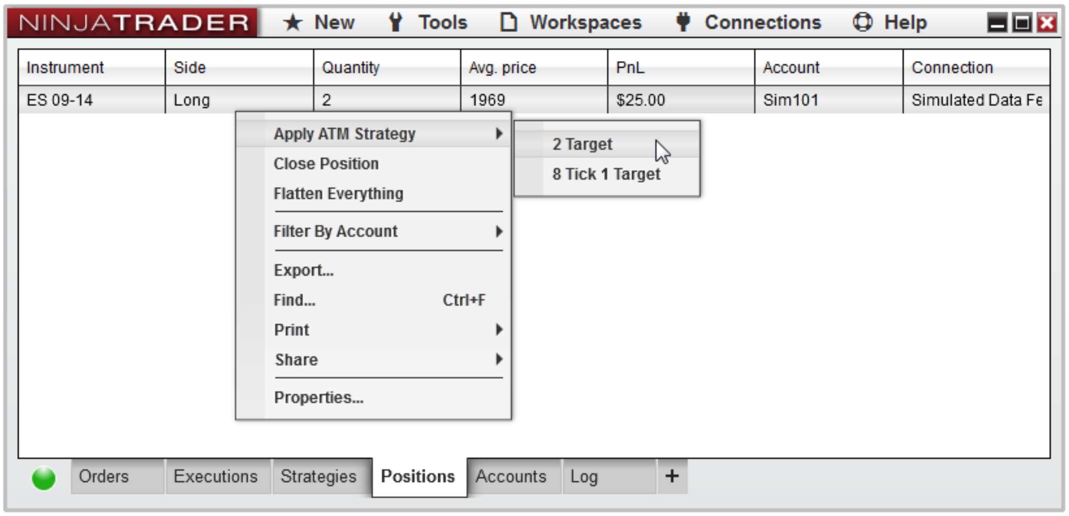
Can I use the Auto Breakeven and Auto Trail strategies together?

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.



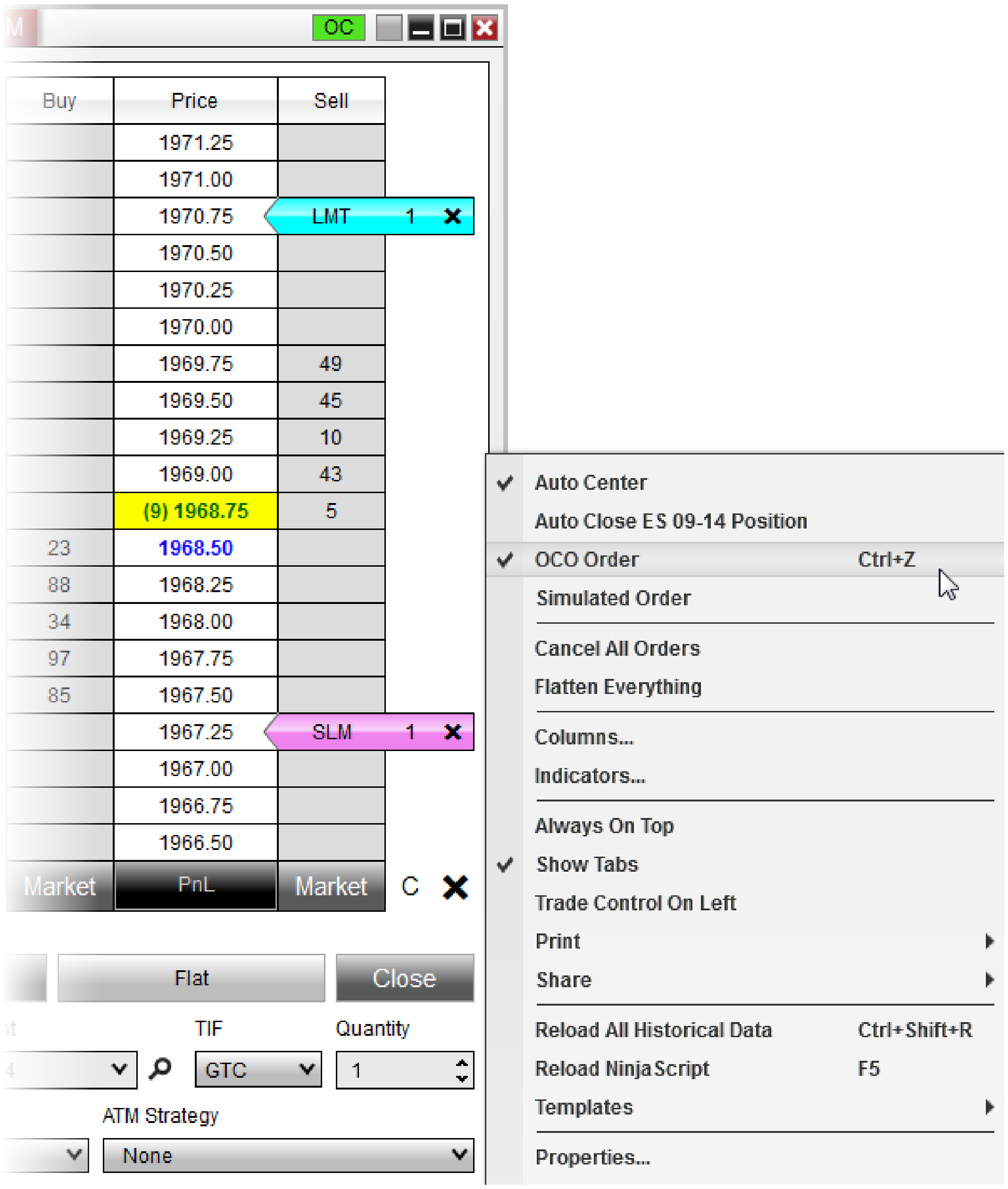
Instrument	Side	Quantity	Avg. price	PnL	Account	Connection
ES 09-14	Long	2	1969	$25.00	Sim101	Simulated Data Fe

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.



Buy	Price	Sell
	1971.25	
	1971.00	
	1970.75	LMT 1
	1970.50	
	1970.25	
	1970.00	
	1969.75	49
	1969.50	45
	1969.25	10
	1969.00	43
	(9) 1968.75	5
23	1968.50	
88	1968.25	
34	1968.00	
97	1967.75	
85	1967.50	
	1967.25	SLM 1
	1967.00	
	1966.75	
	1966.50	
Market	PnL	Market

TIF: GTC
Quantity: 1
ATM Strategy: None

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, Stop Loss, or Profit Target order is rejected, the other will be canceled as a result.

11.2 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 8 are completely achieved through a point and click interface, requiring no programming experience of any kind!

› Alerts Dialog
› Configuring Alerts
› Condition Builder

11.2.1 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*

Note: 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.

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.

11.2.2 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**

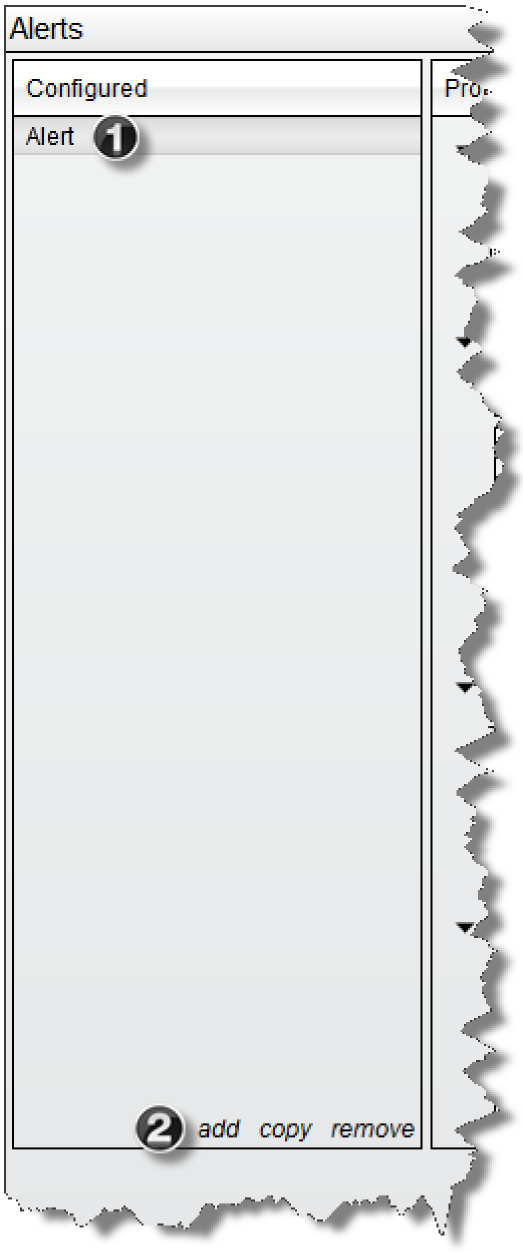
An **Alert Dialog** launched from the **Chart** will list any chart objects (data series, indicators, drawing tools) you have configured on your Chart to be used as alert condition objects.

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

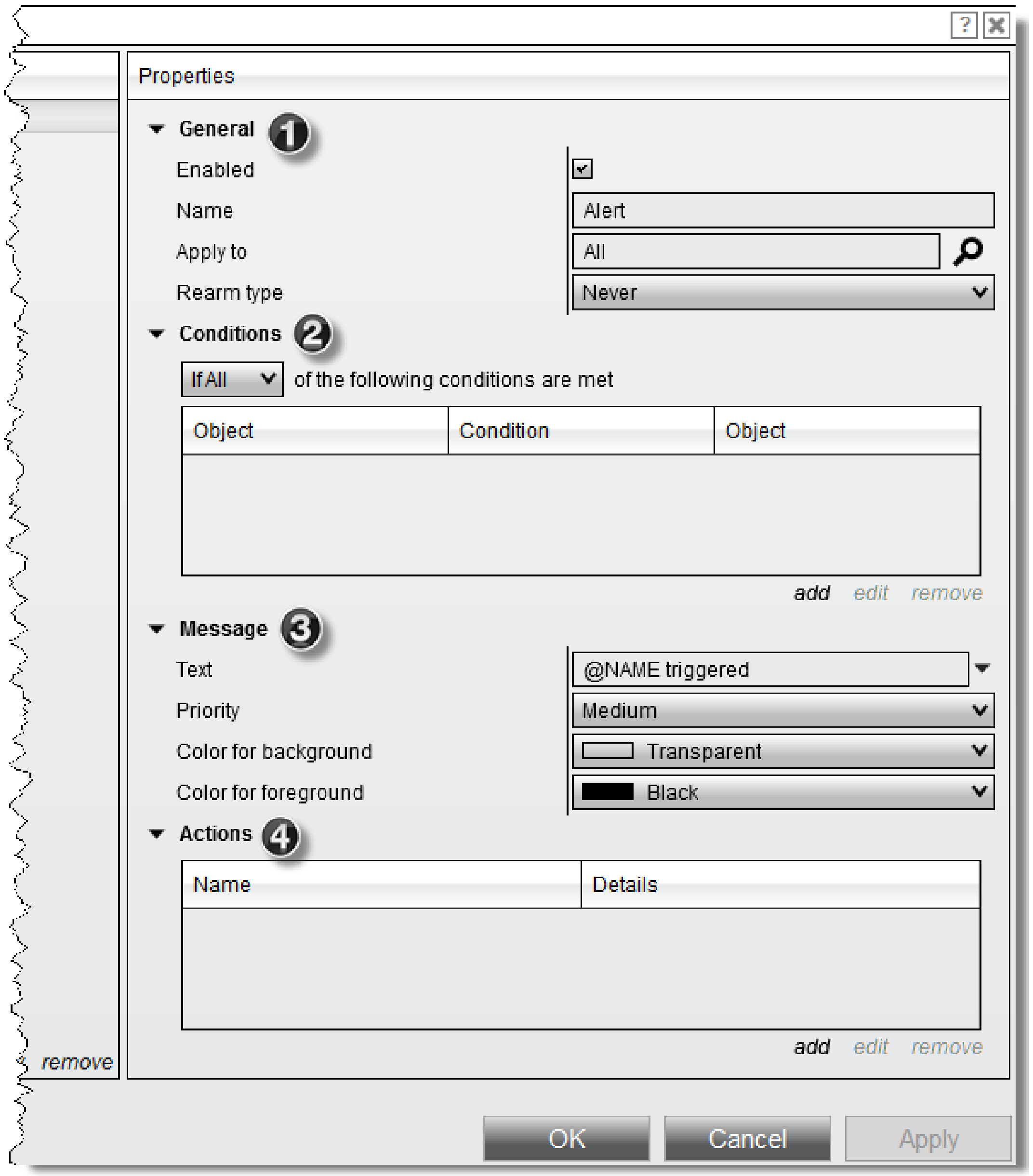


Properties Panel

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



Properties

General 1

Enabled	✓
Name	Alert
Apply to	All
Rearm type	Never

Conditions 2

If All of the following conditions are met

Object	Condition	Object

Message 3

Text	@NAME triggered
Priority	Medium
Color for background	Transparent
Color for foreground	Black

Actions 4

Name	Details

11.2.3 Configuring Alerts

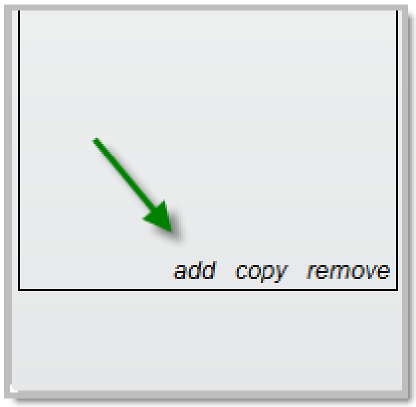
Alerts can be created using conditions which monitor various "objects" which exist on the chart, or market analyzer display. Possible **Condition Objects** include a chart's data series, indicators, drawing tools, or any Market Analyzer column value.

Play Video

▽ How to add a condition object

Adding an alert from the alert dialog

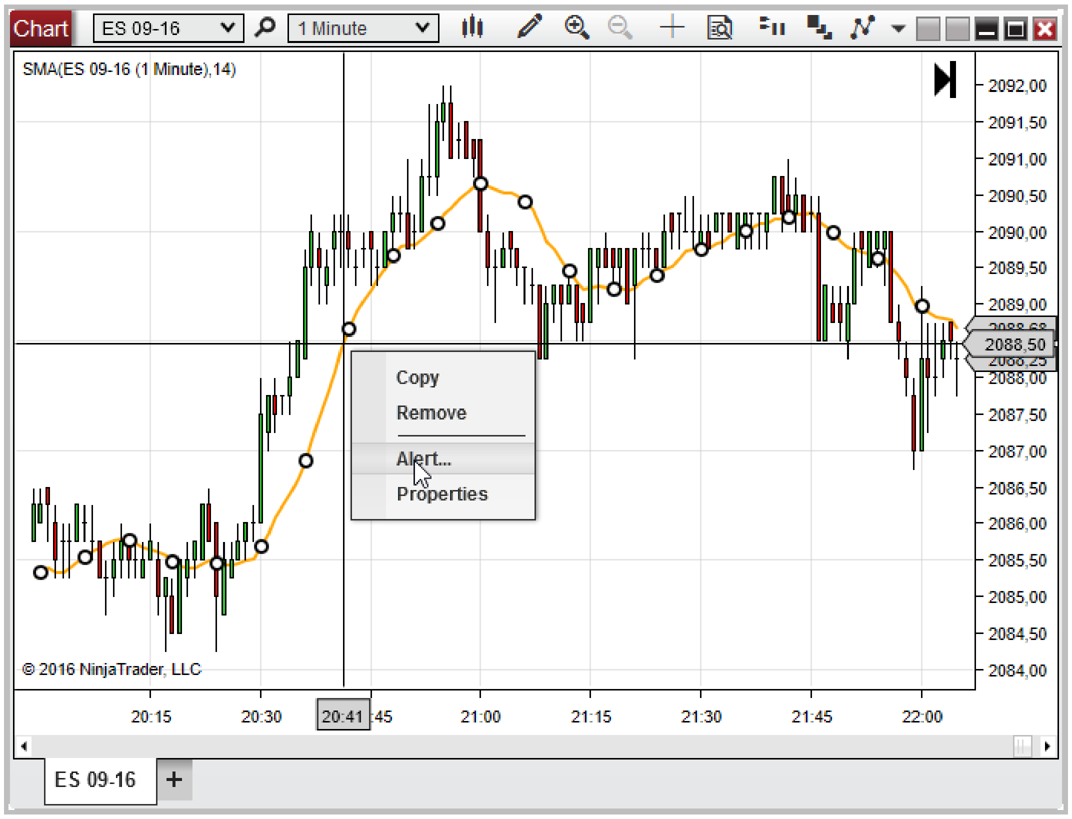
You can create a new generic alert by first accessing the Alerts Dialog window, and selecting the **"add"** text which will add a new alert to your configured alerts panel.



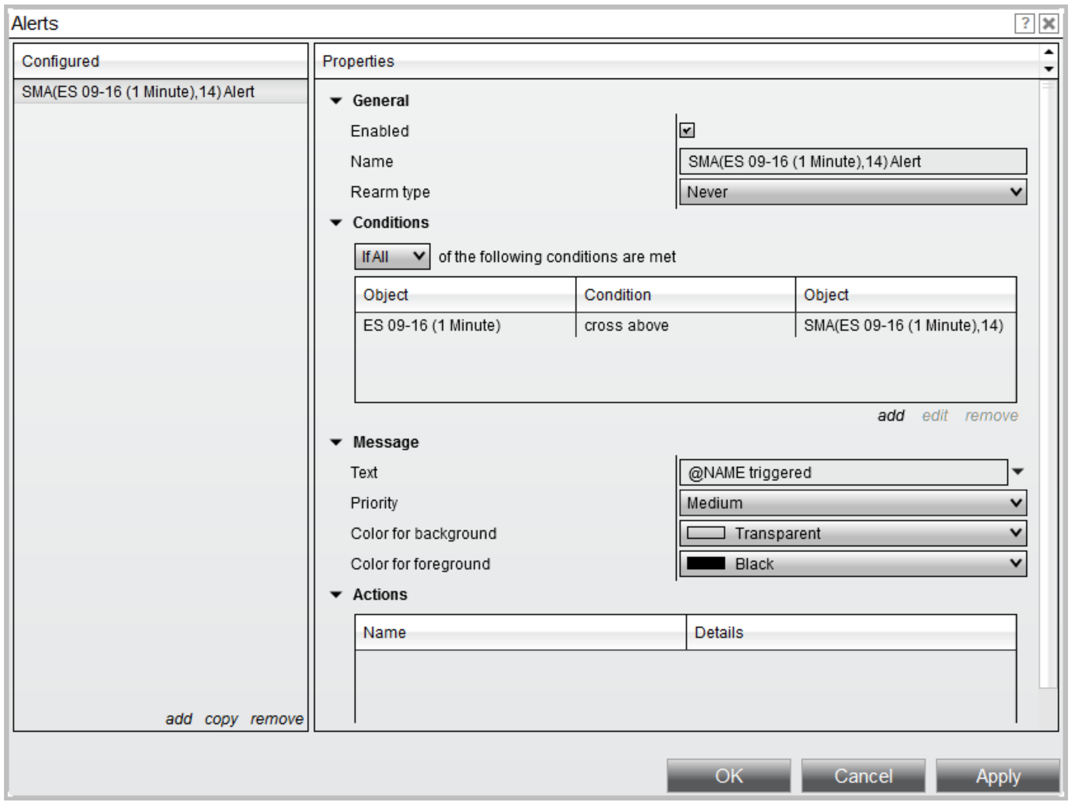
Any suitable object which currently exists on the window or tab will be available to use as a **Condition Object** for the alert. For example, if you have a **Market Analyzer** with several customized columns added, you will be able to use any of those columns as a **Condition Object**. A **Chart** will 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.



SMA(ES 09-16 (1 Minute),14) Alert

General

Enabled	✓
Name	SMA(ES 09-16 (1 Minute),14) Alert
Rearm type	Never

Conditions

If All of the following conditions are met

Object	Condition	Object
ES 09-16 (1 Minute)	cross above	SMA(ES 09-16 (1 Minute),14)

Message

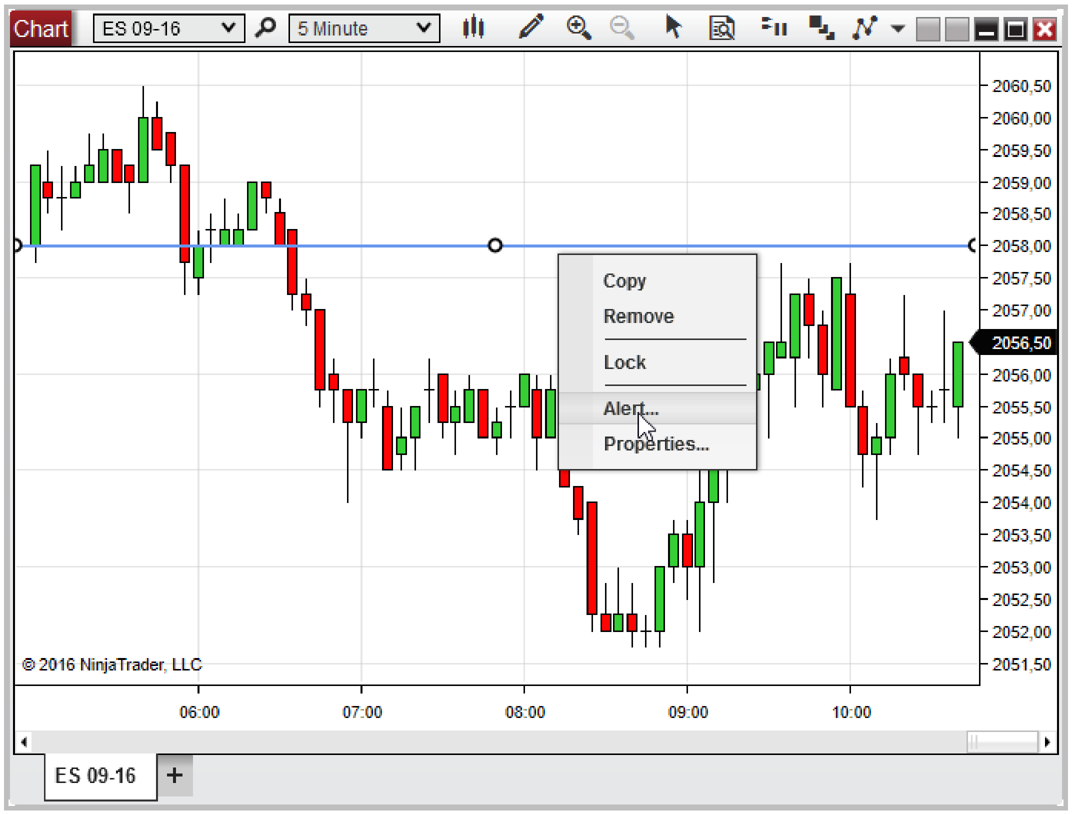
Text	@NAME triggered
Priority	Medium
Color for background	Transparent
Color for foreground	Black

Actions

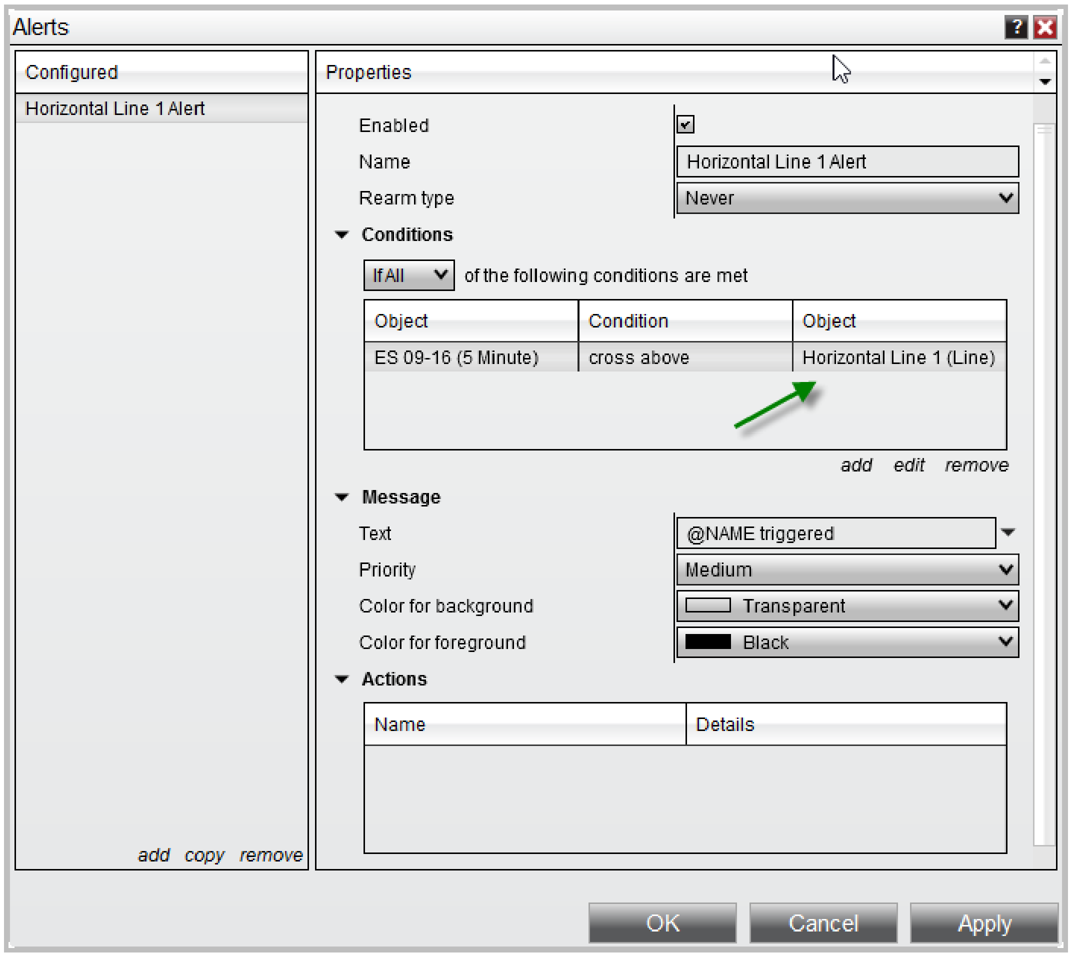
Name	Details

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.


Alerts

Configured
Horizontal Line 1 Alert

Properties

Enabled ☑
Name: Horizontal Line 1 Alert
Rearm type: Never

Conditions

If All — of the following conditions are met

Object	Condition	Object
ES 09-16 (5 Minute)	cross above	Horizontal Line 1 (Line)

Message

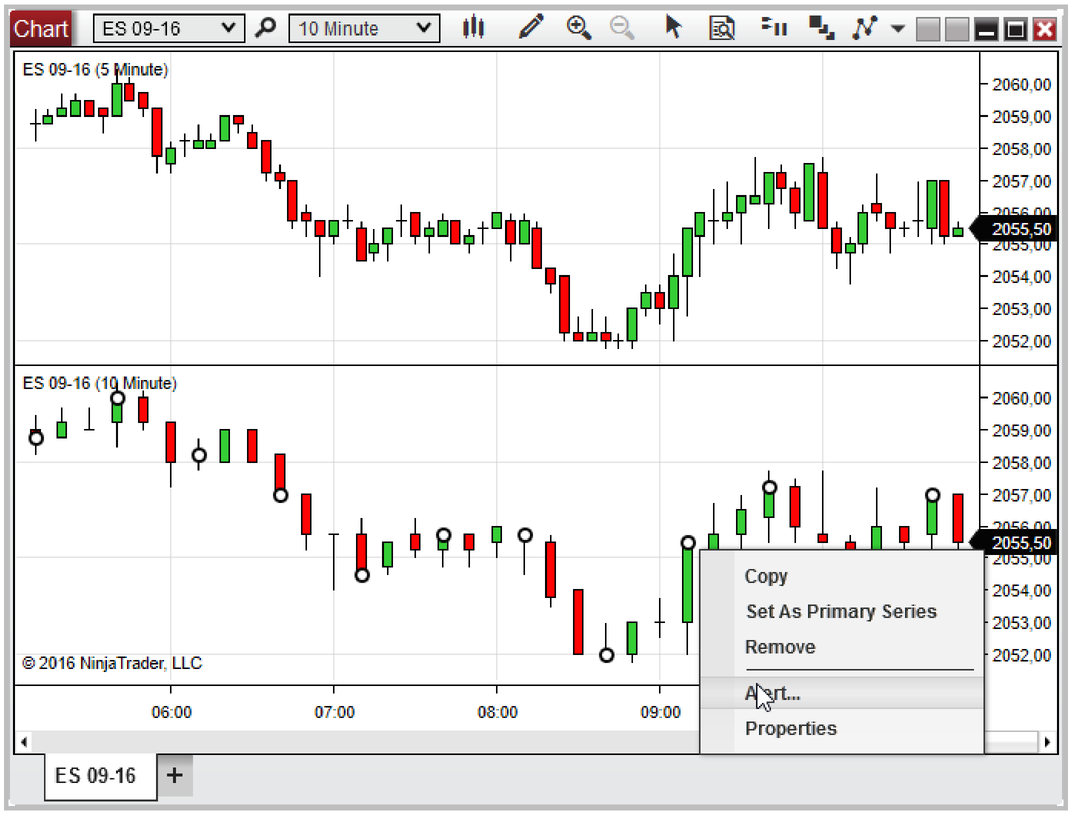
Text: @NAME triggered
Priority: Medium
Color for background: Transparent
Color for foreground: Black

Actions

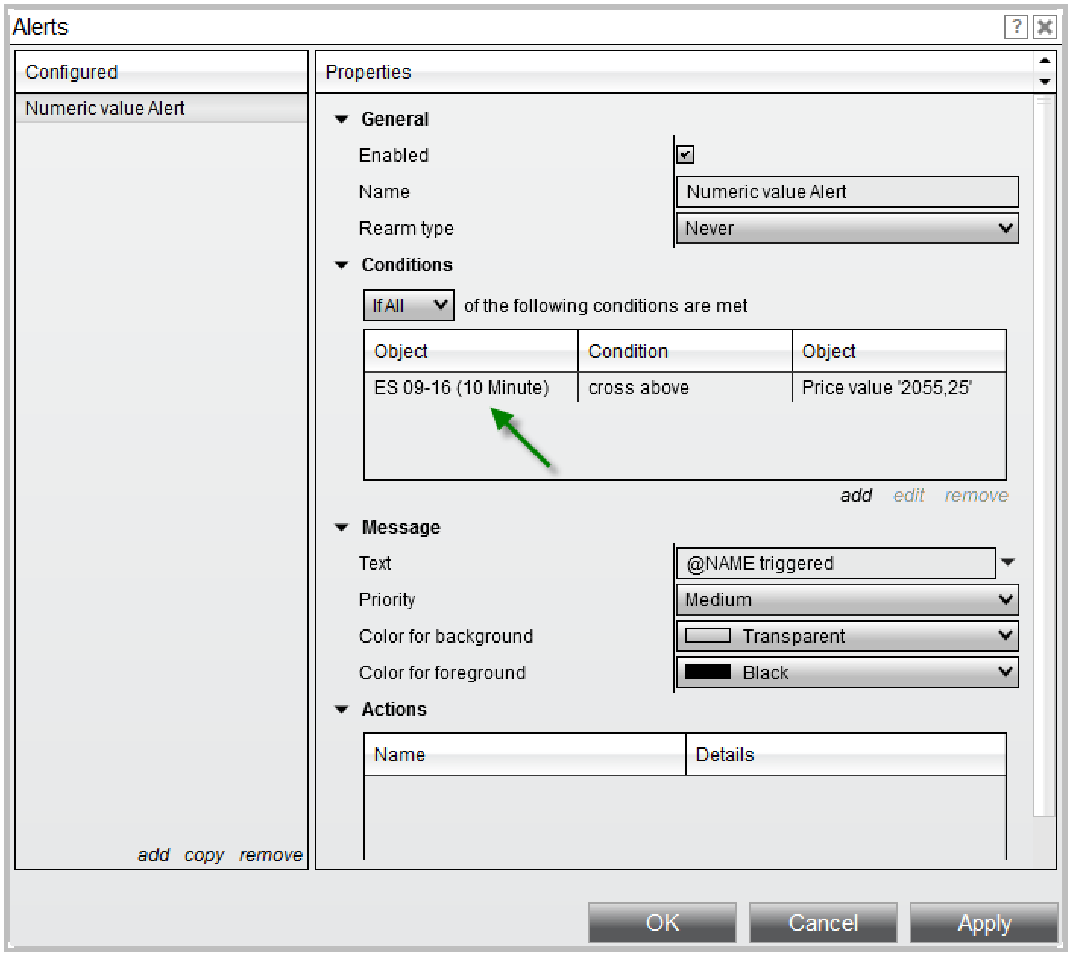
Name	Details

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.


Alerts

Configured

Numeric value Alert

Properties

General

Enabled: ✓

Name: Numeric value Alert

Rearm type: Never

Conditions

If All of the following conditions are met

Object	Condition	Object
ES 09-16 (10 Minute)	cross above	Price value '2055,25'

Message

Text: @NAME triggered

Priority: Medium

Color for background: Transparent

Color for foreground: Black

Actions

Name	Details

Understand the general alert properties

General Alert Properties

The **General** section allow you to configure the following alert properties:

Enable d	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.
Rearm	Sets under what condition the alert will rearm.

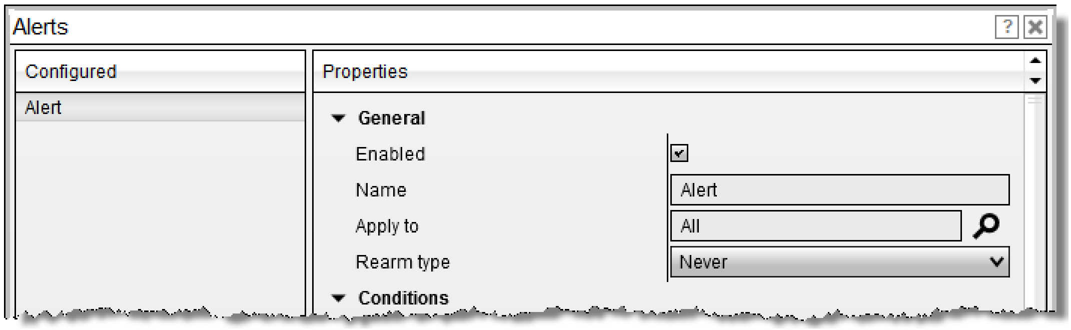
type	Possible options are listed in the table below: Never: The alert will only trigger once and never rearm On timer: Rearm after a specific number of seconds defined in the rearm seconds property On bar close: Rearm on the next bar close for the data series used in the alert condition On condition reversed: Rearm once the condition is no longer true On connect*: Rearm after NinjaTrader has been manually connected to a data provider Remove: Once triggered the alert will never rearm and be removed.
Rearm seconds	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)

Never	The alert will only trigger once and never rearm
On timer	Rearm after a specific number of seconds defined in the rearm seconds property
On bar close	Rearm on the next bar close for the data series used in the alert condition
On condition reversed	Rearm once the condition is no longer true
On connect*	Rearm after NinjaTrader has been manually connected to a data provider
Remove	Once triggered the alert will never rearm and be removed.

*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	Alert will be disabled

	down NinjaTrader	
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


Alerts

Configured: Alert

Properties

General
- Enabled: ✓
- Name: Alert
- Apply to: All
- Rearm type: Never

Conditions

Applying alerts to specific instruments

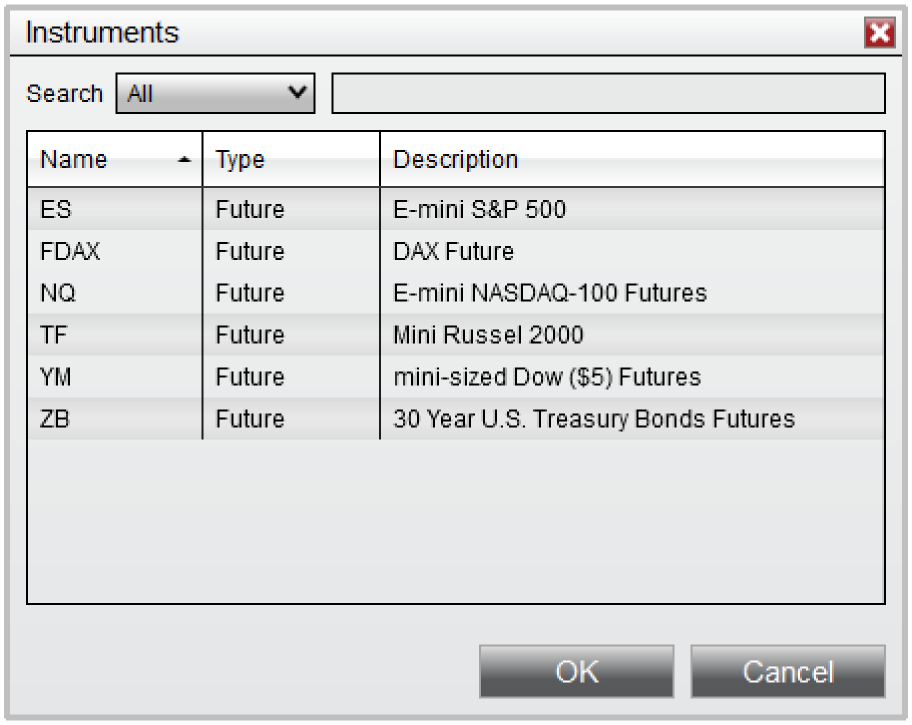
If you are using a window which has multiple instruments, such as a **Market Analyzer**, or a **Chart** with multiple data series, the default behavior will be monitor "**All**" instruments contained in that window tab.

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

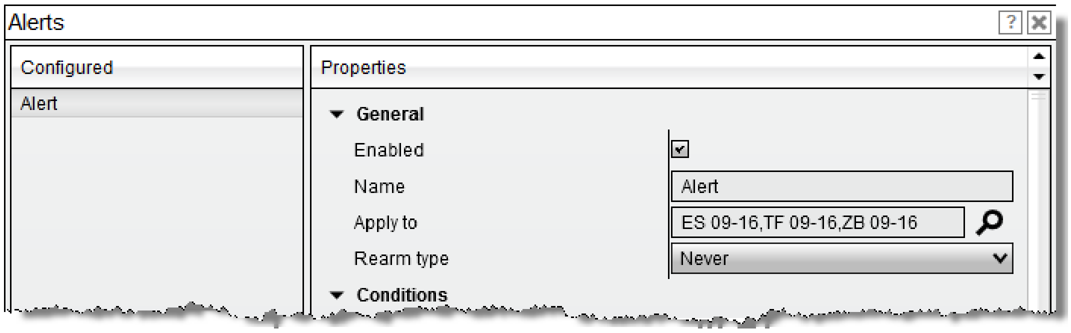
hold down the Shift key, and then click the last instrument.
- To select non-consecutive instruments, press and hold down the Ctrl key, and then click each instrument that you want to select.



Name	Type	Description
ES	Future	E-mini S&P 500
FDAX	Future	DAX Future
NQ	Future	E-mini NASDAQ-100 Futures
TF	Future	Mini Russel 2000
YM	Future	mini-sized Dow ($5) Futures
ZB	Future	30 Year U.S. Treasury Bonds Futures

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.



Enabled	✓
Name	Alert
Apply to	ES 09-16,TF 09-16,ZB 09-16
Rearm type	Never

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

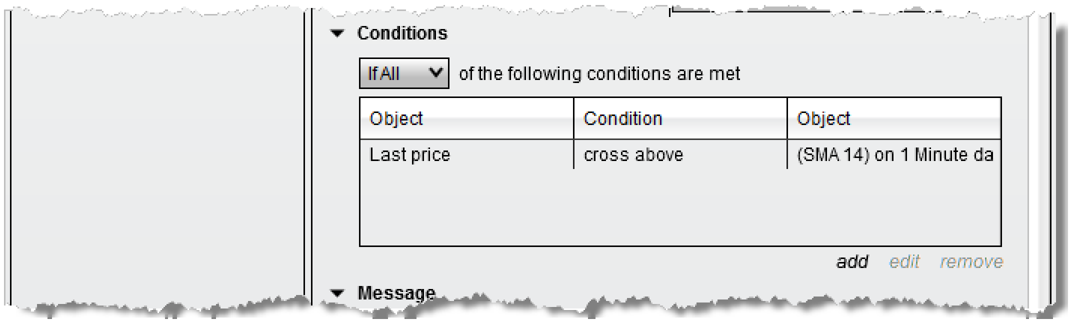
Alert Conditions

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



Object	Condition	Object
Last price	cross above	(SMA 14) on 1 Minute da

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

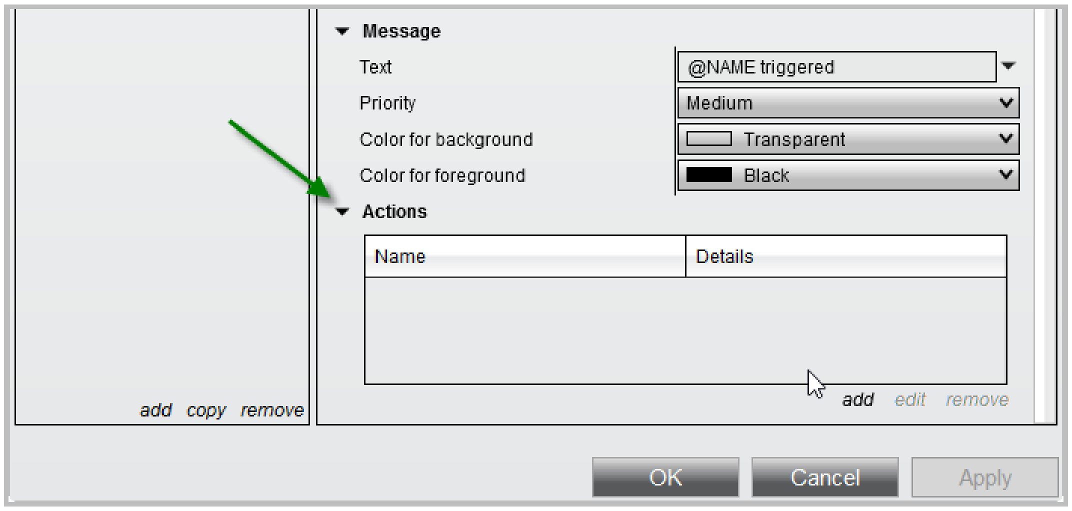
When an alert has been triggered, there are a number of customizable actions that can be taken at that time. These actions include the following:

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	Submit a custom order

Order	

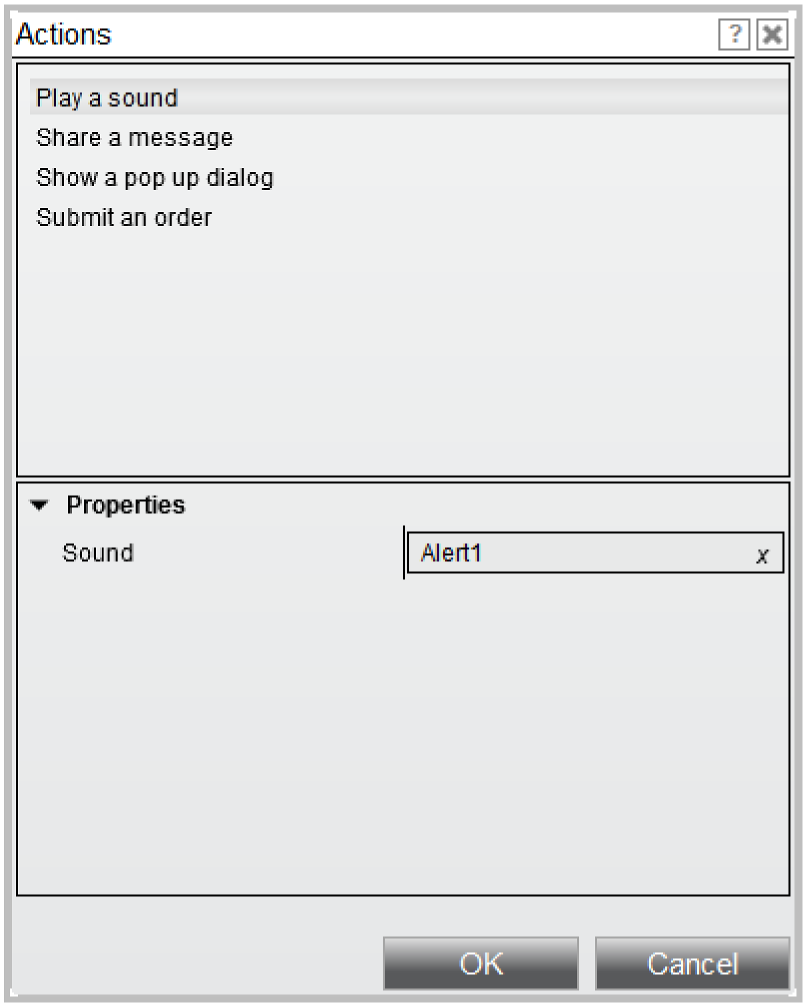
Configuring Actions

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:



Text: @NAME triggered
Priority: Medium
Color for background: Transparent
Color for foreground: Black

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.



Sound: Alert1

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 - A screen shot of the Tab that generated the alert will be sent • 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
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

11.2.4 Condition Builder

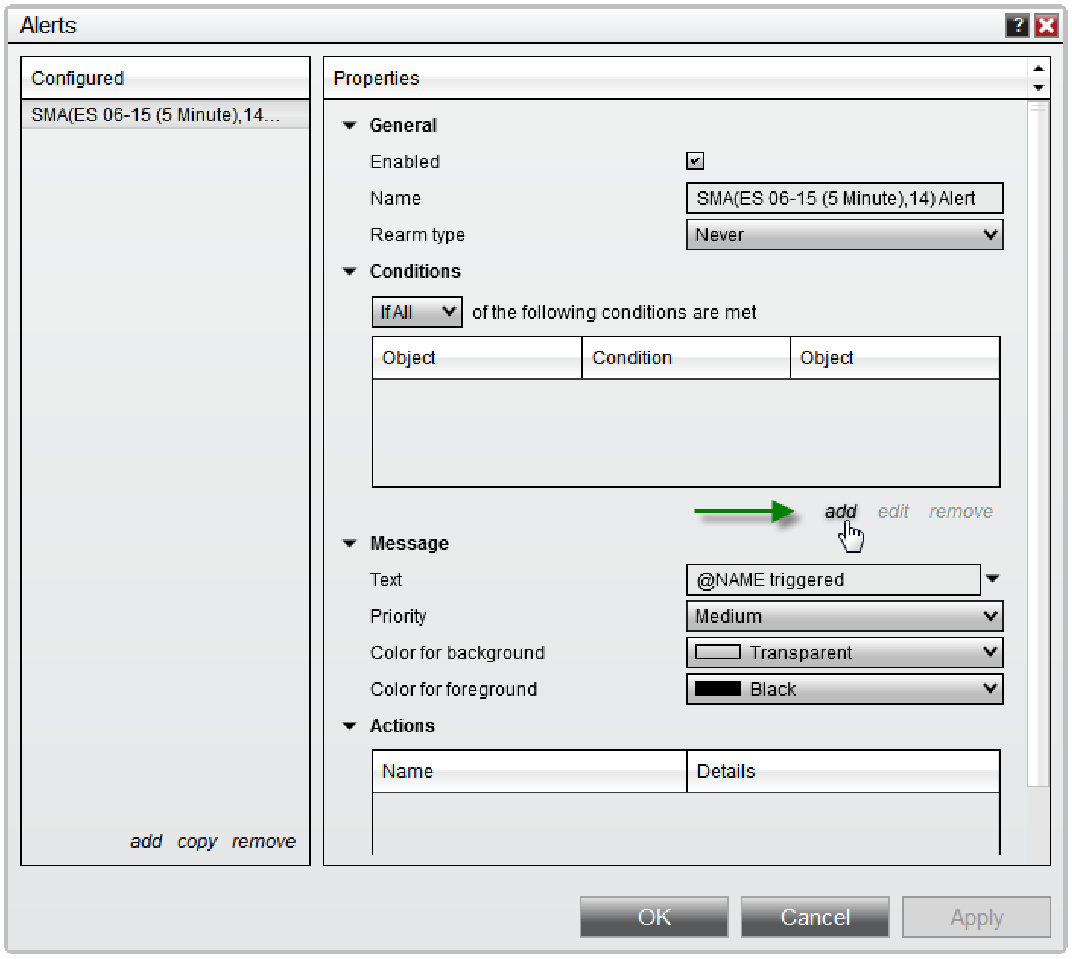
The **Condition Builder** is a very powerful feature that allows you to define complex conditions for your alerting systems without having to know how to program. The sections below assume you have read and understood how to configure the alerts dialog and understand how to select a **Condition Object** 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 without limitations. 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



SMA(ES 06-15 (5 Minute),14...

General
Enabled ✓
Name: SMA(ES 06-15 (5 Minute),14) Alert
Rearm type: Never

Conditions
If All of the following conditions are met

Object	Condition	Object

Message
Text: @NAME triggered
Priority: Medium
Color for background: Transparent
Color for foreground: Black

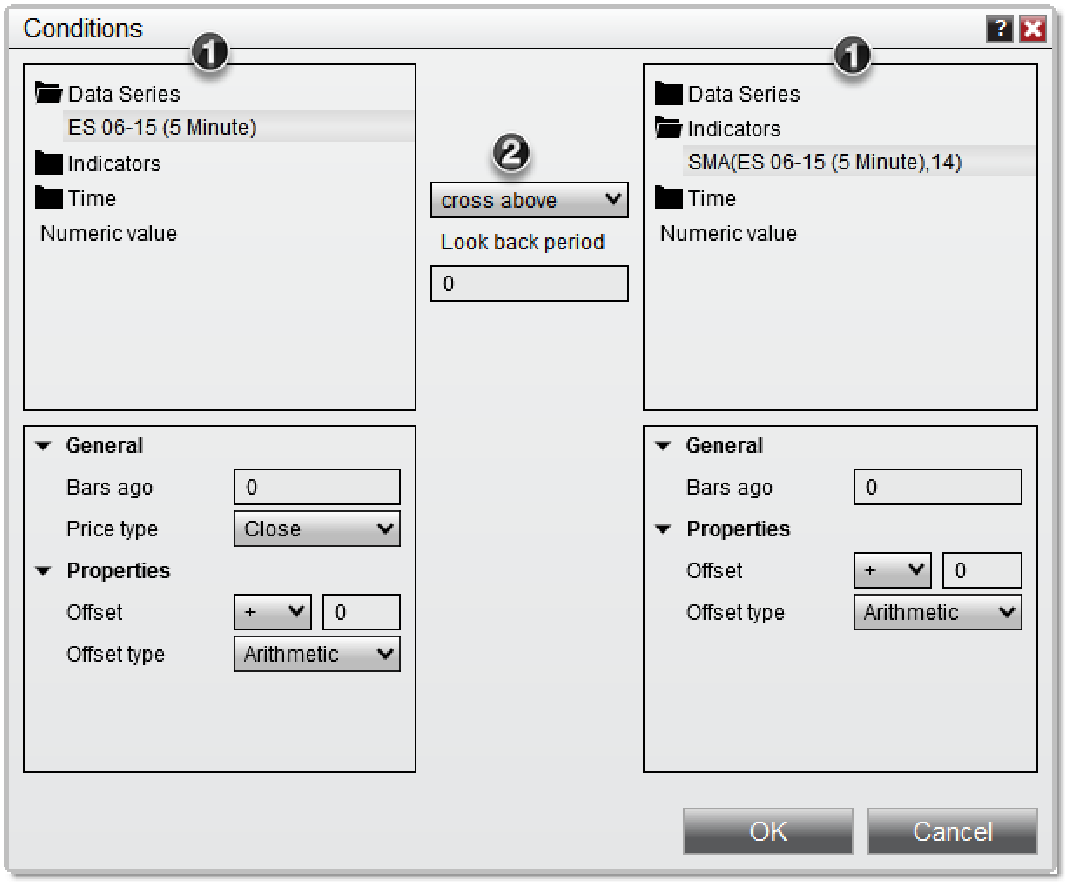
Actions

Name	Details

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

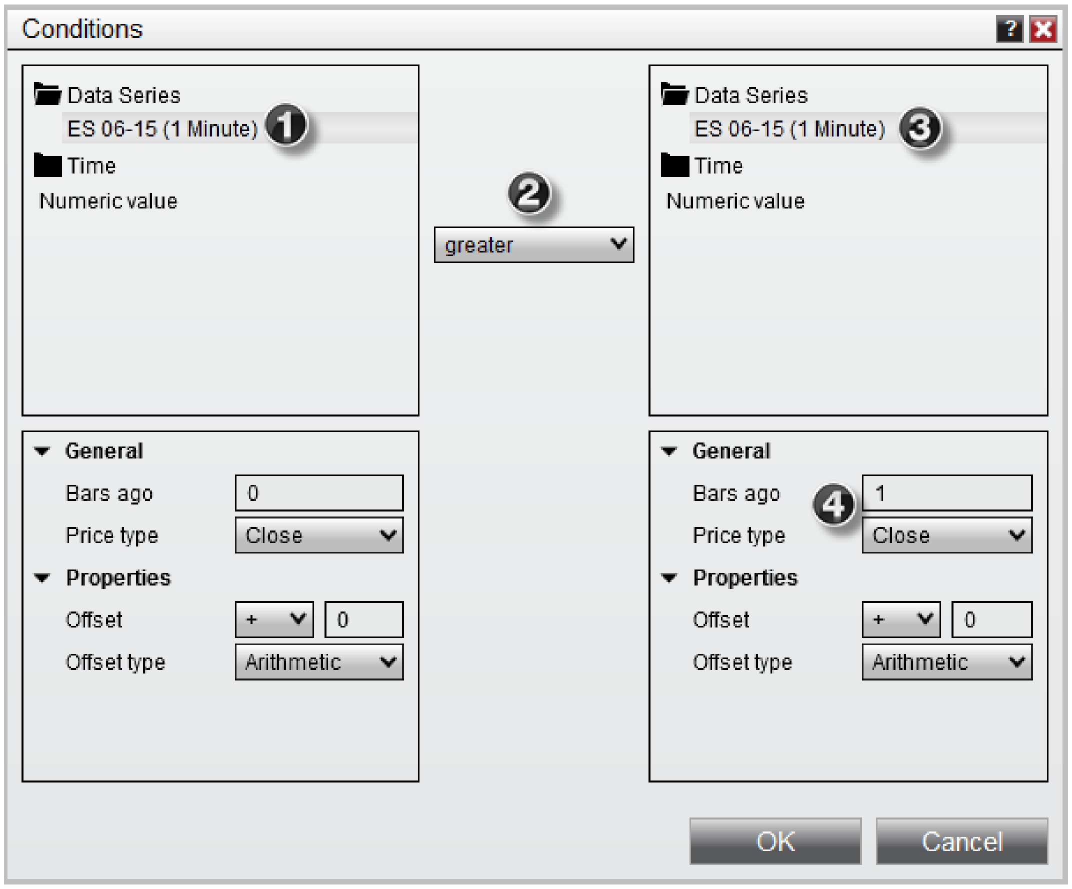
How to make chart price data comparisons

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.

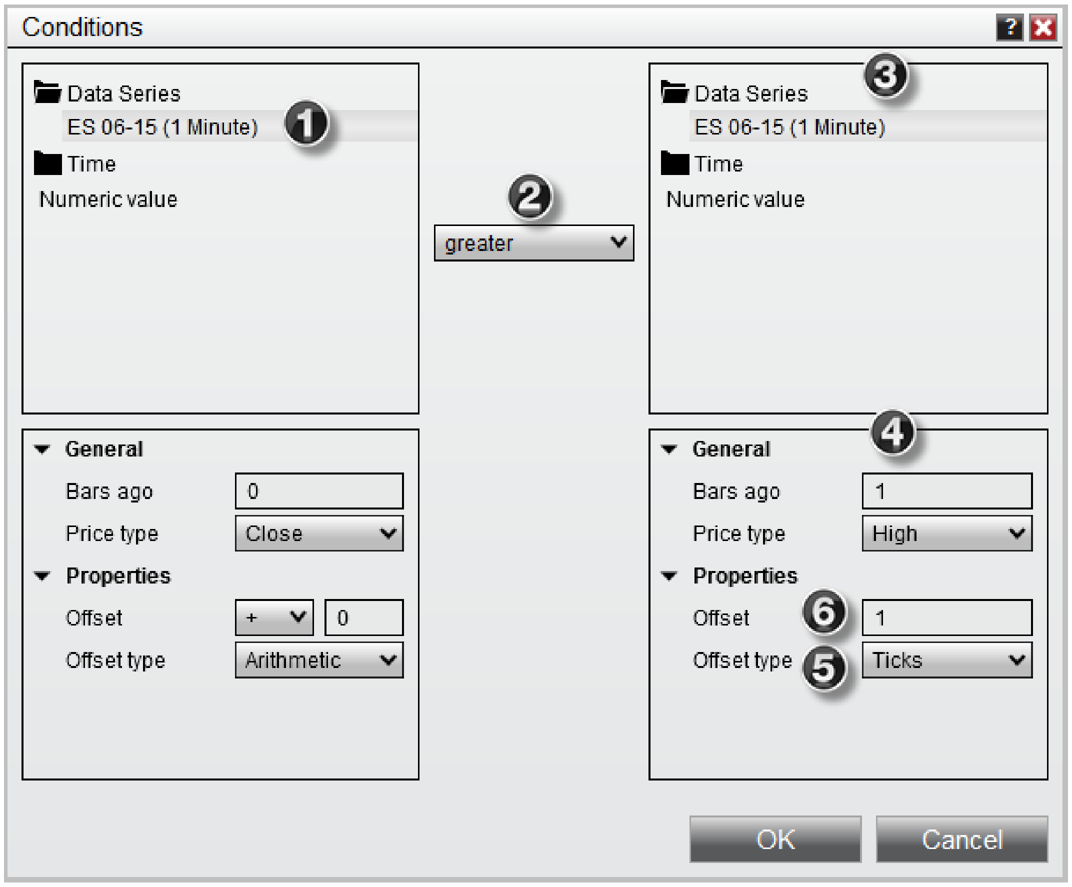
Offset type can be set to:

Arithmetic	Performs simple math functions such as adding, subtracting, multiplying, or dividing from the items value
Pips	Adds or subtracts an absolute pip value (for forex) from the item's value
Percent	Adds or subtracts 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	Adds or subtracts 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"


Conditions

Data Series
ES 06-15 (1 Minute)
Time
Numeric value

greater

Data Series
ES 06-15 (1 Minute)
Time
Numeric value

General	
Bars ago	0
Price type	Close
Properties	
Offset	+ 0
Offset type	Arithmetic

General	
Bars ago	1
Price type	High
Properties	
Offset	1
Offset type	Ticks

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

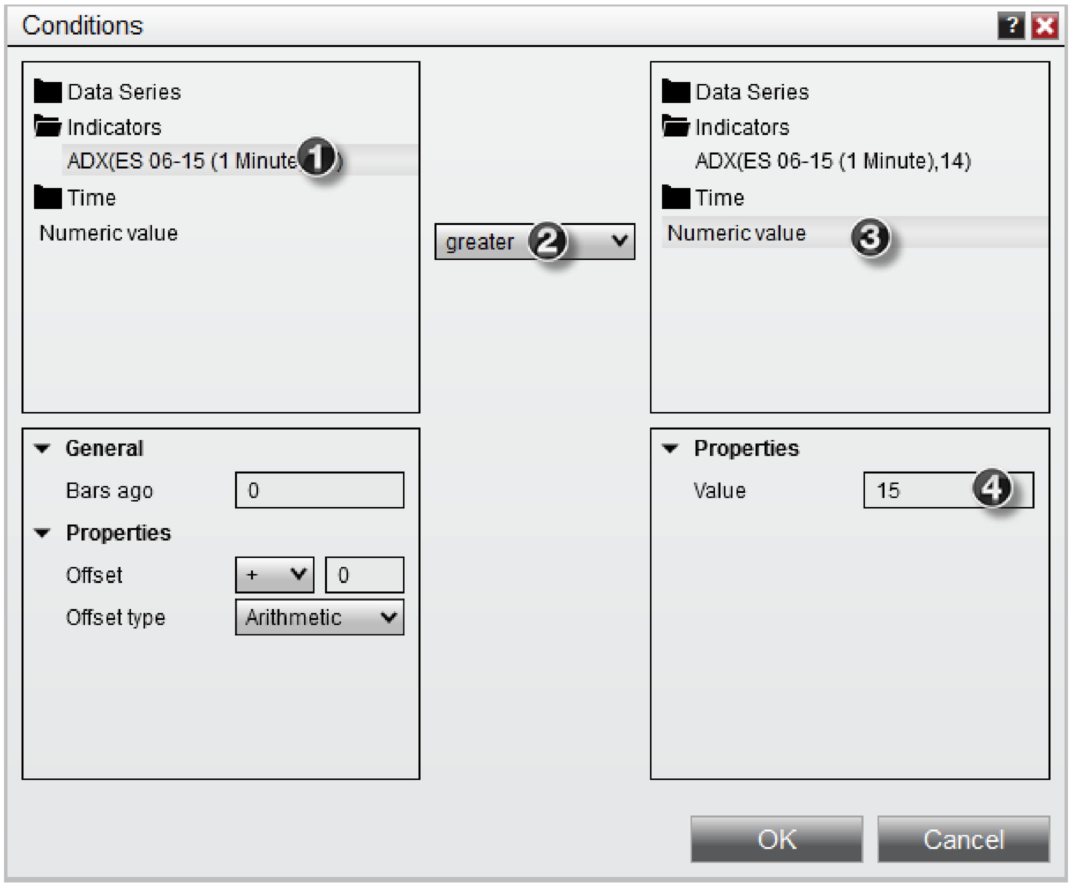
Indicator to Value Comparisons

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"

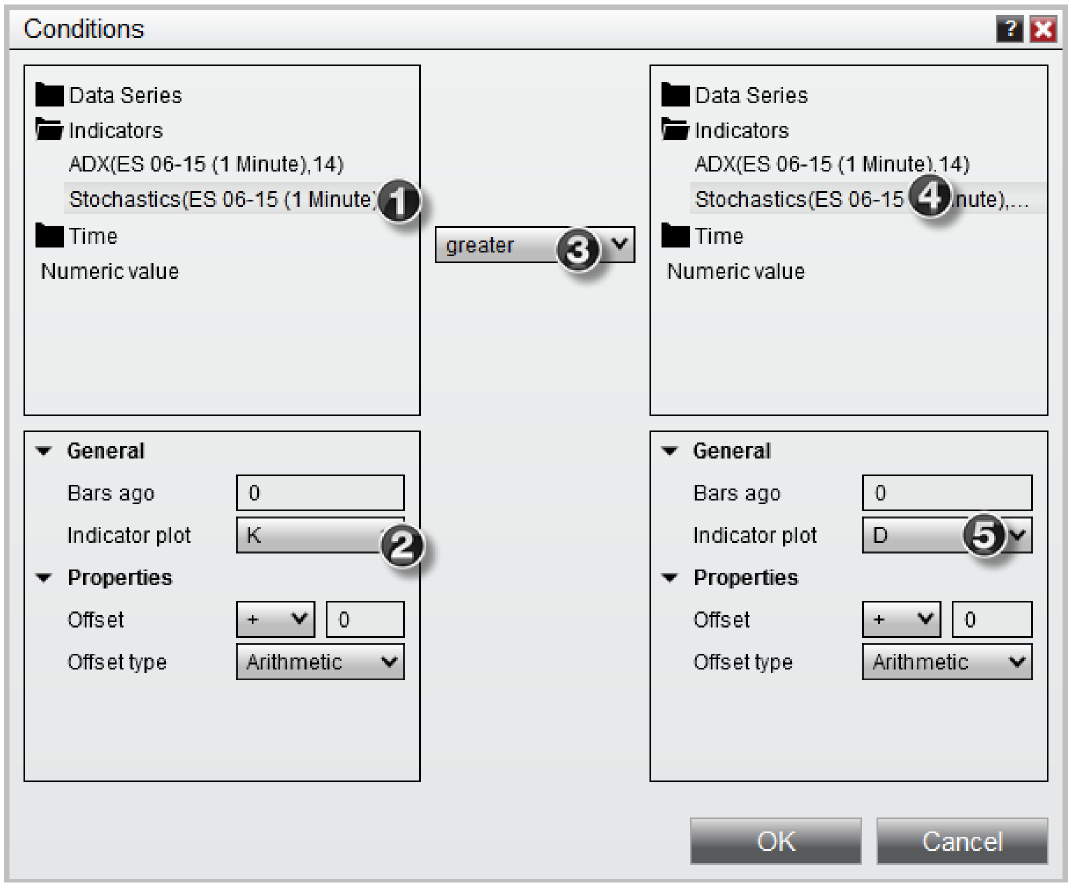
How to compare plot values of multi-plot indicators

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"

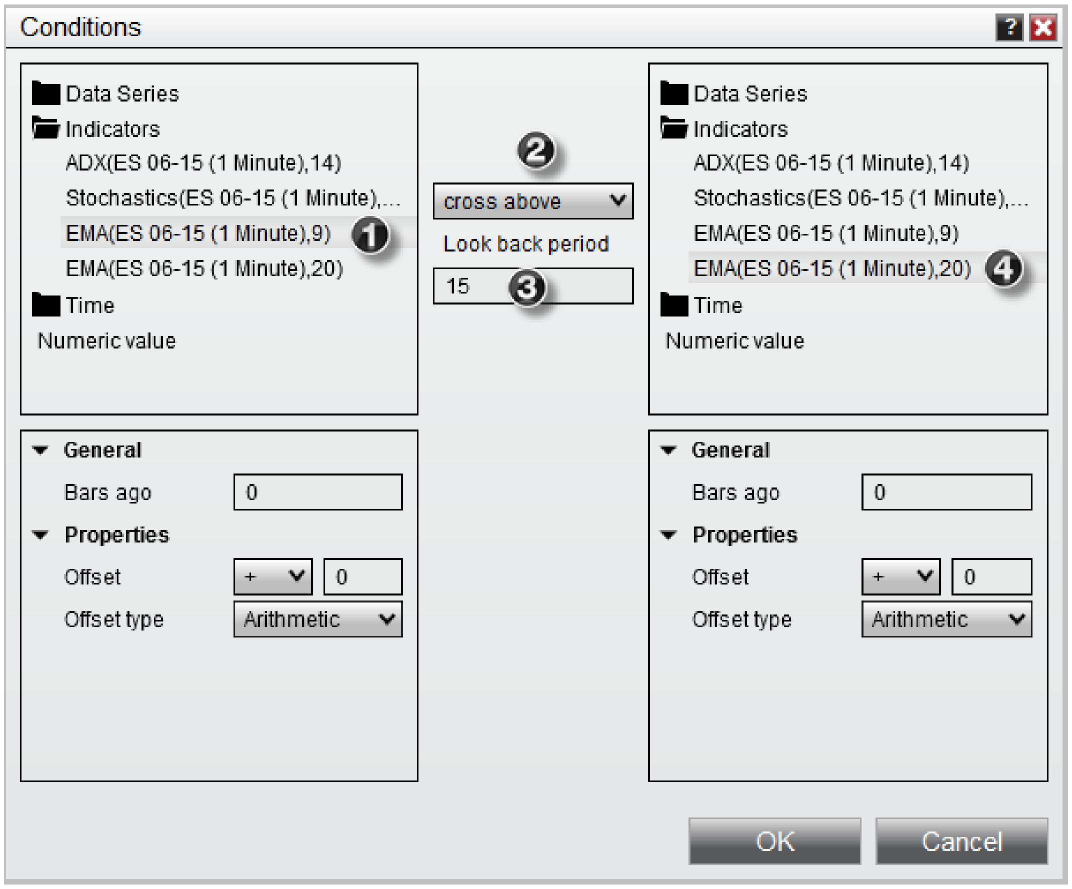
How to create a cross over condition

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


Conditions
Data Series
Indicators
ADX(ES 06-15 (1 Minute),14)
Stochastics(ES 06-15 (1 Minute),...
EMA(ES 06-15 (1 Minute),9)
EMA(ES 06-15 (1 Minute),20)
Time
Numeric value
1
2
3
4
General
Properties
OK
Cancel

cross above

Look back period 15

Bars ago 0

Offset + 0

Offset type Arithmetic

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"

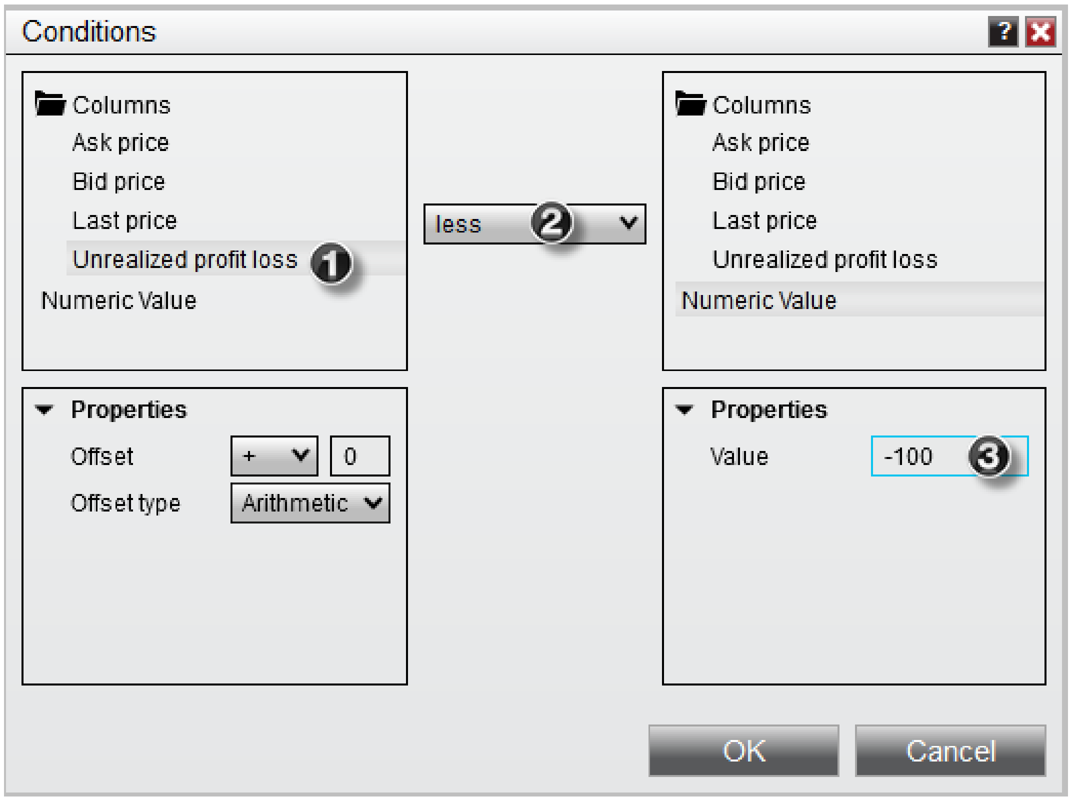
How to compare account position information

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

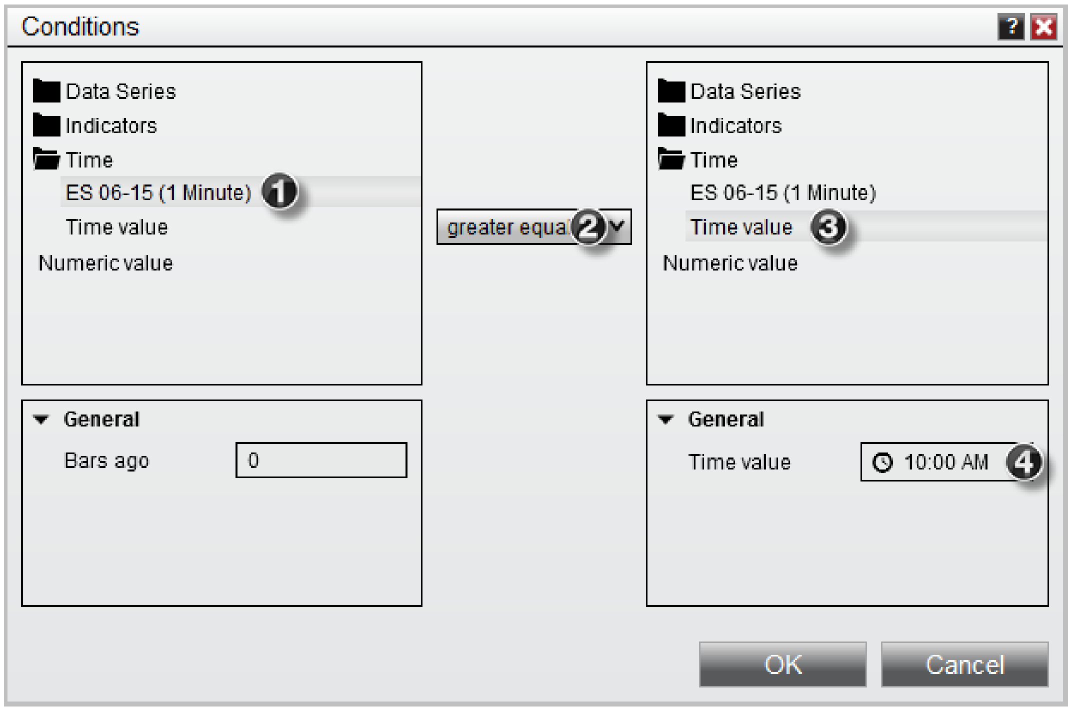
Creating Time Comparisons

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"

11.2.5 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

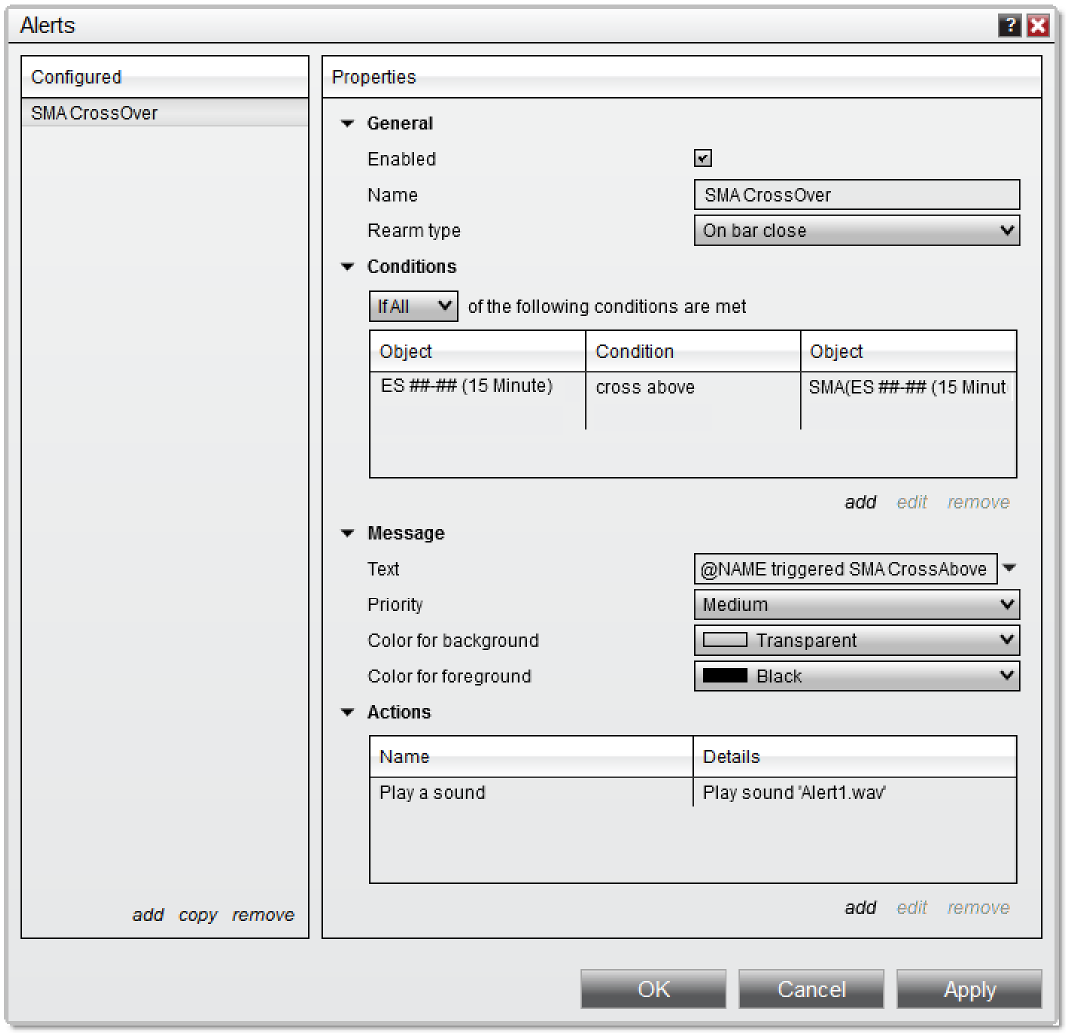
Simple Moving Average Crossover

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.



SMA CrossOver

add copy remove

General

Enabled ✓

Name: SMA CrossOver

Rearm type: On bar close

Conditions

If All of the following conditions are met

Object	Condition	Object
ES ##-## (15 Minute)	cross above	SMA(ES ##-## (15 Minut

add edit remove

Message

Text: @NAME triggered SMA CrossAbove

Priority: Medium

Color for background: Transparent

Color for foreground: Black

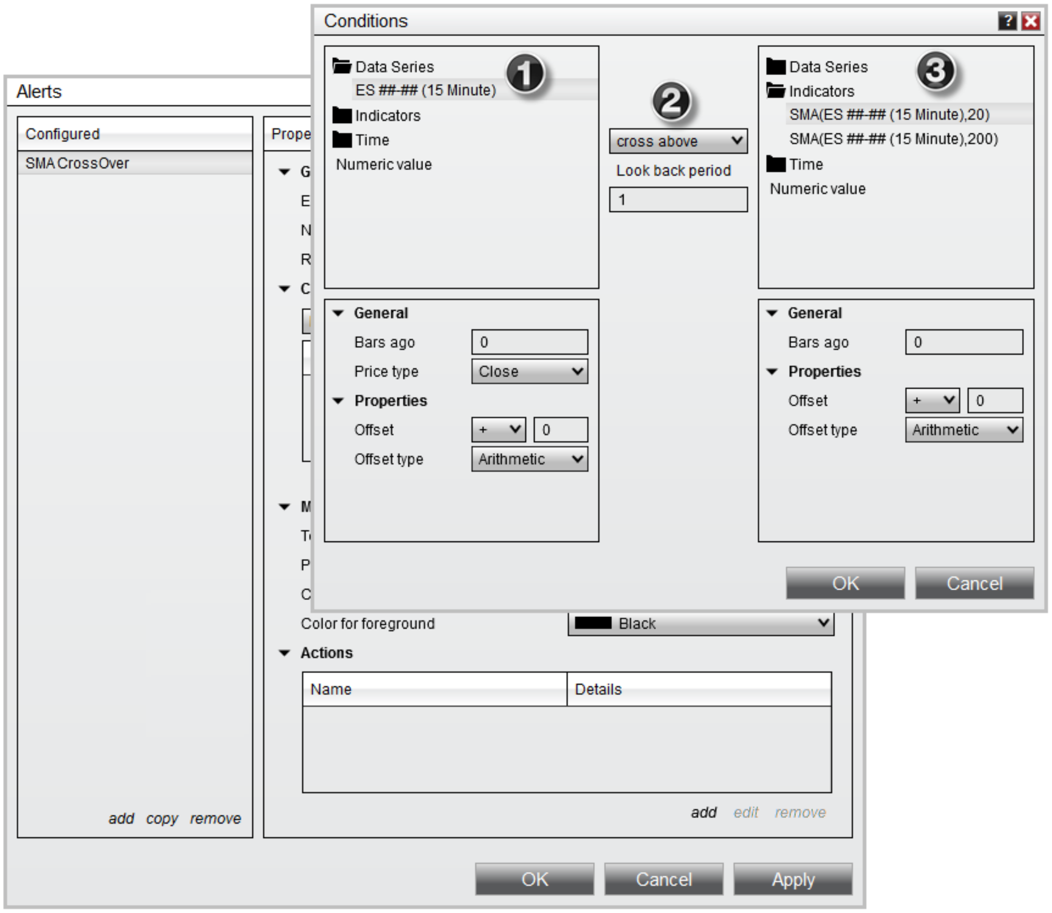
Actions

Name	Details
Play a sound	Play sound 'Alert1.wav'

add edit remove

Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

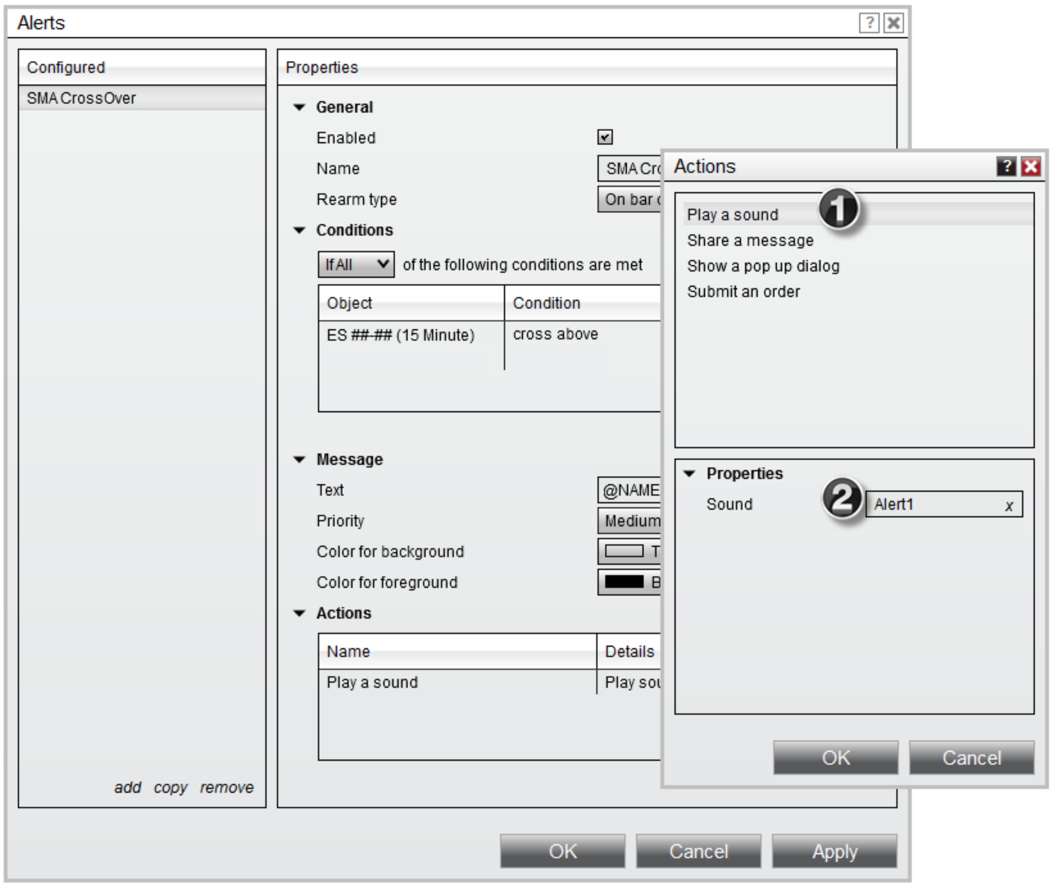


1. In the Conditions window pictured above, the ES data series is selected in the left panel

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."**



SMA CrossOver

General
Enabled ✓
Name SMA Cr
Rearm type On bar c

Conditions
If All of the following conditions are met

Object	Condition
ES ##-## (15 Minute)	cross above

Message
Text @NAME
Priority Medium
Color for background T
Color for foreground B

Actions

Name	Details
Play a sound	Play sou

Properties
Sound Alert1 x

1. In the Actions window, the "Play a Sound" option is selected

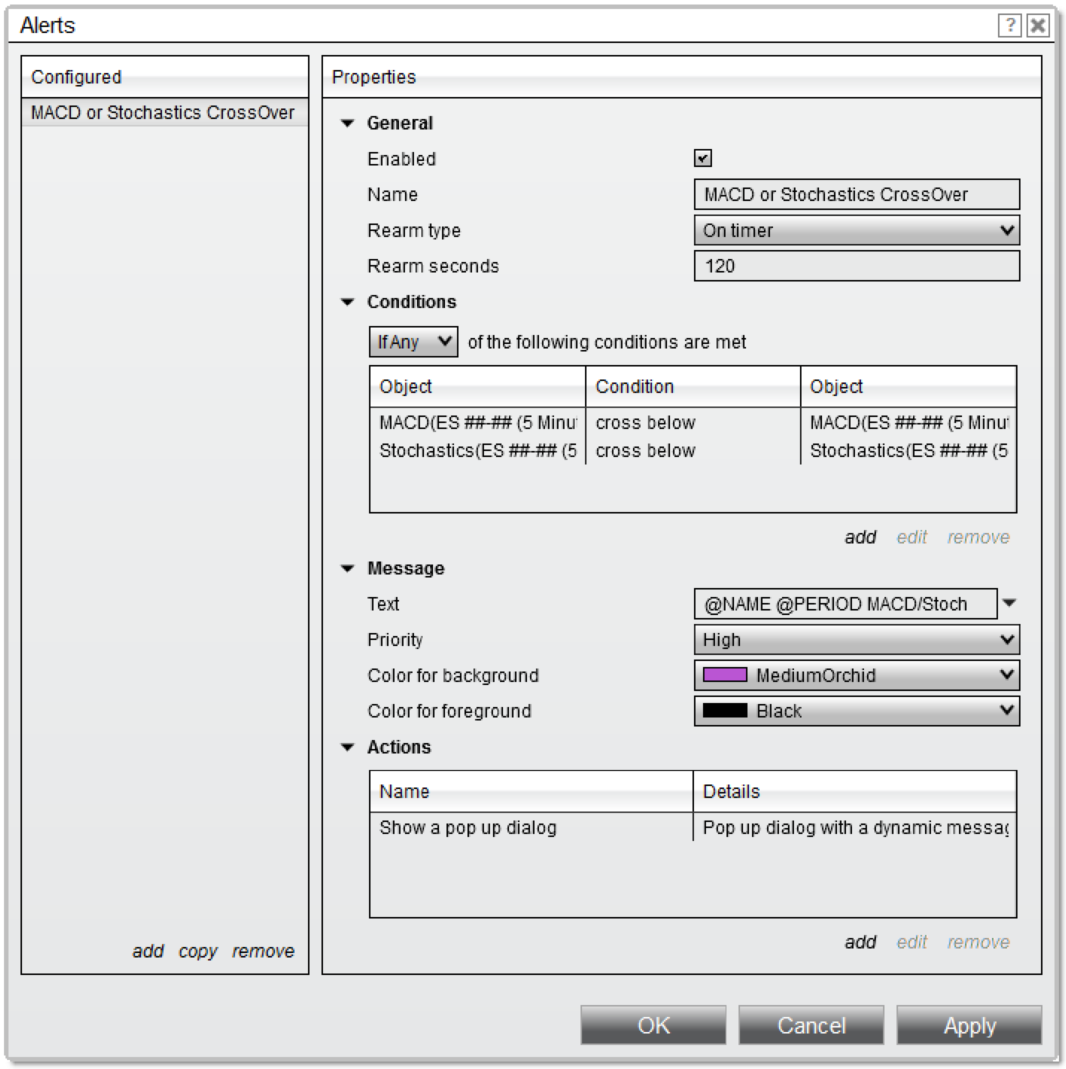
2. A sound named "Alert1" is selected to be played when the alert triggers

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.



MACD or Stochastics CrossOver

General

Enabled	✓
Name	MACD or Stochastics CrossOver
Rearm type	On timer
Rearm seconds	120

Conditions

If Any of the following conditions are met

Object	Condition	Object
MACD(ES ##-## (5 Minu	cross below	MACD(ES ##-## (5 Minu
Stochastics(ES ##-## (5	cross below	Stochastics(ES ##-## (5

Message

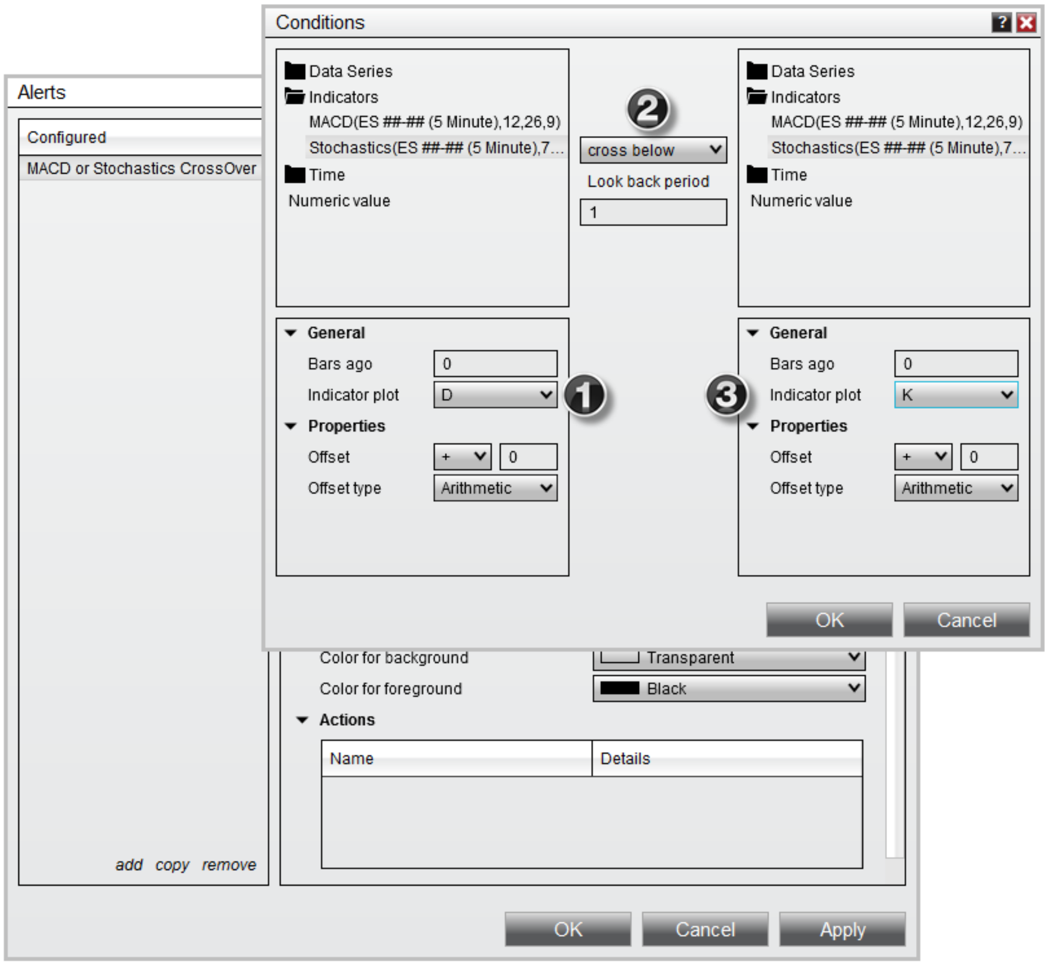
Text	@NAME @PERIOD MACD/Stoch
Priority	High
Color for background	MediumOrchid
Color for foreground	Black

Actions

Name	Details
Show a pop up dialog	Pop up dialog with a dynamic messa

Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

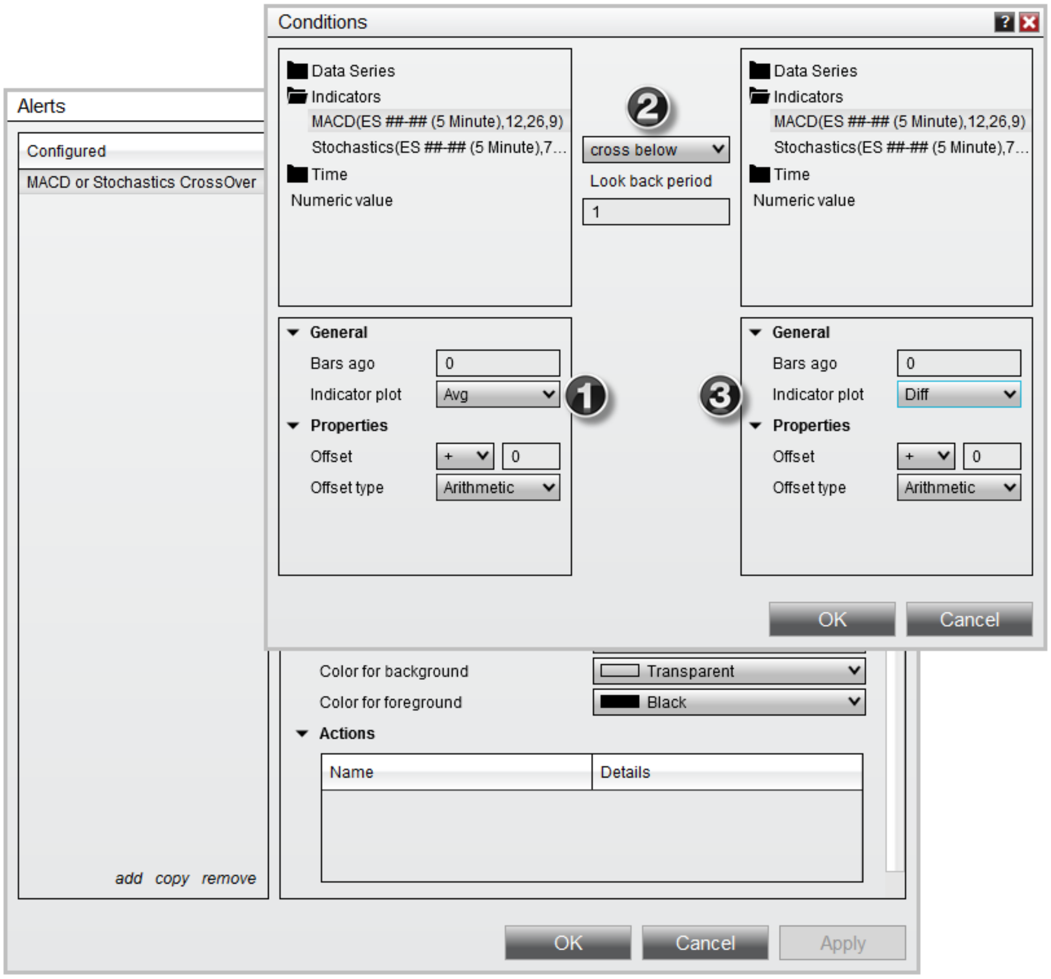


1. In the Conditions window, the D plot of the Stochastics indicator is selected in the left panel

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."

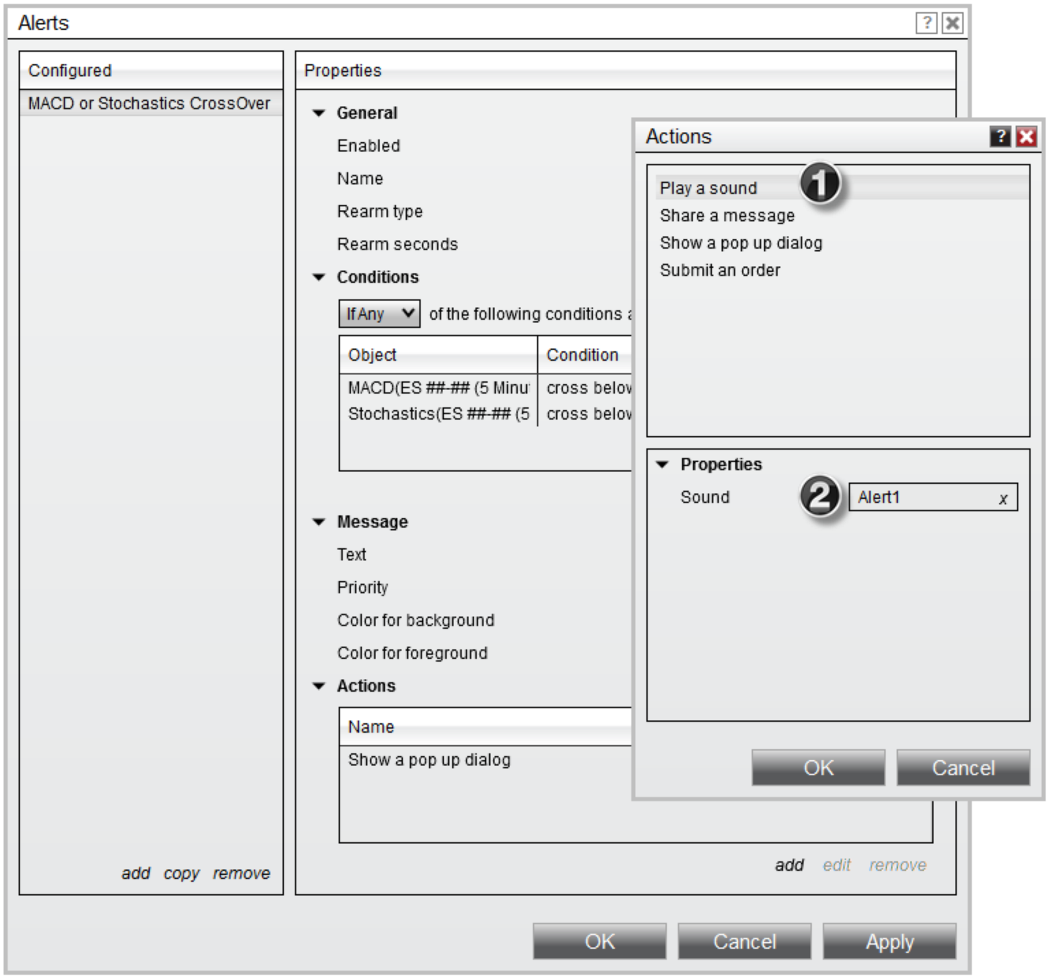


1. In the Conditions window for the second condition, the Avg plot of the MACD indicator is selected in the left panel

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

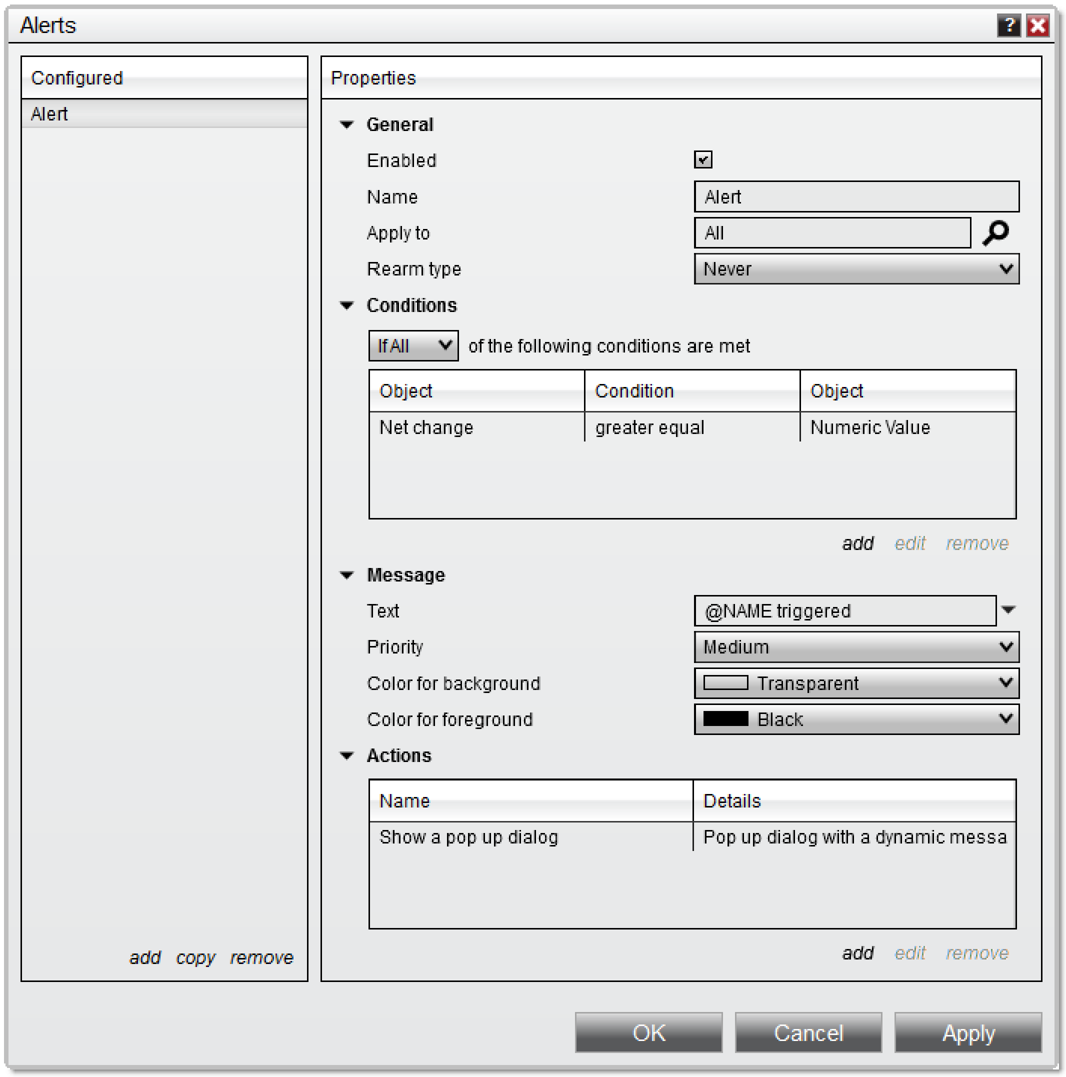
Hot List Analyzer Net Change

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.


Alerts

Configured
Alert

Properties

General

Enabled	✓
Name	Alert
Apply to	All
Rearm type	Never

Conditions

If All of the following conditions are met

Object	Condition	Object
Net change	greater equal	Numeric Value

Message

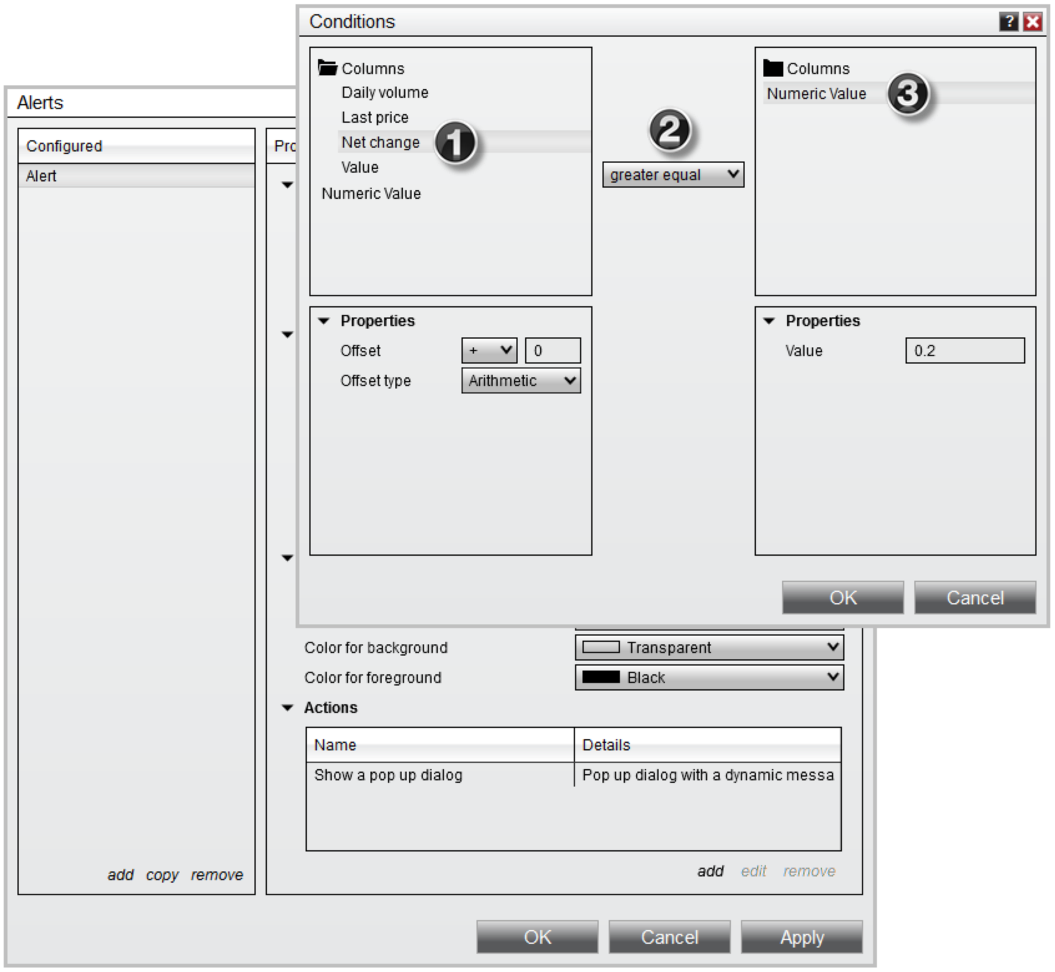
Text	@NAME triggered
Priority	Medium
Color for background	Transparent
Color for foreground	Black

Actions

Name	Details
Show a pop up dialog	Pop up dialog with a dynamic messa

Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

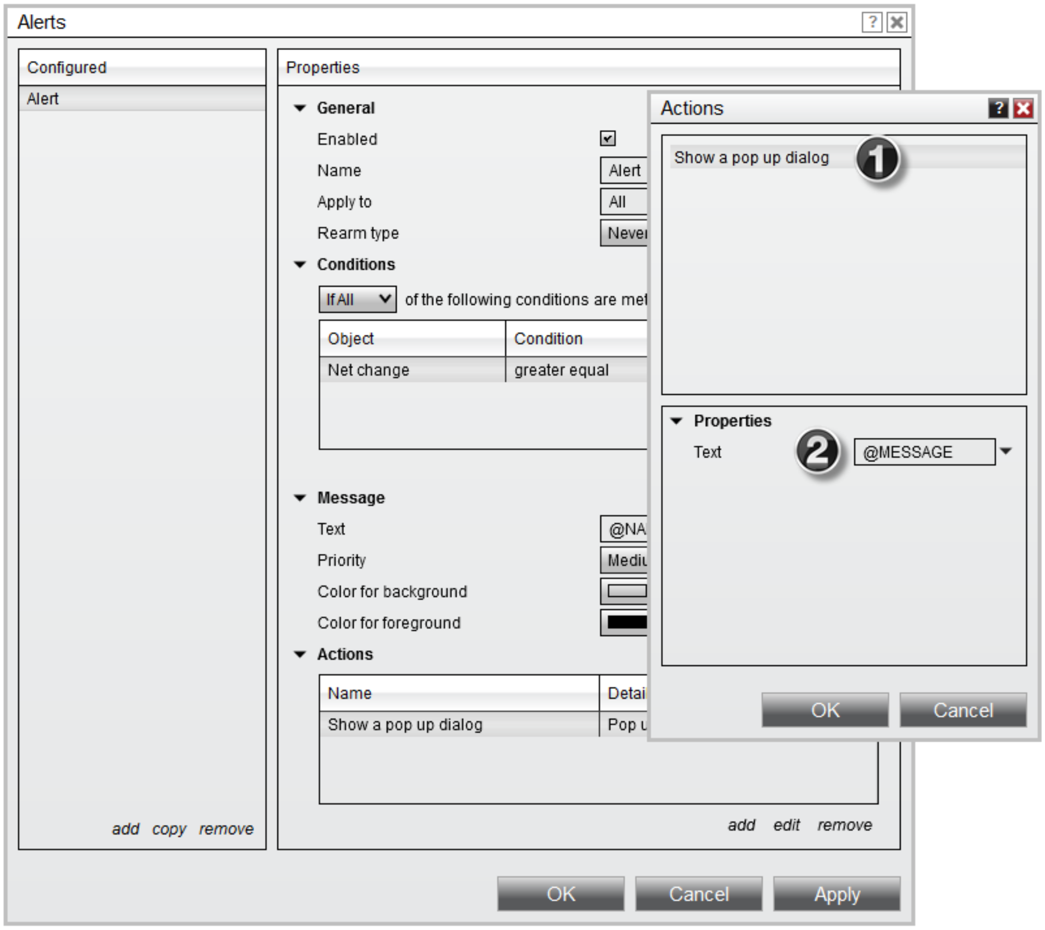


1. In the Conditions window, the Net Change column is selected in the left panel.

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

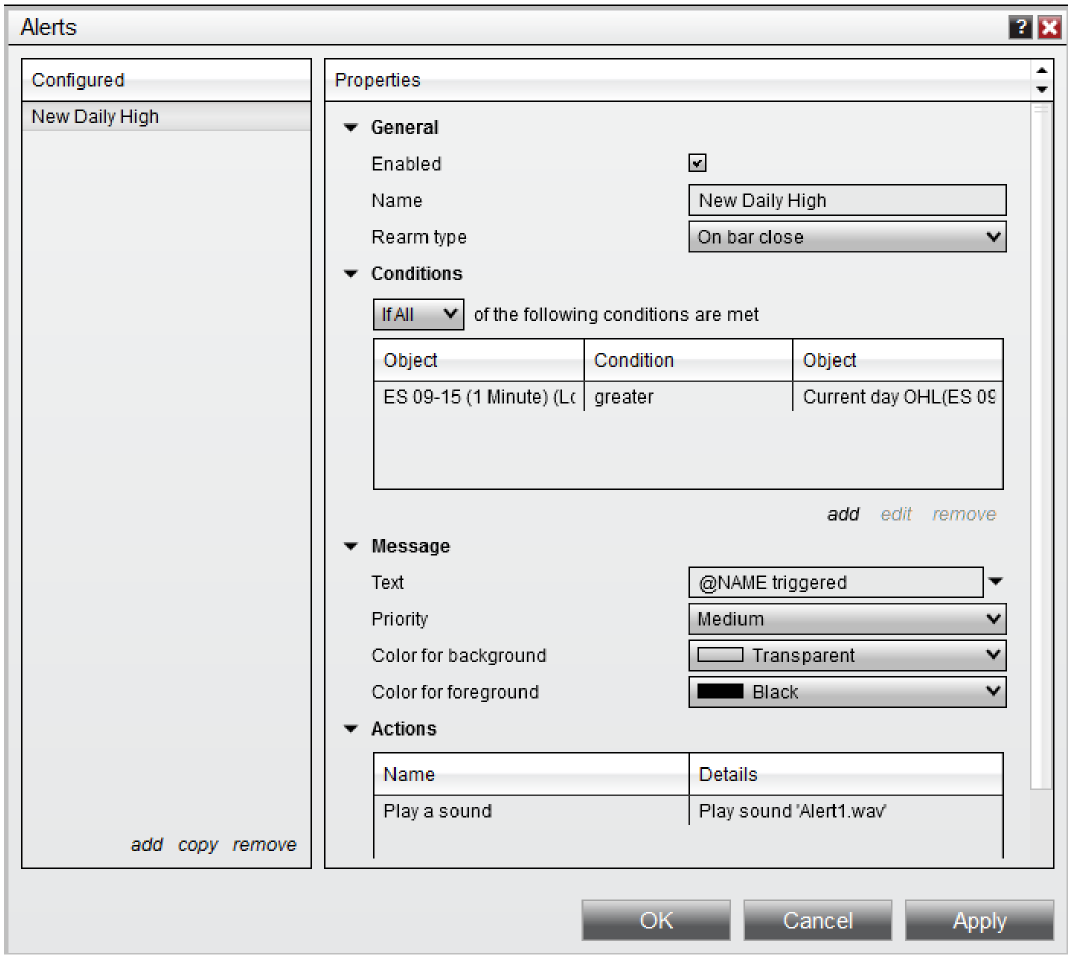
2. "@MESSAGE" is entered for the text to be displayed in the dialog. This will populate the dialog with the message entered in the "Message" section of the Alerts window.

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.



New Daily High

General

Enabled	✓
Name	New Daily High
Rearm type	On bar close

Conditions

If All of the following conditions are met

Object	Condition	Object
ES 09-15 (1 Minute) (Lc	greater	Current day OHL(ES 09

Message

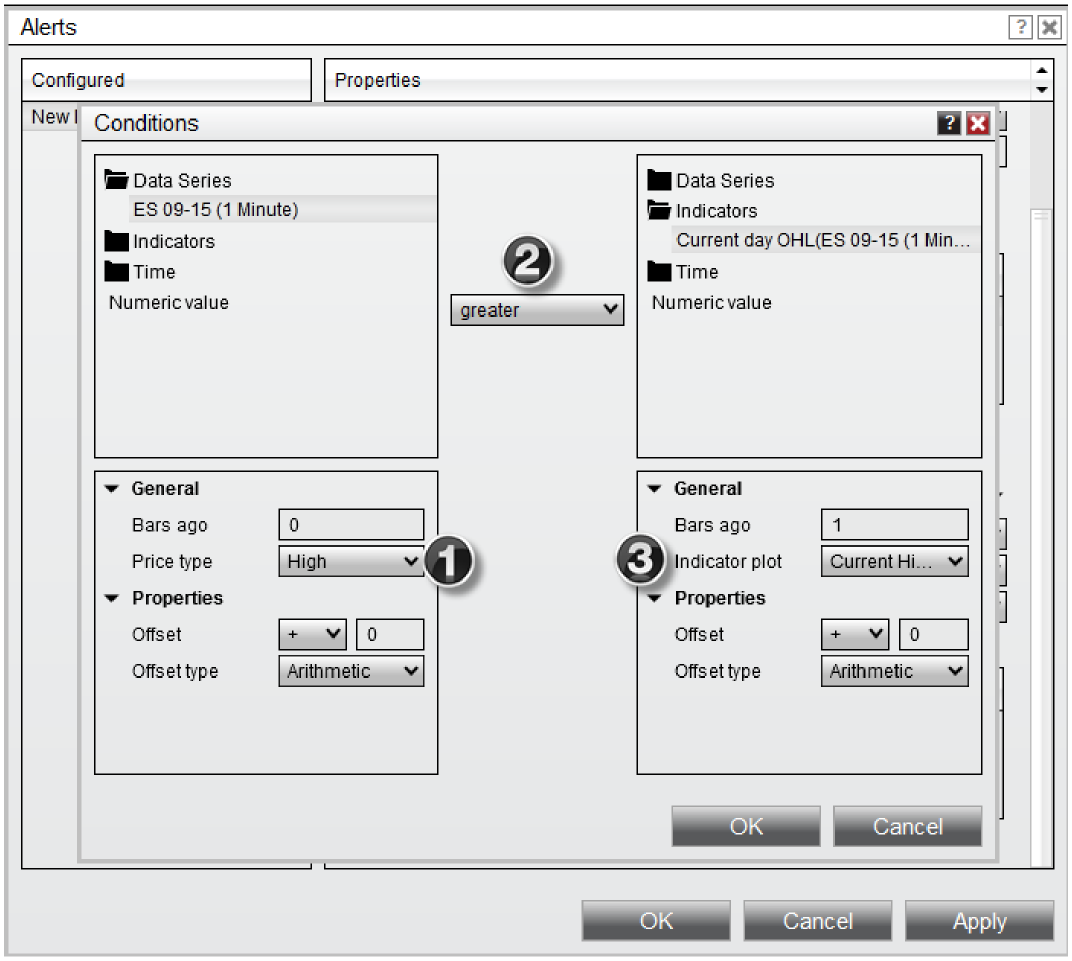
Text	@NAME triggered
Priority	Medium
Color for background	Transparent
Color for foreground	Black

Actions

Name	Details
Play a sound	Play sound 'Alert1.wav'

Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:



Left panel: Data Series: ES 09-15 (1 Minute)

Bars ago	0
Price type	High
Offset	+ 0
Offset type	Arithmetic

Right panel: Indicators: Current day OHL(ES 09-15 (1 Min...

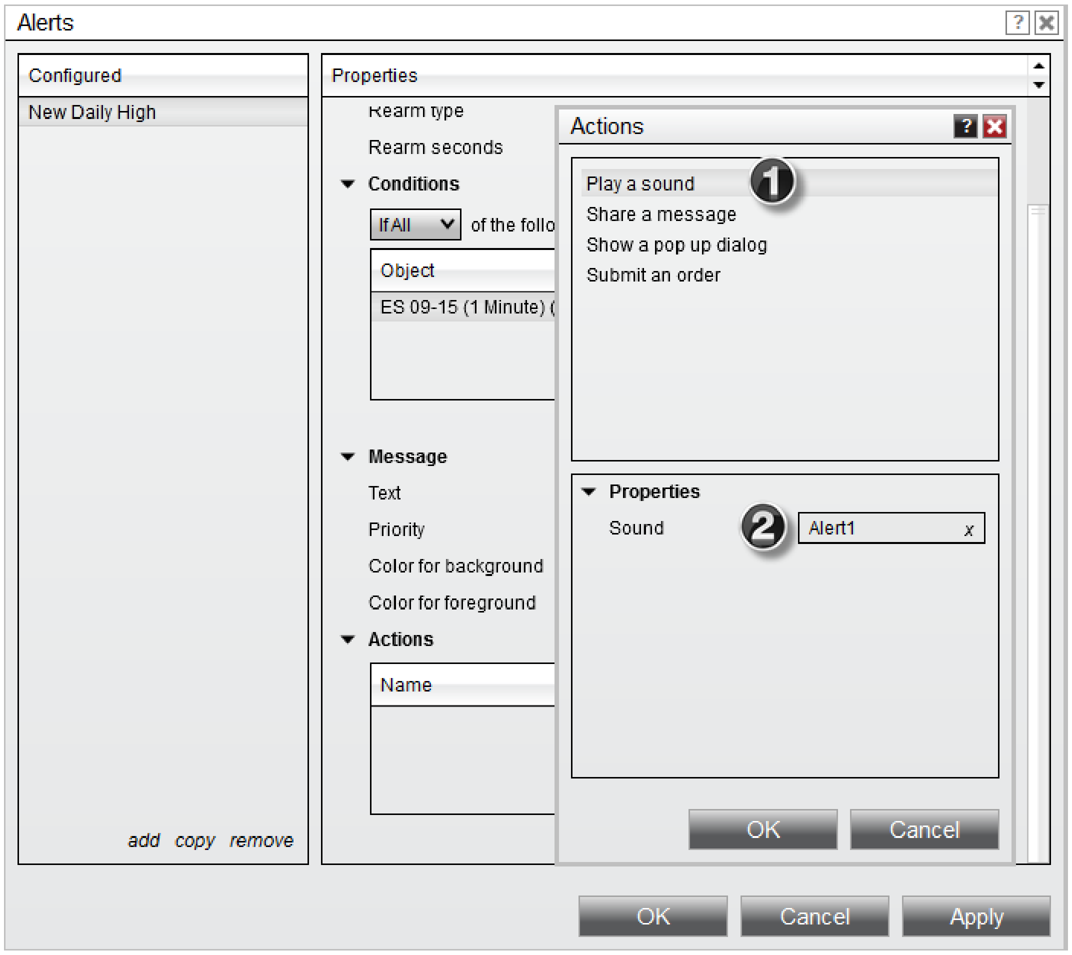
Bars ago	1
Indicator plot	Current Hi...
Offset	+ 0
Offset type	Arithmetic

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

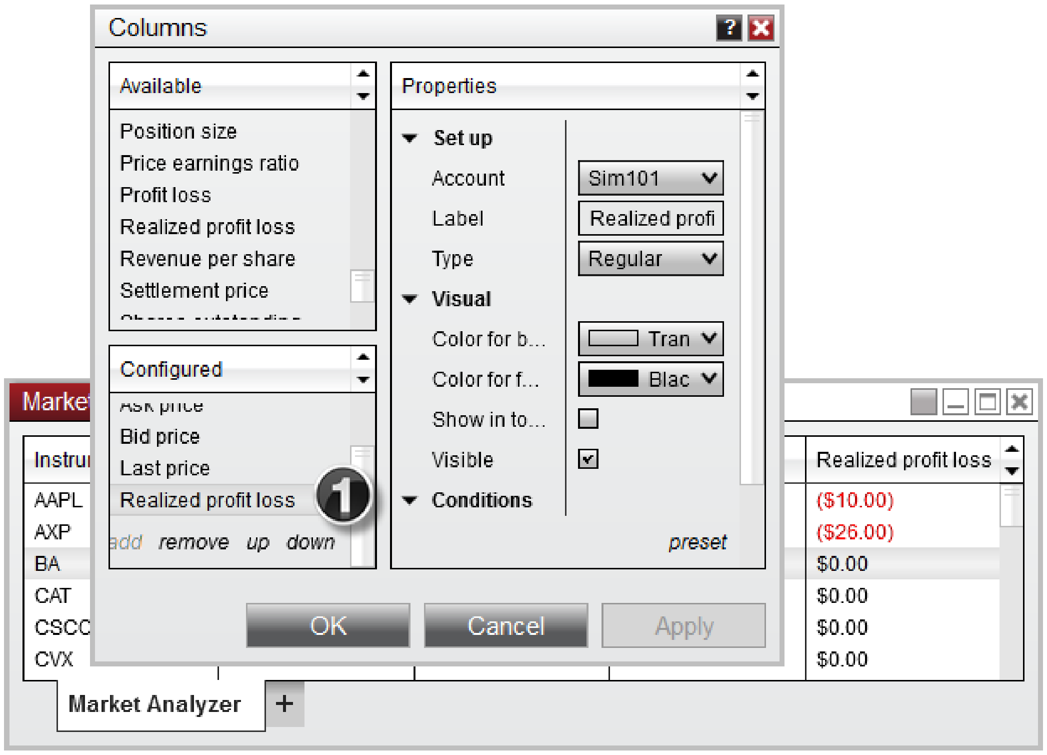
PnL Risk Management

Preparation

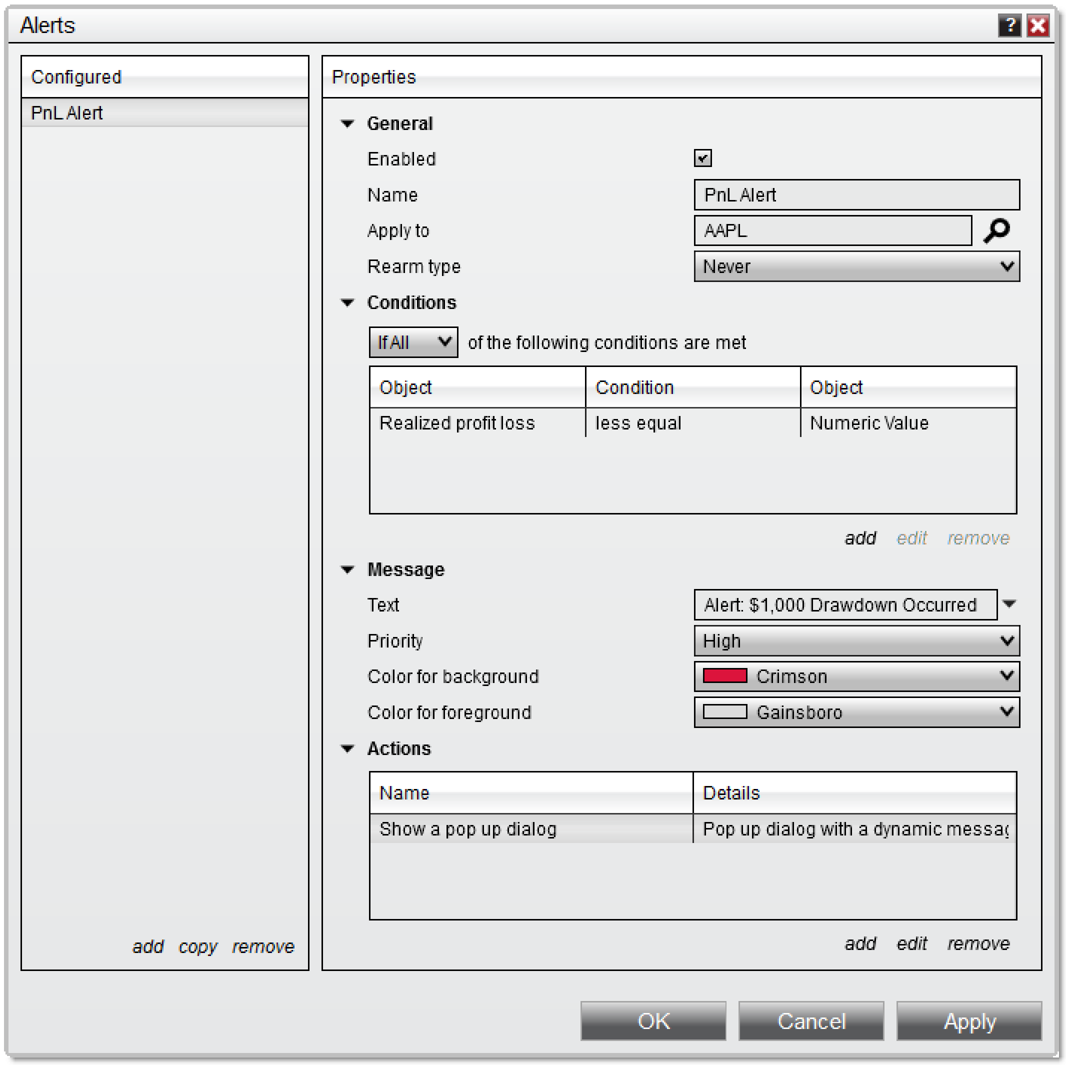
- Open a Market Analyzer
- Apply a "Realized Profit/Loss" column to the Market Analyzer (see image below)

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.


Alerts

Configured
PnL Alert

Properties

General

Enabled	✓
Name	PnL Alert
Apply to	AAPL
Rearm type	Never

Conditions

If All of the following conditions are met

Object	Condition	Object
Realized profit loss	less equal	Numeric Value

Message

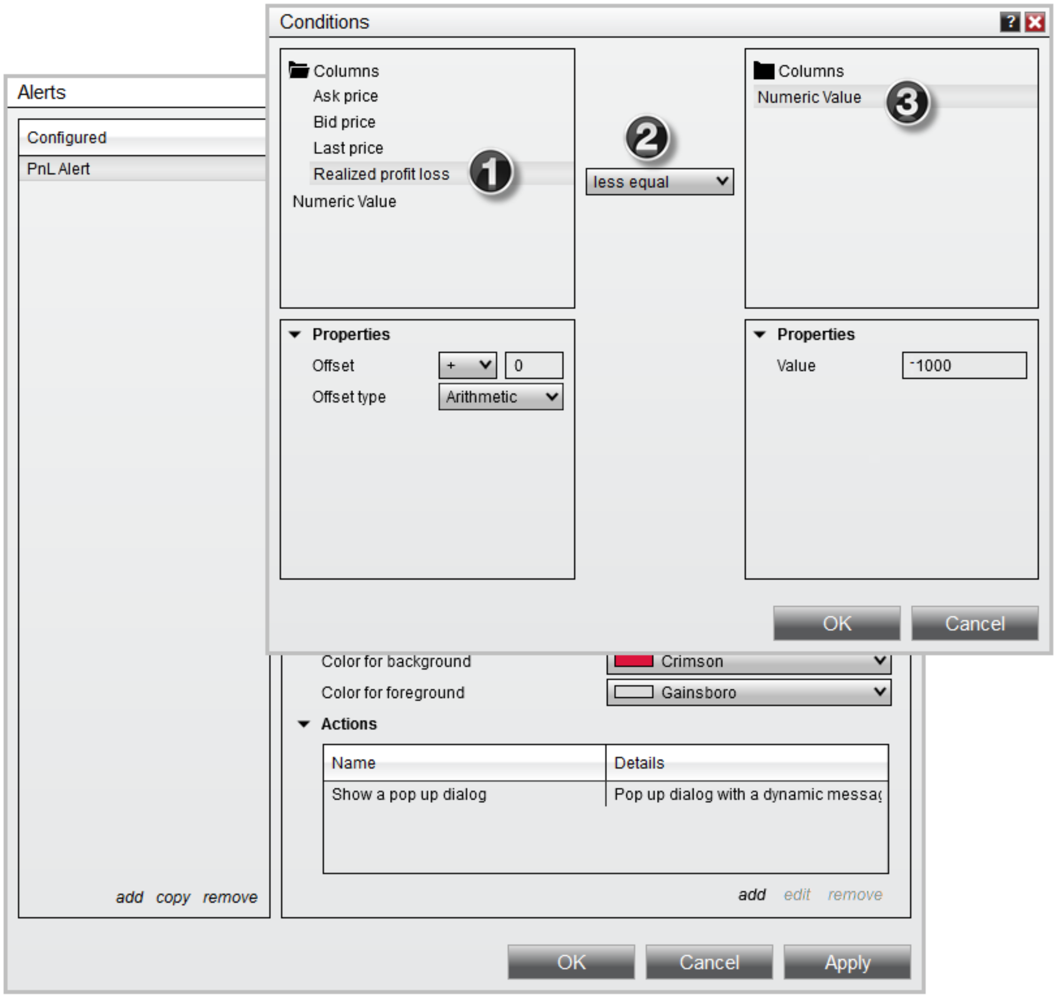
Text	Alert: $1,000 Drawdown Occurred
Priority	High
Color for background	🟥 Crimson
Color for foreground	⬜ Gainsboro

Actions

Name	Details
Show a pop up dialog	Pop up dialog with a dynamic messa

Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

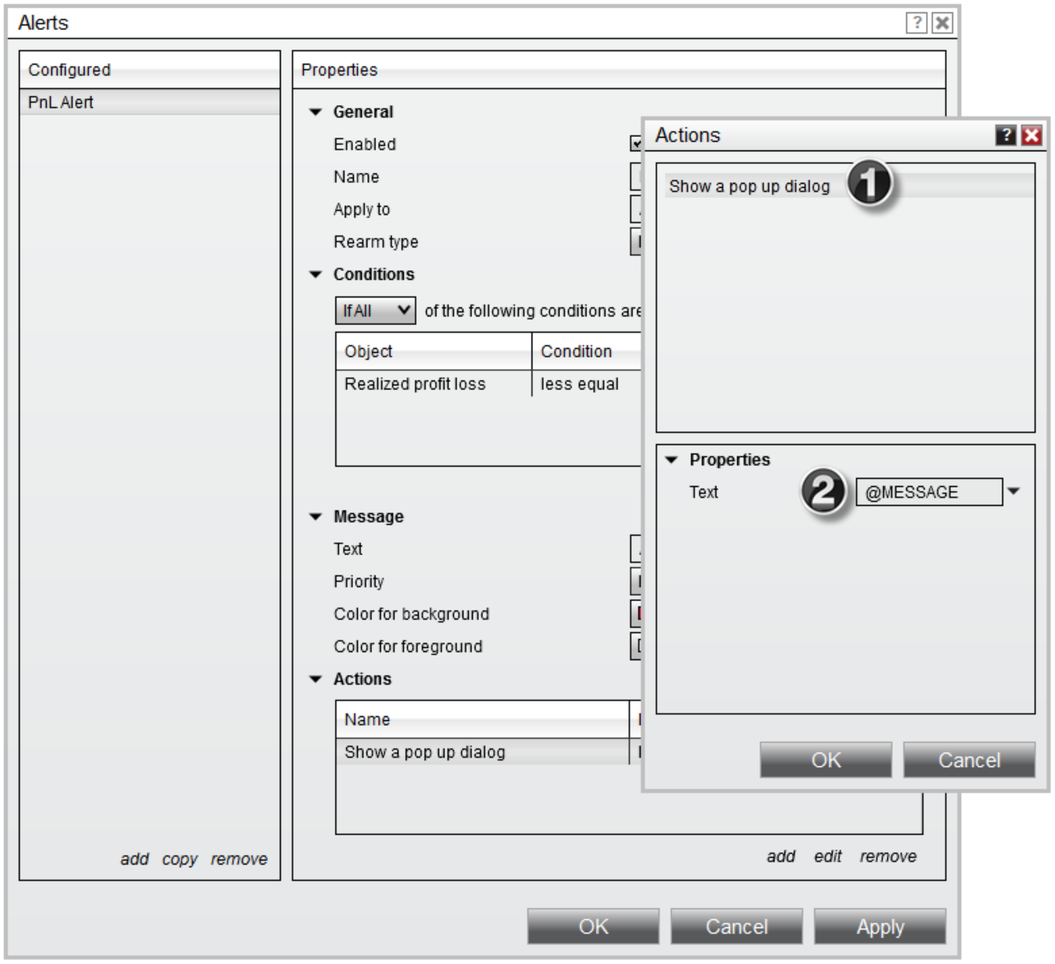


1. In the Conditions window, the "Realized Profit/Loss" column is selected in the right panel. Note that this column was manually added to the Market Analyzer before opening the Alerts window.

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."**



Alerts

Configured

PnL Alert

add copy remove

Properties

General

Enabled

Name

Apply to

Rearm type

Conditions

If All of the following conditions are

Object	Condition
Realized profit loss	less equal

Message

Text

Priority

Color for background

Color for foreground

Actions

Name
Show a pop up dialog

add edit remove

OK Cancel Apply

Actions

Show a pop up dialog 1

Properties

Text 2 @MESSAGE

OK Cancel

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.

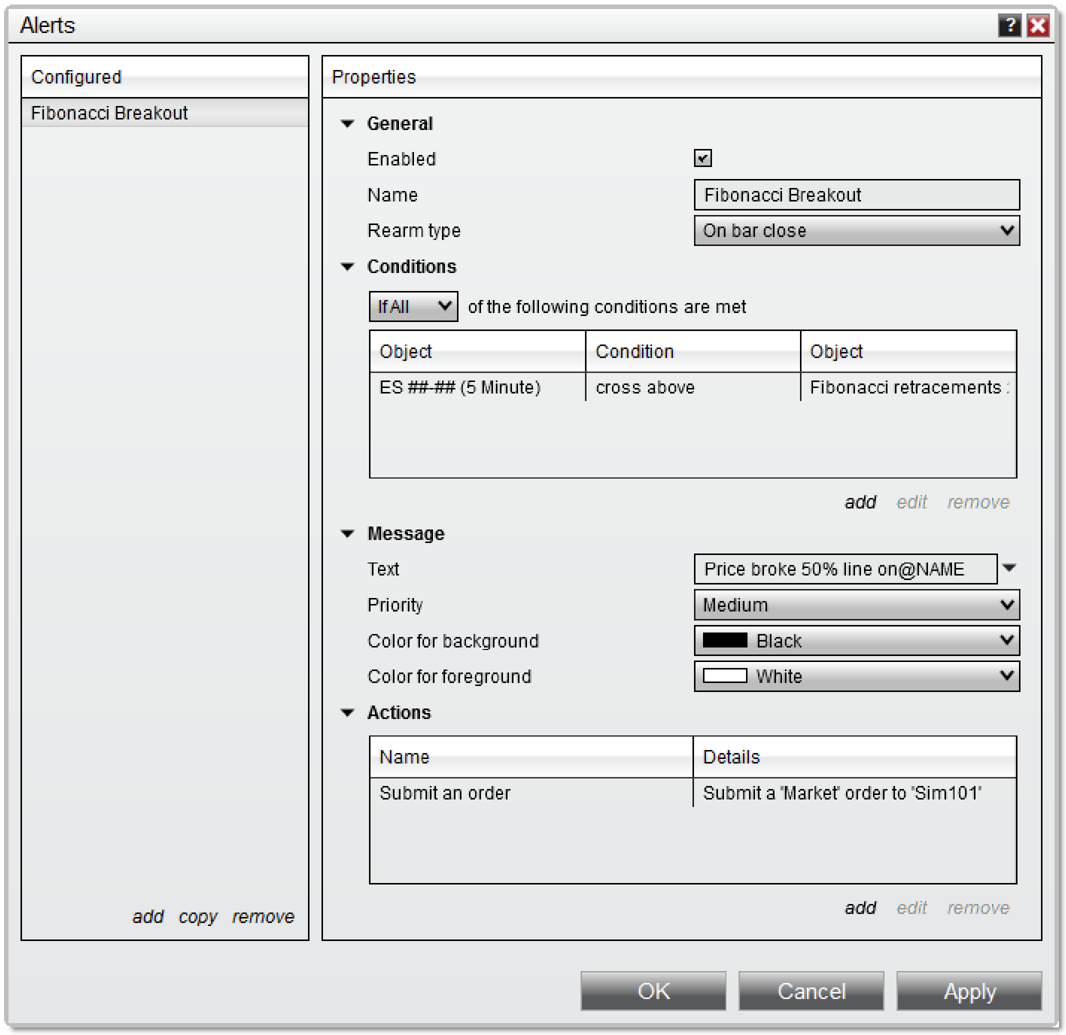
Price and Fibonacci Retracements

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.



Fibonacci Breakout

General

Enabled	✓
Name	Fibonacci Breakout
Rearm type	On bar close

Conditions

If All of the following conditions are met

Object	Condition	Object
ES ##-## (5 Minute)	cross above	Fibonacci retracements :

Message

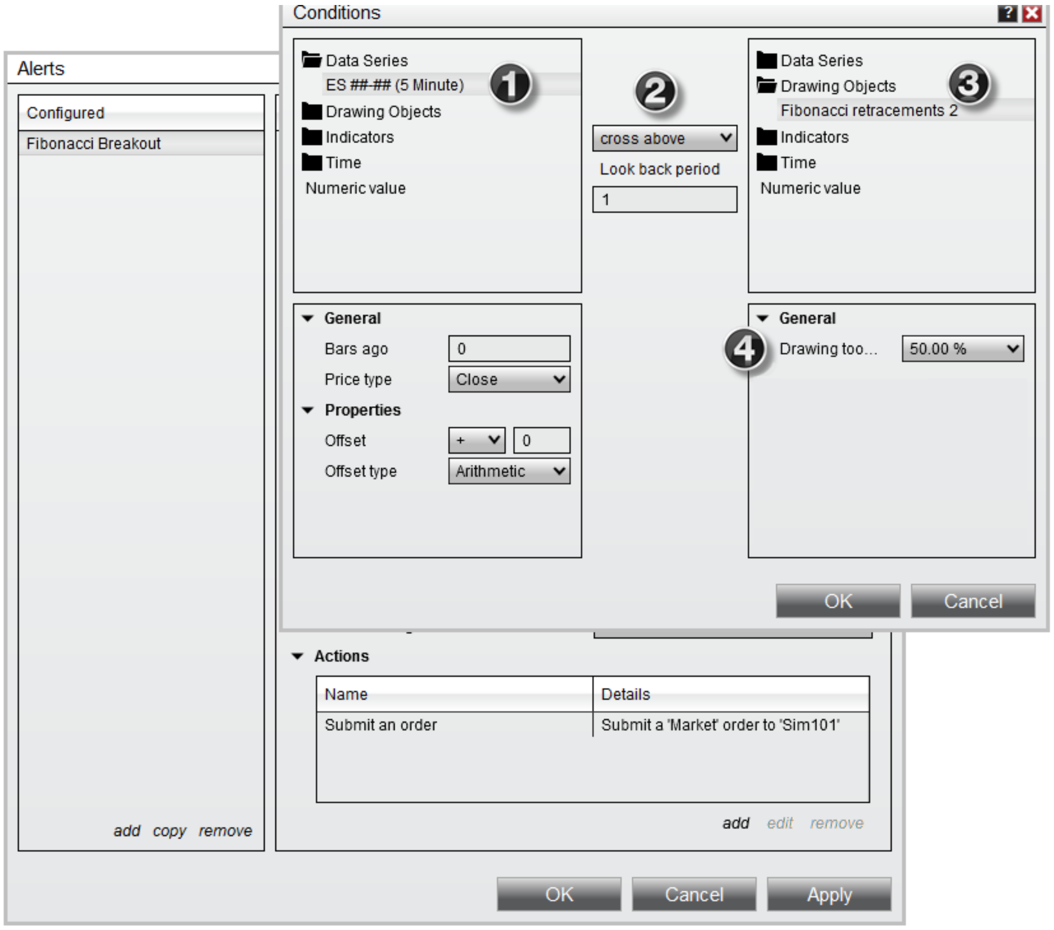
Text	Price broke 50% line on@NAME
Priority	Medium
Color for background	■ Black
Color for foreground	□ White

Actions

Name	Details
Submit an order	Submit a 'Market' order to 'Sim101'

Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:


Conditions

Data Series
ES ##-## (5 Minute)
Drawing Objects
Indicators
Time
Numeric value

cross above
Look back period: 1

Data Series
Drawing Objects
Fibonacci retracements 2
Indicators
Time
Numeric value

General
Bars ago: 0
Price type: Close
Properties
Offset: + 0
Offset type: Arithmetic

General
Drawing too...: 50.00 %

Alerts
Configured
Fibonacci Breakout

Actions

Name	Details
Submit an order	Submit a 'Market' order to 'Sim101'

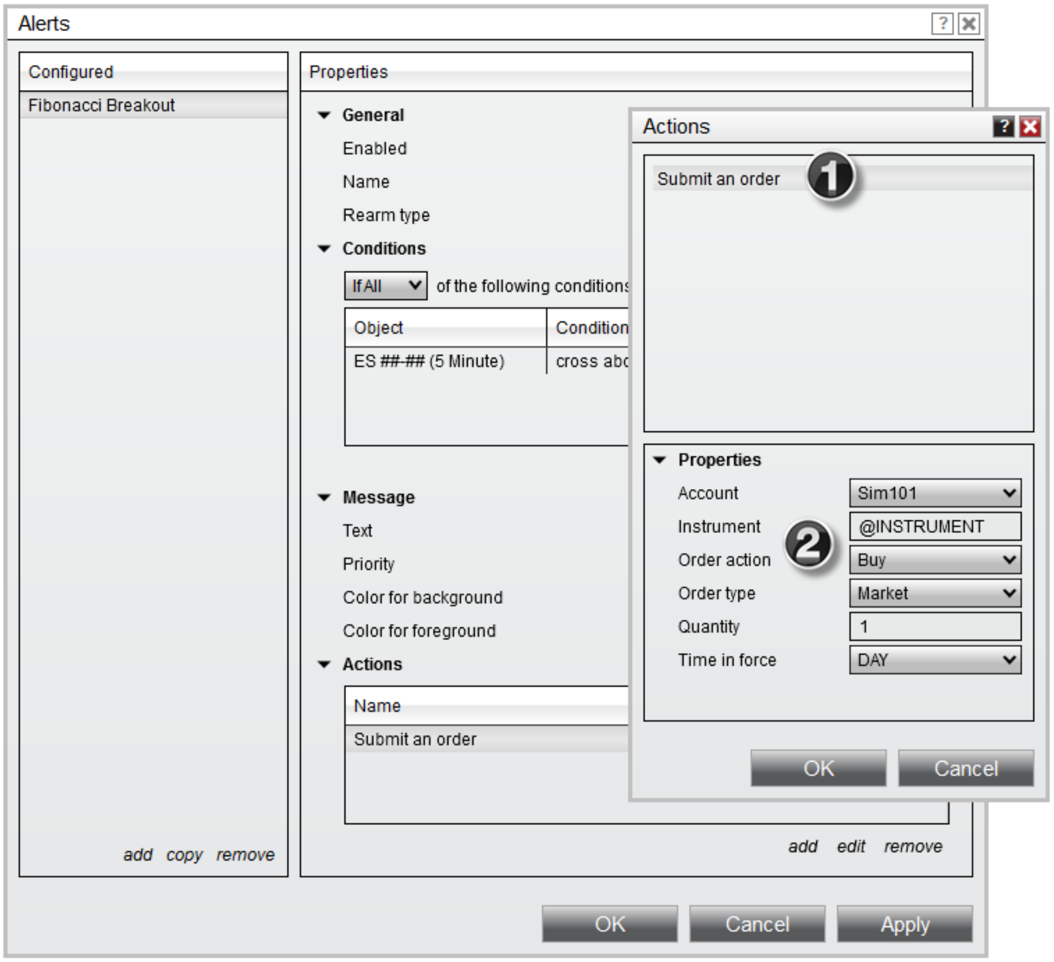
1. The ES ##-## data series is selected in the left panel, and the "Close" price type is selected (which will contain the Last price on the current bar)

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

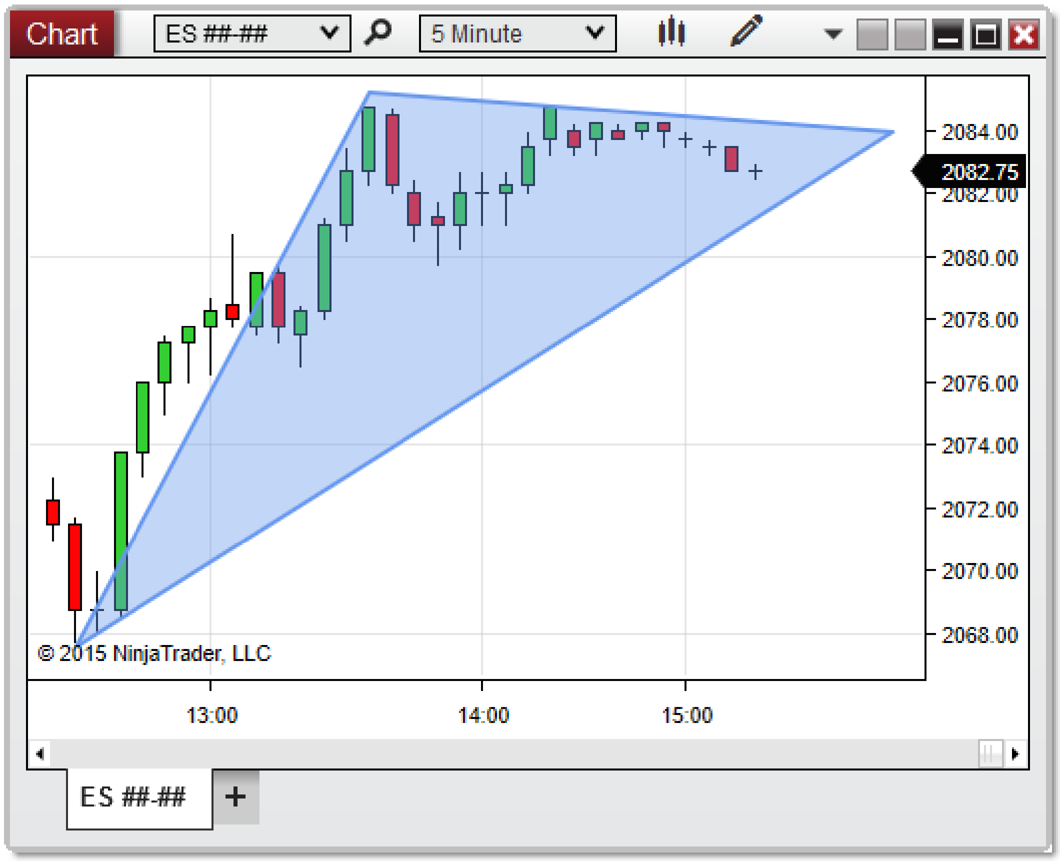
Price and User-Drawn Objects

Preparation

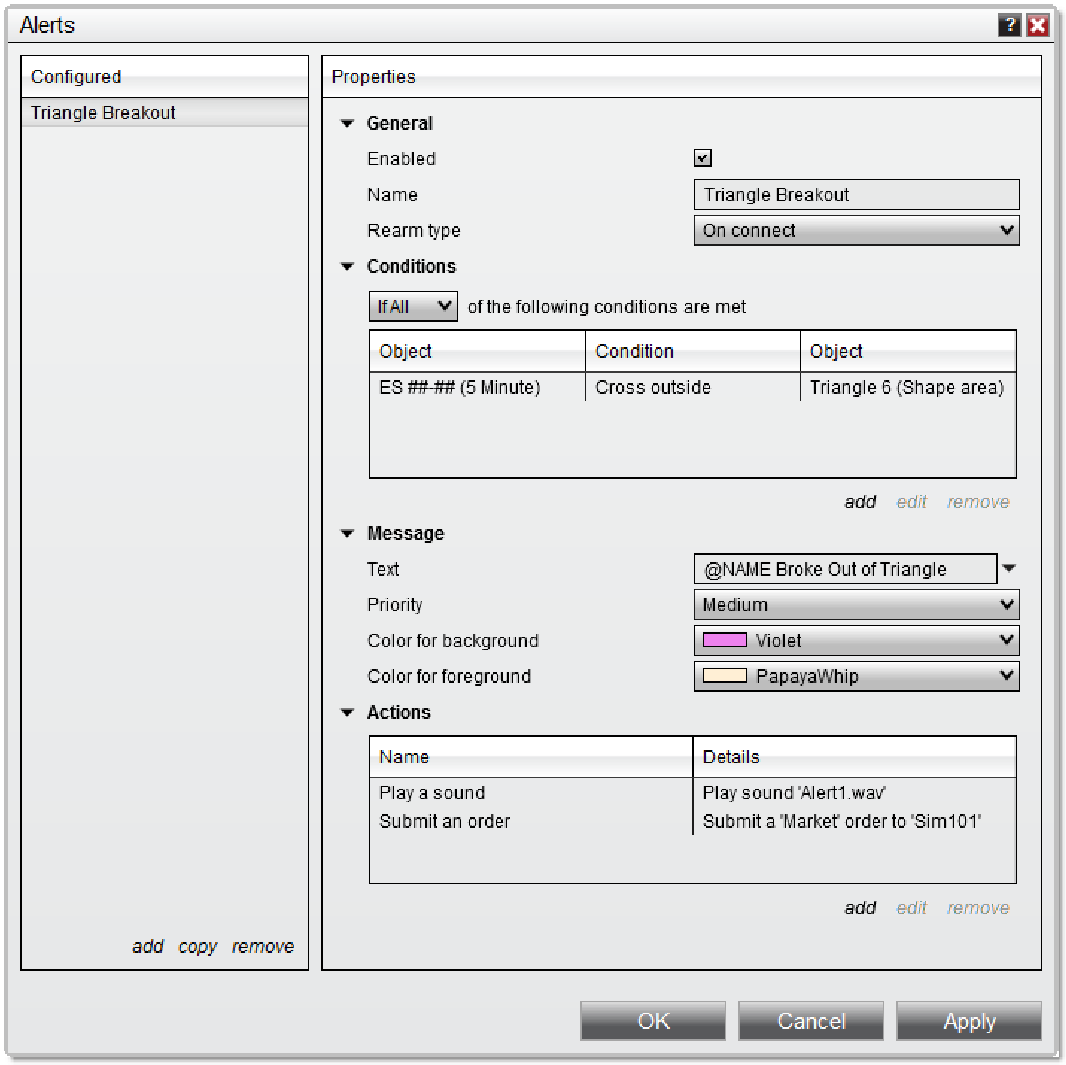
- Open a chart
- Use the Triangle **Drawing Tool** to draw a triangle on the chart. Make sure that the current market price is within the bounds of the triangle.

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.


Alerts

Configured
Triangle Breakout

Properties

General

Enabled ✓

Name: Triangle Breakout

Rearm type: On connect

Conditions

If All of the following conditions are met

Object	Condition	Object
ES ##-## (5 Minute)	Cross outside	Triangle 6 (Shape area)

Message

Text: @NAME Broke Out of Triangle

Priority: Medium

Color for background: Violet

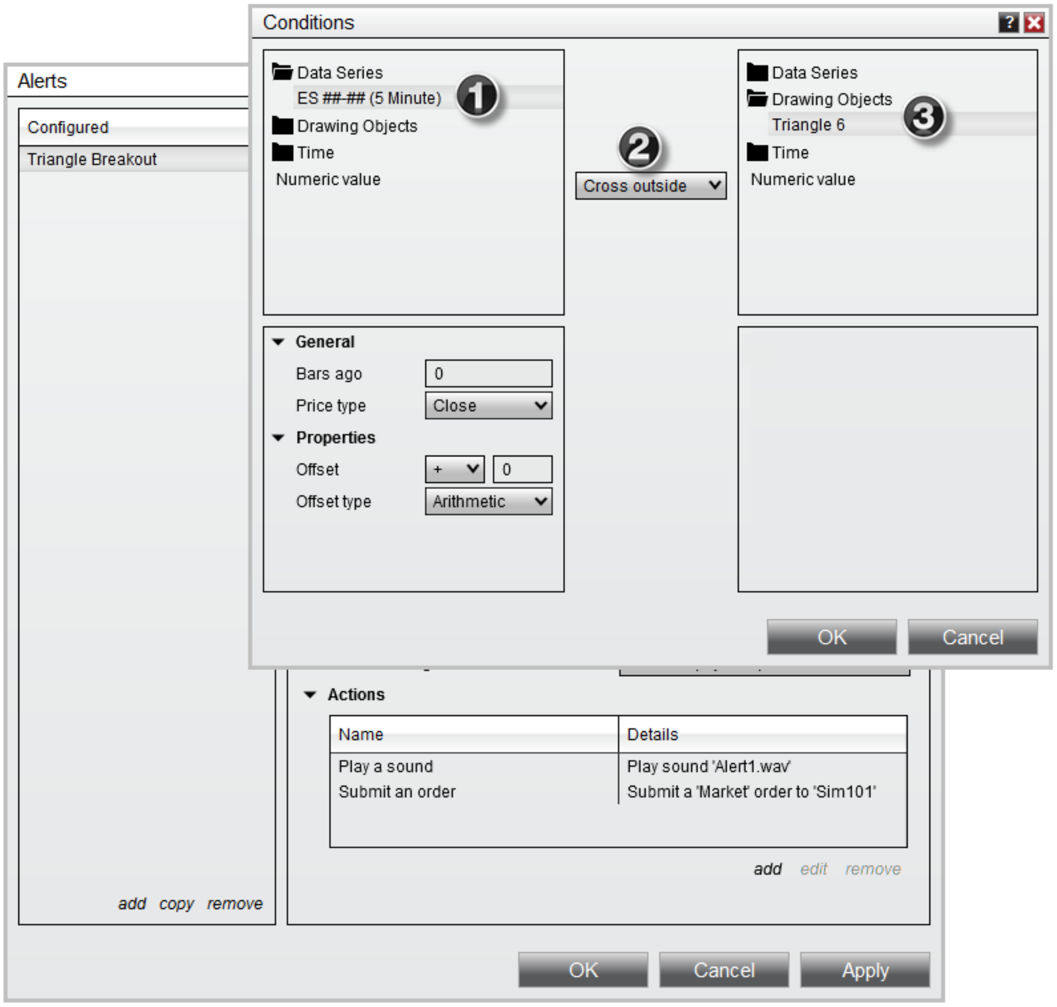
Color for foreground: PapayaWhip

Actions

Name	Details
Play a sound	Play sound 'Alert1.wav'
Submit an order	Submit a 'Market' order to 'Sim101'

Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:


Conditions

Data Series
ES ##-## (5 Minute)
Drawing Objects
Time
Numeric value

Cross outside

Data Series
Drawing Objects
Triangle 6
Time
Numeric value

General
Bars ago: 0
Price type: Close
Properties
Offset: + 0
Offset type: Arithmetic

Alerts
Configured
Triangle Breakout

Actions

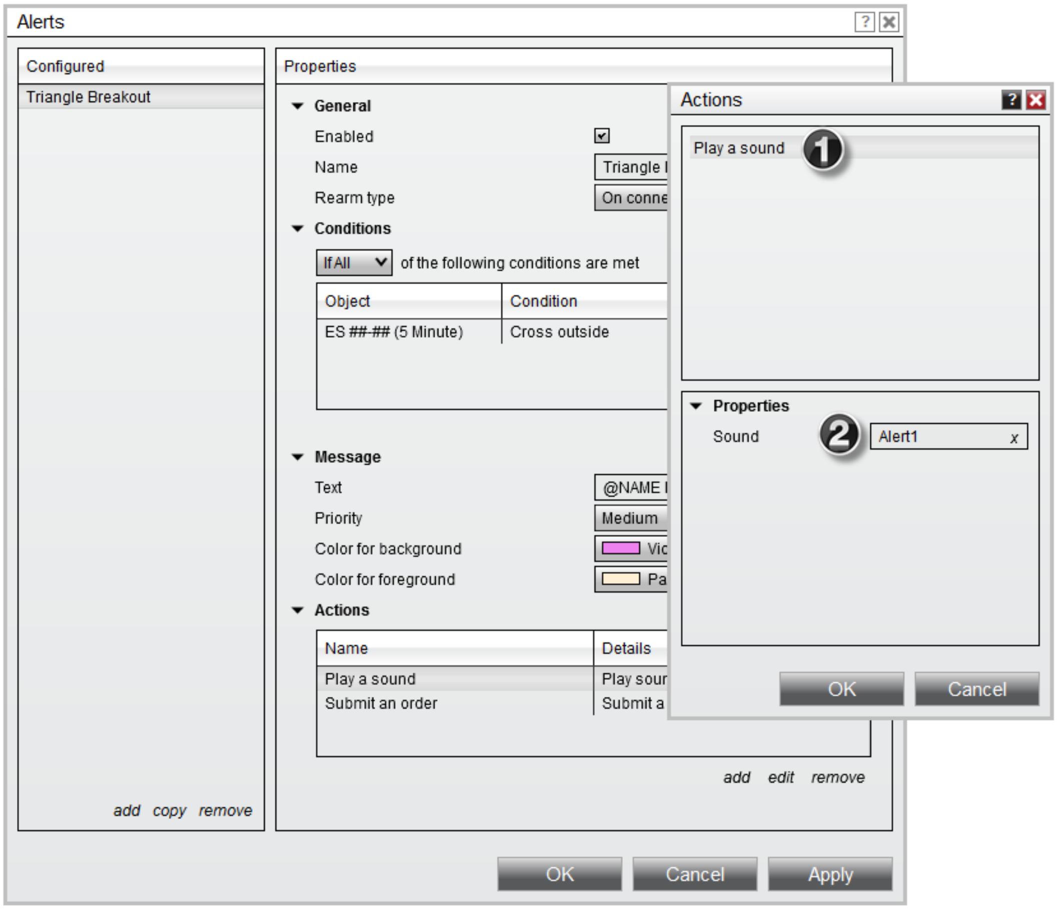
Name	Details
Play a sound	Play sound 'Alert1.wav'
Submit an order	Submit a 'Market' order to 'Sim101'

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

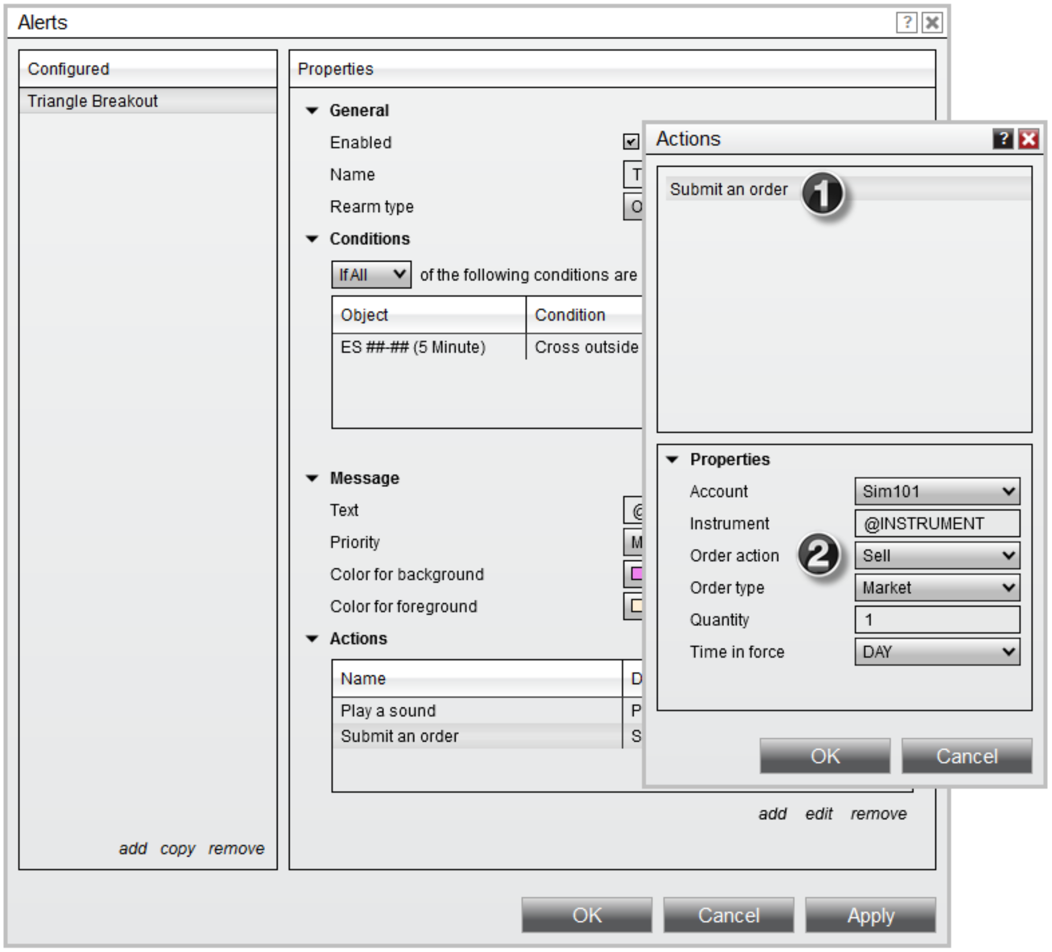
3. The custom-drawn triangle drawn on the chart is selected in the right panel (your object may have a difference name)

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


Alerts

Configured
Triangle Breakout

Properties

General
Enabled ✓
Name T
Rearm type O

Conditions
If All of the following conditions are

Object	Condition
ES ##-## (5 Minute)	Cross outside

Message
Text @
Priority M
Color for background
Color for foreground

Actions

Name	D
Play a sound	P
Submit an order	S

Actions

Submit an order 1

Properties

Account	Sim101
Instrument	@INSTRUMENT
Order action	Sell
Order type	Market
Quantity	1
Time in force	DAY

2

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**.

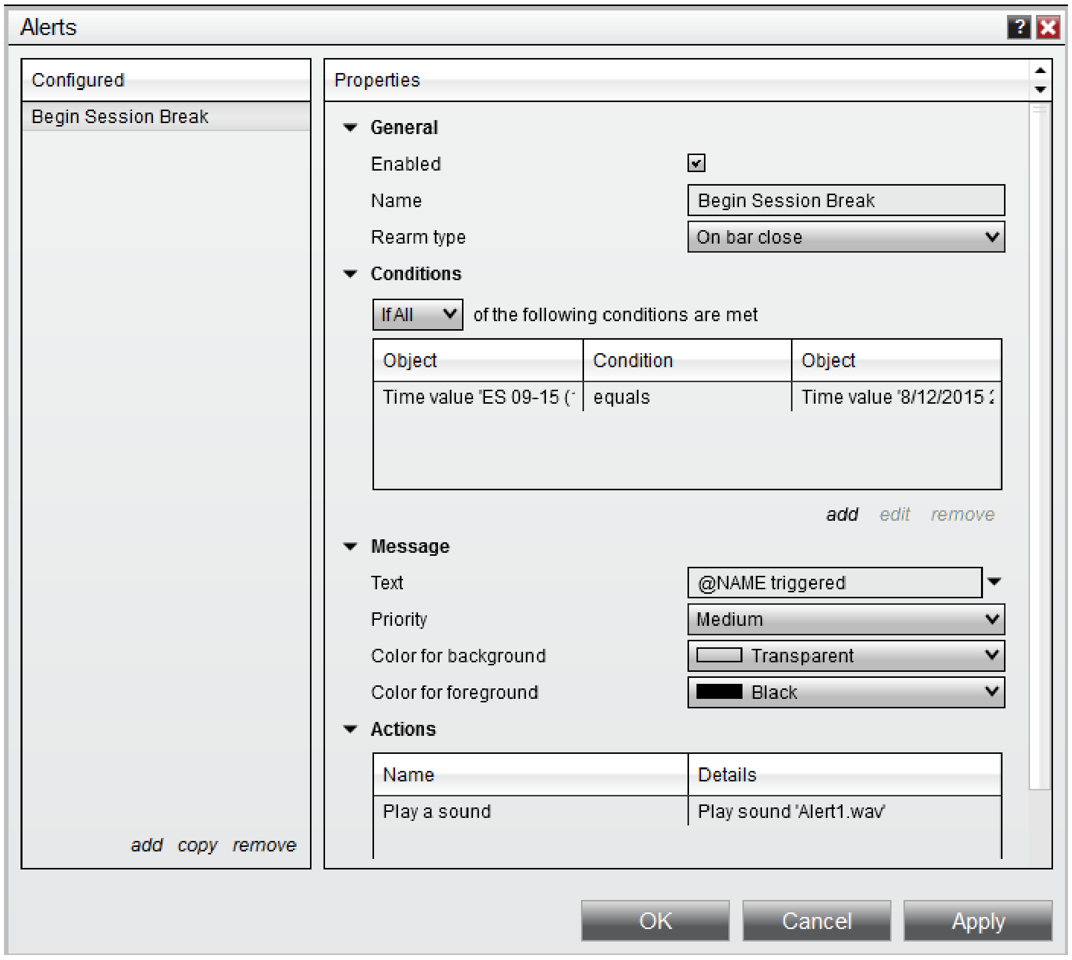
Time-Based Alert

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 equal to 2:15pm. The image below shows the fully configured alert.


Alerts

Configured
Begin Session Break

add copy remove

Properties

General

Enabled	✓
Name	Begin Session Break
Rearm type	On bar close

Conditions

If All of the following conditions are met

Object	Condition	Object
Time value 'ES 09-15 ('	equals	Time value '8/12/2015 :

add edit remove

Message

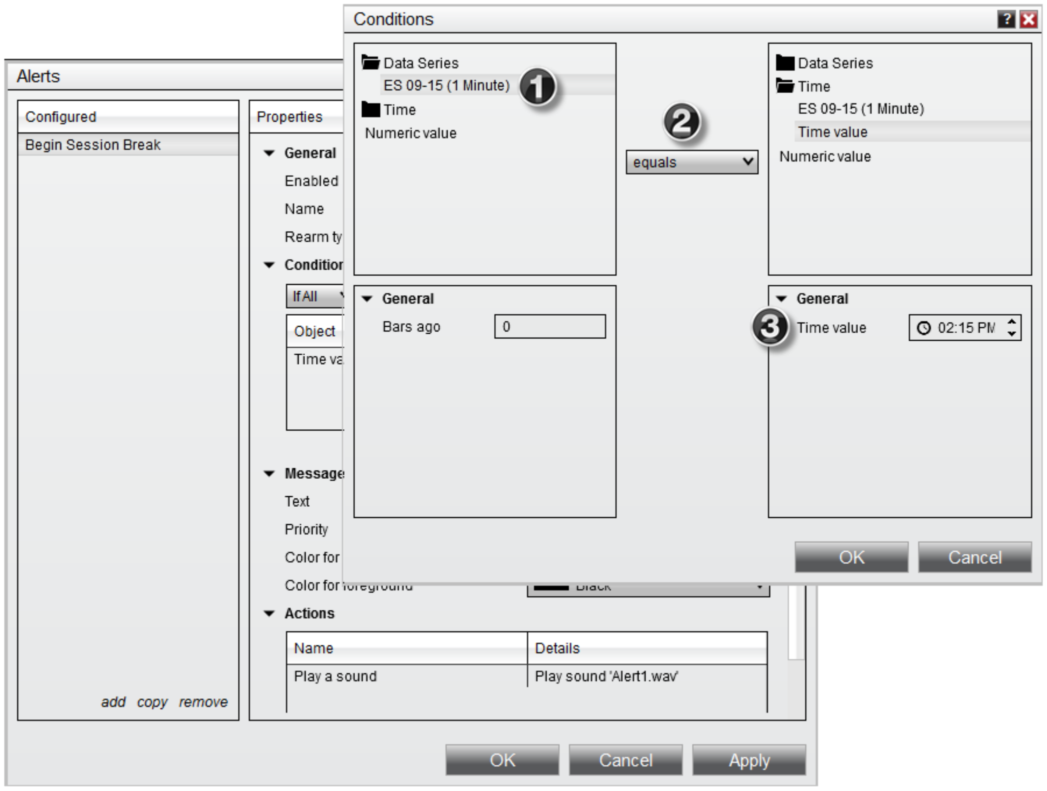
Text	@NAME triggered
Priority	Medium
Color for background	Transparent
Color for foreground	Black

Actions

Name	Details
Play a sound	Play sound 'Alert1.wav'

Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:

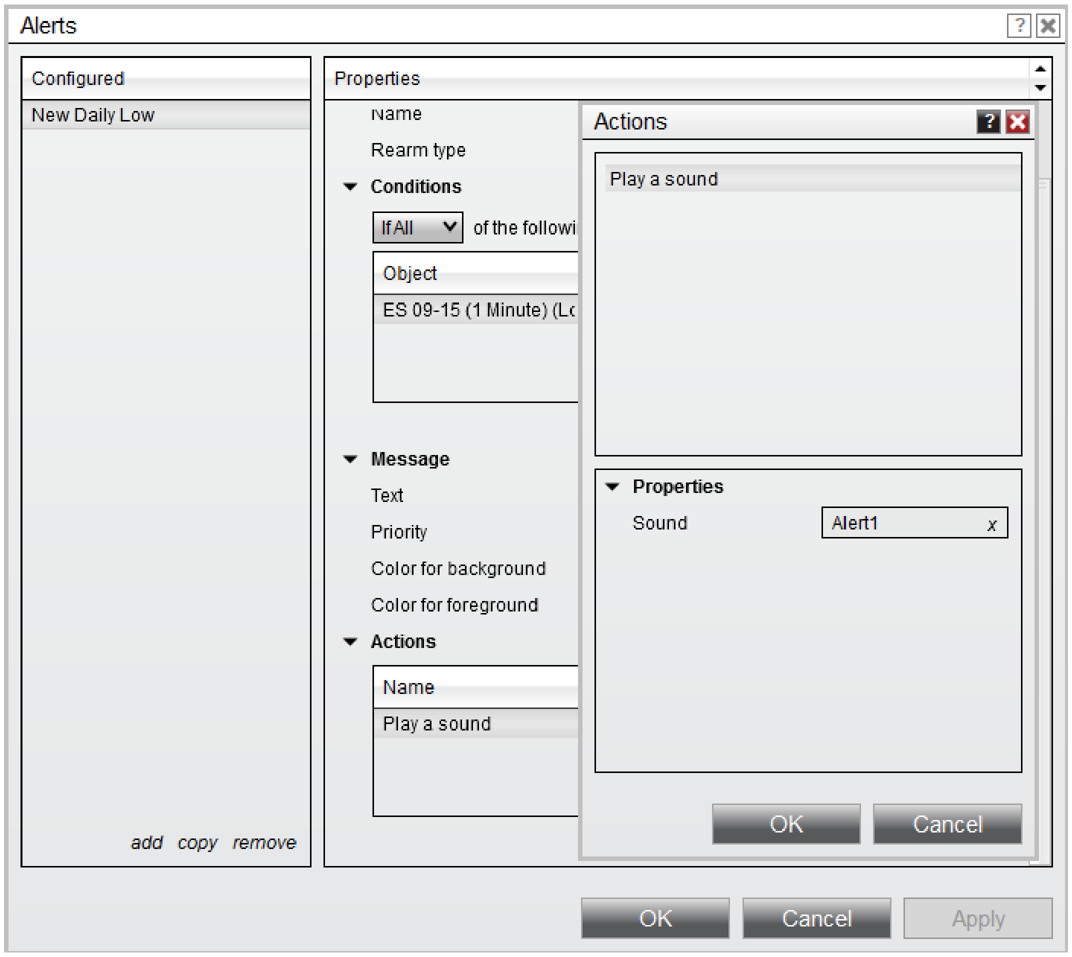


1. The ES ##-## data series is selected in the left panel

2. The "Equals" 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 current time of the ES data series is equal to 2:15pm."**



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

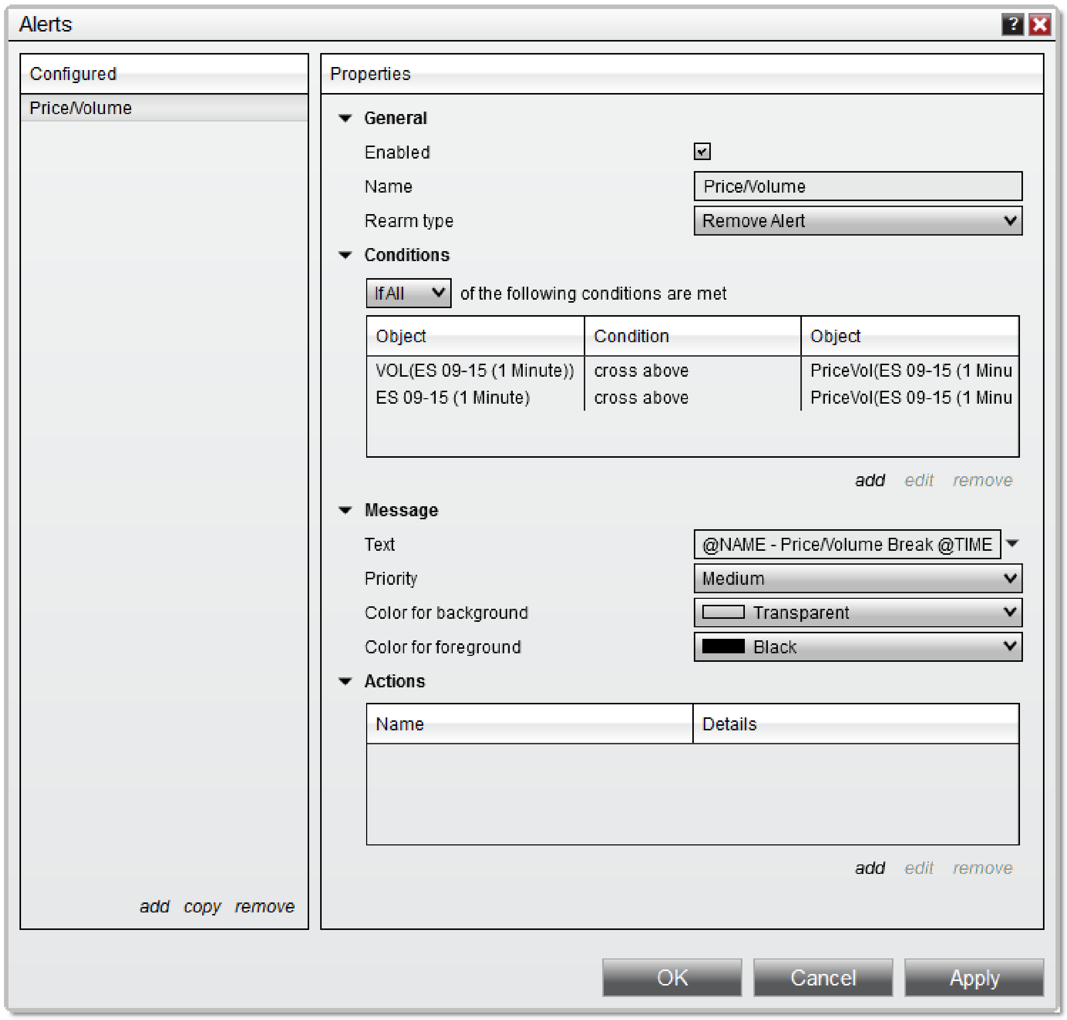
Conditions for custom indicators

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.



Alerts

Configured: Price/Volume

Properties

General
- Enabled: ✓
- Name: Price/Volume
- Rearm type: Remove Alert

Conditions

If All of the following conditions are met

Object	Condition	Object
VOL(ES 09-15 (1 Minute))	cross above	PriceVol(ES 09-15 (1 Minu
ES 09-15 (1 Minute)	cross above	PriceVol(ES 09-15 (1 Minu

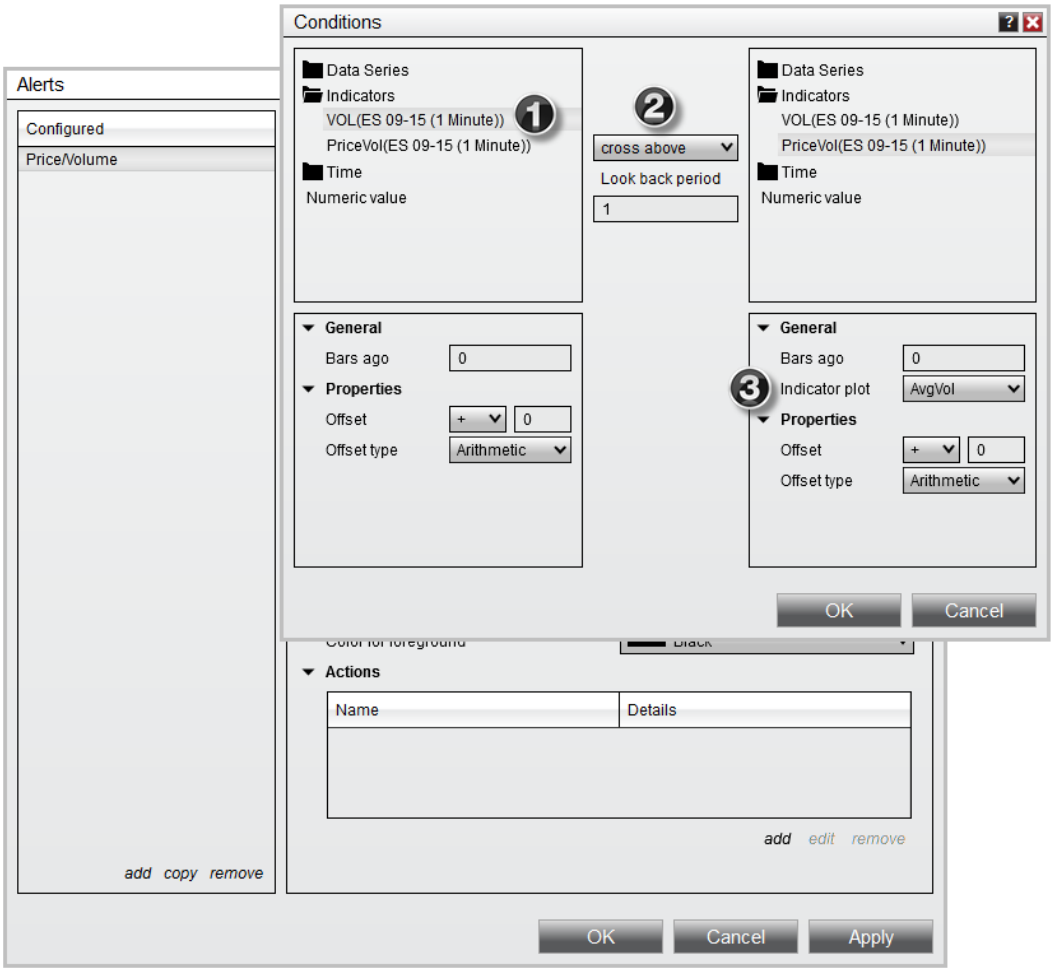
Message
- Text: @NAME - Price/Volume Break @TIME
- Priority: Medium
- Color for background: Transparent
- Color for foreground: Black

Actions

Name	Details

Conditions and Actions

The Conditions and Actions windows for this alert can be seen below:



cross above
Look back period 1

Bars ago 0
Offset + 0
Offset type Arithmetic

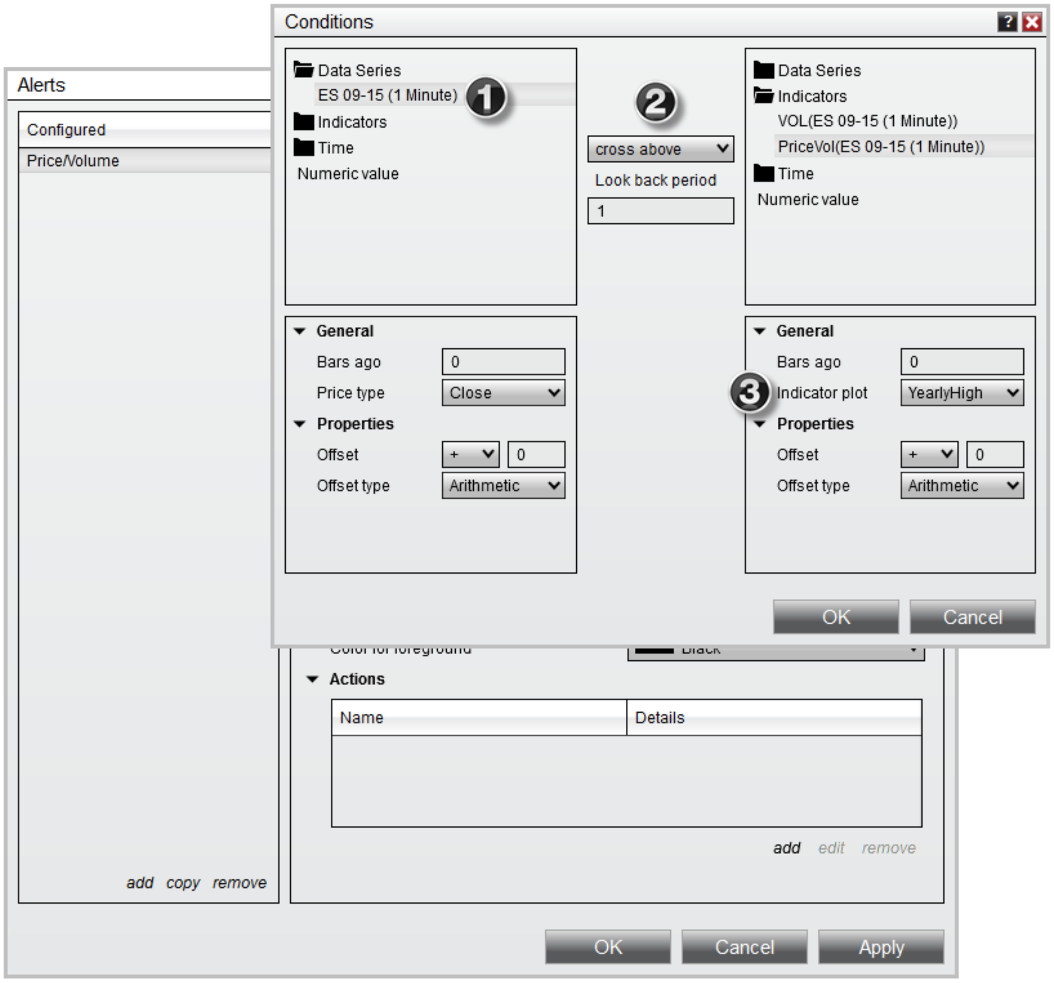
Bars ago 0
Indicator plot AvgVol
Offset + 0
Offset type Arithmetic

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."**


Conditions

Data Series
ES 09-15 (1 Minute)
Indicators
Time
Numeric value

cross above
Look back period
1

Data Series
Indicators
VOL(ES 09-15 (1 Minute))
PriceVol(ES 09-15 (1 Minute))
Time
Numeric value

General	
Bars ago	0
Price type	Close
Properties	
Offset	+ 0
Offset type	Arithmetic

General	
Bars ago	0
Indicator plot	YearlyHigh
Properties	
Offset	+ 0
Offset type	Arithmetic

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.

11.3 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.

› Using the Alerts Log Window
› Alerts Log Properties

11.3.1 Using the Alerts Log Window

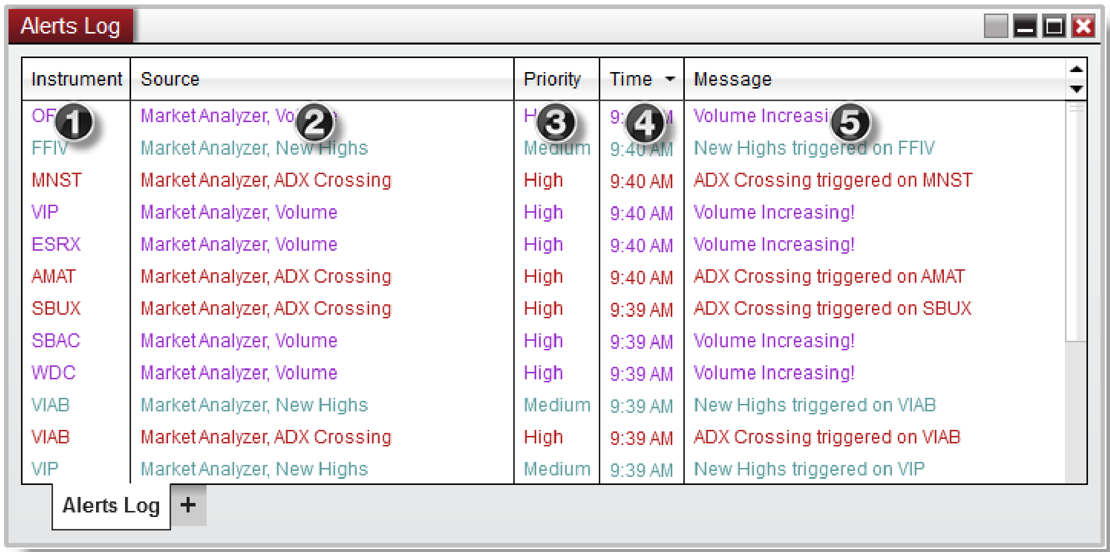
The **Alerts Log** window displays information for each alert that is triggered within NinjaTrader.

Understanding the Alerts Log window

Alerts Log Window Display

When an alert is triggered, the following information is available in the **Alerts Log** window:

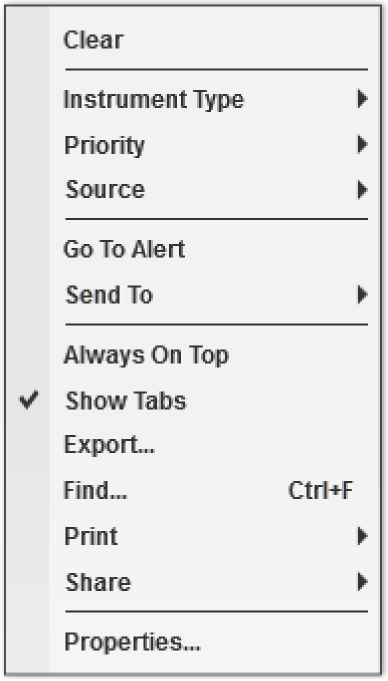
1. Instrument name
2. Source of alert
3. Priority of alert
4. Time of the alert
5. User defined message



Instrument	Source	Priority	Time	Message
OF	Market Analyzer, Vo	H	9: M	Volume Increasi
FFIV	Market Analyzer, New Highs	Medium	9:40 AM	New Highs triggered on FFIV
MNST	Market Analyzer, ADX Crossing	High	9:40 AM	ADX Crossing triggered on MNST
VIP	Market Analyzer, Volume	High	9:40 AM	Volume Increasing!
ESRX	Market Analyzer, Volume	High	9:40 AM	Volume Increasing!
AMAT	Market Analyzer, ADX Crossing	High	9:40 AM	ADX Crossing triggered on AMAT
SBUX	Market Analyzer, ADX Crossing	High	9:39 AM	ADX Crossing triggered on SBUX
SBAC	Market Analyzer, Volume	High	9:39 AM	Volume Increasing!
WDC	Market Analyzer, Volume	High	9:39 AM	Volume Increasing!
VIAB	Market Analyzer, New Highs	Medium	9:39 AM	New Highs triggered on VIAB
VIAB	Market Analyzer, ADX Crossing	High	9:39 AM	ADX Crossing triggered on VIAB
VIP	Market Analyzer, New Highs	Medium	9:39 AM	New Highs triggered on VIP

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	Forex, Future, Index, Option, Stock, CFD

Type	
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.

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.

11.3.2 Alerts Log Properties

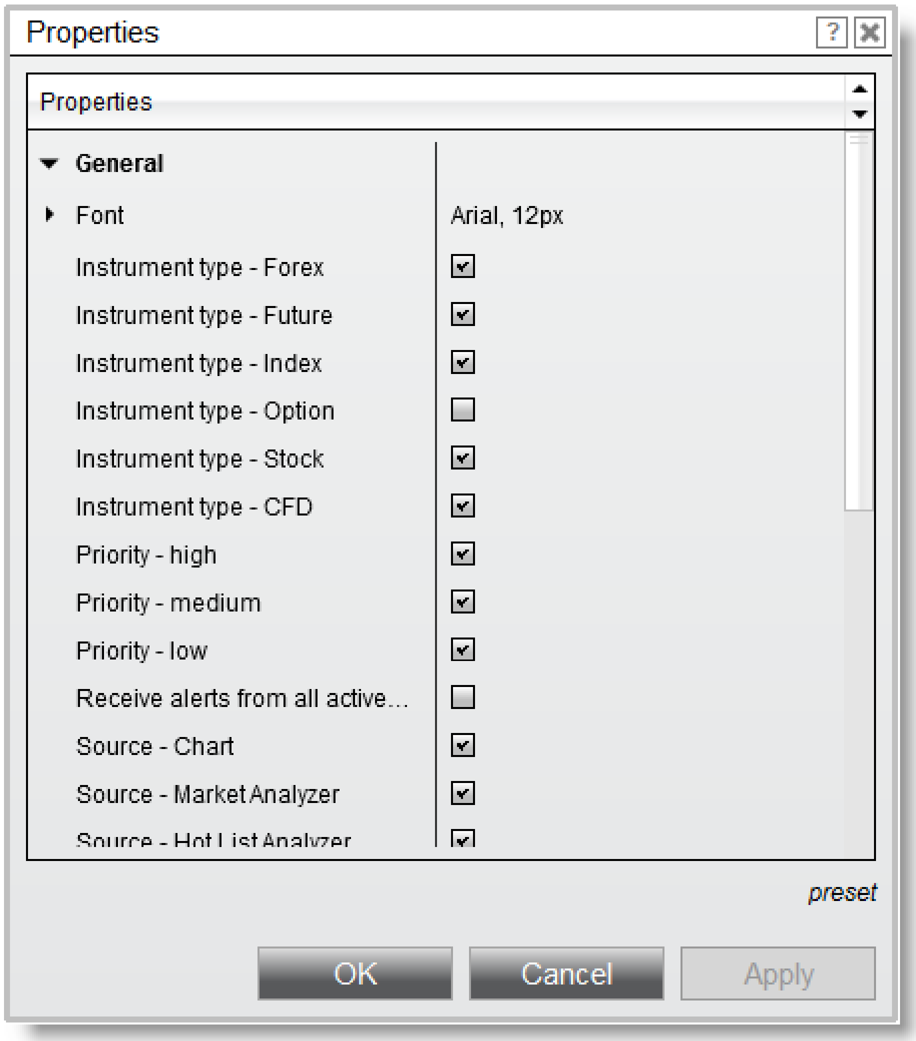
The **Alerts Log** window can be customized through the **Alerts Log Properties** window.

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:



Properties

Properties

General

Font	Arial, 12px
Instrument type - Forex	✓
Instrument type - Future	✓
Instrument type - Index	✓
Instrument type - Option	
Instrument type - Stock	✓
Instrument type - CFD	✓
Priority - high	✓
Priority - medium	✓
Priority - low	✓
Receive alerts from all active...	
Source - Chart	✓
Source - Market Analyzer	✓
Source - Hot List Analyzer	✓

Property Definitions

General	
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.

11.3.3 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.

11.4 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

11.4.1 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).

› What can I do and how?
› Commands and Valid Parameters
› Initialization
› File Interface
› DLL Interface
› TradeStation

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.

11.4.1.1 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

11.4.1.2 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
INSTRUMENT	Instrument name
ACTION	BUY, SELL
QTY	Any integer value
ORDER TYPE	MARKET, LIMIT, STOP, STOPLIMIT
LIMIT	Any decimal value (use decimals not commas

PRICE	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**	**Aty**	**OrderTYpe**	**LimitPrice**	**StopPrice**	**TIF**	**OCOID**	**OrderID**	**Strategy**	**StrategyID**
CANCEL										R		O

CANCELALLORDERS												
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.

11.4.1.3 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

11.4.1.4 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

11.4.1.4.1 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.

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]
```

11.4.1.4.2 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.

11.4.1.5 DLL Interface

DLL Functions Overview

The native C/C++ DLL Interface functions are contained in NTDirect.dll located in the C:\WINDOWS\system32 folder. 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

11.4.1.5.1 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. A return value of 0 indicates success and -1 indicates an error. 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.

11.4.1.6 TradeStation Email Integration

The TradeStation Email Interface is targeted toward individuals who are not 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, Cancelled 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

11.4.1.6.1 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

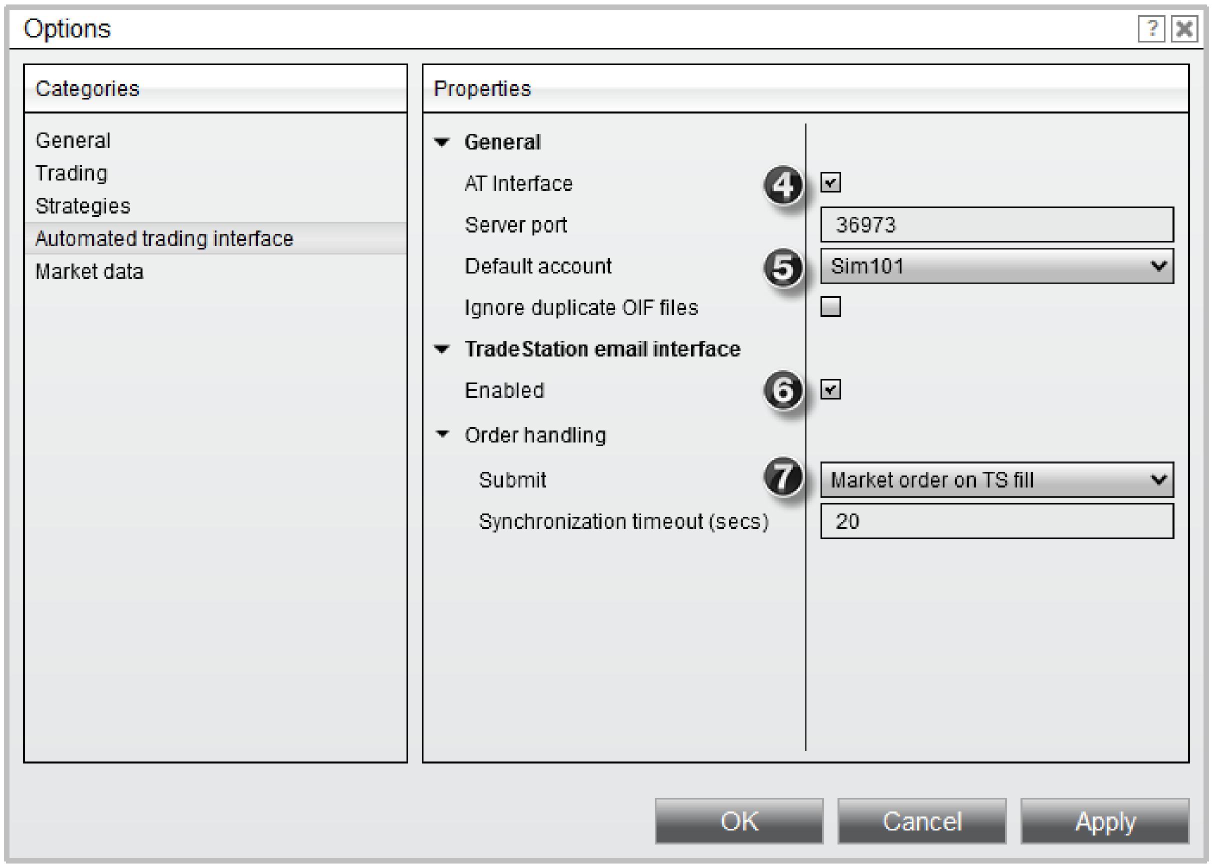
This will ensure accurate processing of your automated signals.

11.4.1.6.2 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



Options

Categories: General, Trading, Strategies, Automated trading interface, Market data

Properties

General
- AT Interface: ✓ (4)
- Server port: 36973
- Default account: Sim101 (5)
- Ignore duplicate OIF files: ☐

TradeStation email interface
- Enabled: ✓ (6)

Order handling
- Submit: Market order on TS fill (7)
- Synchronization timeout (secs): 20

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)

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

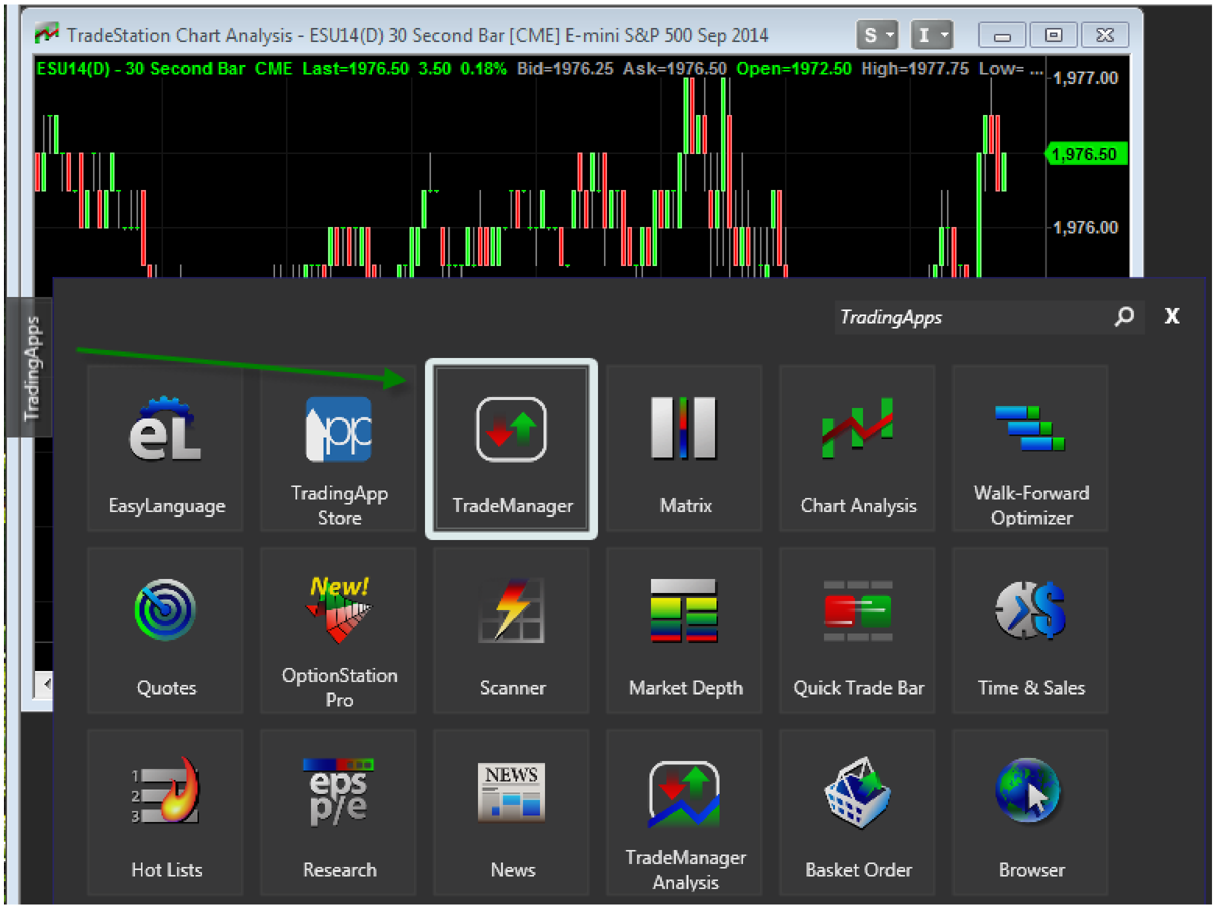
10. Antivirus software which scans outgoing emailing can impair the link between TradeStation and NinjaTrader. If your PC has Antivirus software installed and scans outgoing mail, each mail notification sent from TradeStation to NinjaTrader will be scanned and therefore add significant delay in automatically 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

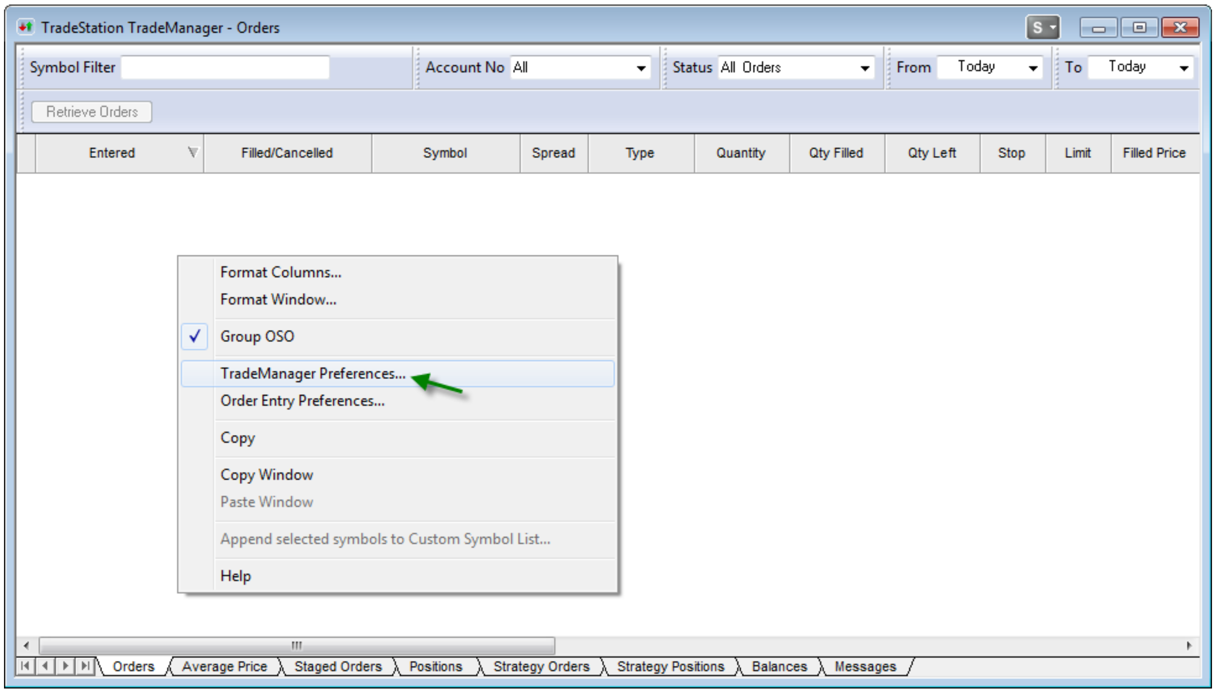
11. Start TradeStation
12. Set up your workspace options

Setting Up TradeStation Email Notification

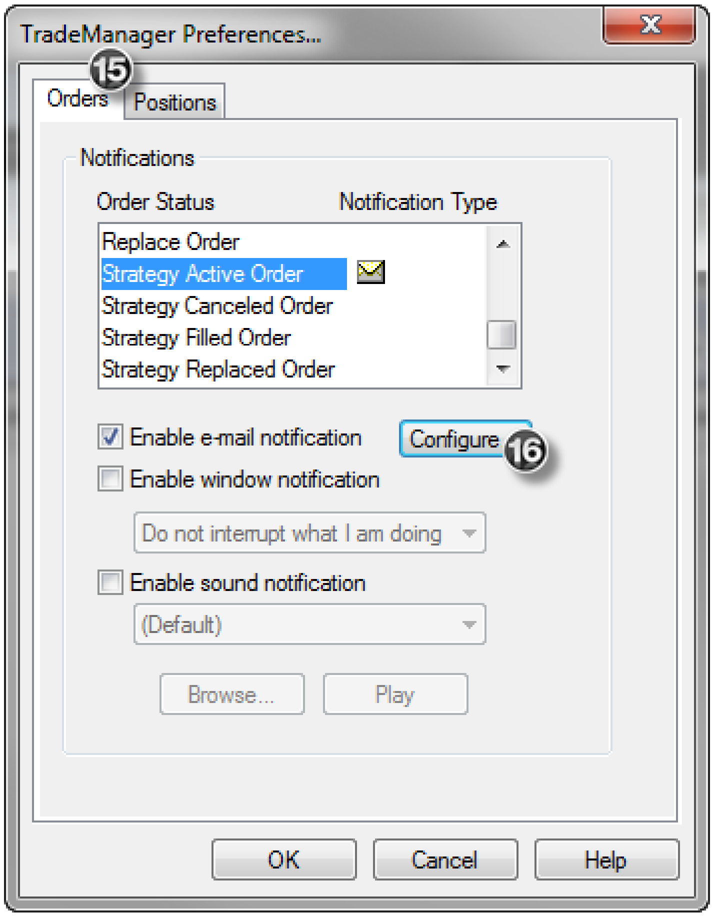
13. Add a **TradeManager** window to your workspace by clicking on the TradingApps panel on left pane as per the image below.



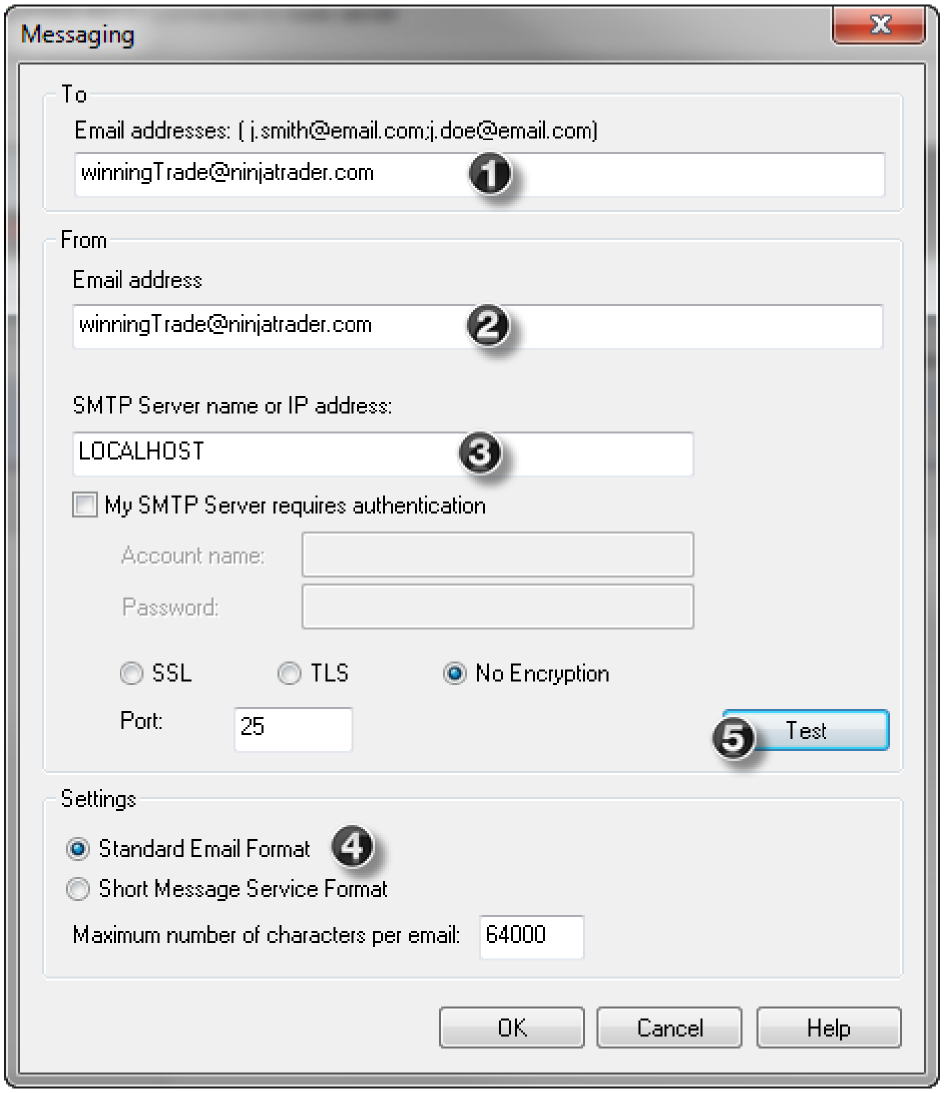
14. Once the TradeManager window appears, right click on this window, and then select the menu name TradeManager preferences



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


Messaging

To

Email addresses: (j.smith@email.com;j.doe@email.com)

winningTrade@ninjatrader.com

From

Email address

winningTrade@ninjatrader.com

SMTP Server name or IP address:

LOCALHOST

My SMTP Server requires authentication

Account name:

Password:

SSL TLS [✓ No Encryption]

Port: 25

Settings

[✓ Standard Email Format]

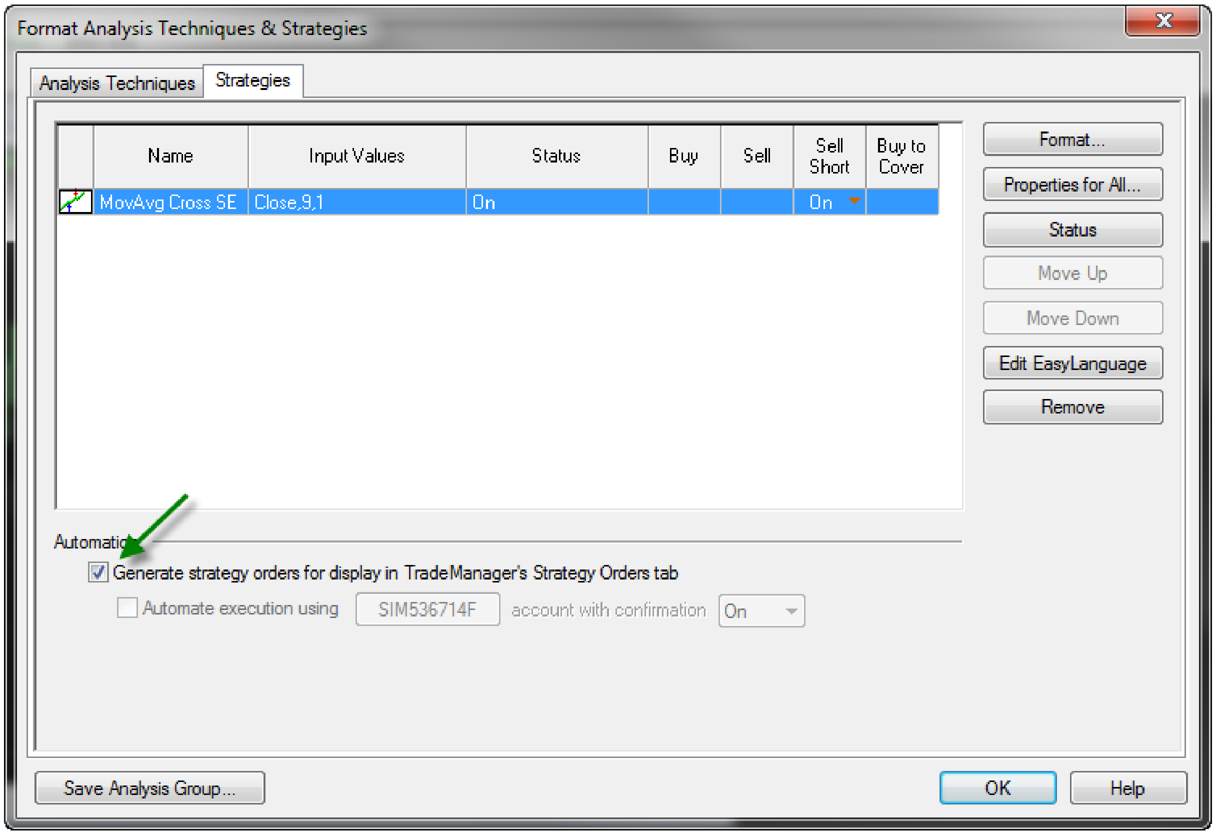
Short Message Service Format

Maximum number of characters per email: 64000

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



	Name	Input Values	Status	Buy	Sell	Sell Short	Buy to Cover
	MovAvg Cross SE	Close,9,1	On			On	

Automation

☑ Generate strategy orders for display in TradeManager's Strategy Orders tab

☐ Automate execution using SIM536714F account with confirmation On

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!

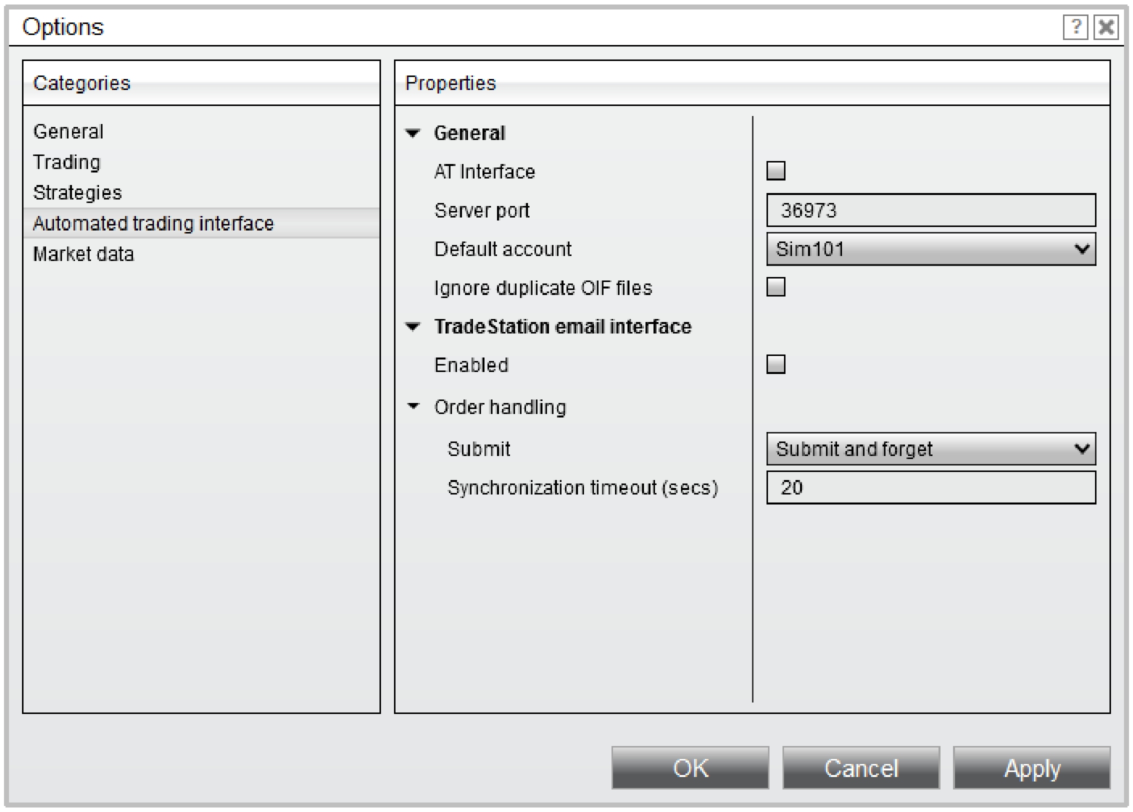
11.4.1.6.3 Symbol Mapping

Please see the TradeStation Symbol Mapping section.

11.4.1.6.4 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.



General
- AT Interface: ☐
- Server port: 36973
- Default account: Sim101
- Ignore duplicate OIF files: ☐

TradeStation email interface
- Enabled: ☐

Order handling
- Submit: Submit and forget
- Synchronization timeout (secs): 20

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".

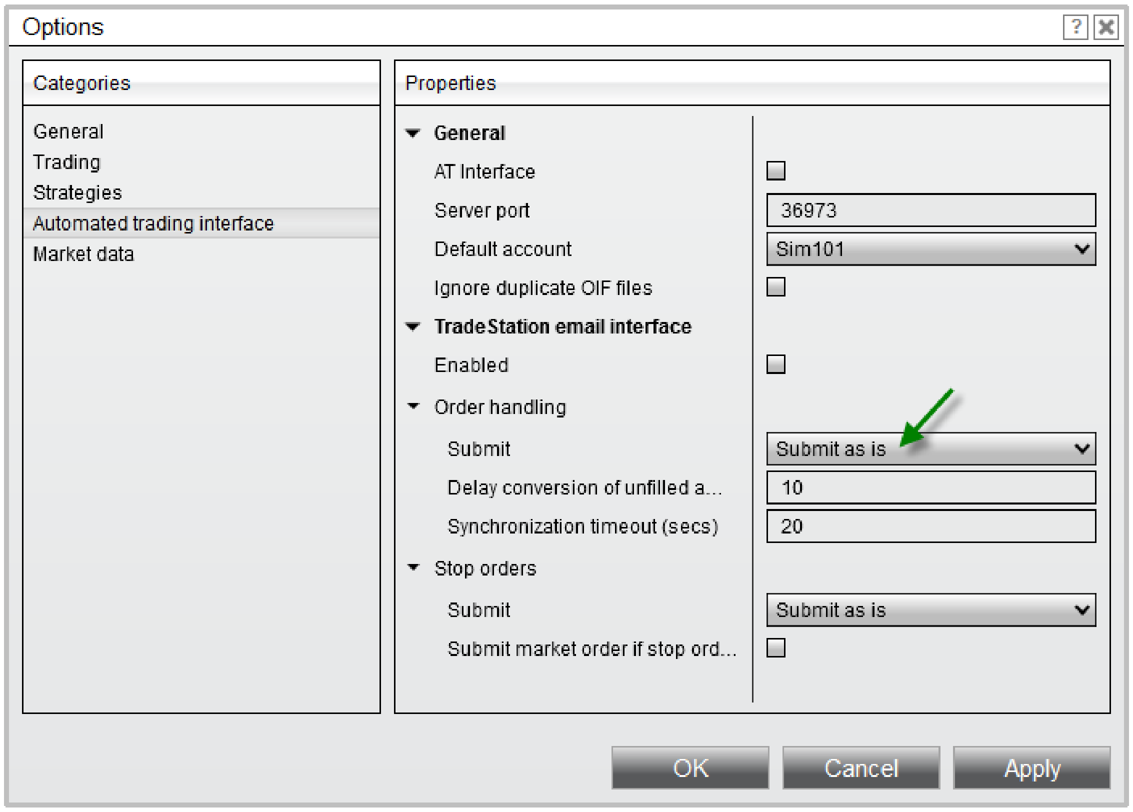
Understanding special handling for FX through GAIN Capital/FOREX.com

Special Handling for FX through GAIN Capital/FOREX.com

GAIN 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 GAIN 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.

11.4.1.6.5 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**".


Options

Categories: General, Trading, Strategies, Automated trading interface, Market data

Properties

General	
AT Interface	☐
Server port	36973
Default account	Sim101
Ignore duplicate OIF files	☐
TradeStation email interface	
Enabled	☐
Order handling	
Submit	Submit as is
Delay conversion of unfilled a...	10
Synchronization timeout (secs)	20
Stop orders	
Submit	Submit as is
Submit market order if stop ord...	☐

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.

11.4.1.6.6 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;"

11.4.2 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.

11.4.2.1 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.

11.4.2.2 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 8 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 8 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

11.4.2.3 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.**

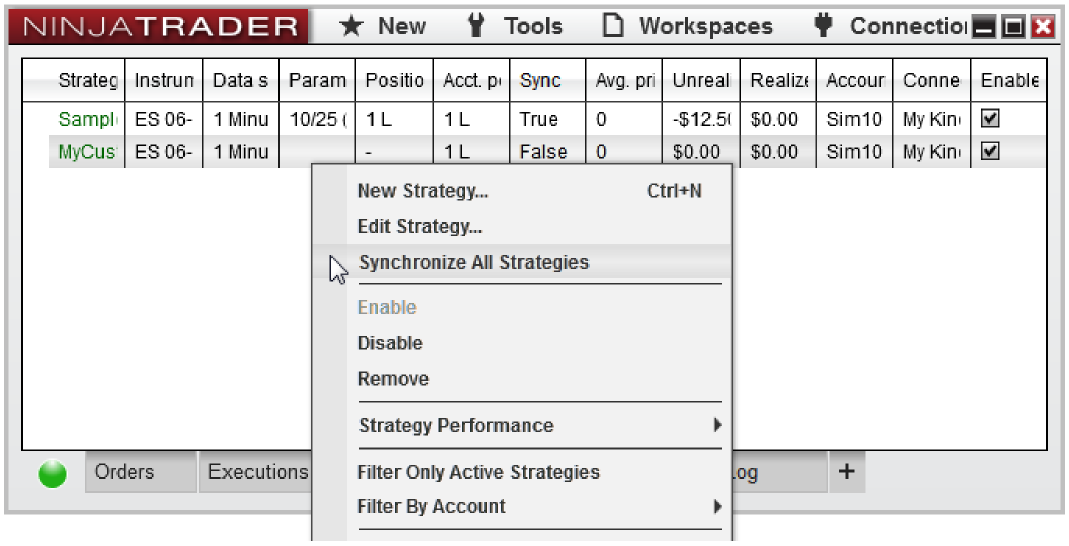
- Only one strategy with this setting can be started at a time for an individual account and instrument.
- The account and instrument the strategy is started on must not have any working orders which were submitted outside of the strategy. If an order is detected, the strategy can not be started until 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.**



Strateg	Instrun	Data s	Param	Positio	Acct. p	Sync	Avg. pri	Unreal	Realiz	Accour	Conne	Enable
Sampl	ES 06-	1 Minu	10/25 (	1 L	1 L	True	0	-$12.5(	$0.00	Sim10	My Kin	✓
MyCus	ES 06-	1 Minu		-	1 L	False	0	$0.00	$0.00	Sim10	My Kin	✓

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.

11.4.2.4 Running a NinjaScriptStrategy from a Chart

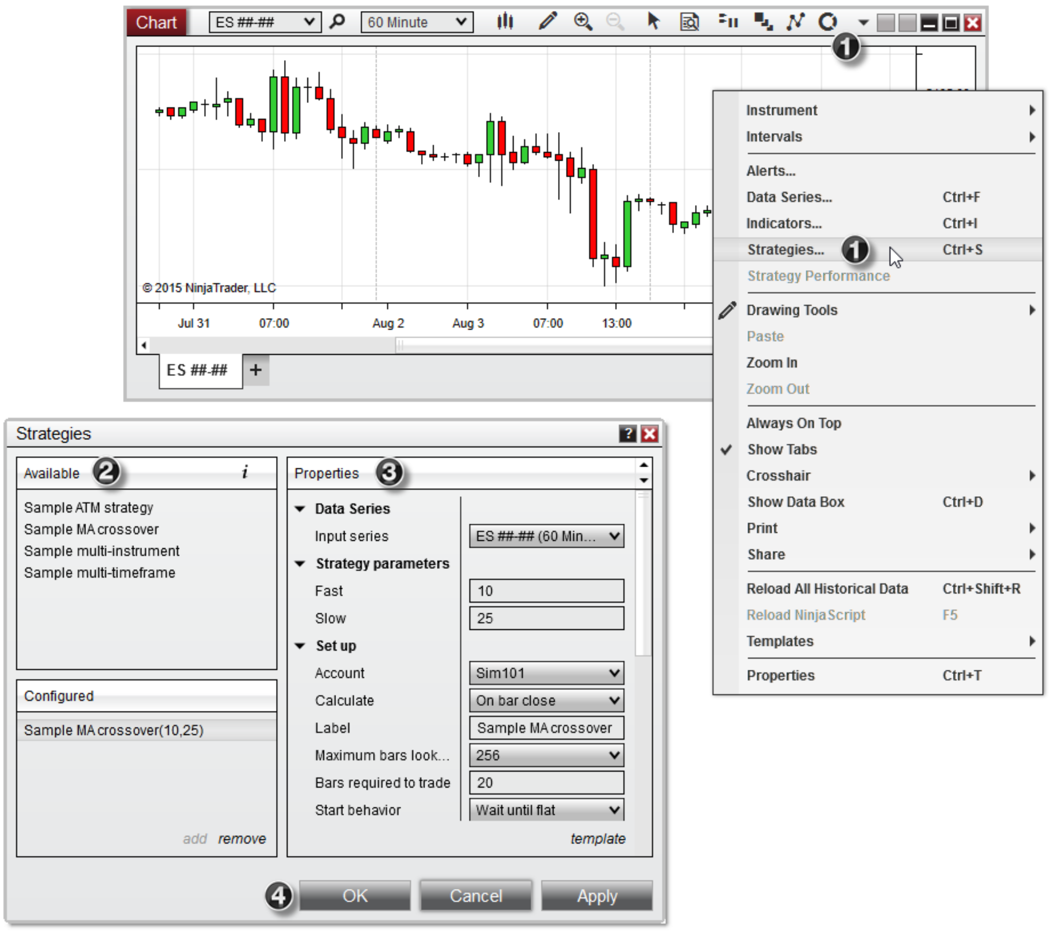
You can run a NinjaScript strategy in real-time in a live or simulation account within a NinjaTrader chart.

▽ 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.



Data Series

Input series	ES ##-## (60 Min...

Strategy parameters

Fast	10
Slow	25

Set up

Account	Sim101
Calculate	On bar close
Label	Sample MA crossover
Maximum bars look...	256
Bars required to trade	20
Start behavior	Wait until flat

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.
- Running and disabled strategies are displayed in the **Control Center** Strategies tab

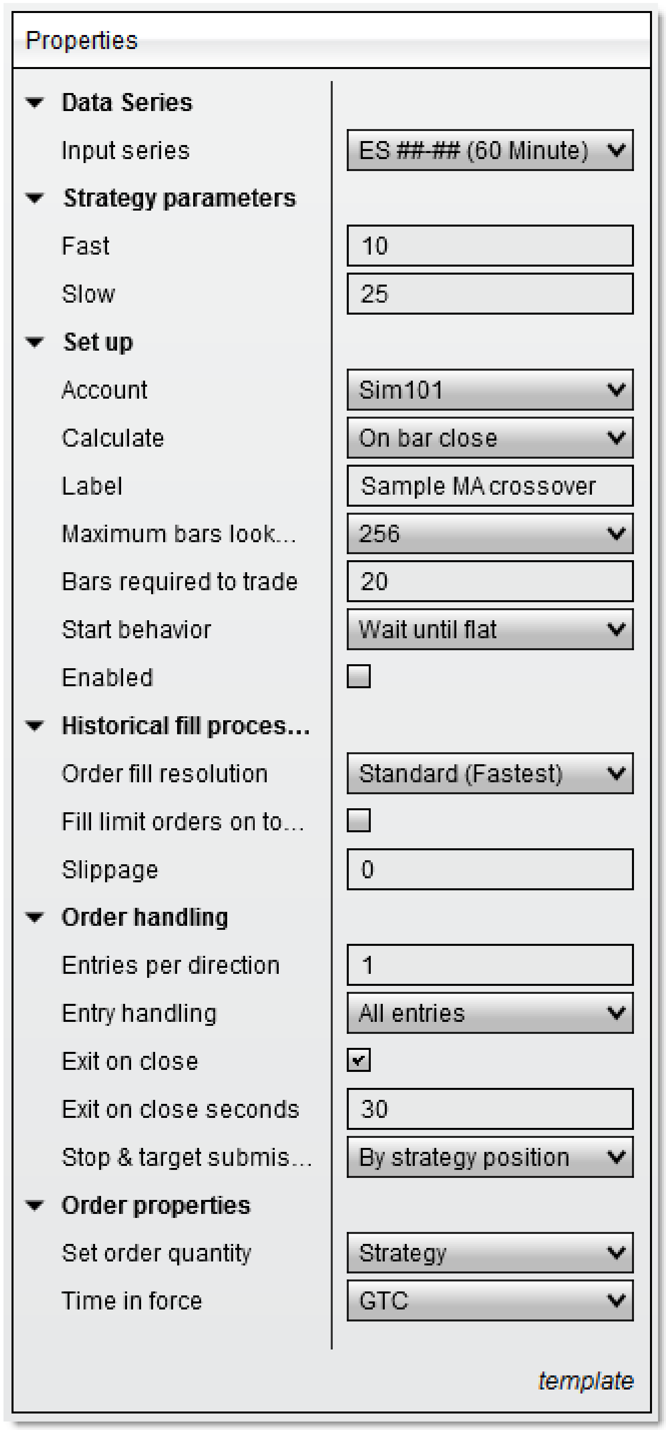
Terminating a NinjaScript Strategy

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:



Properties

Data Series

Input series	ES ##-## (60 Minute)

Strategy parameters

Fast	10
Slow	25

Set up

Account	Sim101
Calculate	On bar close
Label	Sample MA crossover
Maximum bars look...	256
Bars required to trade	20
Start behavior	Wait until flat
Enabled	☐

Historical fill proces...

Order fill resolution	Standard (Fastest)
Fill limit orders on to...	☐
Slippage	0

Order handling

Entries per direction	1
Entry handling	All entries
Exit on close	✓
Exit on close seconds	30
Stop & target submis...	By strategy position

Order properties

Set order quantity	Strategy
Time in force	GTC

template

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 • "Account Size" - Uses the "Max Order Size" property in the Risk Template associated with the instrument and account on which the strategy is running
Time in force	Sets the order's time in force. Possible values are DAY and GTC

11.4.2.5 Running a NinjaScriptStrategy 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)

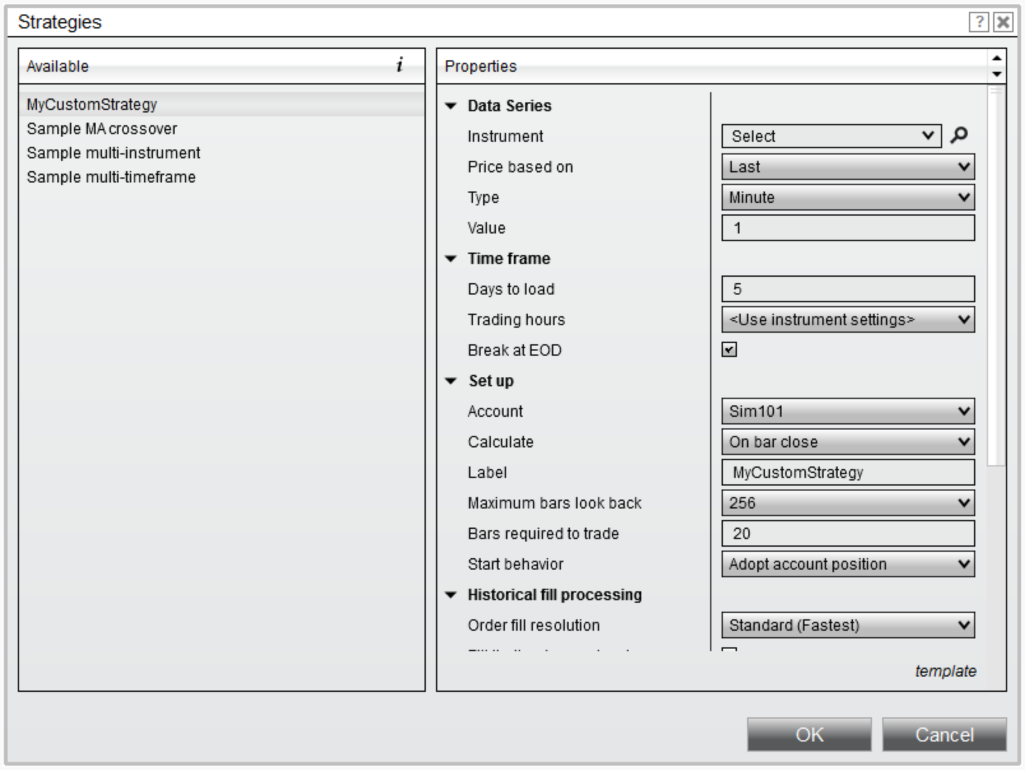
The image below shows the adjustable properties for a strategy available in the Strategies tab of the Control Center (see the "*How to run a NinjaScript strategy in* *from the Strategies tab*" section above):

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 "Understanding Historical Data" 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	Enables or disables the filling of limit orders on a single touch of price action.
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	
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 submissi on	Sets how stop and target orders are submitted
Order properti es	
Set order quantity	Sets how the order size is determined, options are: • **Account size** - Allows you to set a virtual account value that is used to determine maximum order size based on margin settings per instrument set in the Instrument Manager"

	• **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



MyCustomStrategy
Sample MA crossover
Sample multi-instrument
Sample multi-timeframe

Data Series	
Instrument	Select
Price based on	Last
Type	Minute
Value	1
Time frame	
Days to load	5
Trading hours	<Use instrument settings>
Break at EOD	☑
Set up	
Account	Sim101
Calculate	On bar close
Label	MyCustomStrategy
Maximum bars look back	256
Bars required to trade	20
Start behavior	Adopt account position
Historical fill processing	
Order fill resolution	Standard (Fastest)

11.4.2.6 Working with Strategy Templates

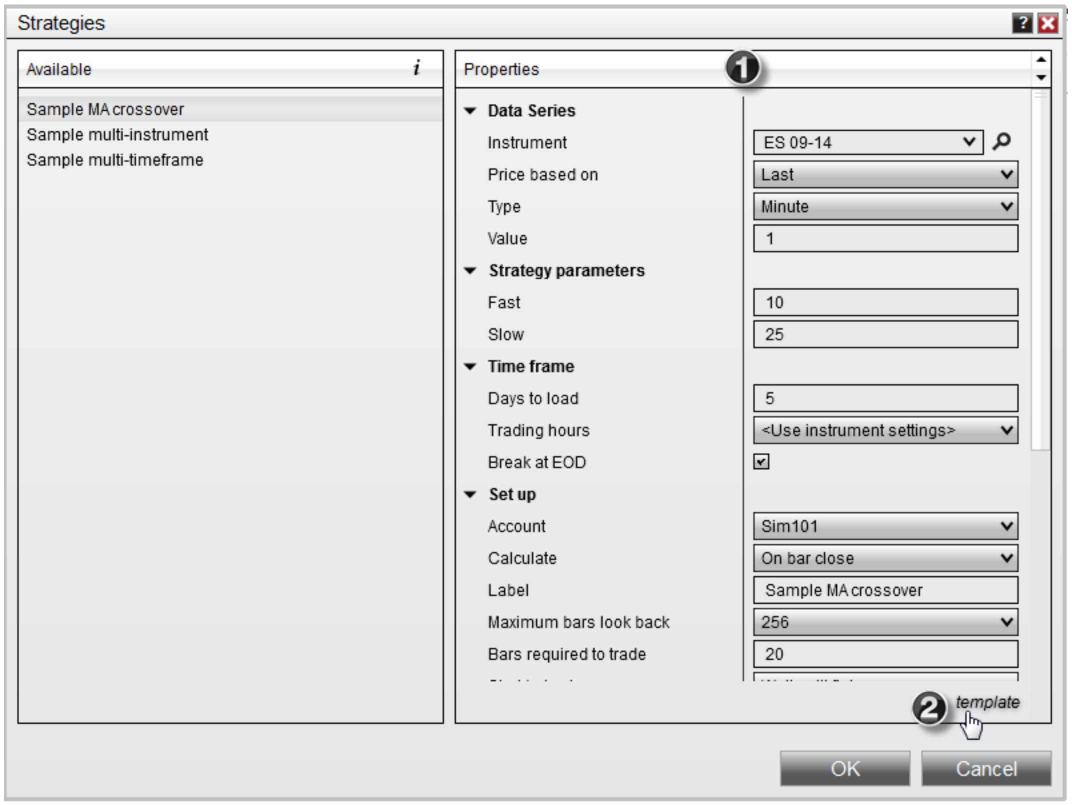
NinjaTrader allows you to save your **Strategy properties** as a template that can be loaded or set as the default for new instance of a **Strategy** when starting the strategy to be used in real-time or for backtesting purposes. There is no limit to the number of templates you can save.

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



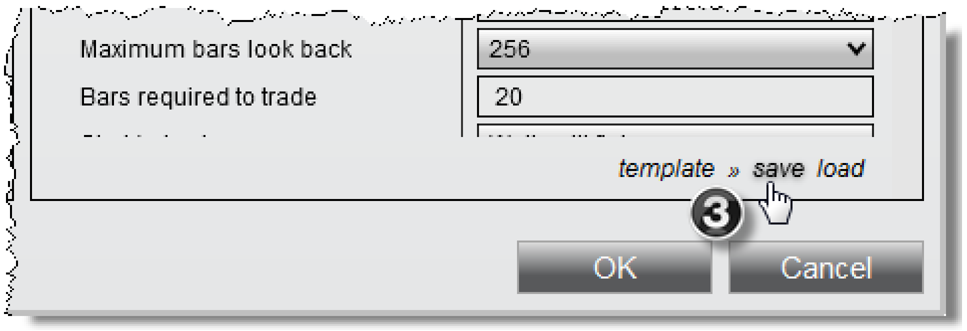
Strategies

Available
Sample MA crossover
Sample multi-instrument
Sample multi-timeframe

Properties

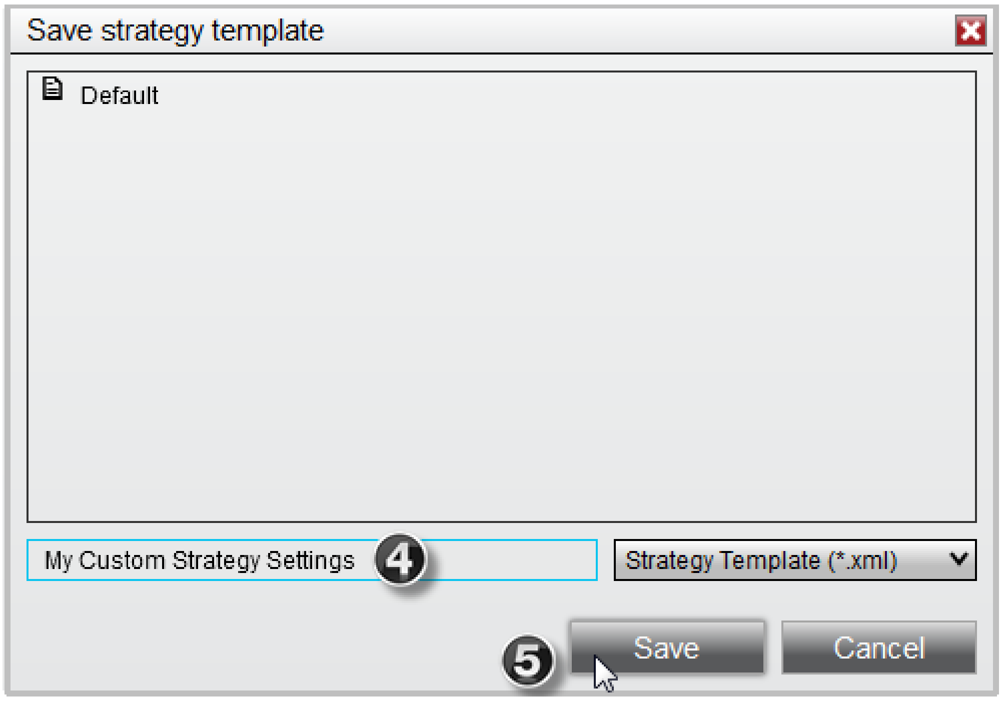
Data Series	
Instrument	ES 09-14
Price based on	Last
Type	Minute
Value	1
Strategy parameters	
Fast	10
Slow	25
Time frame	
Days to load	5
Trading hours	<Use instrument settings>
Break at EOD	✓
Set up	
Account	Sim101
Calculate	On bar close
Label	Sample MA crossover
Maximum bars look back	256
Bars required to trade	20

3. Select the option **save** which will open a **Save Strategy Template** dialog window



Maximum bars look back	256
Bars required to trade	20

4. Enter a custom *name to identify the strategy template
5. Click the **Save** button



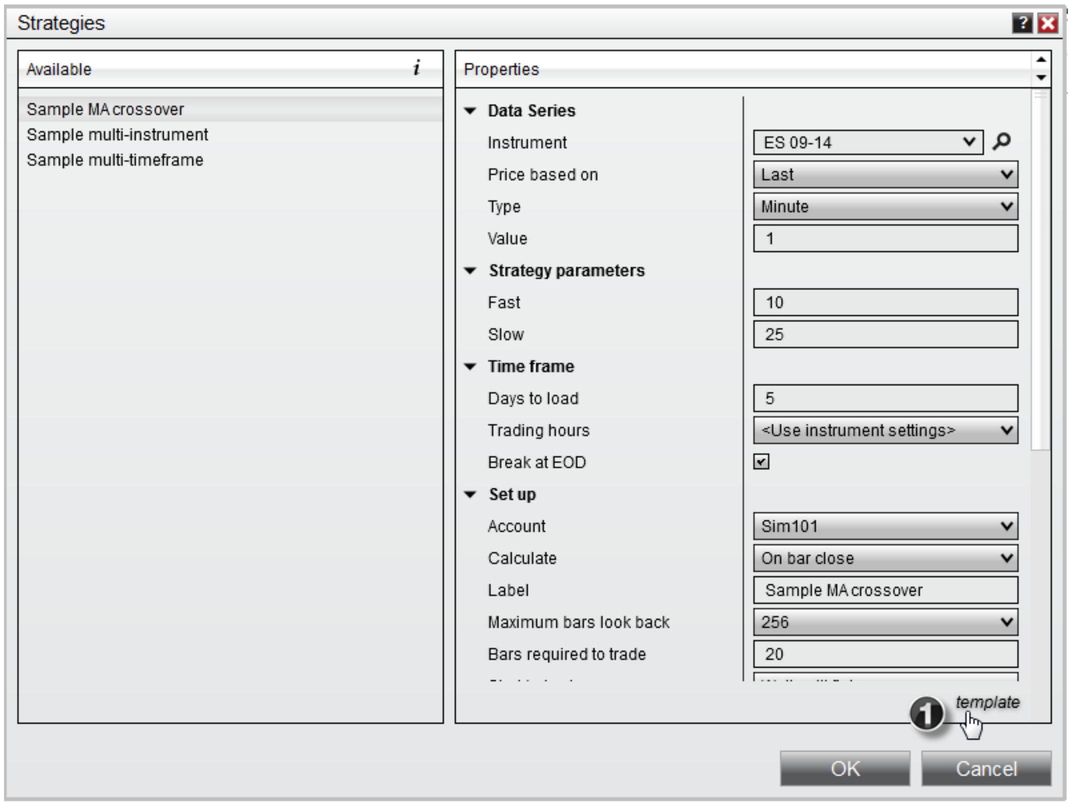
Tip: If you wish to save your **strategy properties** as the **default** values used when recalling these settings, you can call the strategy template name "**Default**" which will automatically load when a new instance of the strategy has been initiated.

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


Strategies

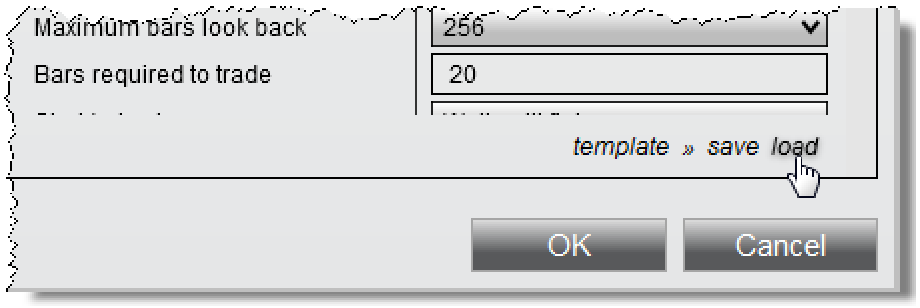
Available

Sample MA crossover
Sample multi-instrument
Sample multi-timeframe

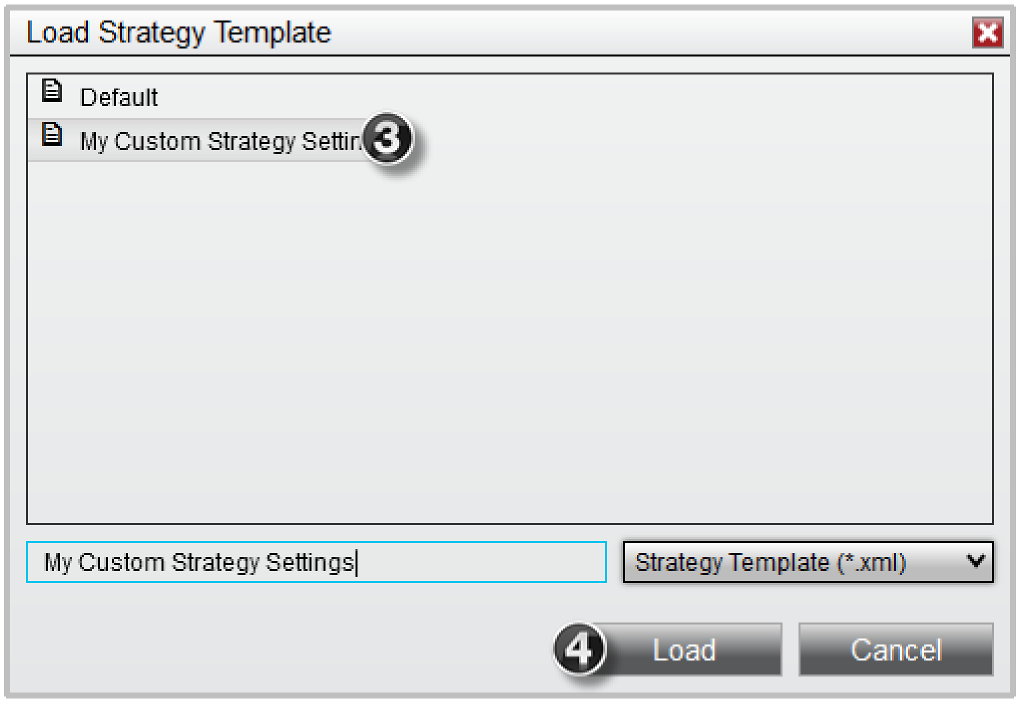
Properties

Data Series	
Instrument	ES 09-14
Price based on	Last
Type	Minute
Value	1
Strategy parameters	
Fast	10
Slow	25
Time frame	
Days to load	5
Trading hours	<Use instrument settings>
Break at EOD	✓
Set up	
Account	Sim101
Calculate	On bar close
Label	Sample MA crossover
Maximum bars look back	256
Bars required to trade	20

2. Select the option **load** which will open a **Load Strategy Template** dialog window


Maximum bars look back 256
Bars required to trade 20

3. Select the desired template name from the list of templates
4. Click the **Load** button

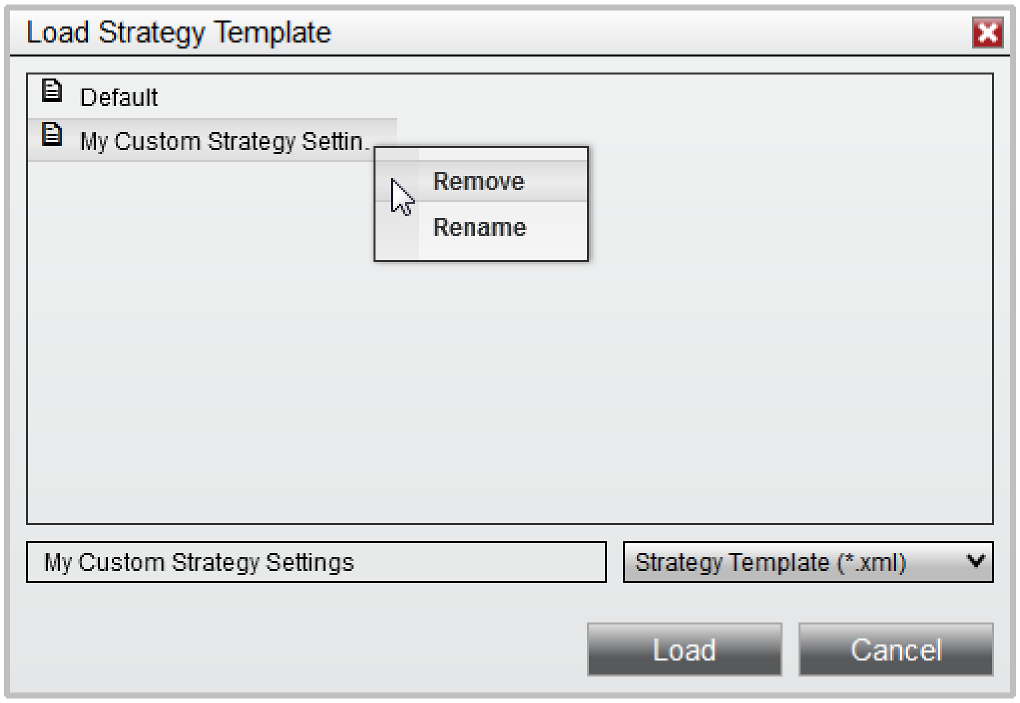

Load Strategy Template
Default
My Custom Strategy Settin
3
My Custom Strategy Settings
Strategy Template (*.xml)
4
Load
Cancel

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


Load Strategy Template
Default
My Custom Strategy Settin.
Remove
Rename
My Custom Strategy Settings
Strategy Template (*.xml)
Load
Cancel

Tip: If you wish to rename an existing template, you can select **Rename** from the same menu

11.4.2.7 Running FX Strategies

It is important to understand how order quantities behave when running a real-time FX strategy in a live brokerage account. This is relevant for:

- Currency and point based performance calculations
- Actual order size being submitted to your live brokerage account

Running FX NinjaScript Strategies in the Strategy Analyzer, Simulated Data Feed Connection or Playback Connection

Running an FX strategy in the Strategy Analyzer for a historical backtest, in real-time connected to either the Simulated Data Feed or Playback order quantities will always represent the total units of the base currency being traded. A lot size of one equals a base currency unit of one and DOES NOT equal a typical "standard" lot size of 100,000. Therefore, if you want to trade a standard lot you would use an order quantity of 100,000.

Running FX NinjaScript Strategies on a Live Brokerage Connection

Running an FX strategy on a live brokerage connection either in a live funded account or a simulation (Sim101) account order quantities will reflect the lot size convention that your brokerage trades in. For example, if your brokerage has a lot size of 1 that equals 100,000 units of the base currency (a "standard" lot) then use a value of 1 to trade 100,000 base units.

CRITICAL

Based on the above information, if you are backtesting in the Strategy Analyzer using order quantities of 100,000 and you now wish to trade live in your brokerage account where 1 lot is equal to 100,000 MAKE SURE that you adjust your strategy's order quantity from 100,000 to 1 to ensure you are trading the correct quantity.

11.5 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.

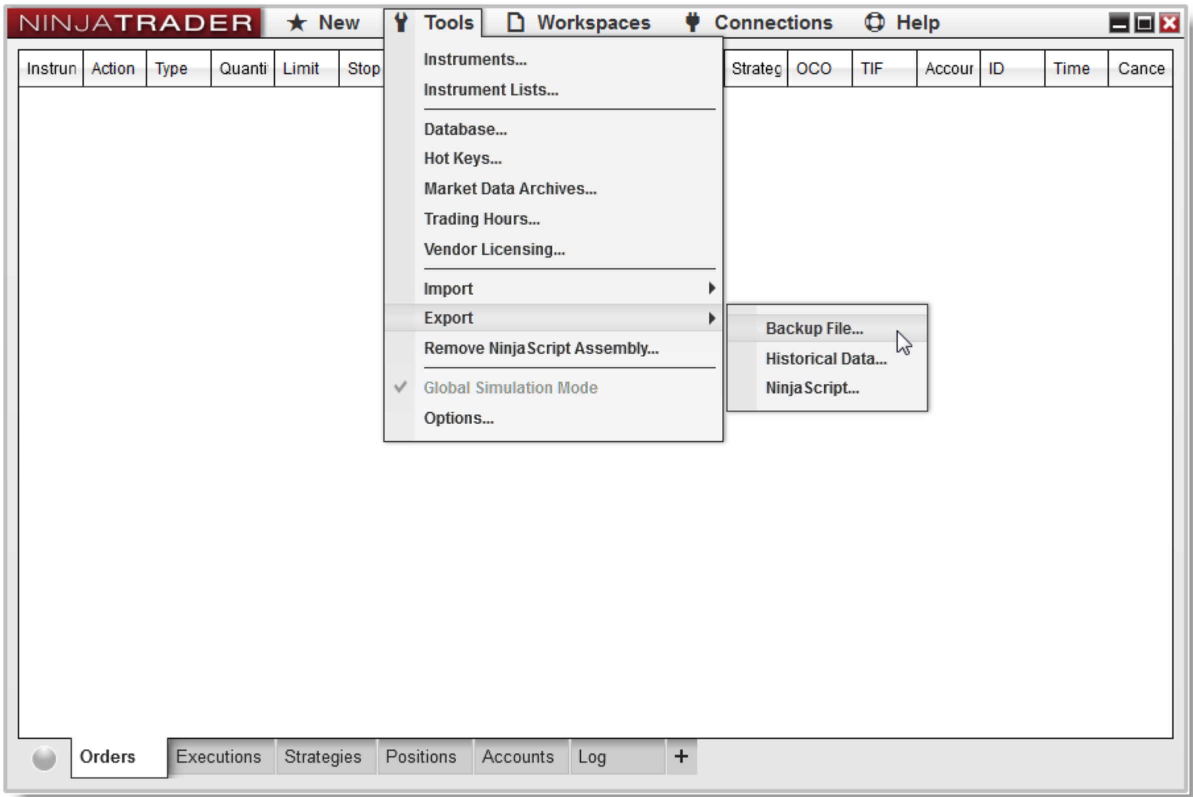
› Creating a Backup Archive
› Restoring a Backup Archive

11.5.1 Creating a Backup Archive

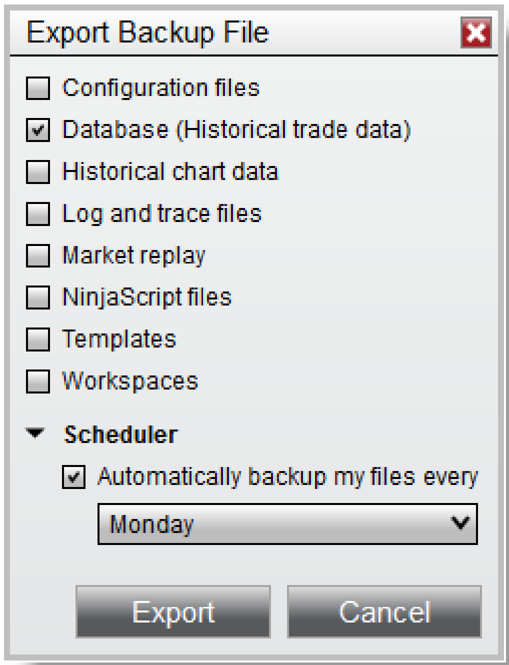
Running your first backup

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


Export Backup File

- [ ] Configuration files
- [x] Database (Historical trade data)
- [ ] Historical chart data
- [ ] Log and trace files
- [ ] Market replay
- [ ] NinjaScript files
- [ ] Templates
- [ ] Workspaces

Scheduler

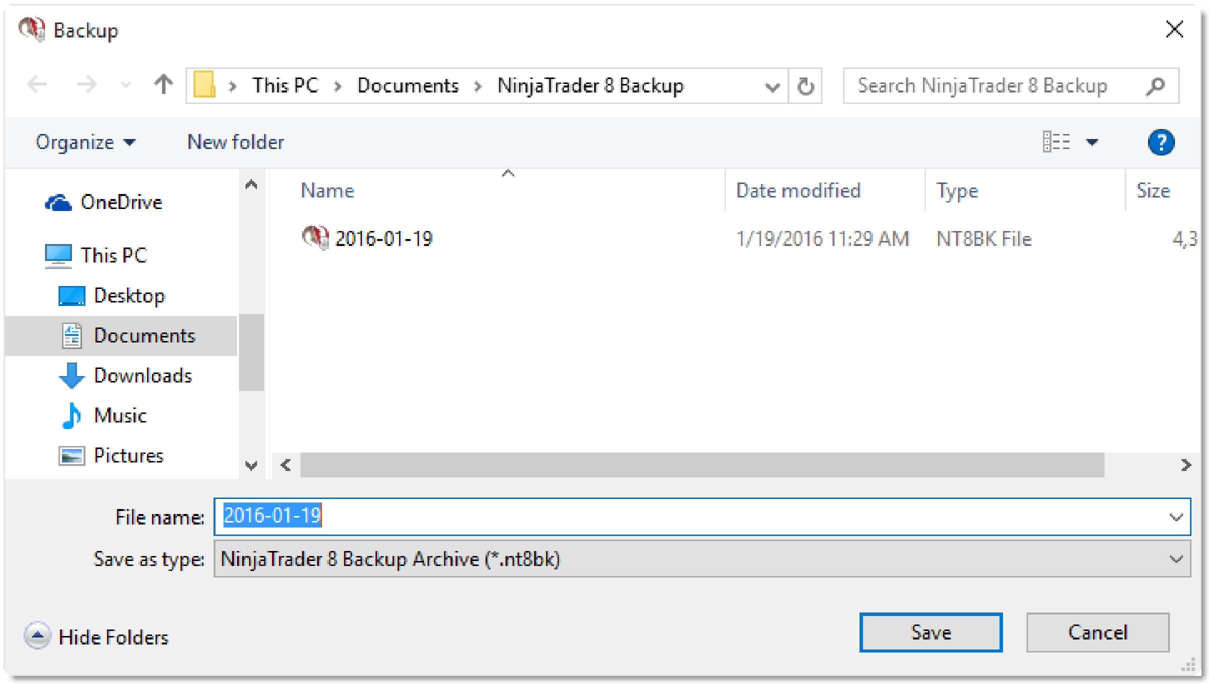
- [x] Automatically backup my files every

Monday

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.



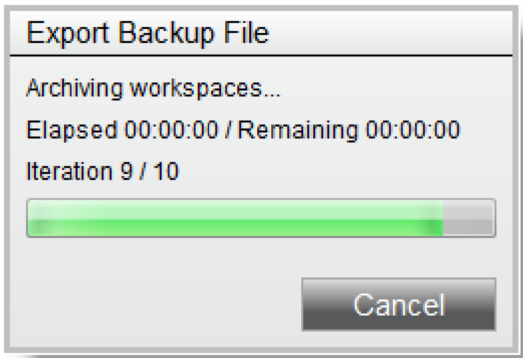
Name	Date modified	Type	Size
2016-01-19	1/19/2016 11:29 AM	NT8BK File	4,3

File name: 2016-01-19

Save as type: NinjaTrader 8 Backup Archive (*.nt8bk)

6. Select the **Backup** button.

You will now be presented with a status bar indicating the estimated time and progress of the backup.


Export Backup File

Archiving workspaces...

Elapsed 00:00:00 / Remaining 00:00:00

Iteration 9 / 10

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

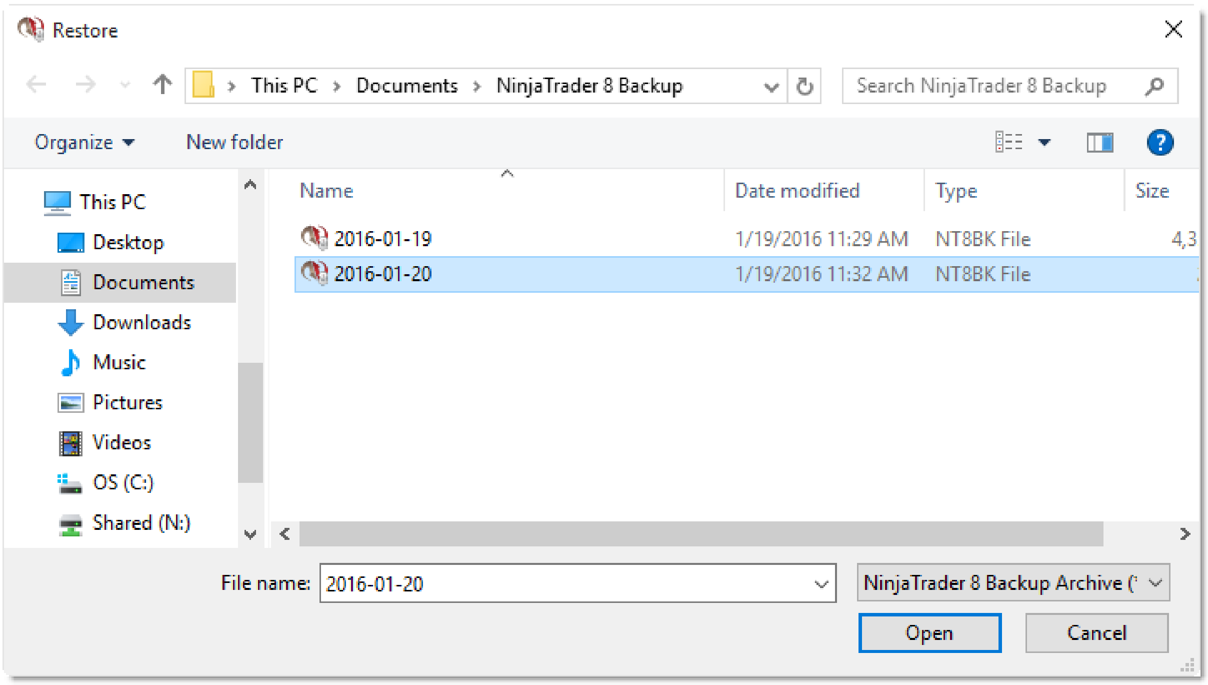
When running a backup, there is an option labeled "Please remind me to back up my files every..." which if selected, will allow you to specify a day of the week on which to receive a reminder via a popup notification. You can configure NinjaTrader to run this backup Daily, Weekly, or Monthly.

Note: NinjaTrader **must** be running for the a scheduled backup to run. If NinjaTrader was not running at the time the backup was scheduled, the backup will be run the next time NinjaTrader is shut down.

11.5.2 Restoring a Backup Archive

Complete the following steps to restore a Backup Archive.

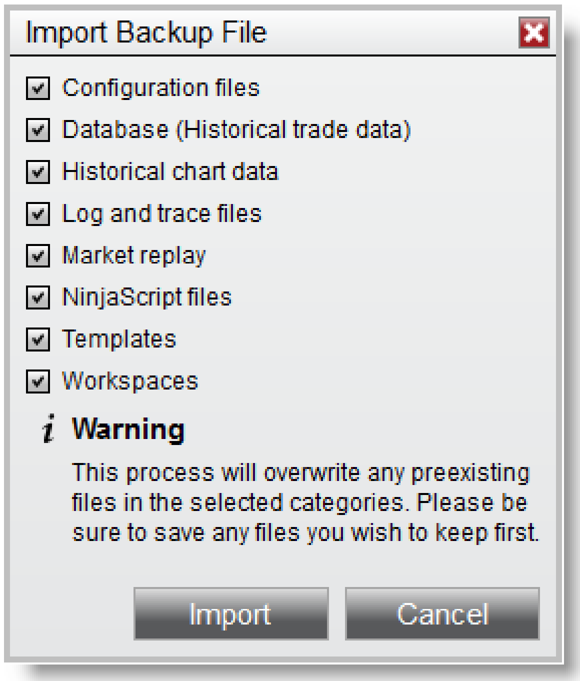
1. From within the Control Center window select the **Tools** menu. Then select the menu **Import** and the **Backup File...** menu item
2. Select the backup archive to restore from the "Restore" file dialog window



Name	Date modified	Type	Size
2016-01-19	1/19/2016 11:29 AM	NT8BK File	4,3
2016-01-20	1/19/2016 11:32 AM	NT8BK File	

File name: 2016-01-20

3. Press the "**Restore**" button
4. Select the items you wish to restore


Import Backup File

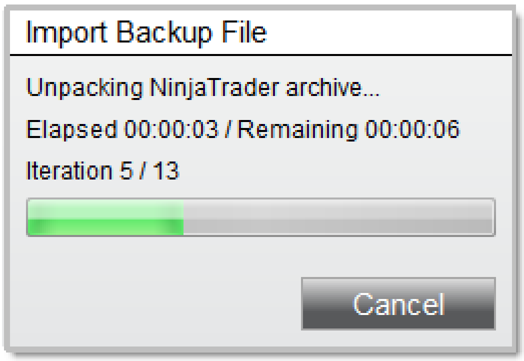
- ✓ Configuration files
- ✓ Database (Historical trade data)
- ✓ Historical chart data
- ✓ Log and trace files
- ✓ Market replay
- ✓ NinjaScript files
- ✓ Templates
- ✓ Workspaces

Warning

This process will overwrite any preexisting files in the selected categories. Please be sure to save any files you wish to keep first.

5. Press the **"Import"** button

You will now be presented with a status bar indicating the estimated time and progress of the import.


Import Backup File

Unpacking NinjaTrader archive...

Elapsed 00:00:03 / Remaining 00:00:06

Iteration 5 / 13

11.6 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

Trading Features

Misc

11.6.1 Creating a Chart

The following section covers how to open a NinjaTrader chart.

Play Video

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.

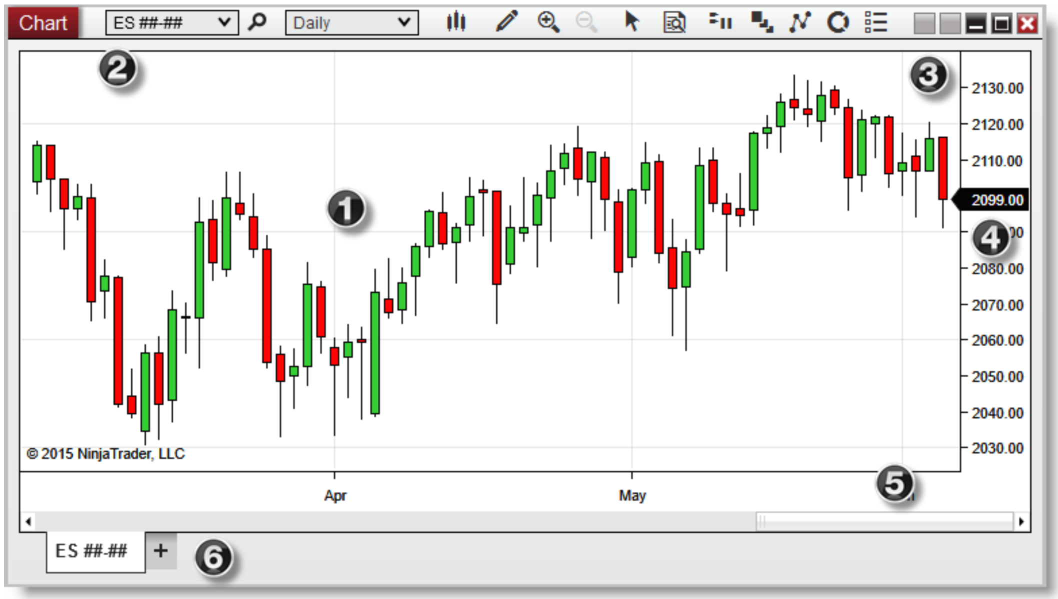
Selecting an Instrument

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.

11.6.2 Navigating a Chart

The following section covers navigation and display of NinjaTrader charts.

Play Video

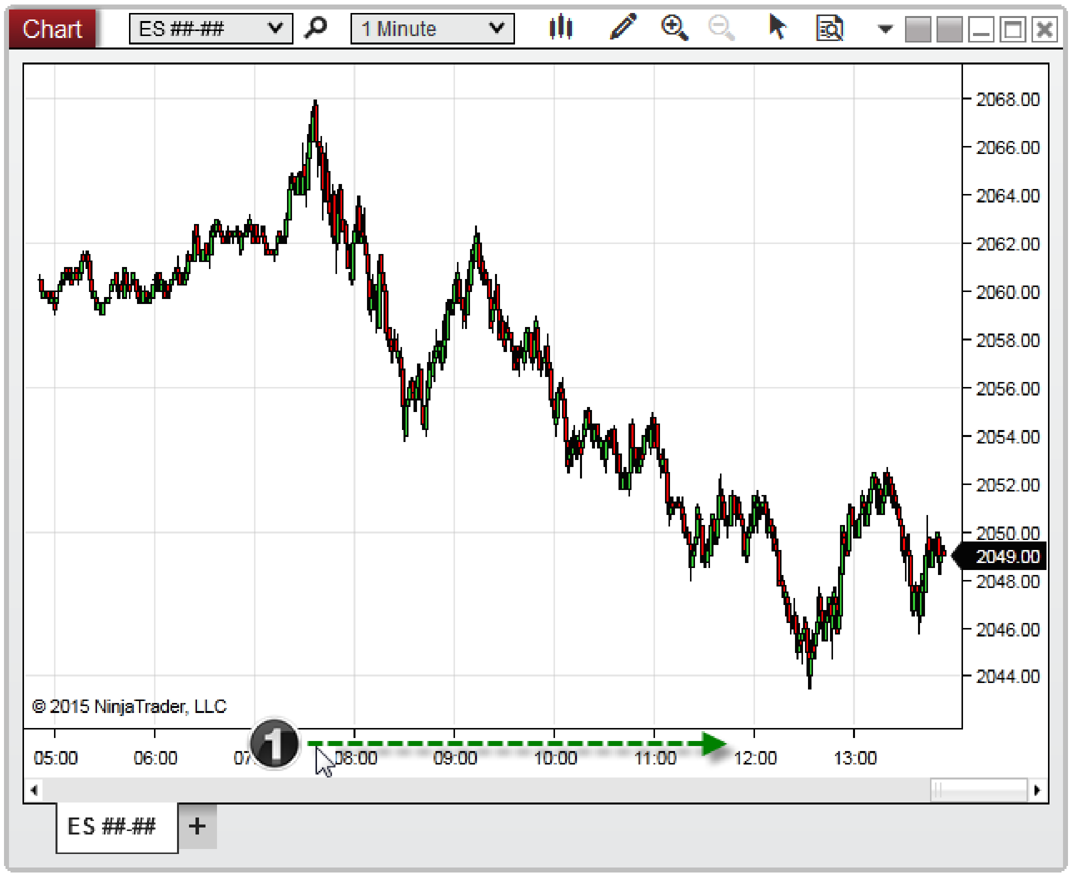
How to change the horizontal scale and time range of a chart

Horizontal Scaling

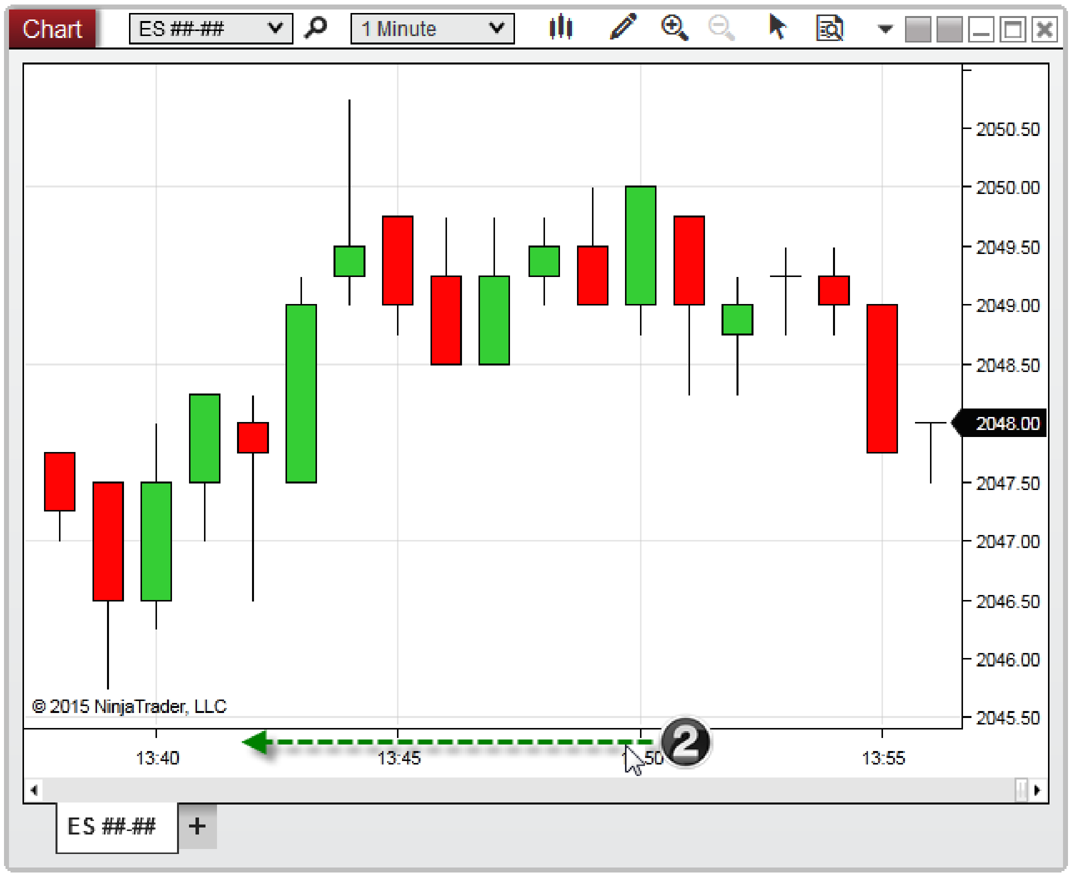
To compress or decompress the horizontal axis, left mouse click in the x-axis margin and move the mouse cursor to the left or right. Alternatively, use the Hot

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

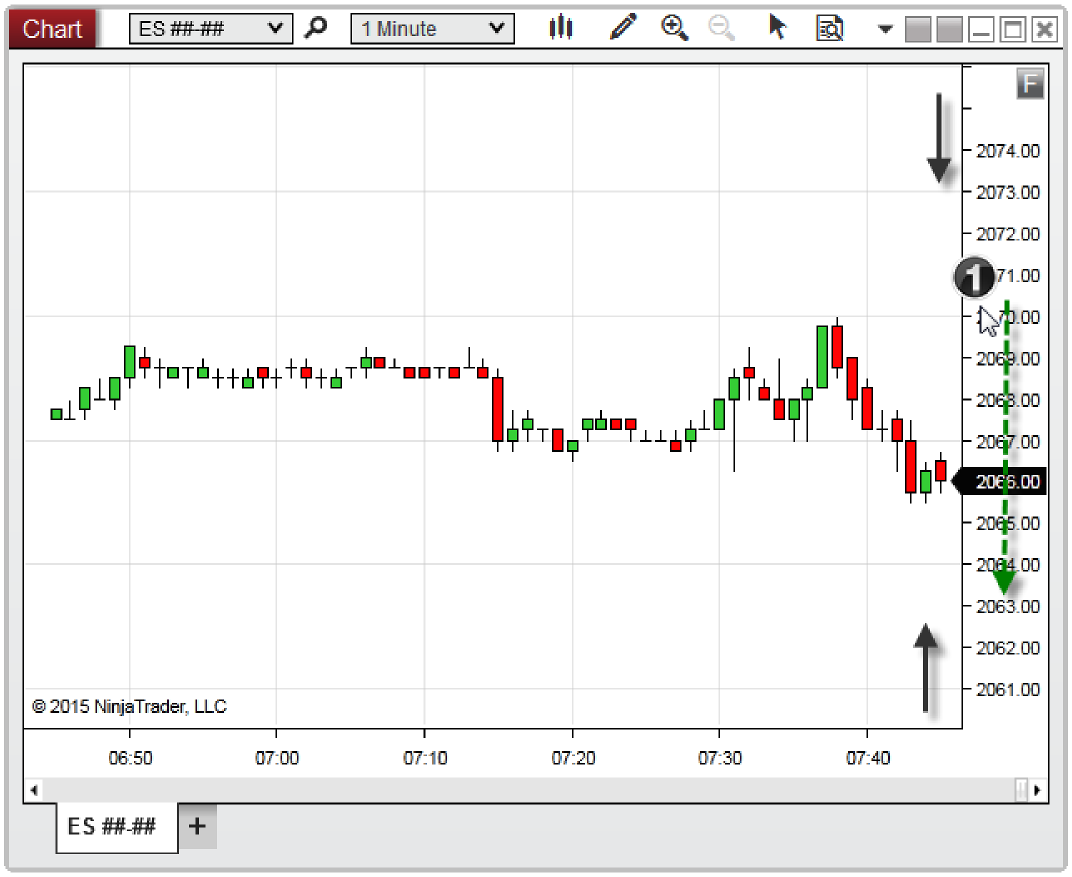


How to change the vertical scale and price range of a chart

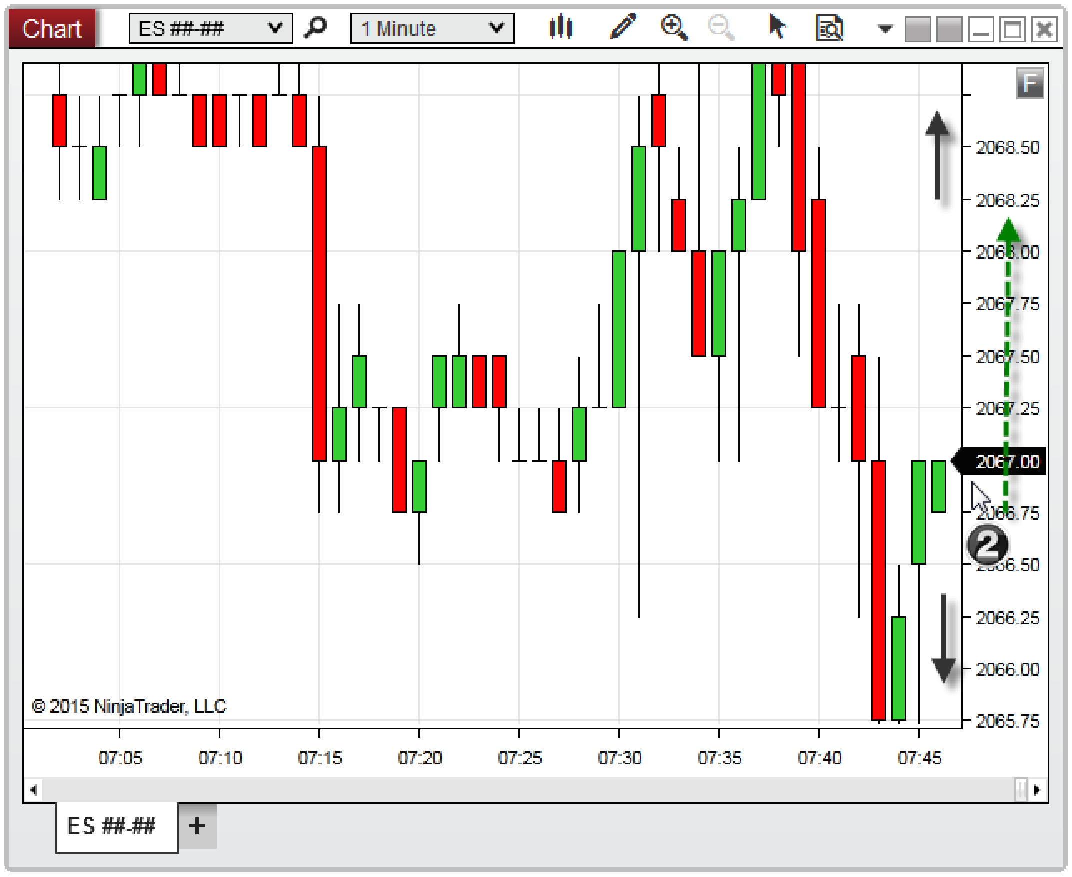
Vertical Scaling

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



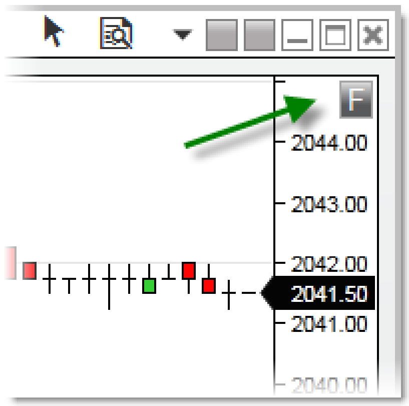
2. Click and drag up will stretch the chart's price scale



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)

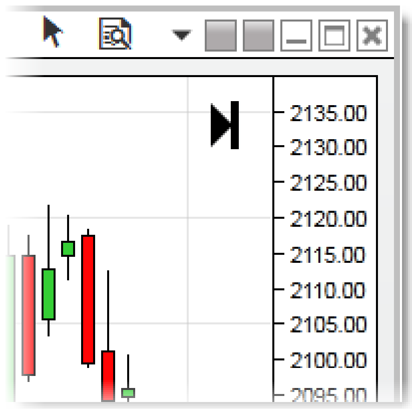
You can pan the chart left or right via the following controls:

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 arrow key)	Forward one page at a time
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.

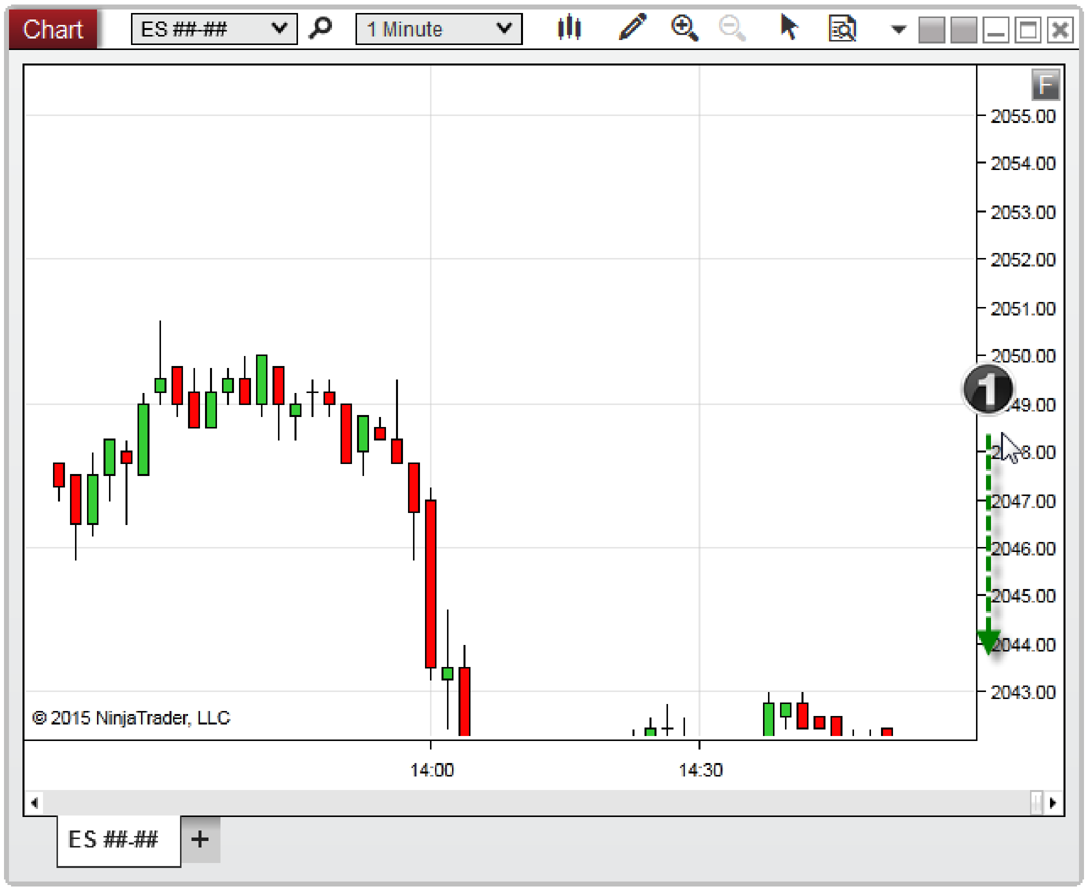


Vertical Scrolling (panning chart up or down)

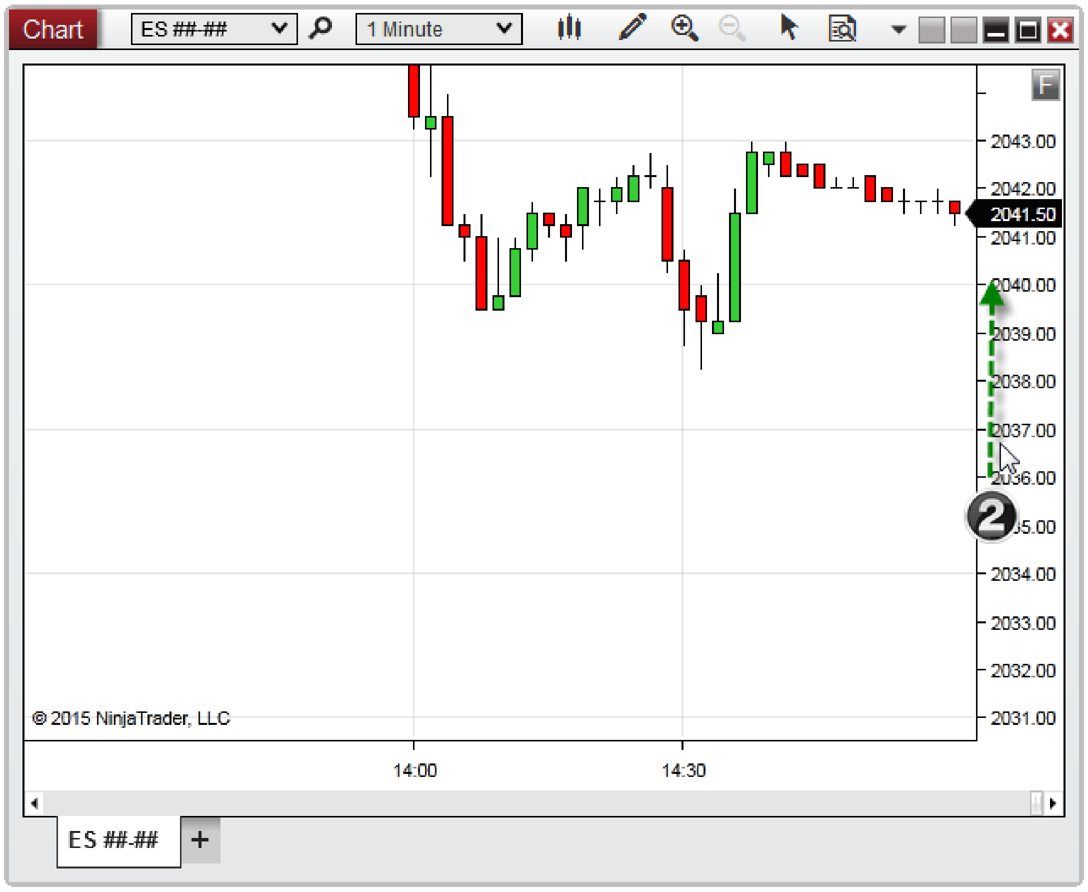
To pan the chart up or down:

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.

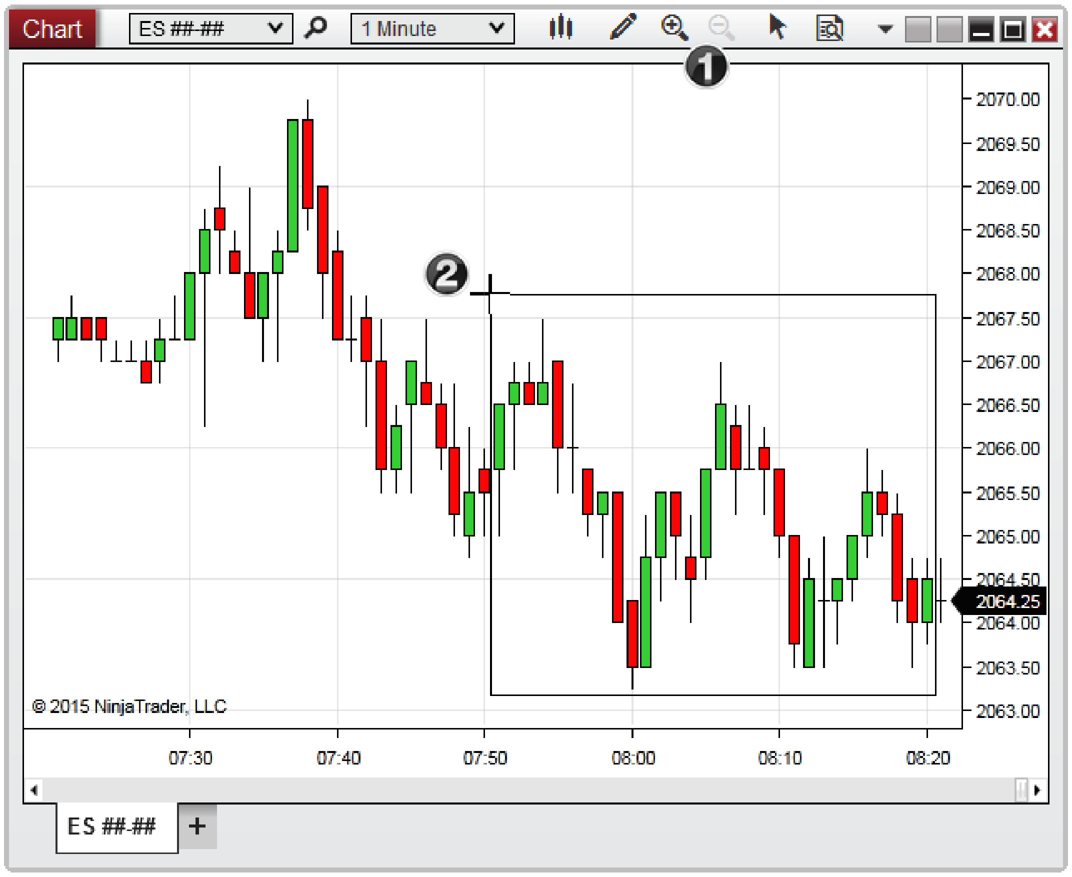
How to zoom in and out in a chart

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:

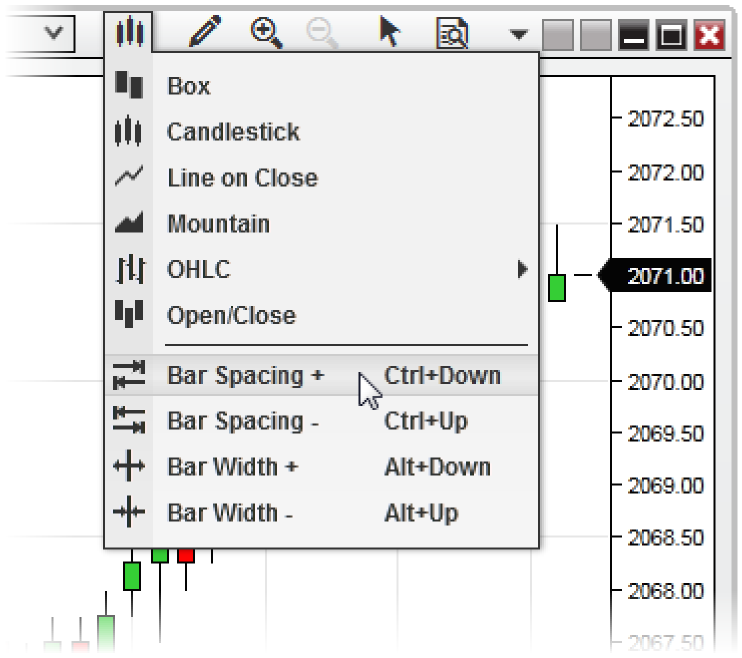
- CTRL + Up arrow key decreases bar spacing
- CTRL + Down arrow key increases bar spacing

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



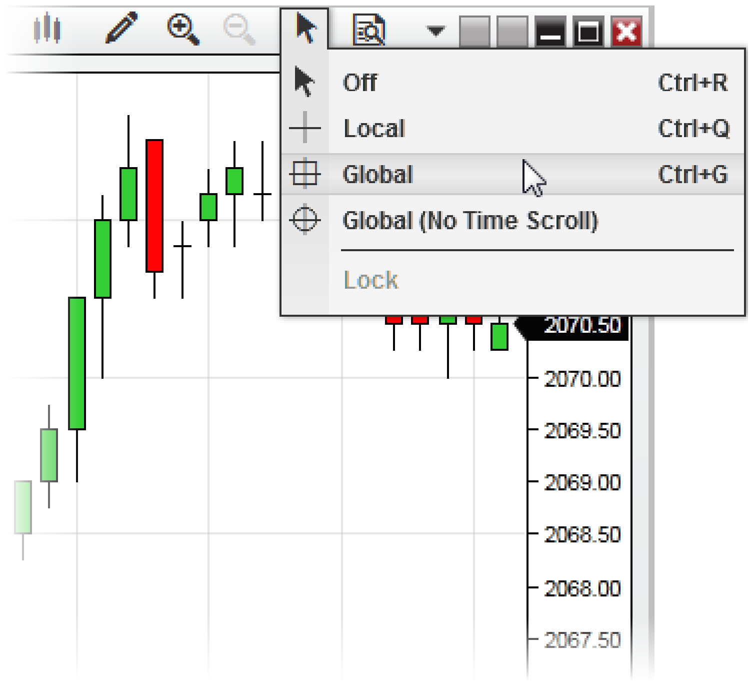
Note: On a multi-series chart, before changing bar spacing or width, you must select the Data Series you want to adjust by left mouse clicking on it. If none is selected, the primary Data Series of the chart will be adjusted.

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)



11.6.3 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

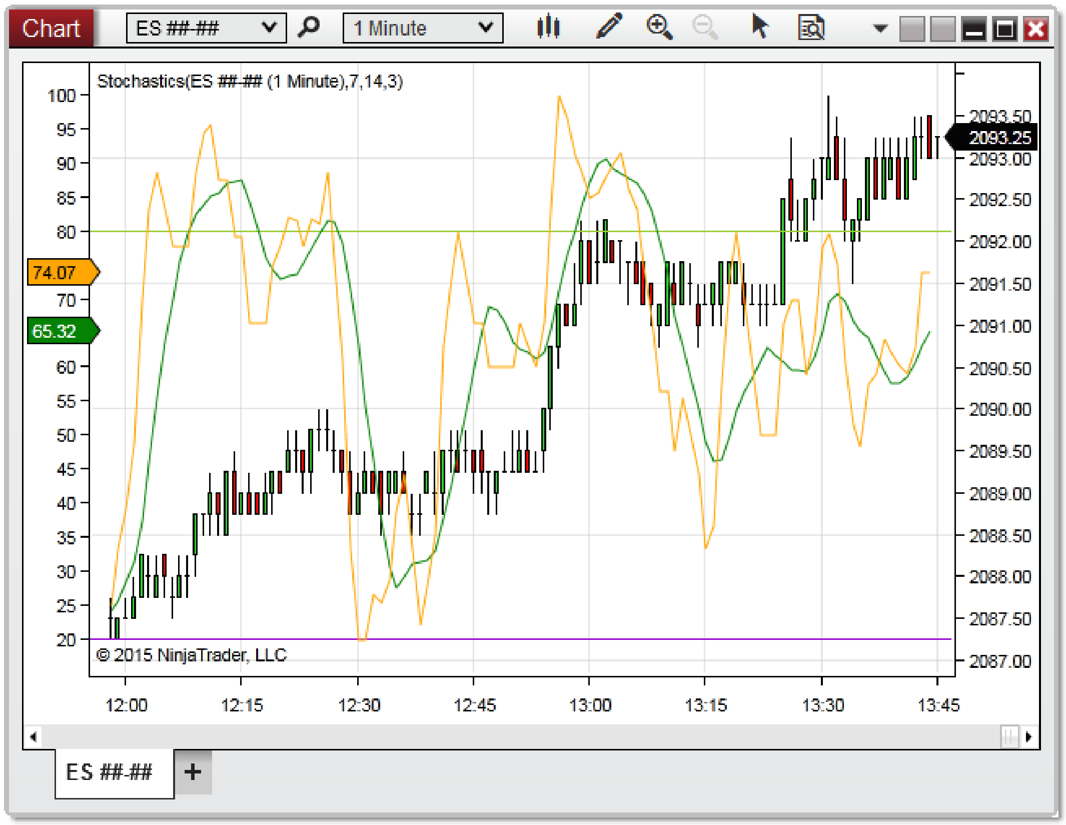
Panel Scales

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.

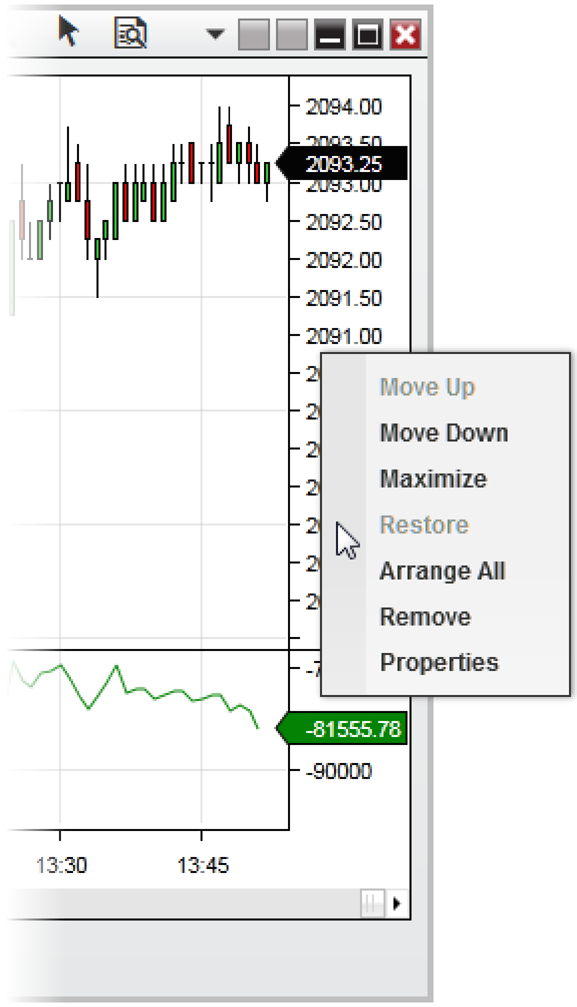
Tip: In addition to changing a chart objects scale justification property via the **Data Series window** or **Indicators window**, you can drag and drop a chart object onto different scales. Please see the section "*How to drag and drop chart objects"* section located on the Working with Chart Objects 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.



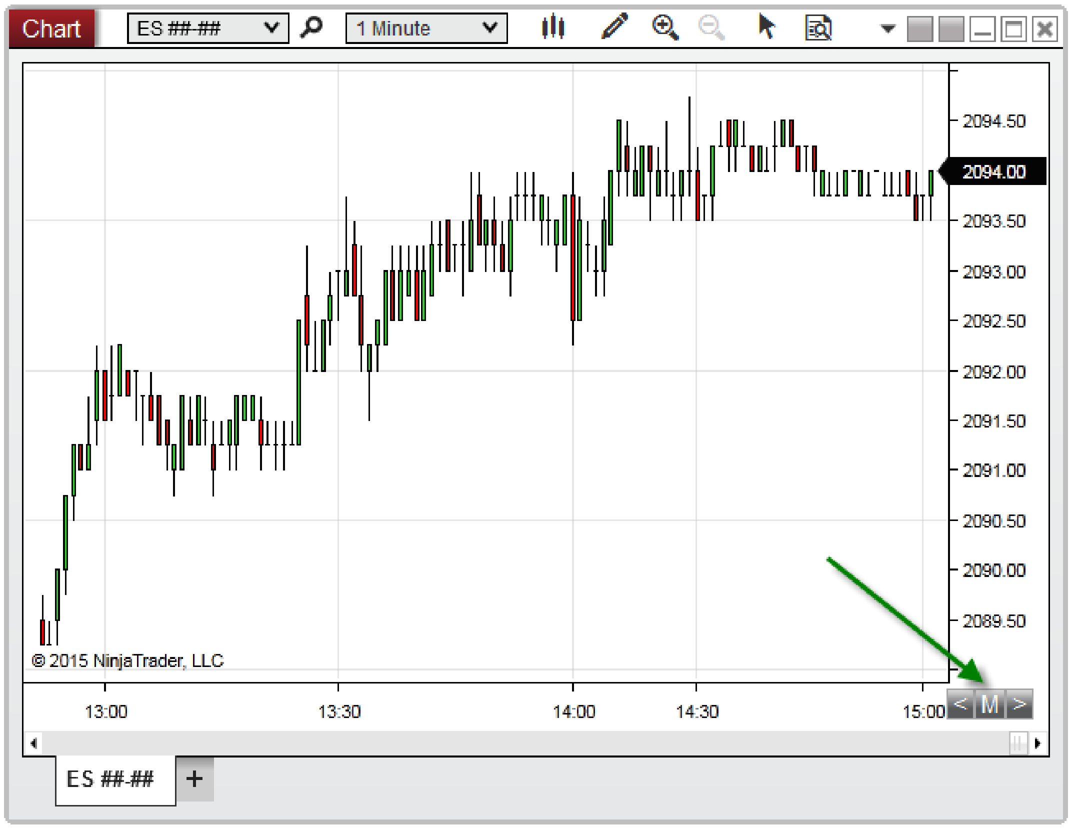
The following actions are available:

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	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

value	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 "**Points**" 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**"

11.6.4 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**.

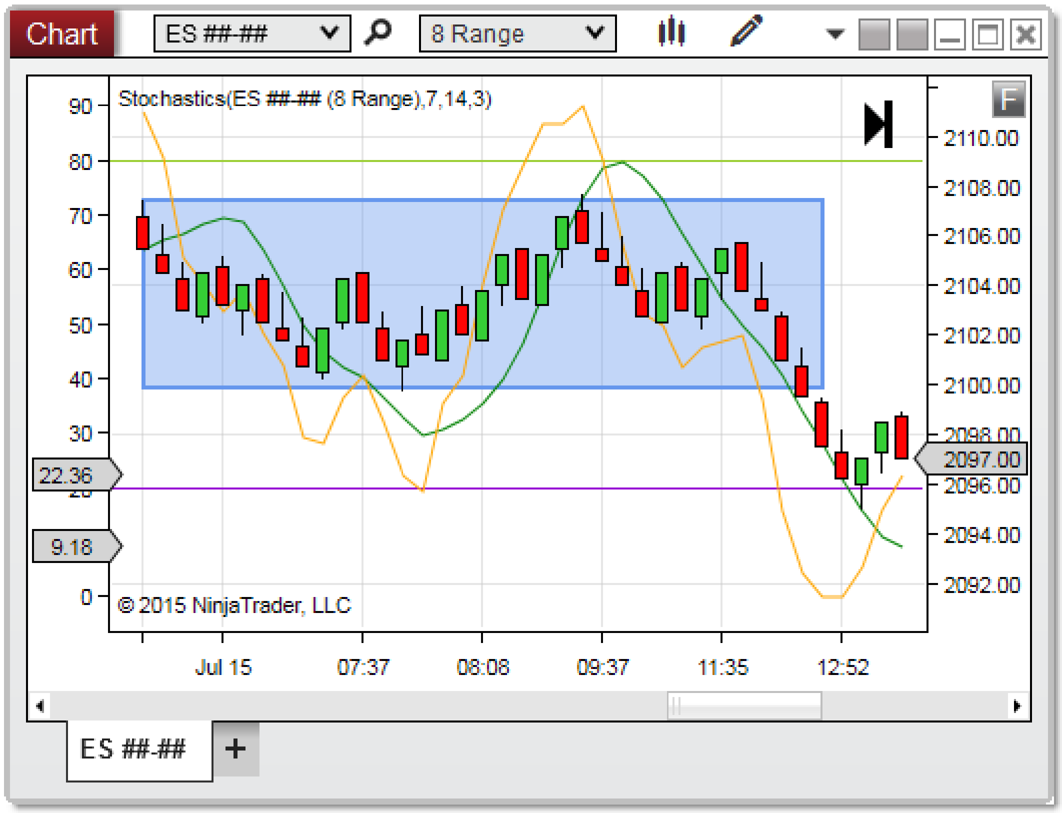
How to change the z-order (paint order) of a chart object

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.

11.6.5 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**.

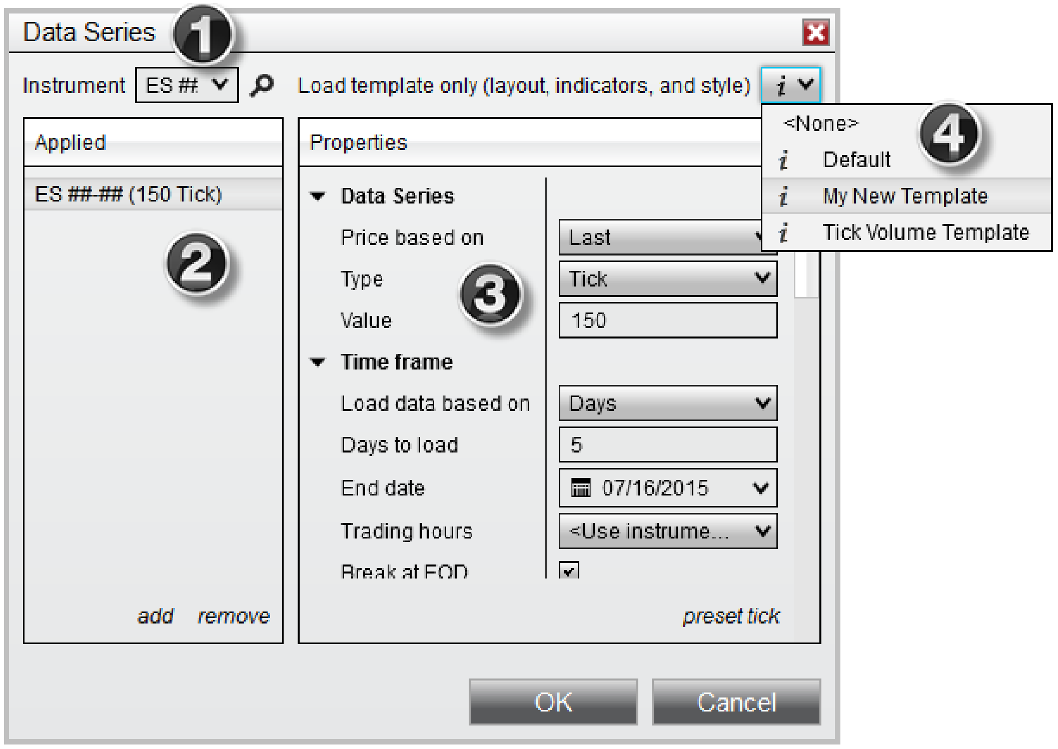
Accessing the Data Series Window

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.


Data Series

Instrument: ES ##

Load template only (layout, indicators, and style)

<None>
Default
My New Template
Tick Volume Template

Applied
ES ##-## (150 Tick)

Properties

Data Series
Price based on: Last
Type: Tick
Value: 150

Time frame
Load data based on: Days
Days to load: 5
End date: 07/16/2015
Trading hours: <Use instrume...
Break at EOD: ✓

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

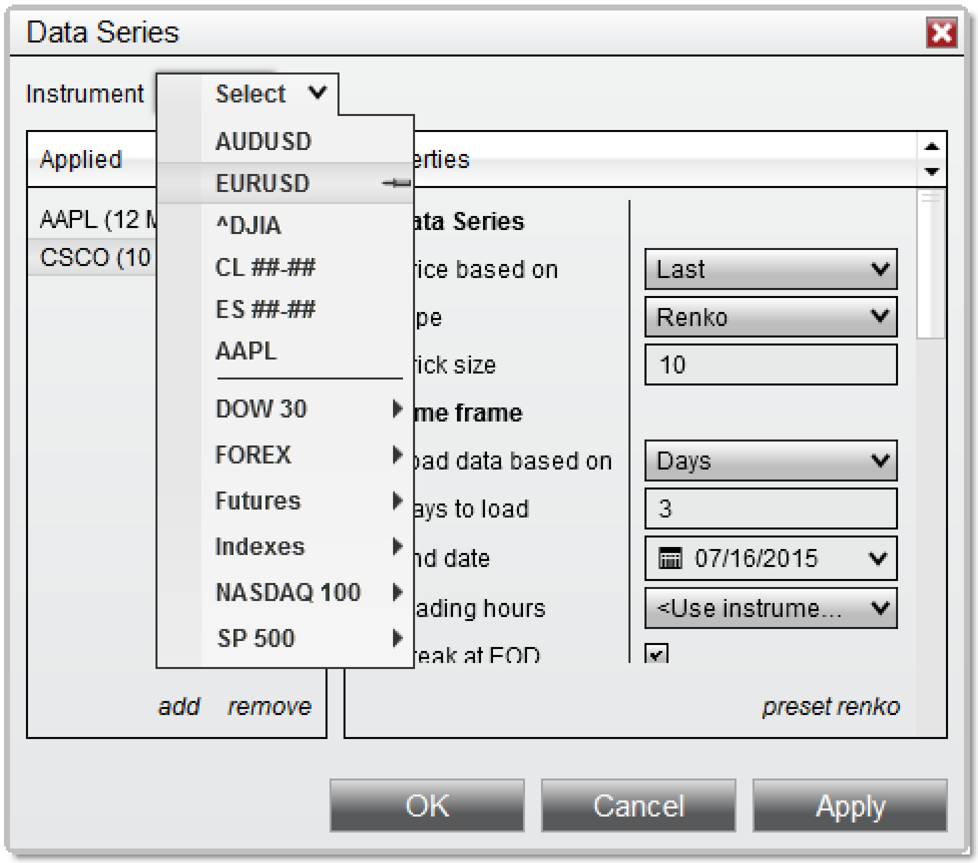
Adding a Data Series

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).



Last
Renko
10
Days
3
07/16/2015
<Use instrume...

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**.

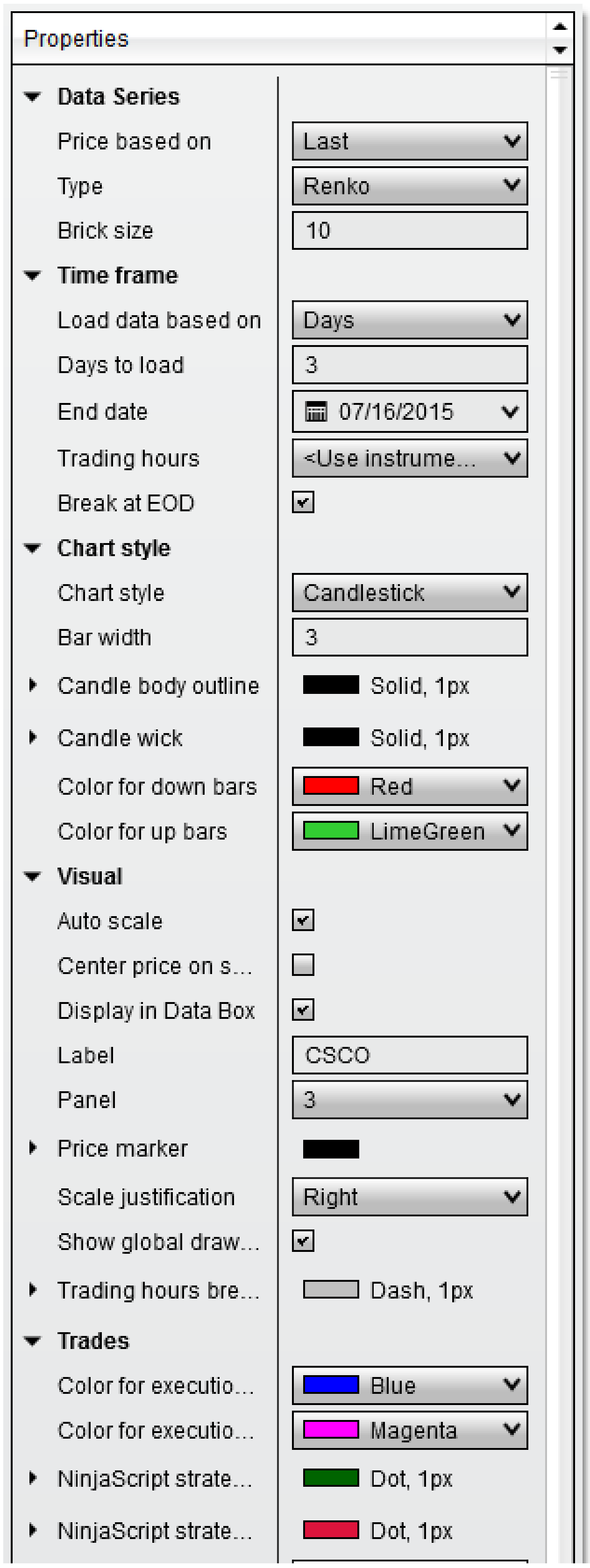
How to edit Data Series parameters

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.



Properties

Data Series

Price based on	Last
Type	Renko
Brick size	10

Time frame

Load data based on	Days
Days to load	3
End date	07/16/2015
Trading hours	<Use instrume...
Break at EOD	✓

Chart style

Chart style	Candlestick
Bar width	3
Candle body outline	Solid, 1px
Candle wick	Solid, 1px
Color for down bars	🟥 Red
Color for up bars	🟩 LimeGreen

Visual

Auto scale	✓
Center price on s...	☐
Display in Data Box	✓
Label	CSCO
Panel	3
Price marker	■
Scale justification	Right
Show global draw...	✓
Trading hours bre...	Dash, 1px

Trades

Color for executio...	🟦 Blue
Color for executio...	🟪 Magenta
NinjaScript strate...	Dot, 1px
NinjaScript strate...	Dot, 1px

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, a non-time-based bar, such as a Range or Renko bar, will be cut off at the end of the **Trading Hours** session regardless of whether it is fully complete. 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
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	Sets the color, dash style, and width for the lines connecting entries and exits of unprofitable trades taken by a NinjaScript

Trade Line	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.

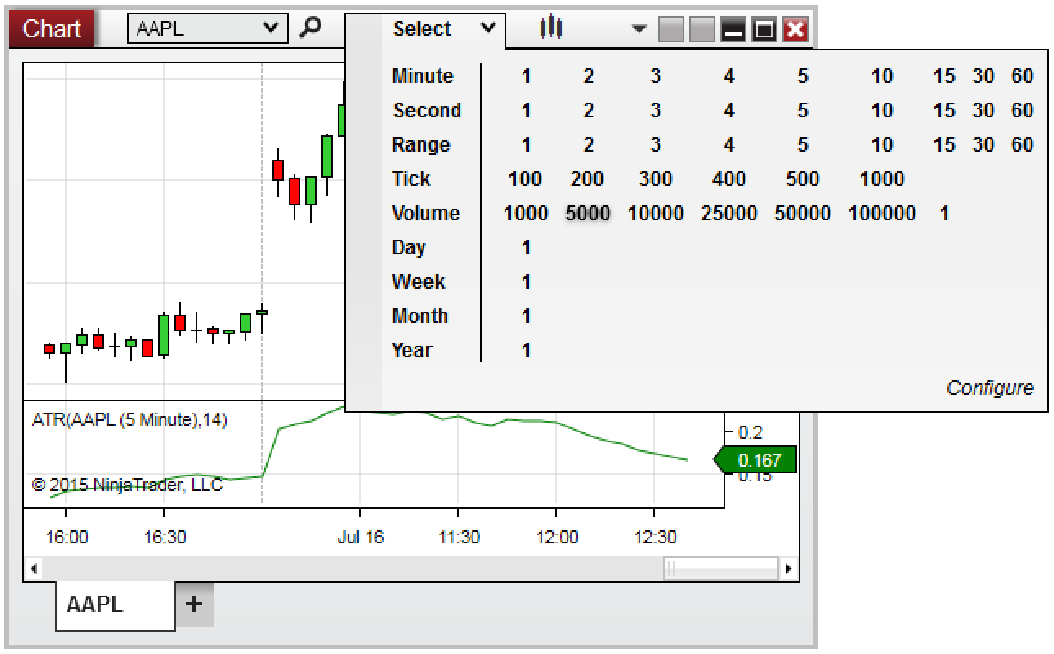
Changing an Instrument via the Chart Toolbar

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.



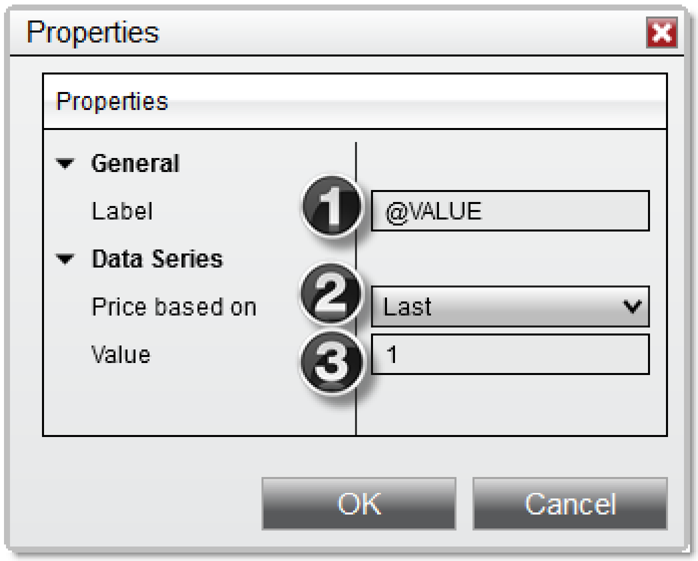
Minute	1	2	3	4	5	10	15	30	60
Second	1	2	3	4	5	10	15	30	60
Range	1	2	3	4	5	10	15	30	60
Tick	100	200	300	400	500	1000			
Volume	1000	5000	10000	25000	50000	100000	1		
Day	1								
Week	1								
Month	1								
Year	1								

Configure

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.

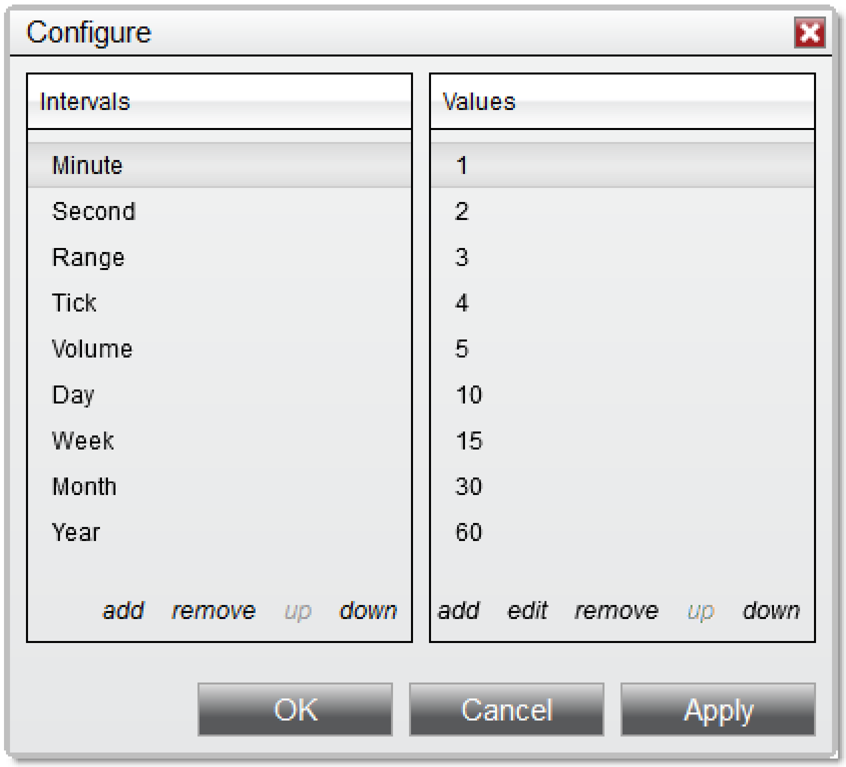
3. The "Value" field sets the value to be used for the interval, based on the **Interval Type**.

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.

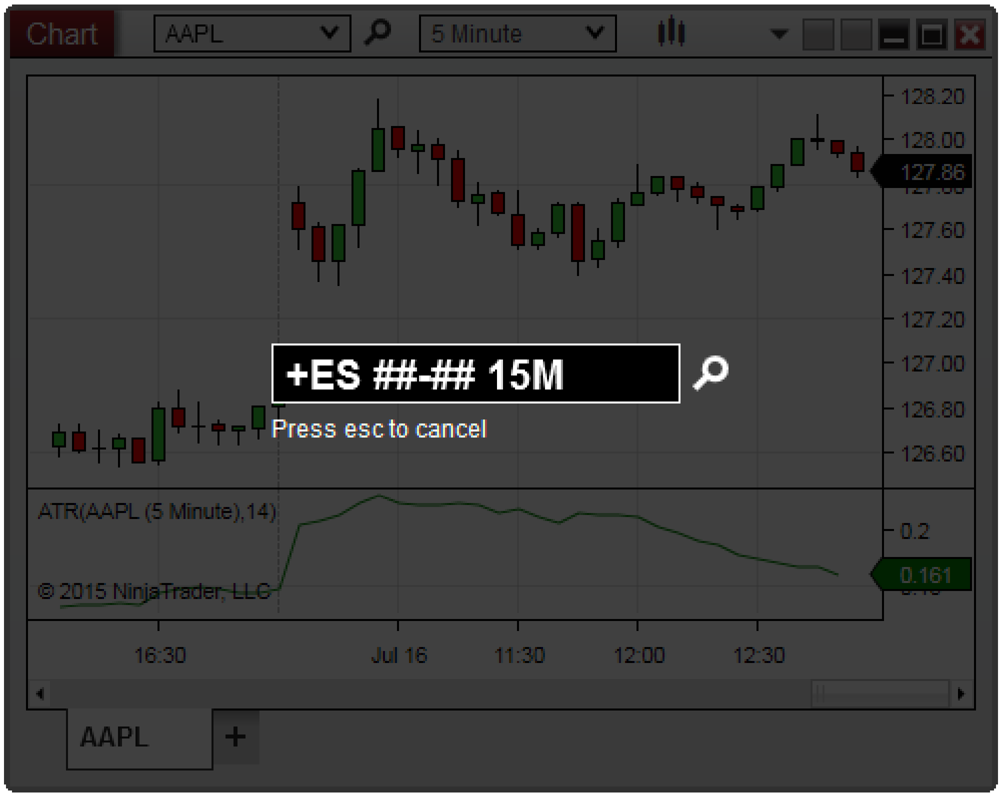

Configure

Intervals	Values
Minute	1
Second	2
Range	3
Tick	4
Volume	5
Day	10
Week	15
Month	30
Year	60

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
MO	Month
Y	Year
RE	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). S*elect 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.

11.6.6 Working with Multiple Data Series

Multiple Data Series

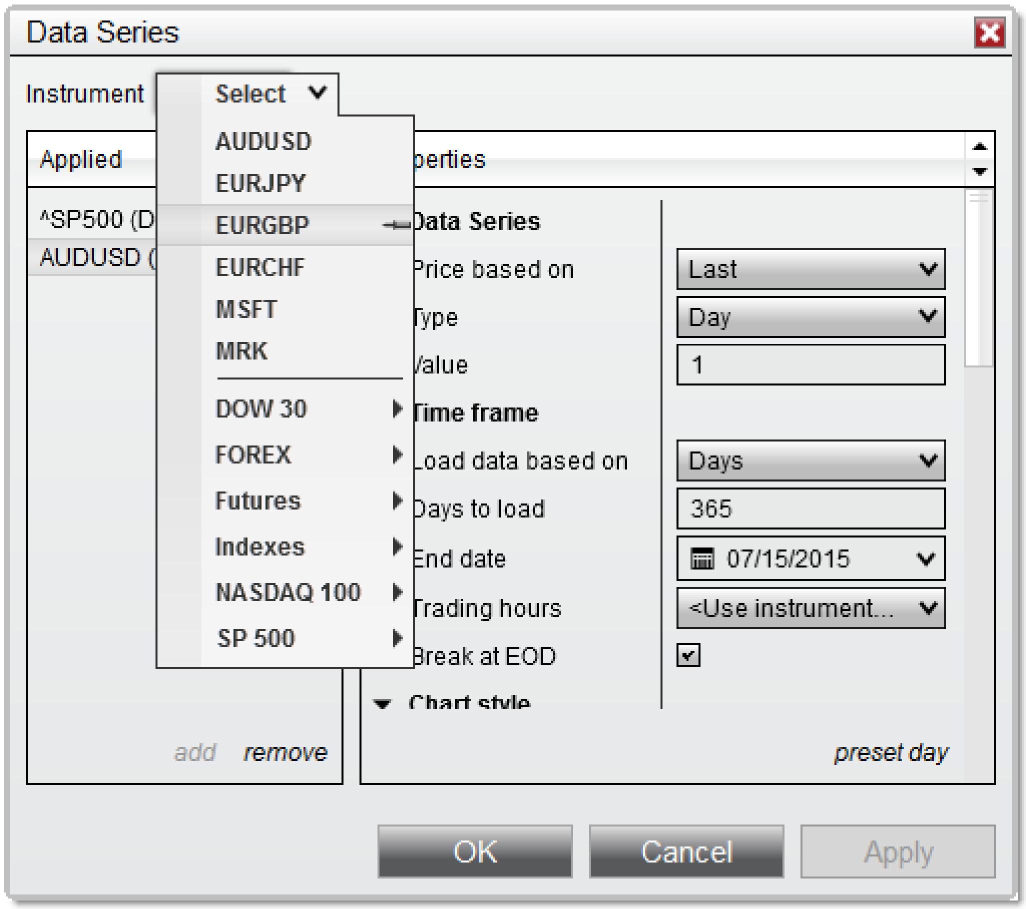
Multiple **Data Series** objects can be be viewed within a single chart window, and there are several ways to add **Data Series** to a chart.

▽ 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



Data Series

Price based on	Last
Type	Day
Value	1

Time frame

Load data based on	Days
Days to load	365
End date	07/15/2015
Trading hours	<Use instrument...
Break at EOD	✓

Chart style

In the image above, we can use the **Instrument Selector** to add a **Data Series** to a chart which is already open.

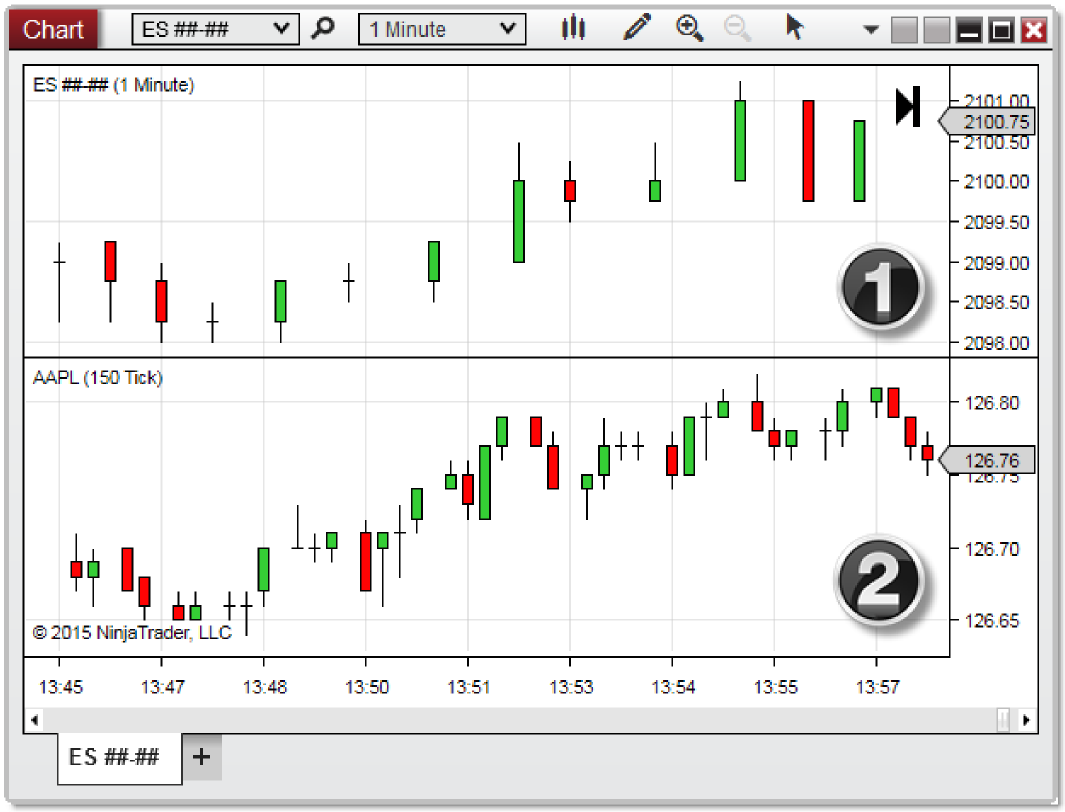
Managing Multiple Data Series

Multiple Data Series

The image below shows two Data Series plotted within one chart window:

1. ES ##-## (1 Min)

2. AAPL (150 Tick)

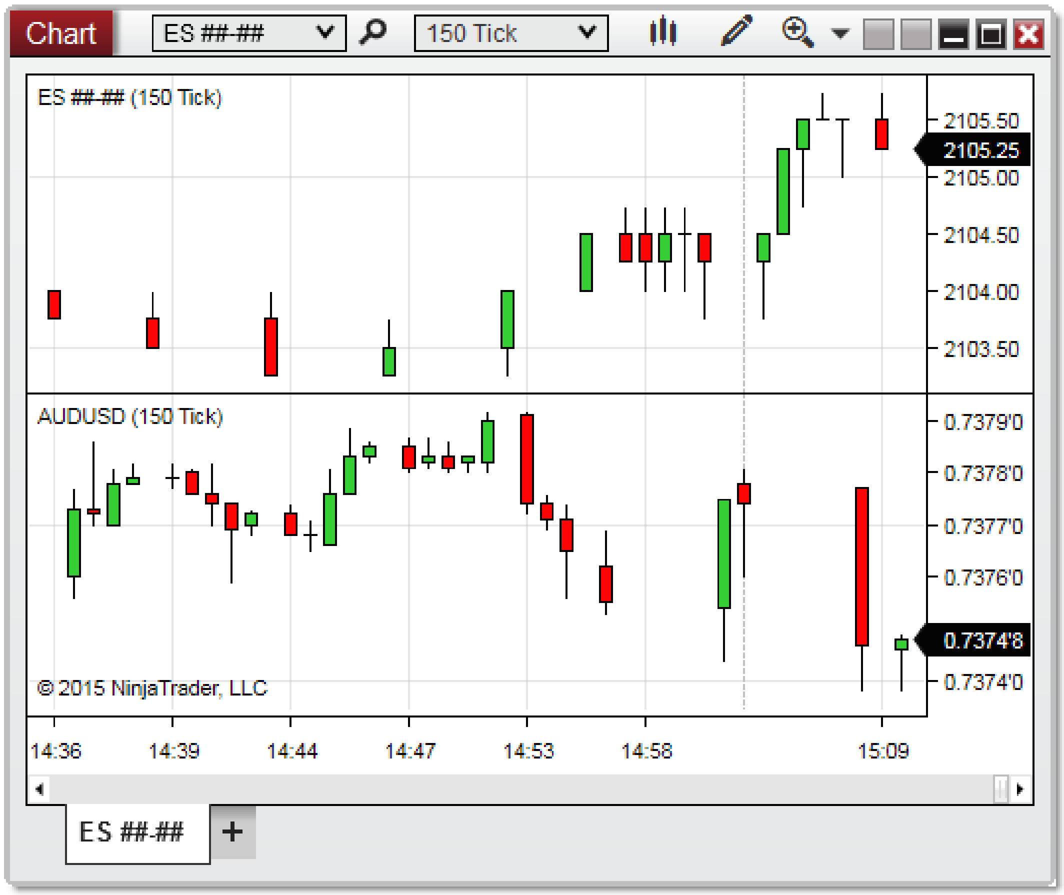


Each instrument is placed in its own panel by default, with the scale shown in the right margin of the chart. Many separate panels can be displayed within a single chart window. Instruments and indicators can alternatively be plotted within a single panel, as well. The scale of each **Data Series** can 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.

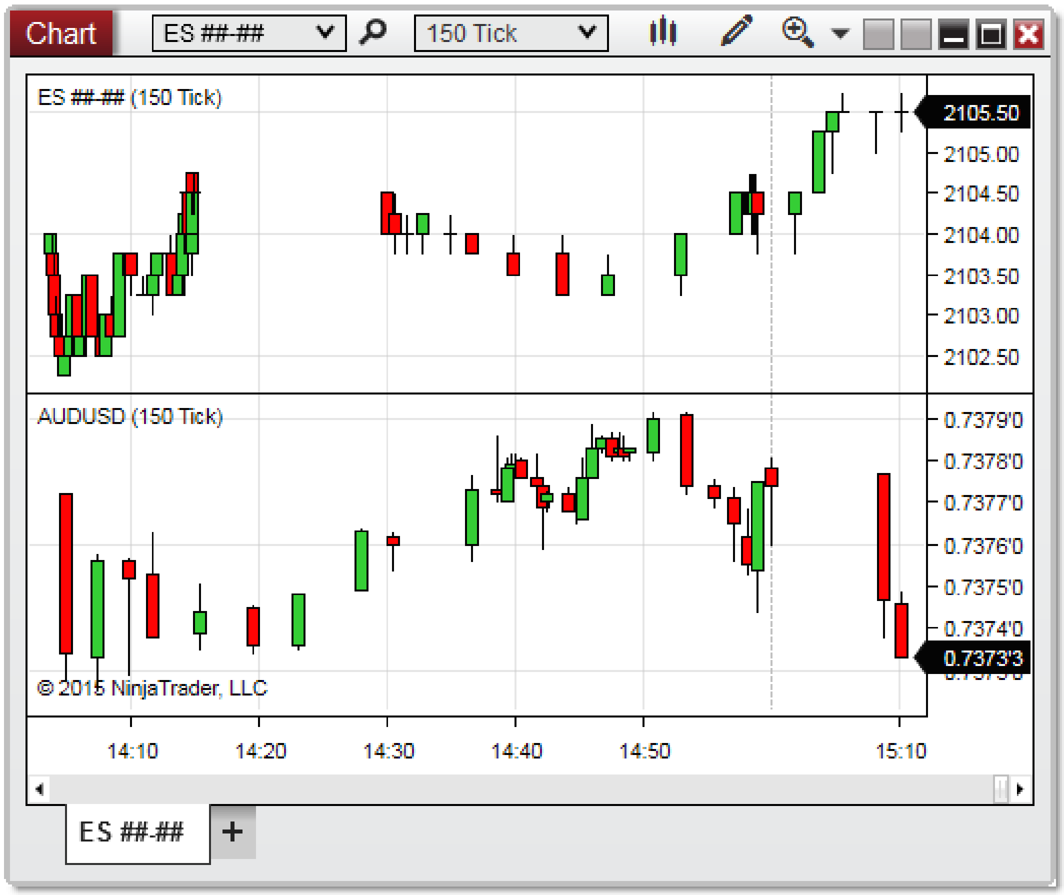
Tip: When more than one panel is displayed in a chart, you can temporarily maximize a panel to fill the entire chart window by right mouse clicking in the price axis of that specific panel, then selecting the **Maximize** option. To restore the panel to its original size and placement, you can then right 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. 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).

11.6.7 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**.

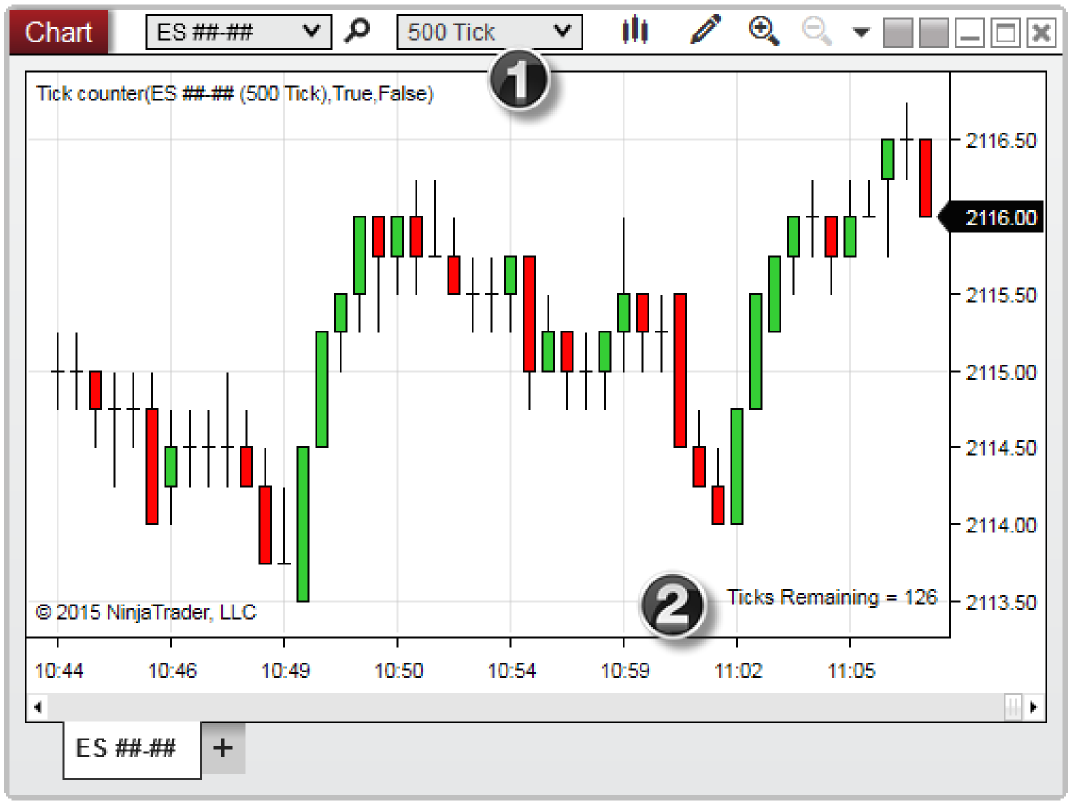
Notes:

- 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) 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.

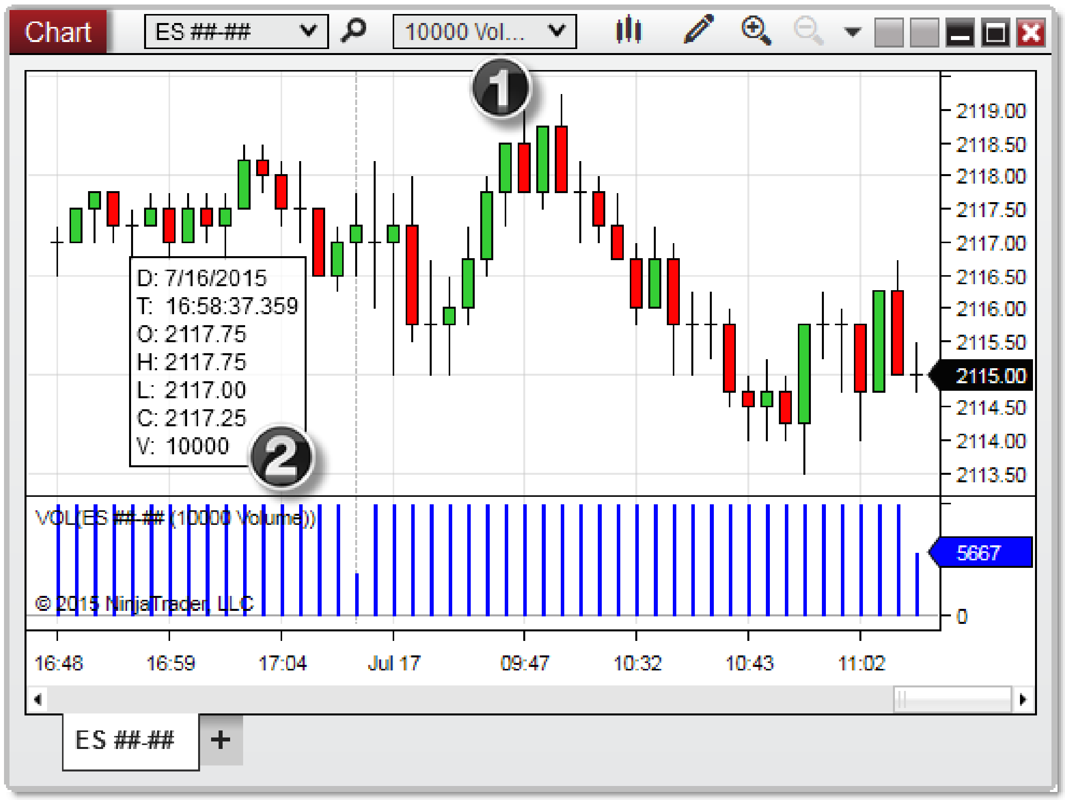


1. Each historical bar in the the 500 Tick chart shown above plots a total of 500 ticks.
2. The "Tick Counter" indicator has been applied to the chart to show the number of ticks remaining in the current bar.

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.

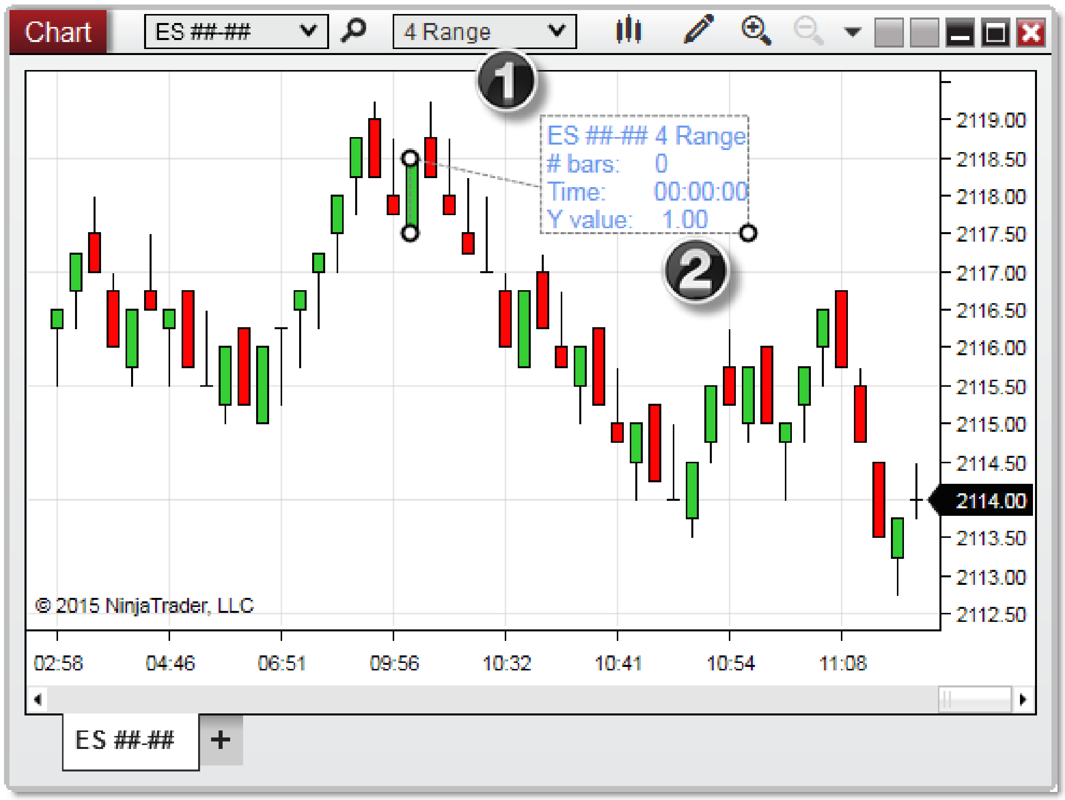
Understanding Range bars

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.

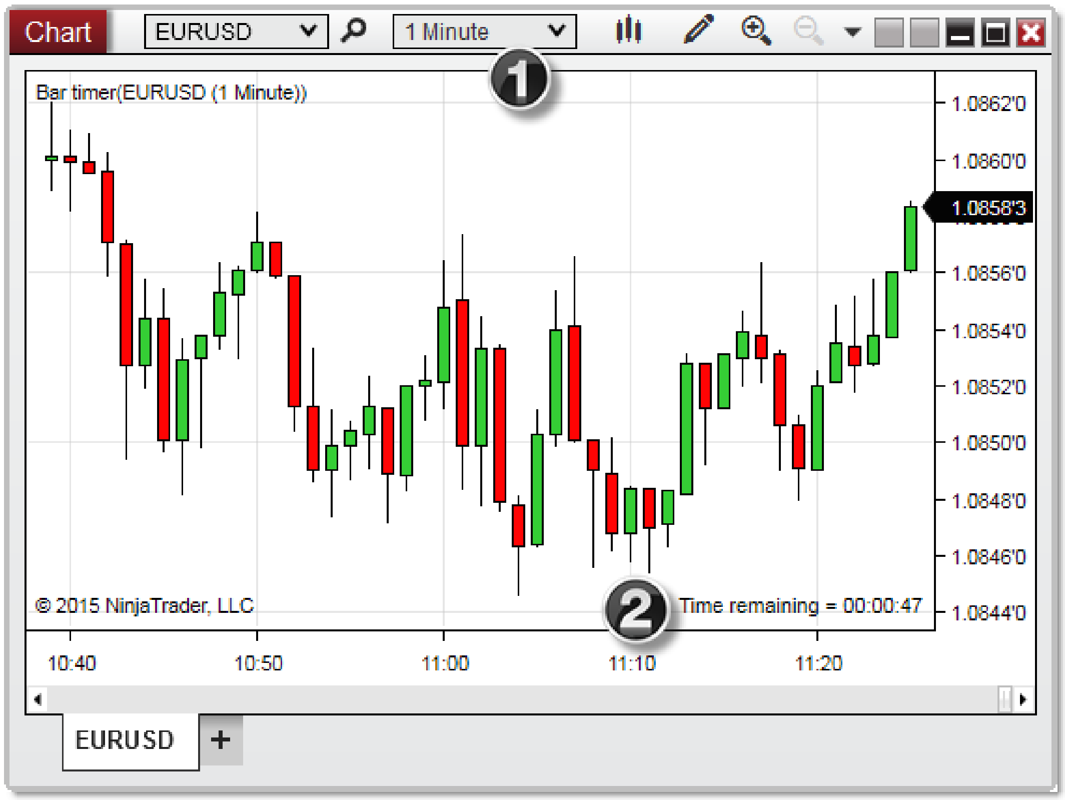


1. Each historical bar in the 4 **Range** chart shown below represents exactly 4 ticks of price movement.
2. The **Ruler** Drawing Tool verifies that each bar consists of 4 ticks. (The "Y value" of 1.00 shown in the **Ruler's** display flag is equivalent to 4 ticks for the e-mini S&P 500 continuous contract 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.

Understanding Heiken Ashi bars

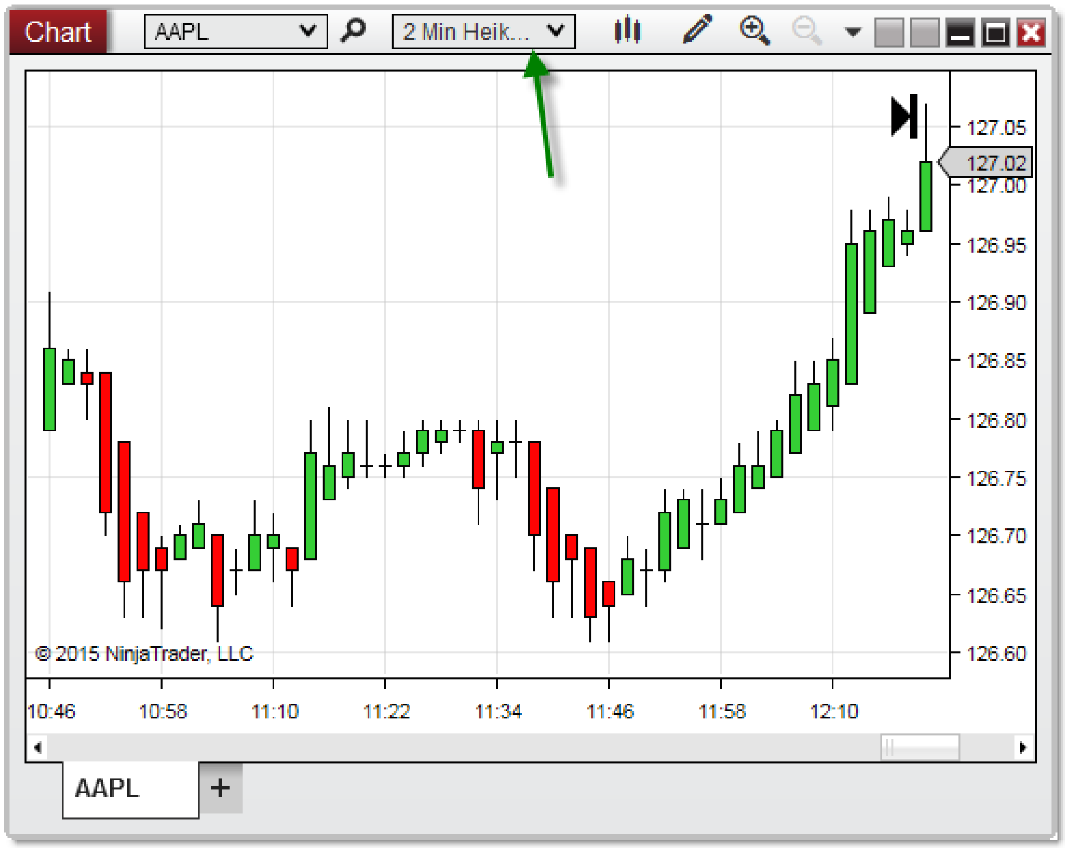
Heiken Ashi Bars

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:



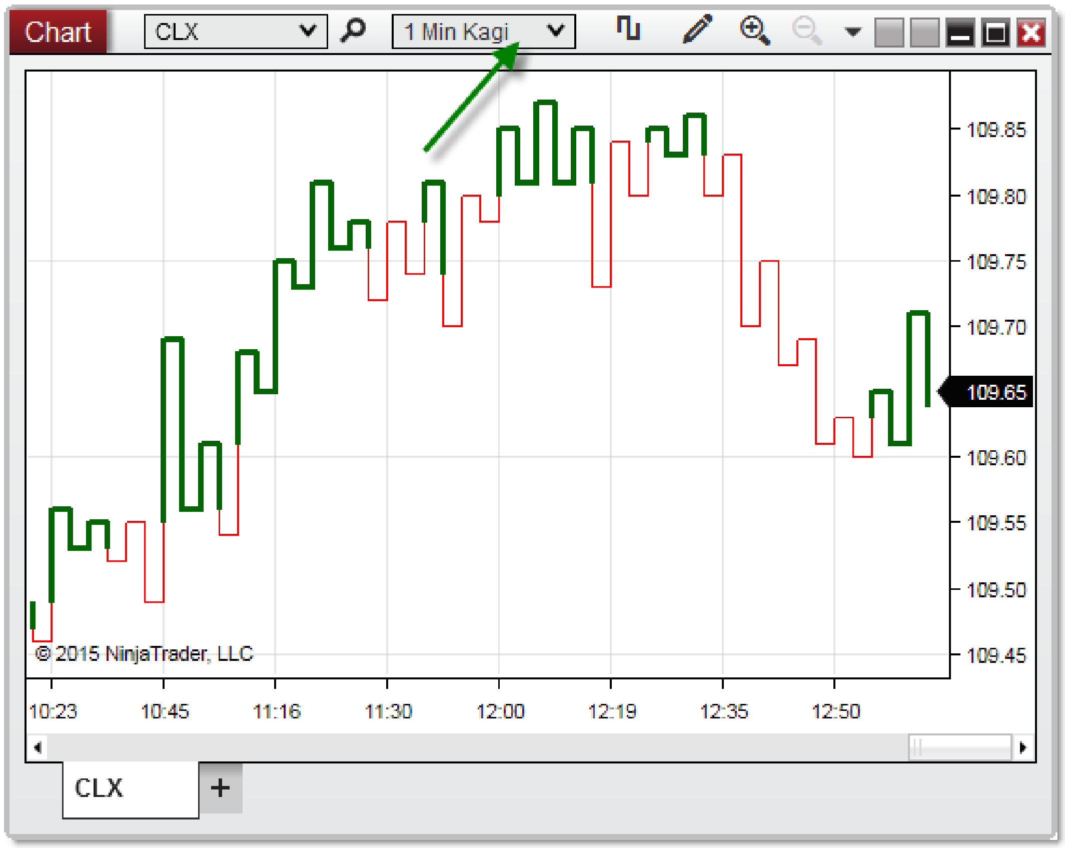
Understanding Kagi bars

Kagi Bars

Kagi bars are based solely on price movement with no regard to time or volume.

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**.

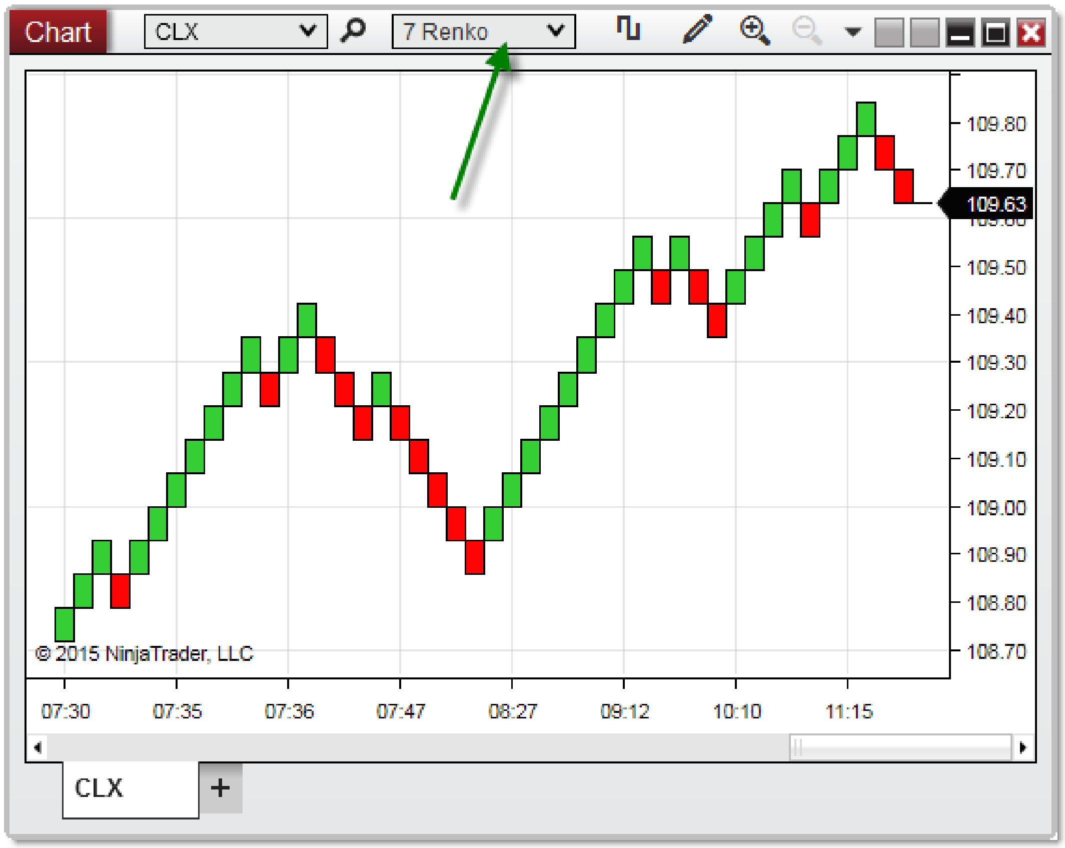
For example, suppose the price of an instrument is heading down and the **Reversal** is set to 2 ticks. The line will continue to plot downward until price reverses more than 2 ticks. At this point, the line will change direction, but stay red by default. Once the last **Kagi** bar High is exceeded, 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 solely on price movement with no regard to time or volume. 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

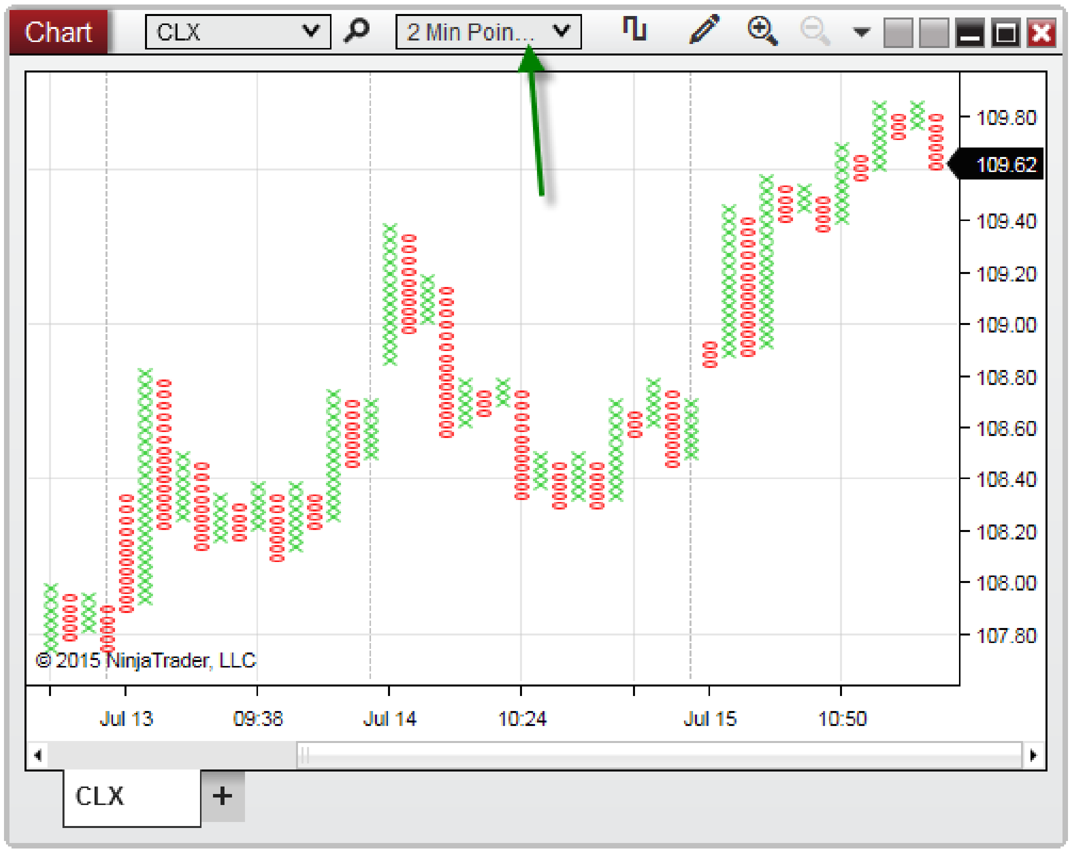
PointAndFigure Bars

PointAndFigure bars are built based solely on price movement with no regard to time or volume. 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.

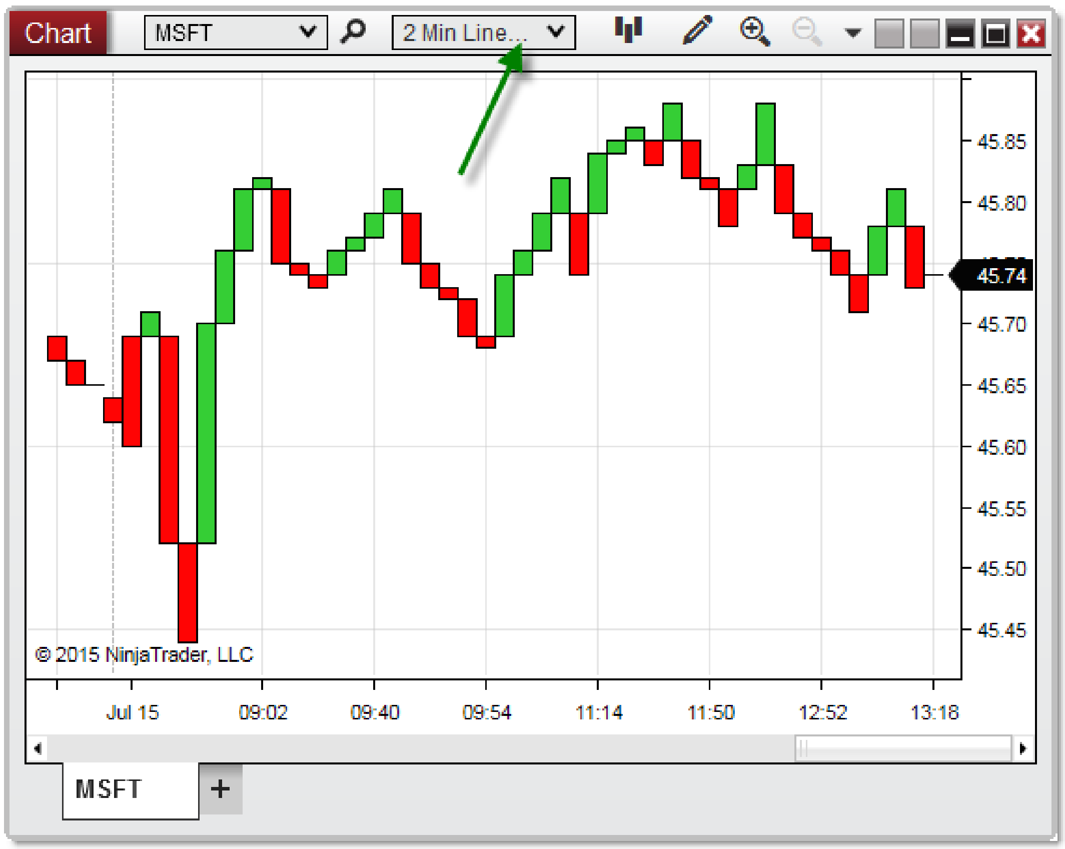


Understanding Line Break bars

Line Break Bars

Line Break bars are based solely on price movement with no regard to time or volume. **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.



11.6.8 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.

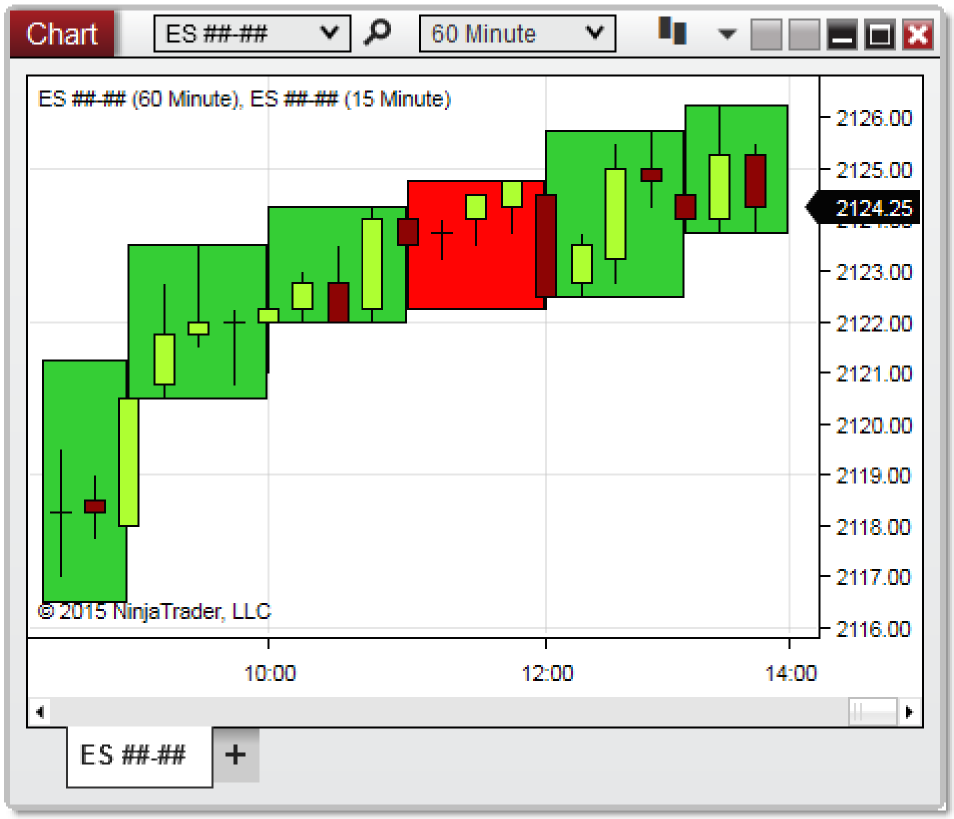
Understanding the Box Chart Style

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.

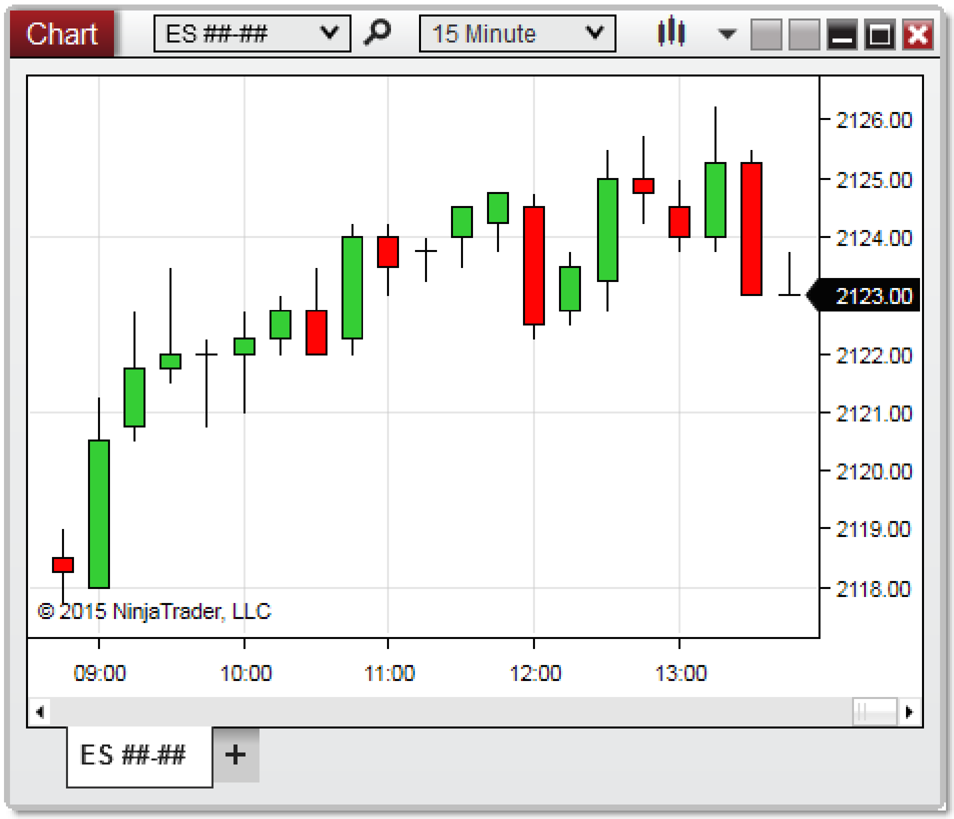
Understanding the Candlestick Chart Style

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**.

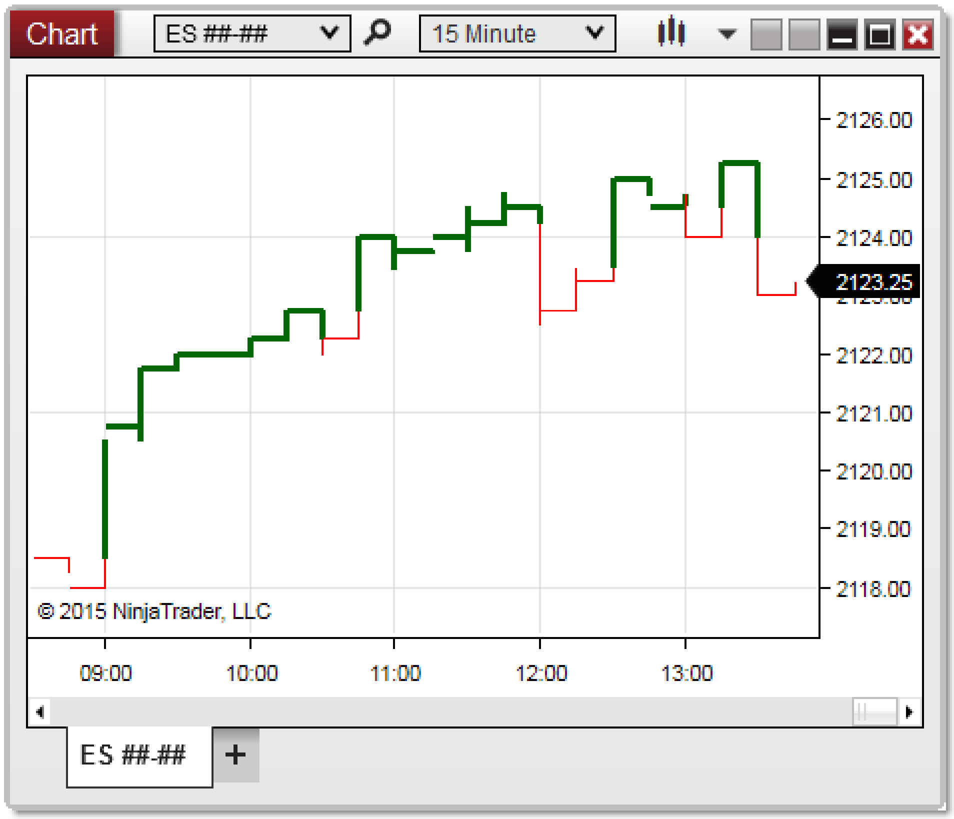
Understanding the Kagi Line Chart Style

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.



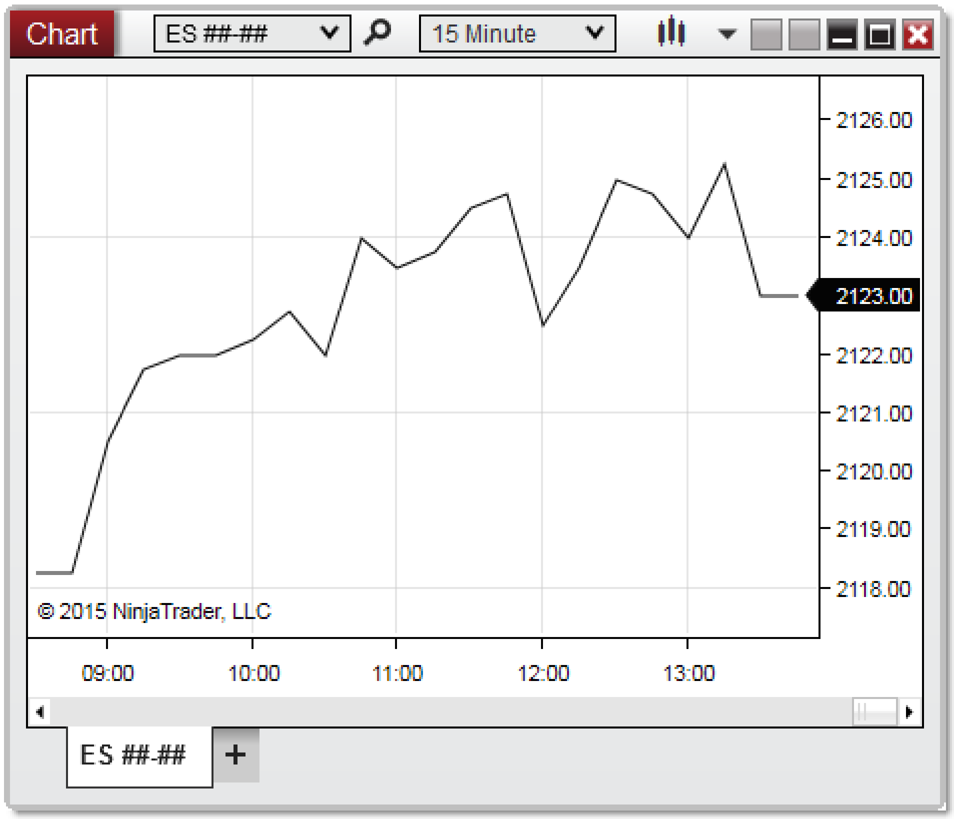
Understanding the Line on Close Chart Style

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.



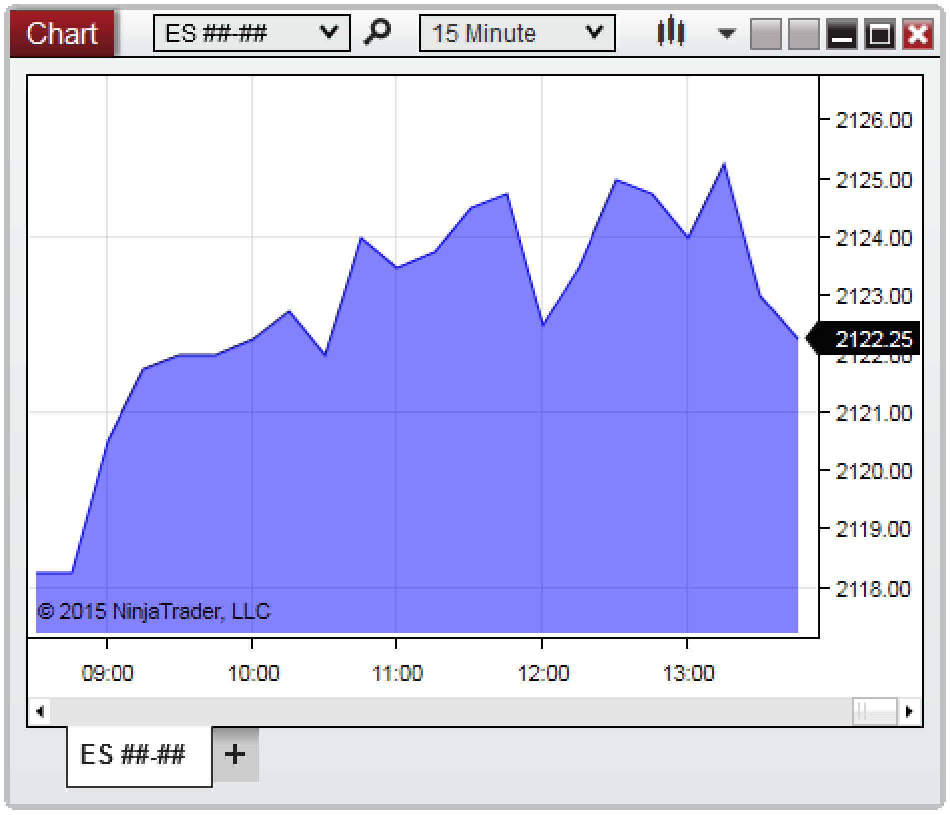
Understanding the Mountain Chart Style

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.

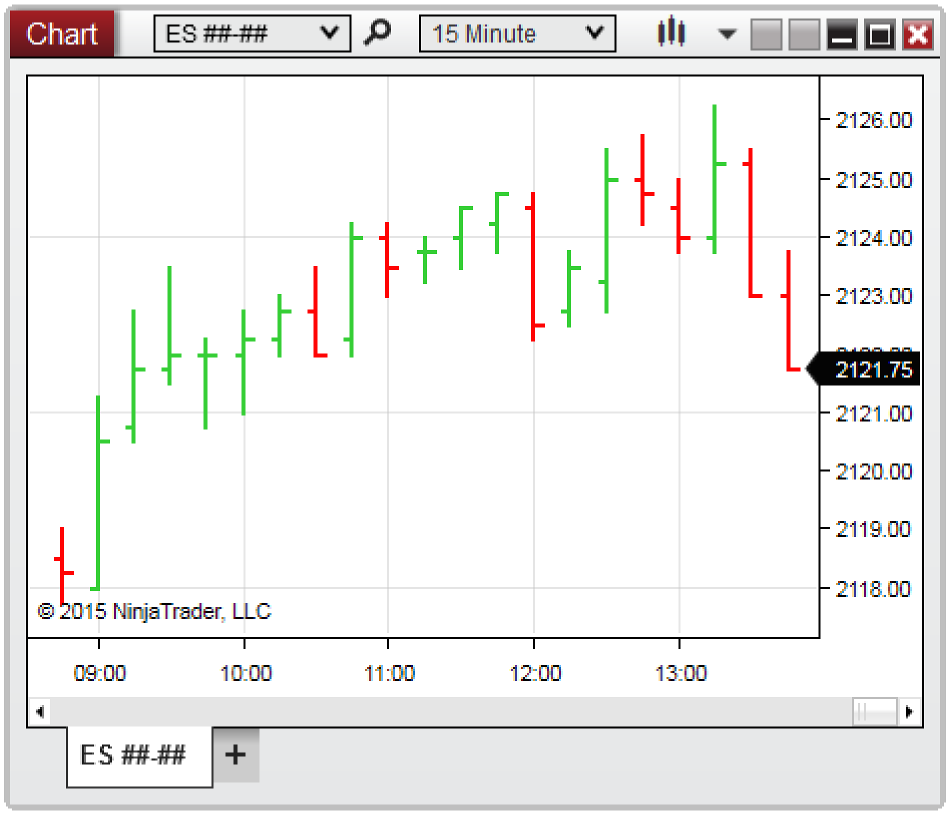
Understanding the OHLC Chart Style

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).



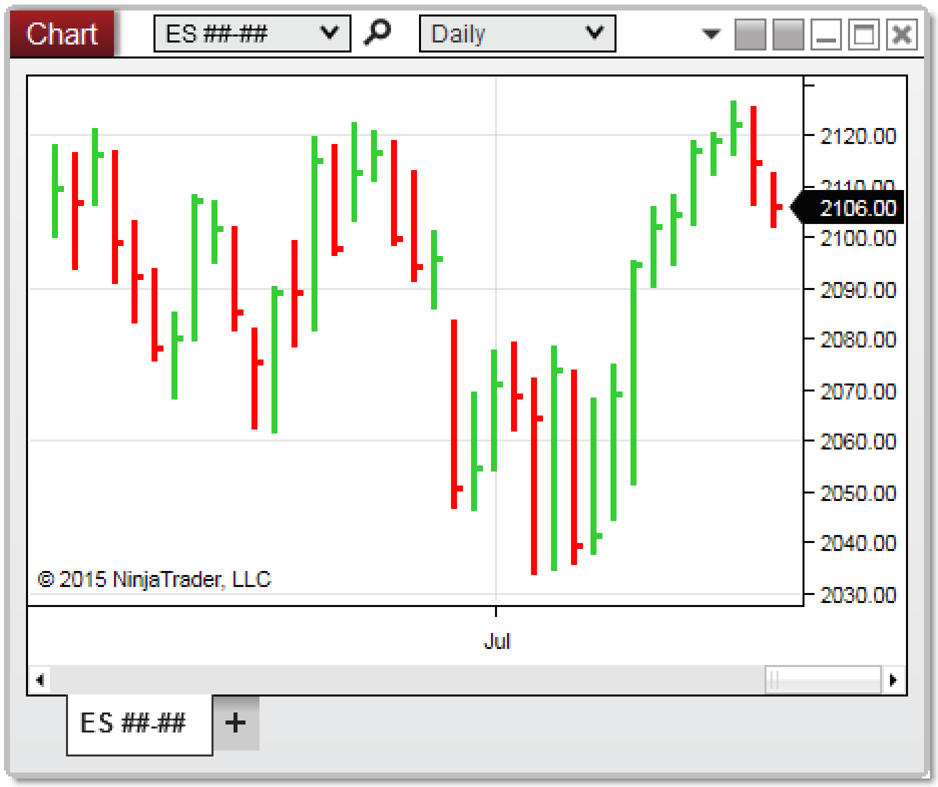
Understanding the HLC Chart Style

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.

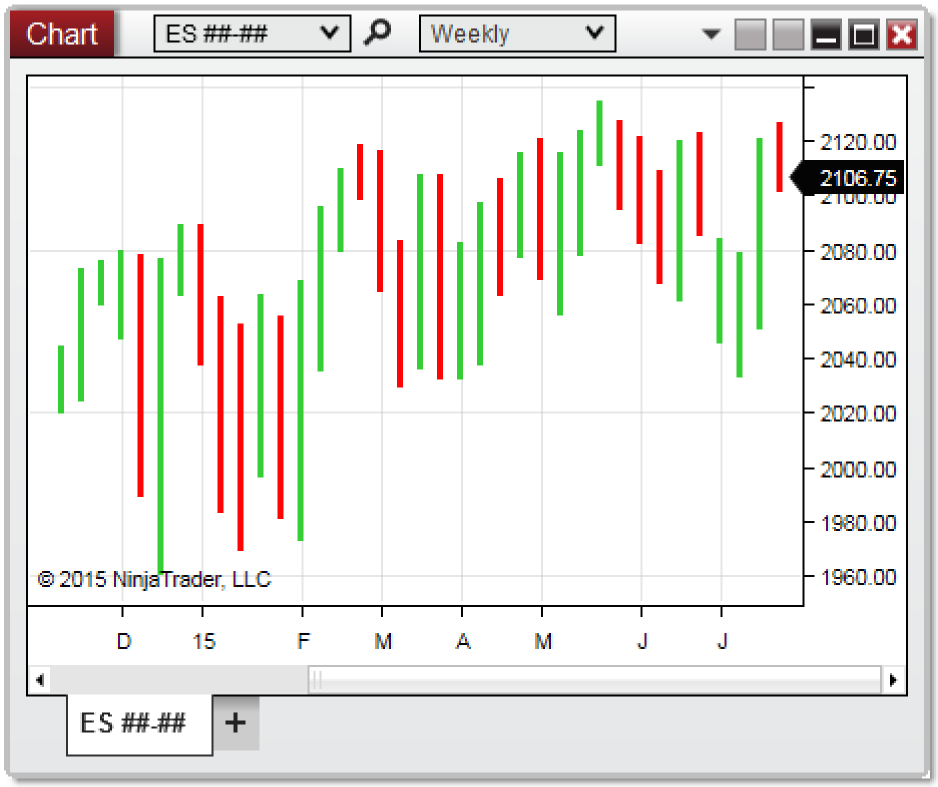
Understanding the HiLo Chart Style

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.

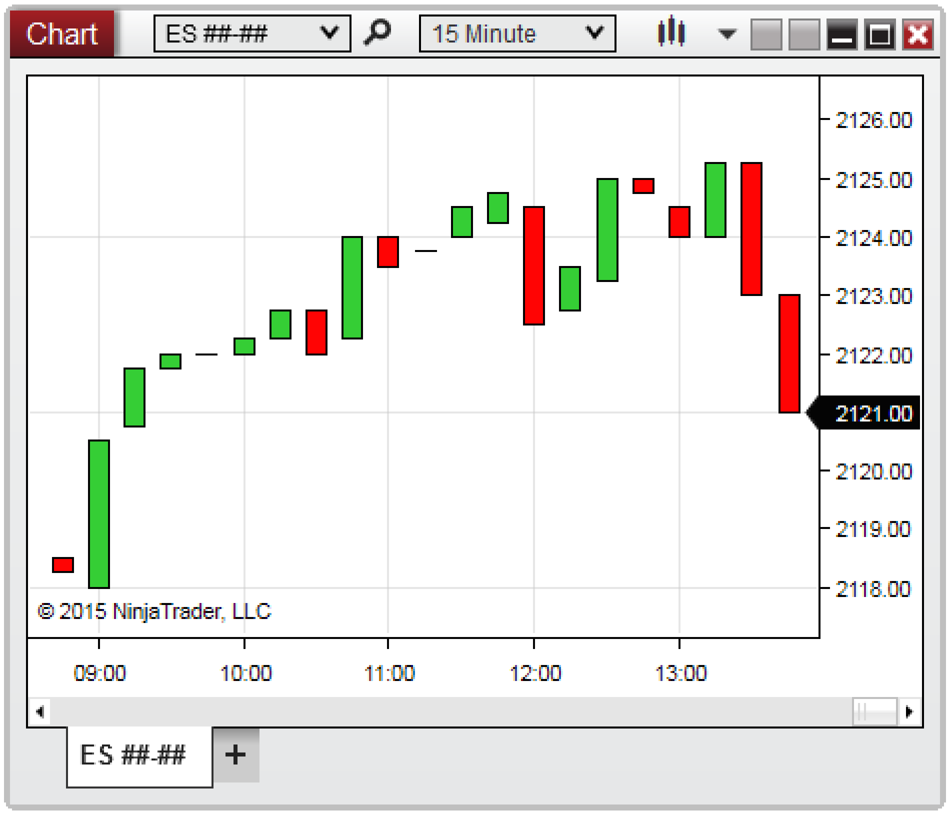
Understanding the Open/Close Chart Style

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**.

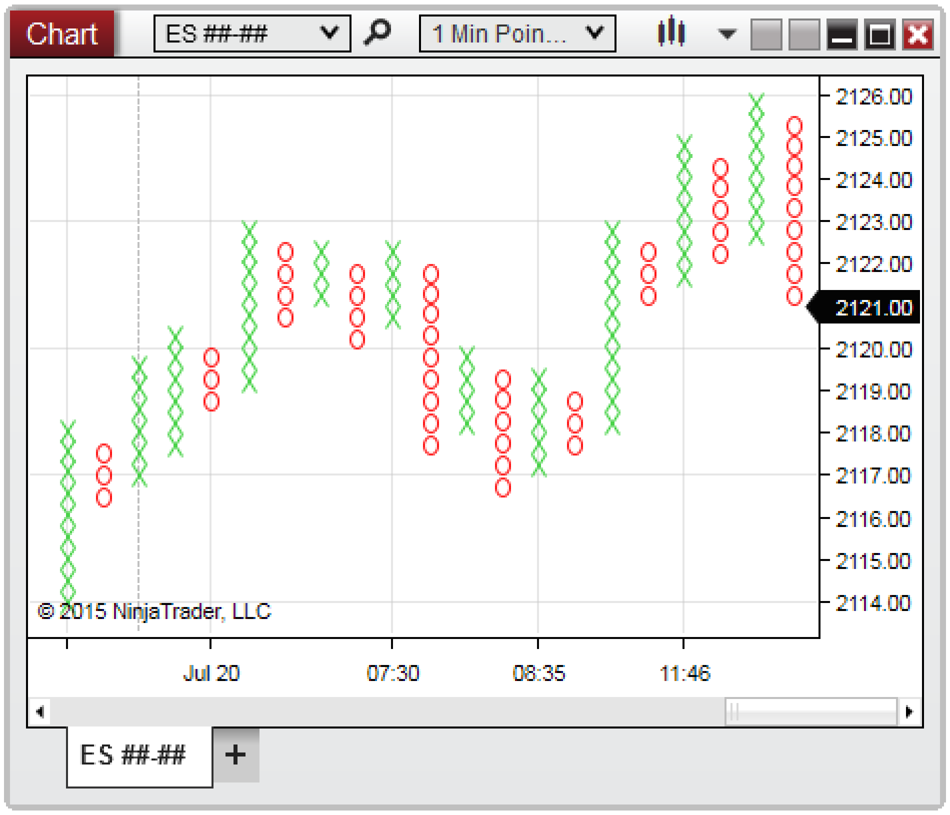
Understanding the Point and Figure Chart Style

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.



11.6.9 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.

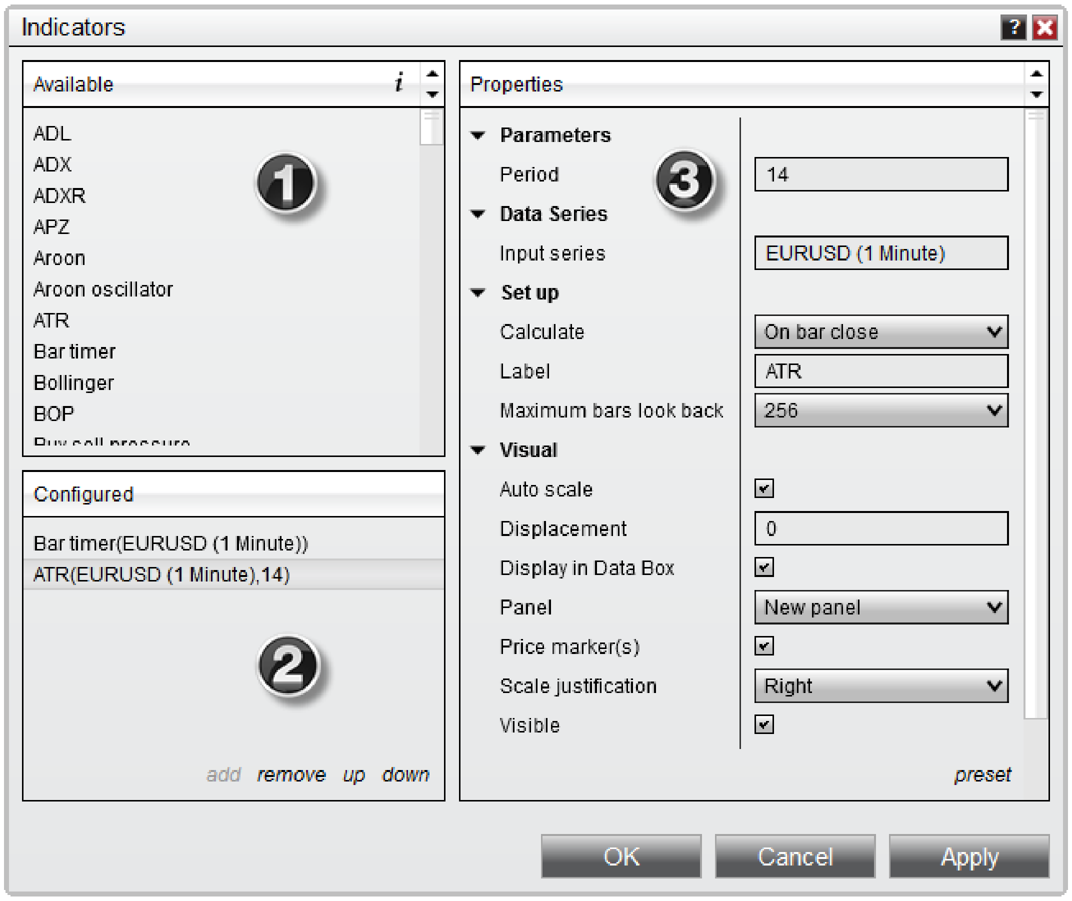
Accessing the Indicators Window from a Chart

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.



ADL
ADX
ADXR
APZ
Aroon
Aroon oscillator
ATR
Bar timer
Bollinger
BOP

Bar timer(EURUSD (1 Minute))
ATR(EURUSD (1 Minute),14)

Parameters	
Period	14
Data Series	
Input series	EURUSD (1 Minute)
Set up	
Calculate	On bar close
Label	ATR
Maximum bars look back	256
Visual	
Auto scale	✓
Displacement	0
Display in Data Box	✓
Panel	New panel
Price marker(s)	✓
Scale justification	Right
Visible	✓

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

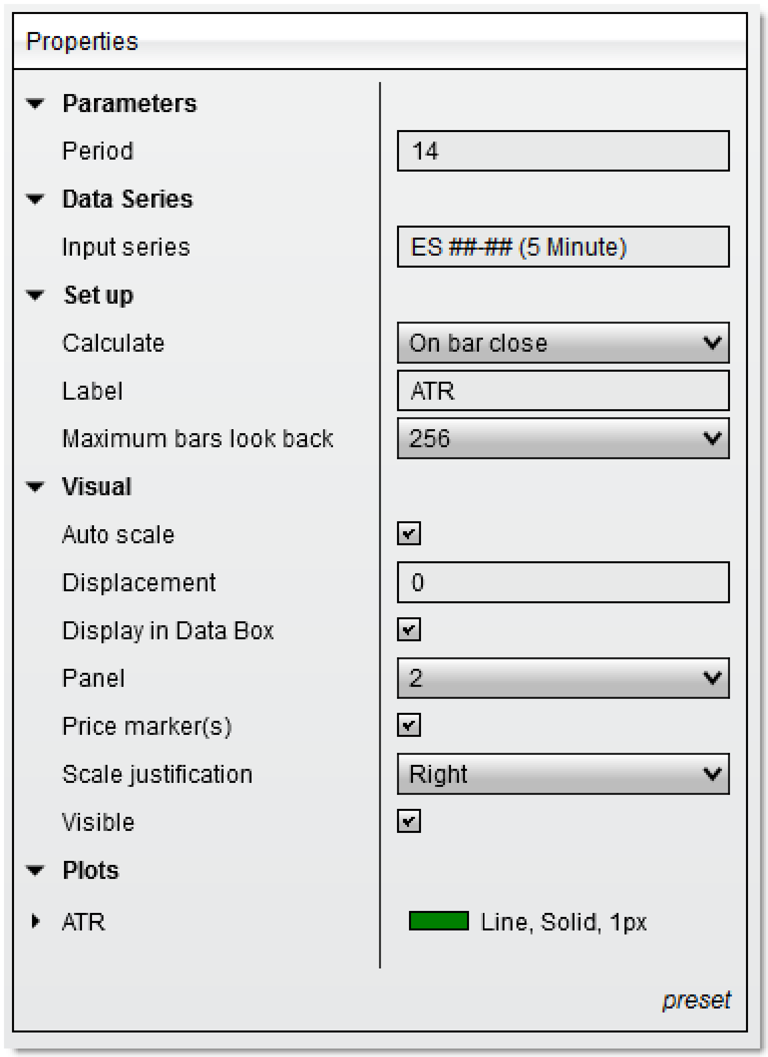
Editing an Indicator

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:



Properties

▾ **Parameters**	
Period	14
▾ **Data Series**	
Input series	ES ##-## (5 Minute)
▾ **Set up**	
Calculate	On bar close
Label	ATR
Maximum bars look back	256
▾ **Visual**	
Auto scale	✓
Displacement	0
Display in Data Box	✓
Panel	2
Price marker(s)	✓
Scale justification	Right
Visible	✓
▾ **Plots**	
▸ ATR	🟩 Line, Solid, 1px

preset

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 axis on which the indicator will be plotted within its selected panel. See the Navigating a chart page for additional information.
Visible	Enables or disables visibility 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 - Run calculations at the close of a bar if that bar's Close price differs from its Open price

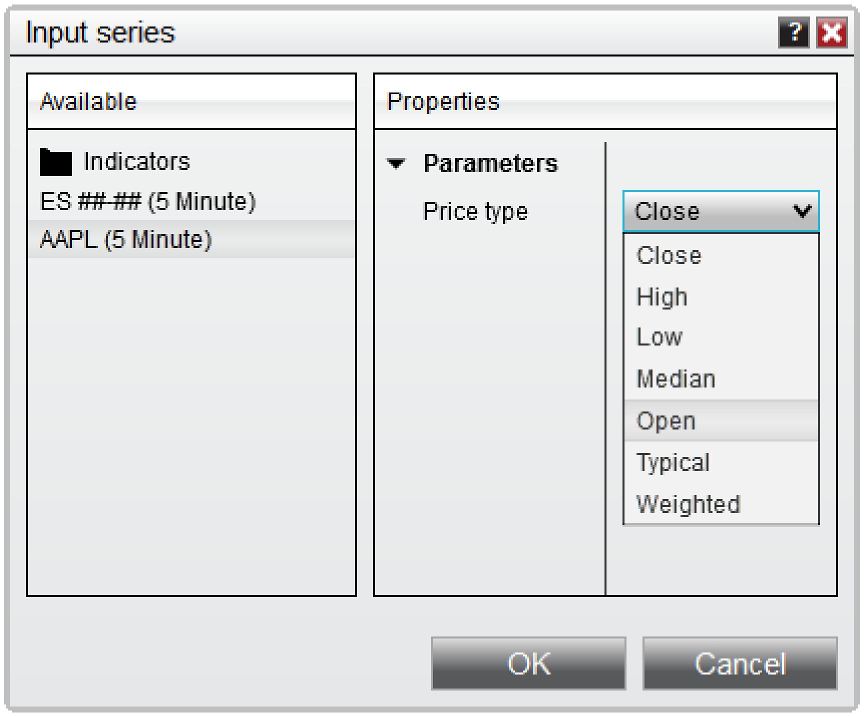
Saving an Indicator's Parameters as Default

You can optionally save your customized indicator's parameters as default. Doing so 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**.

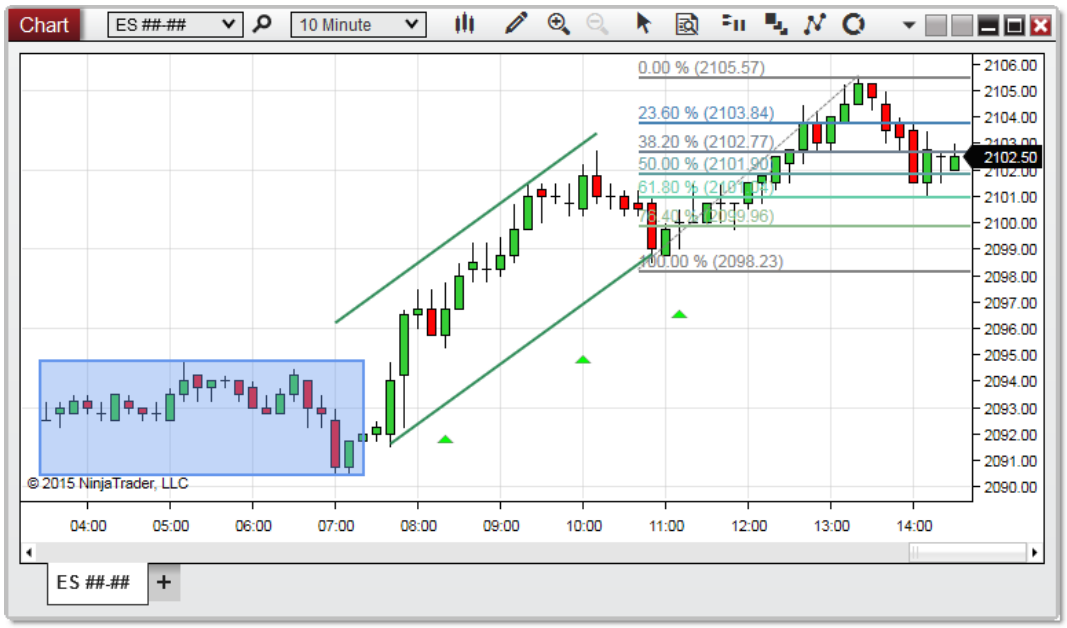
11.6.10 Working with Drawing Tools & Objects

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.

▽ 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.

Ruler Tool

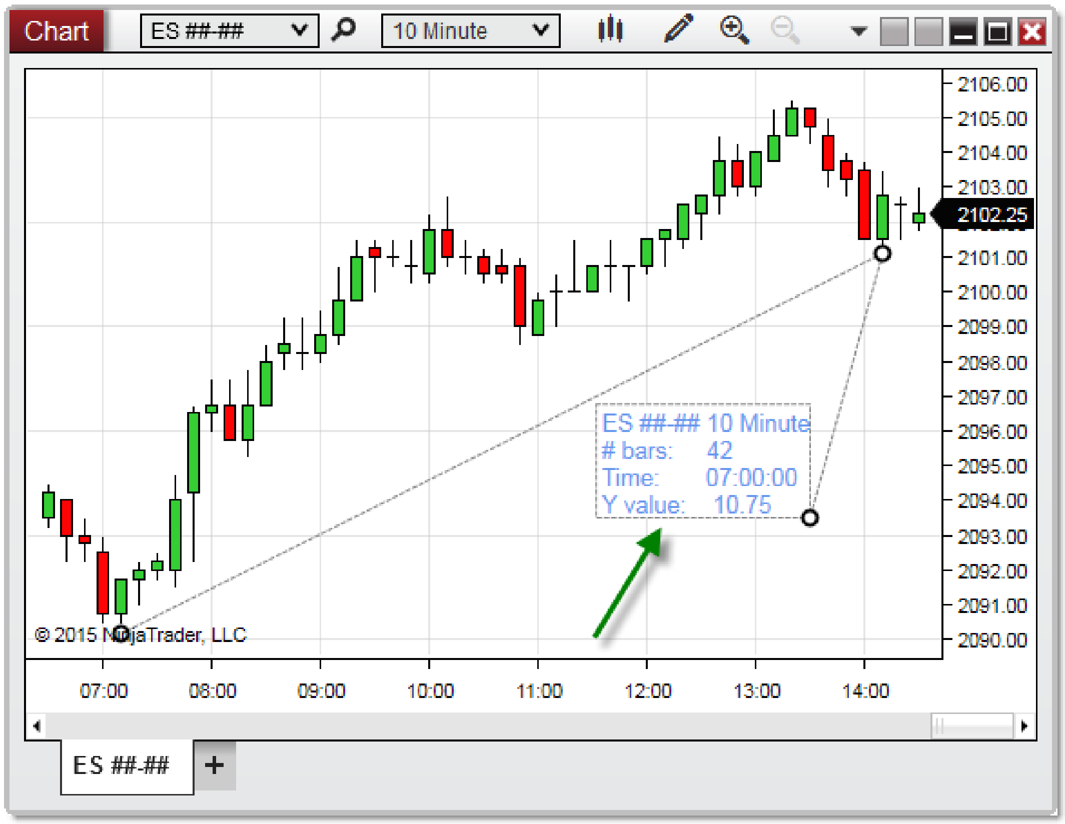
The **Ruler** measures the number of bars, length of time, and y-axis distance between two anchor points for a **Data Series.** The measurement data is attached in a flag at a third, independent anchor point.

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.


ES ##-## 10 Minute
# bars: 42
Time: 07:00:00
Y value: 10.75

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.

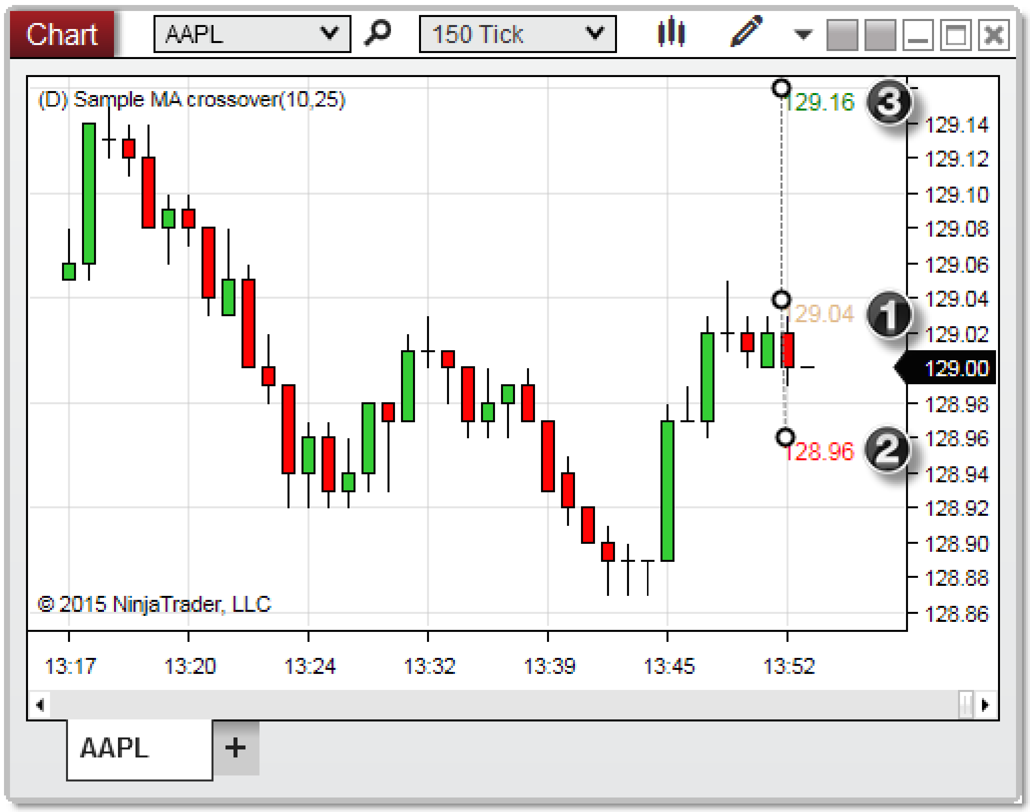
Risk-Reward Tool

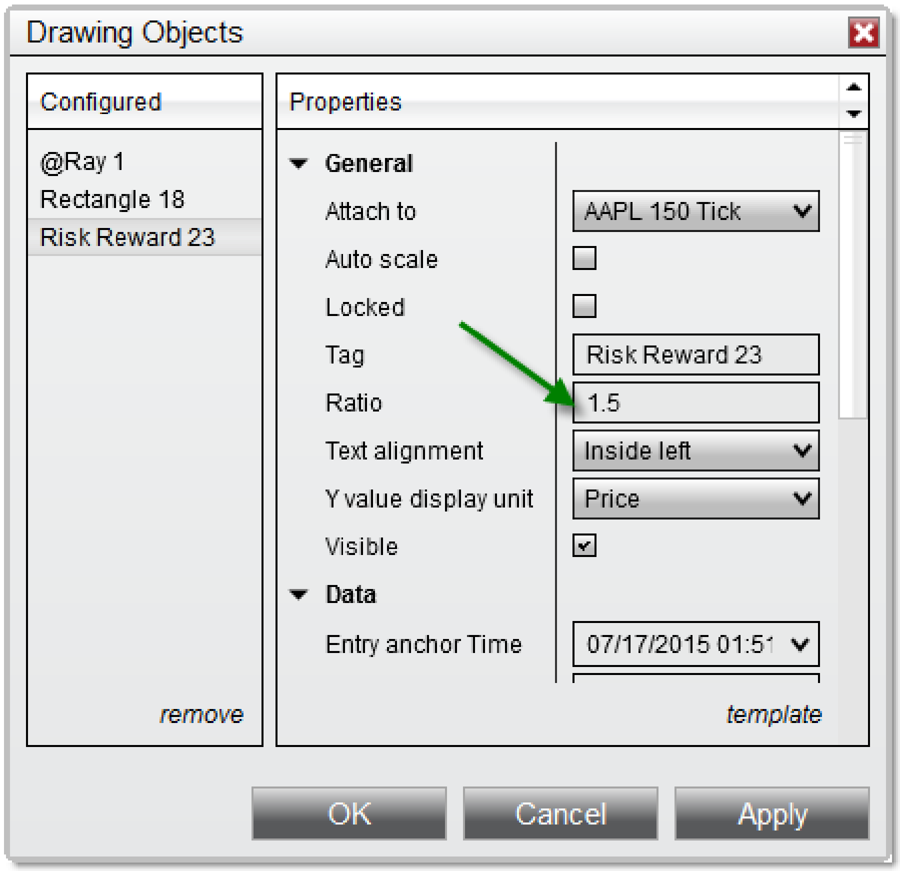
The **Risk-Reward** tool can help you to determine the placement of your profit targets to achieve a specific risk/reward ratio on any trade.

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.





Drawing Objects

Configured

@Ray 1
Rectangle 18
Risk Reward 23

Properties

General
Attach to	AAPL 150 Tick
Auto scale	☐
Locked	☐
Tag	Risk Reward 23
Ratio	1.5
Text alignment	Inside left
Y value display unit	Price
Visible	✓

Data
Entry anchor Time	07/17/2015 01:51

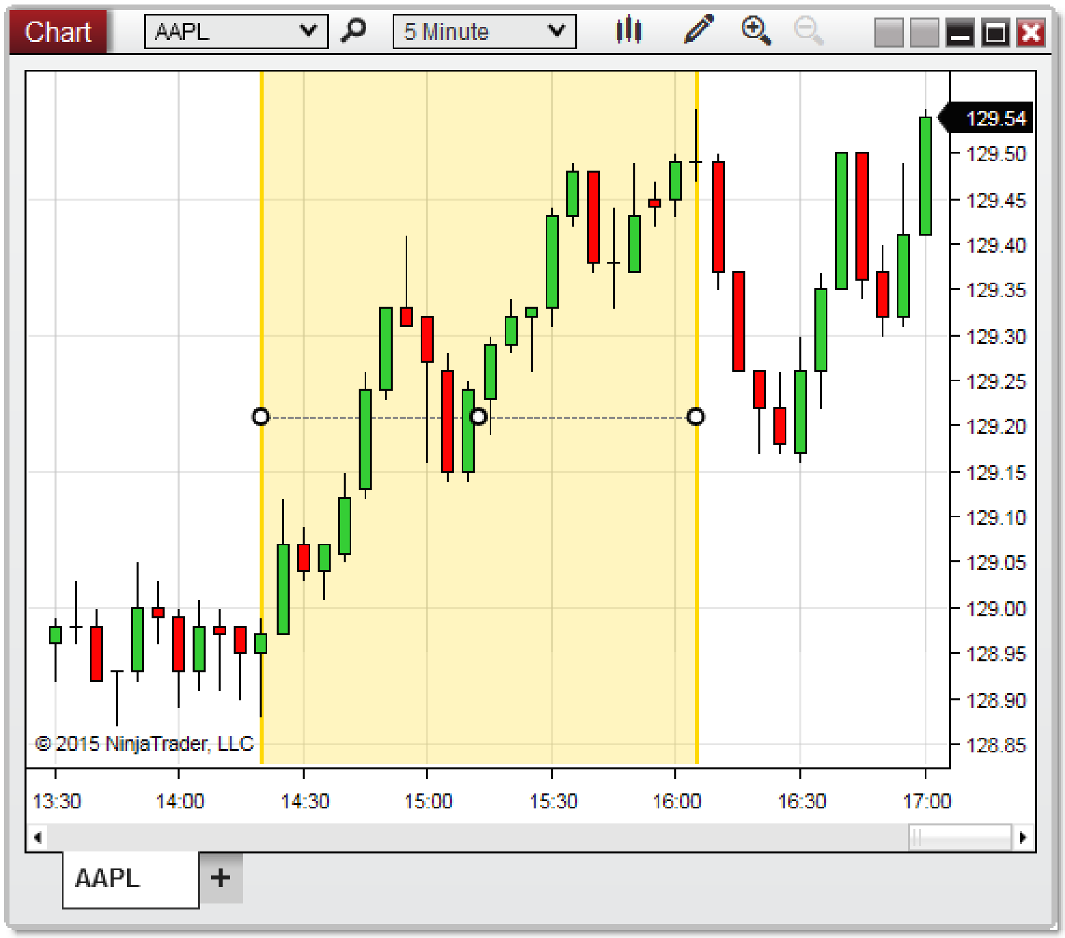
1. In the images above, the first anchor point is set at 129.04, with a risk/reward ratio of 1.5
2. The second anchor point (the maximum risk) is set at 128.96
3. Based on the $0.08 distance between the first and second anchor points (129.04 - 128.96), the third anchor point is automatically placed at 129.16 to achieve a 1:1.5 risk/reward ratio (129.04 + (0.08 * 1.5))

Region Highlight X / Region Highlight Y

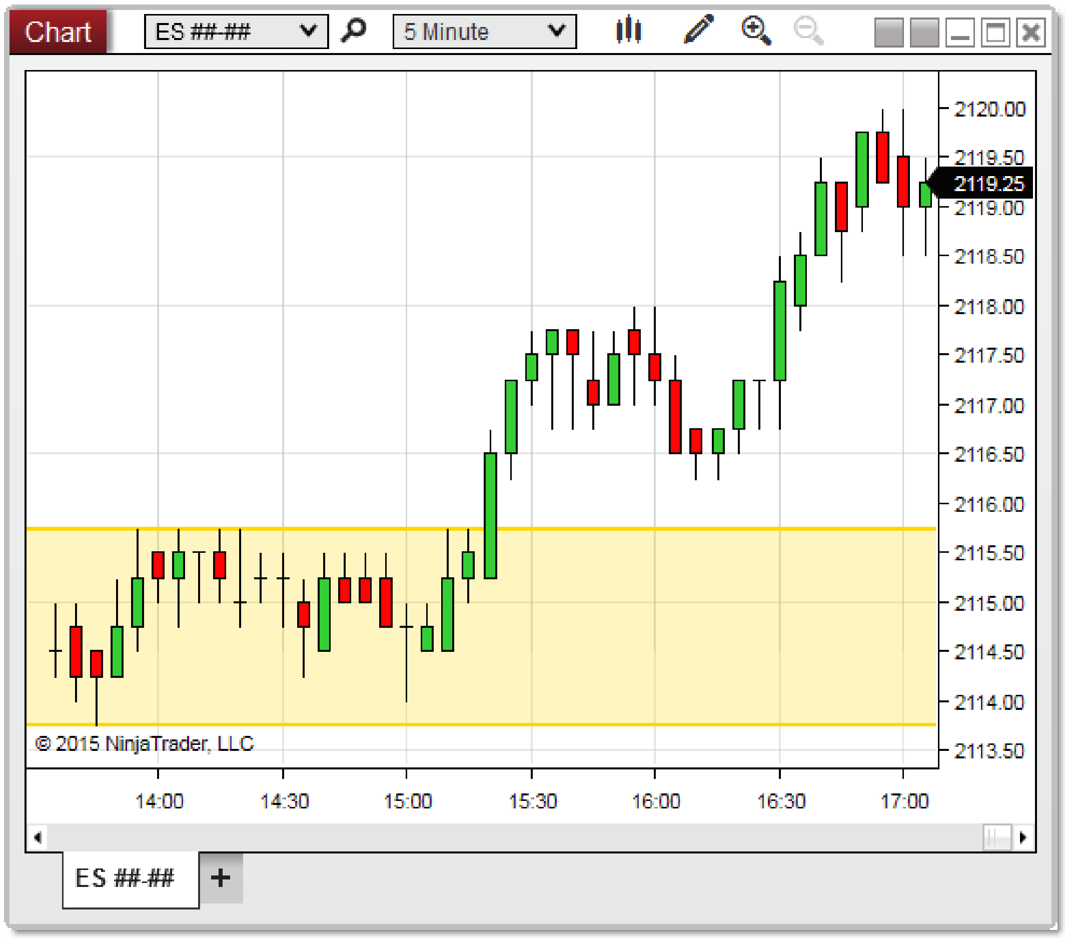
The **Region Highlight X** and **Region Highlight Y** tools allow you to highlight or shade an entire horizontal or vertical region on a chart. The **Highlight Region X** tool will highlight a horizontal region, and the highlighting will extend indefinitely upwards and downwards, keeping the highlight 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
Fibonacci Retracements	F8
Fibonacci Extensions	F9
Fibonacci Time Extensions	F10
Fibonacci Circle	F11
Andrew's Pitchfork	Ctrl + F8
Gann Fan	Ctrl + F9

© 2017 NinjaTrader, LLC

Regression Channel	Ctrl + F10
Trend Channel	Ctrl + 2
Ellipse	Ctrl + F11
Rectangle	Ctrl+ F12
Triangle	Ctrl + F6
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 **Data Series** OHLC and indicator price values only
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.

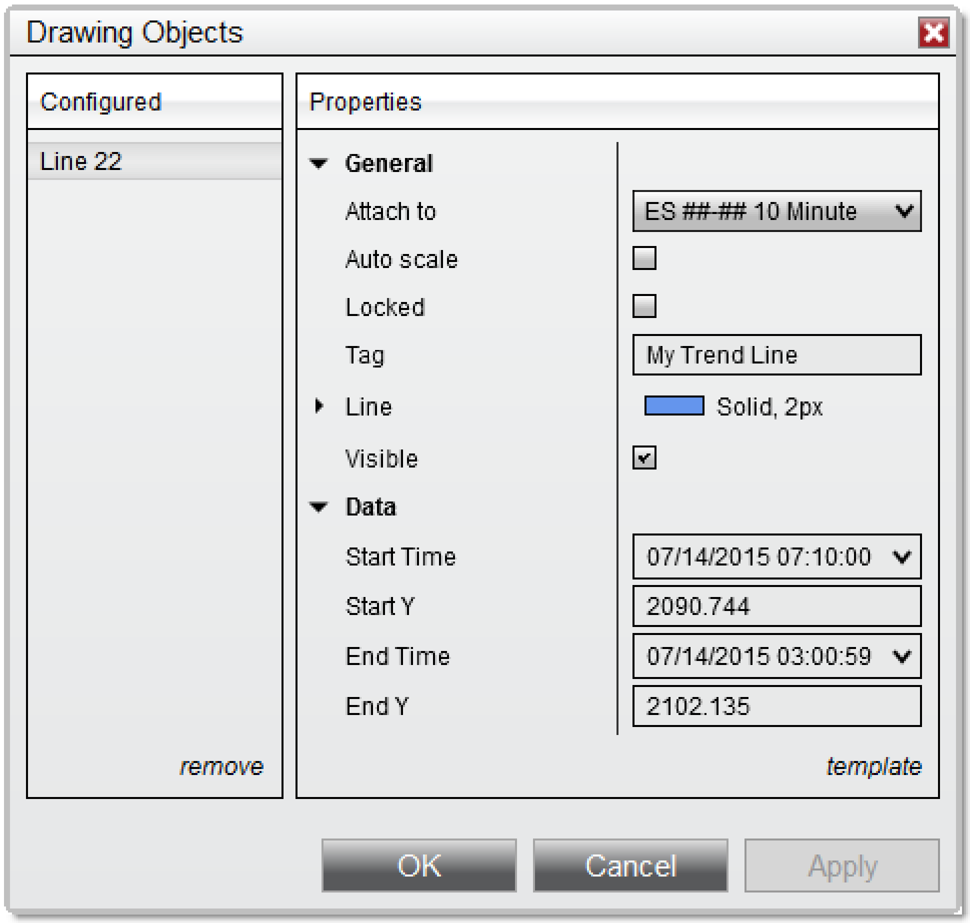
The **Drawing Objects** dialogue is also accessible directly from the **Drawing Tools** menu on the chart toolbar, or by right mouse clicking in a chart, then selecting the **Drawing Tools** menu item.

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**.


Drawing Objects

Configured

Line 22

remove

Properties

General

Attach to	ES ##-## 10 Minute
Auto scale	☐
Locked	☐
Tag	My Trend Line
Line	Solid, 2px
Visible	✓

Data

Start Time	07/14/2015 07:10:00
Start Y	2090.744
End Time	07/14/2015 03:00:59
End Y	2102.135

template

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)
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.

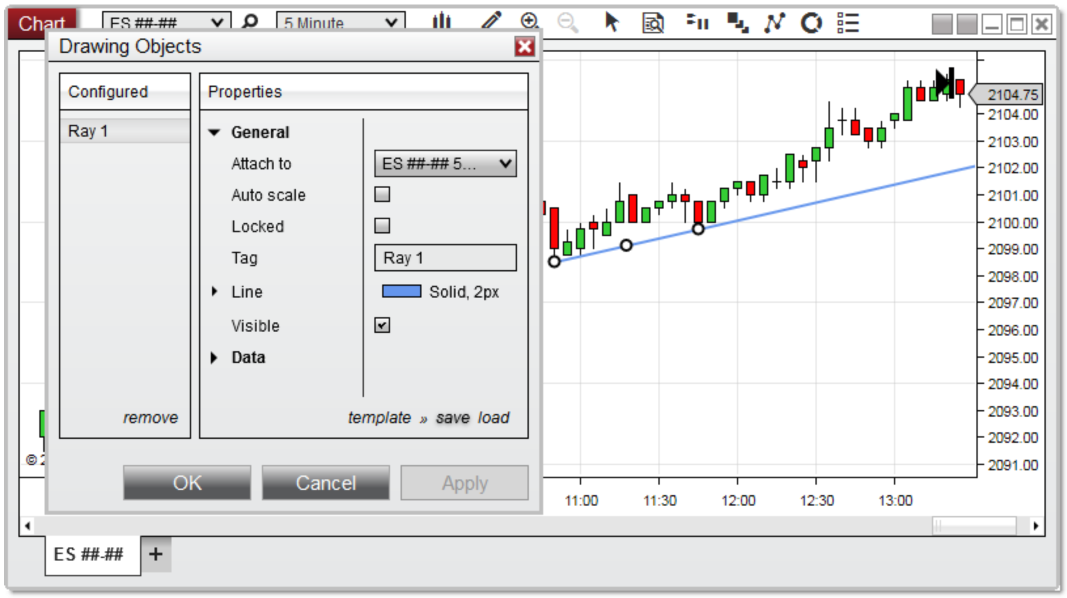
Saving Drawing Object Templates

To save **Drawing Object** settings as default:

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 or select an existing template to overwrite it.
4. Click the **Save** button when finished

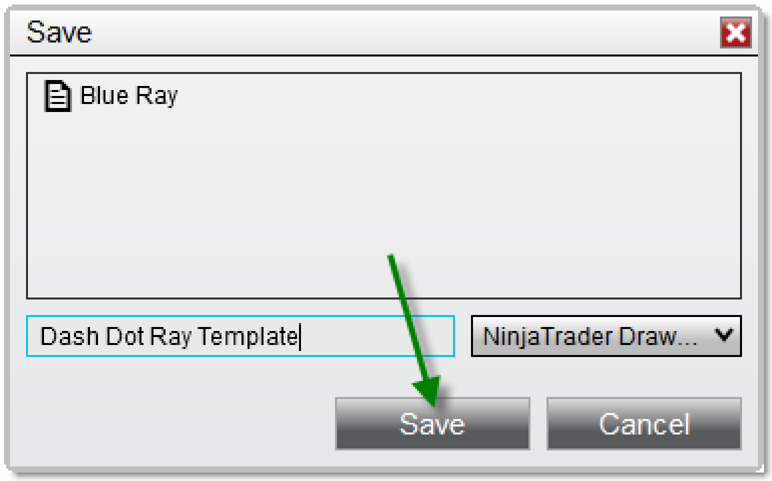
If you wish to load a previously saved template, you can select the **load** option after left mouse clicking on the **template** text.

In the image below, a template will be saved for the **Ray** drawing tool.



General
Attach to: ES ##-## 5...
Auto scale: ☐
Locked: ☐
Tag: Ray 1
Line: Solid, 2px
Visible: ✓
Data
template » save load

In the image below, we can enter a name for the new **Ray** template and save it for future use.



Tip: Saving a new **Drawing Object** template with the name "Default" will make it the new default template for that **Drawing Tool**. The new default template will then be automatically applied to any new **Drawing Object** of that particular type in the future.

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:

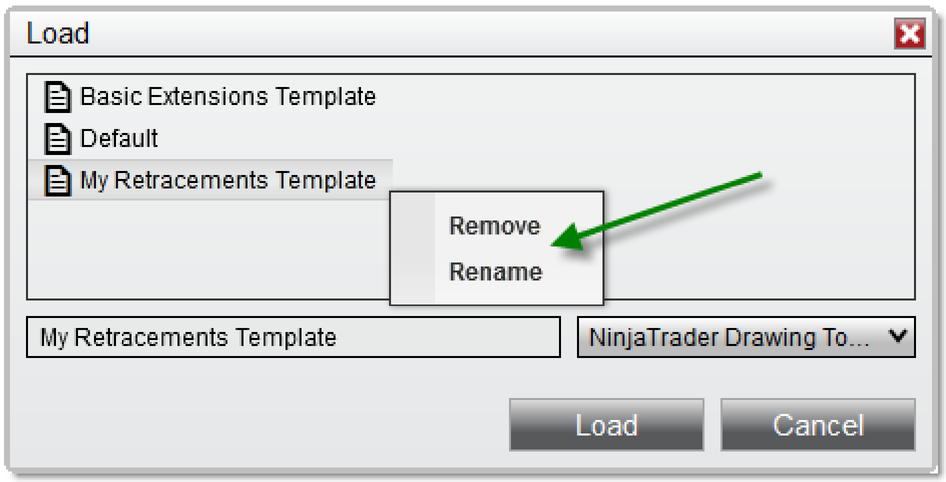
3. Left mouse click on the **template** text, then select either the **Save** or **Load** menu items
1. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the template for removal from the list of templates, then select the **Remove** menu item.

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

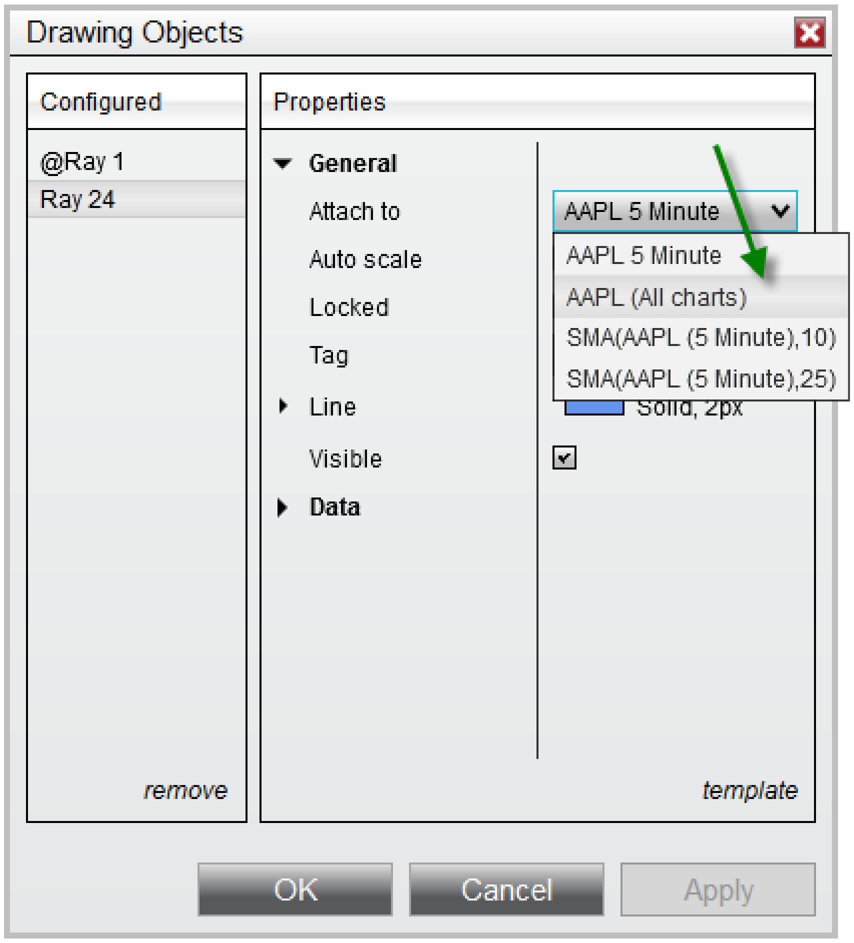
Drawing Objects can be applied to a specific chart (local), or to all charts of the same instrument (global).

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.


Drawing Objects

Configured

@Ray 1

Ray 24

remove

Properties

General

Attach to: AAPL 5 Minute

AAPL 5 Minute

AAPL (All charts)

SMA(AAPL (5 Minute),10)

SMA(AAPL (5 Minute),25)

Auto scale

Locked

Tag

Line: Solid, 2px

Visible: ✓

Data

template

Note: **Global Drawing Objects** will be automatically deleted after 20 days of not being viewed. This is done to conserve PC resources.

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.

Understanding drawing object levels

Drawing Object Levels

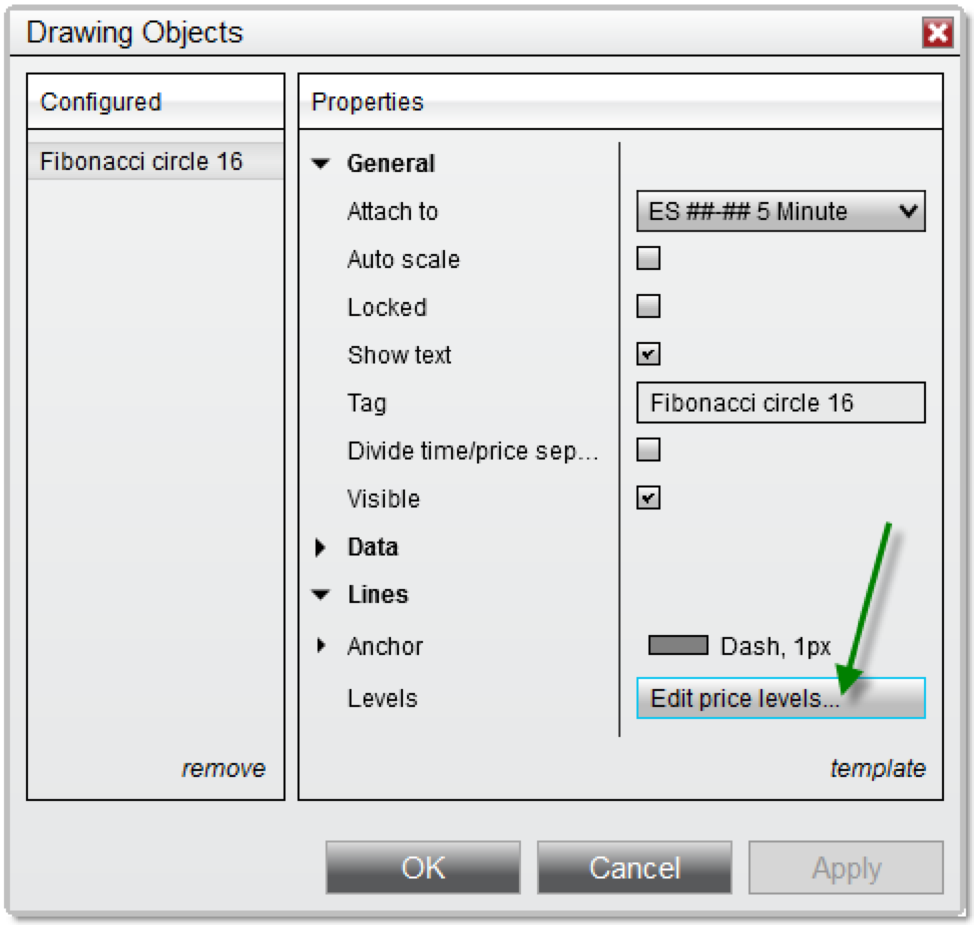
Drawing Tools that include lines drawn at multiple, customizable price levels, such as **Fibonacci Retracements**, include a "Levels" property which can be used to add, remove, or edit levels displayed in objects drawn with that **Drawing Tool**.

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.


Drawing Objects

Configured
- Fibonacci circle 16

Properties

General	
Attach to	ES ##-## 5 Minute
Auto scale	☐
Locked	☐
Show text	✓
Tag	Fibonacci circle 16
Divide time/price sep...	☐
Visible	✓
Data	
Lines	
Anchor	Dash, 1px
Levels	Edit price levels...

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.

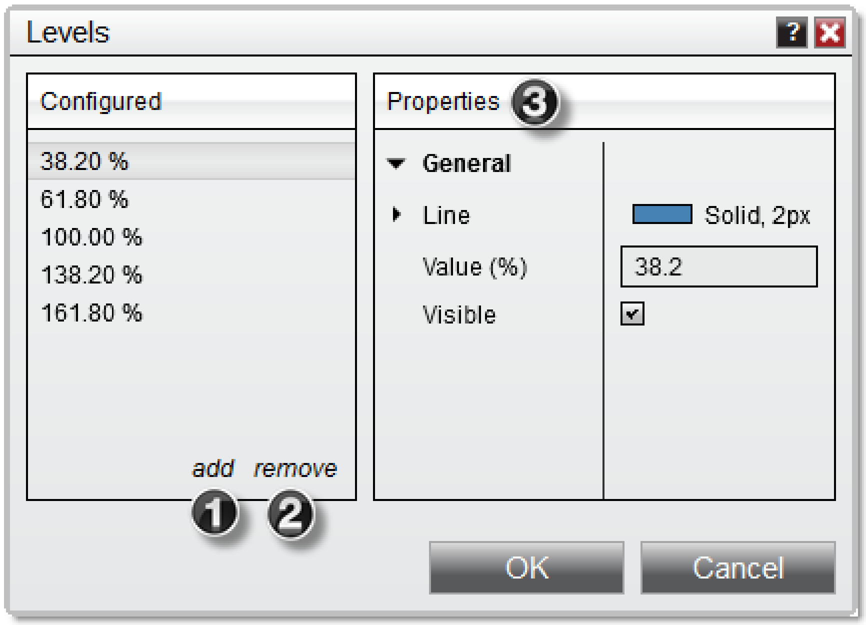
Note: The value property of a level is always expressed in percentage terms, and the placement of the line corresponding to that level will be based upon the anchor points you set for that particular **Drawing Object**.

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

11.6.11 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.

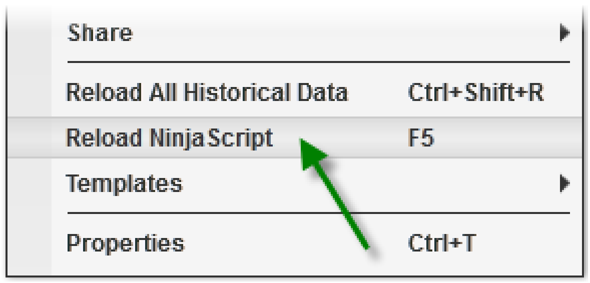
Working with automated strategies in a chart

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

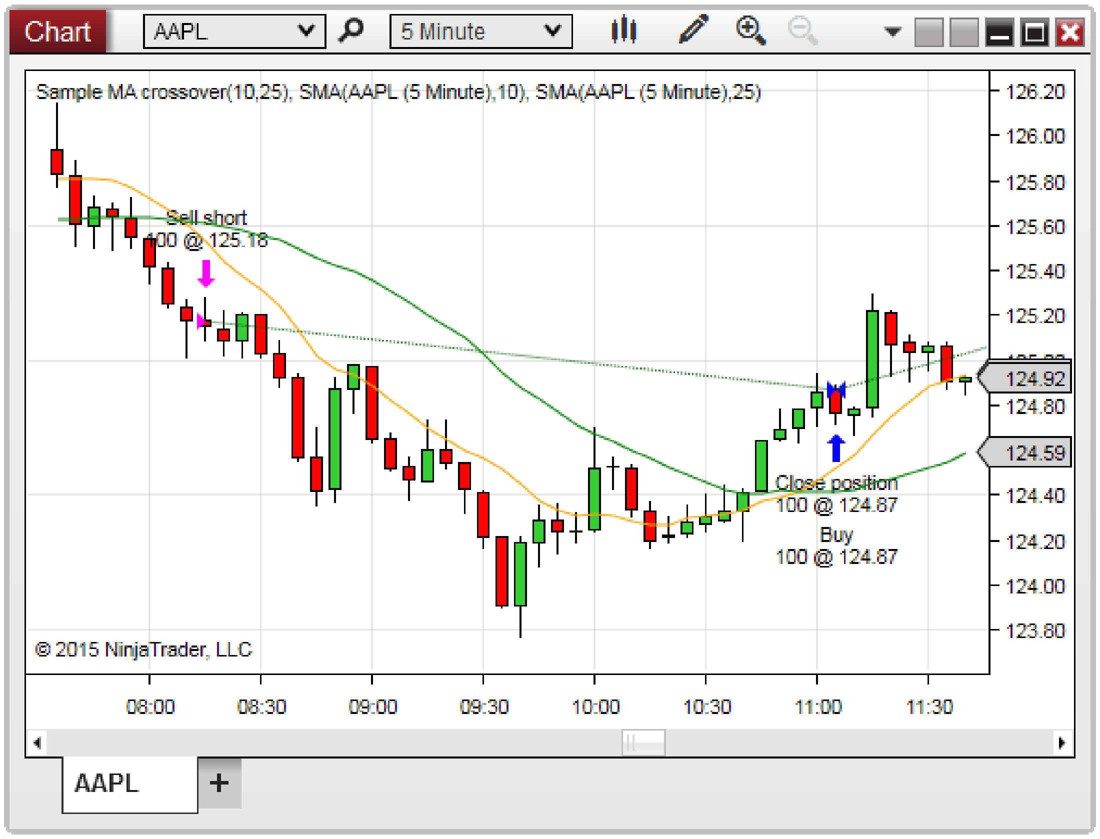
An automated strategy can be reloaded by right mouse clicking in the chart and selecting the menu item **Reload NinjaScript**. Reloading an automated strategy will remove the existing instance of the strategy and add a new one in the chart.



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.

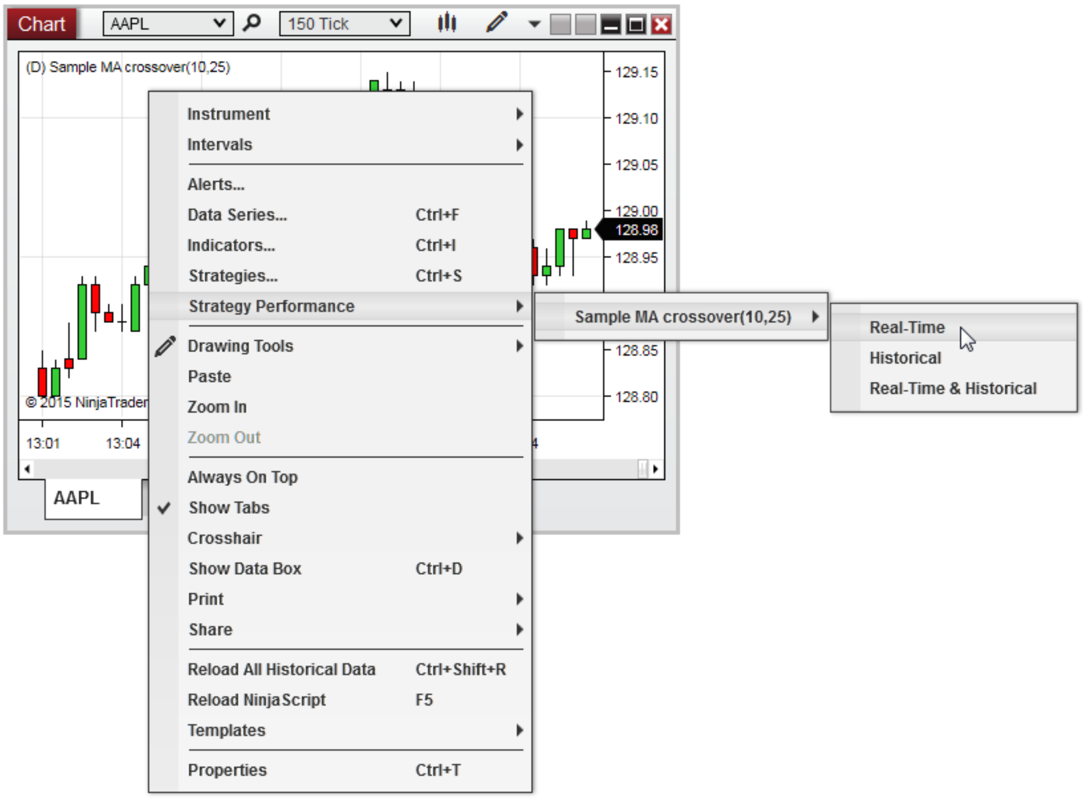


Note: You can view historical trades when a strategy is applied to a chart because the IncludeTradeHistoryInBacktest property is set to **true** by default when a strategy is applied to a chart. You can set this property to false in your code for leaner memory management, at the cost of not being able 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:

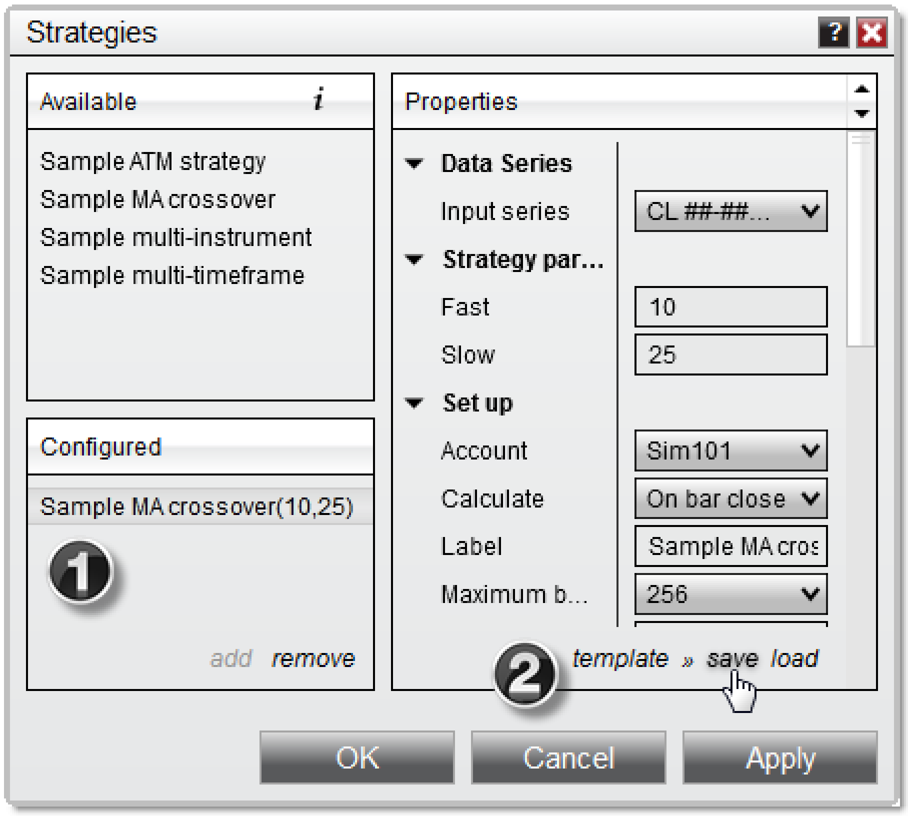
- Account defaults to the **Sim101** account
- Enabled defaults to False

Saving Custom Strategy Settings

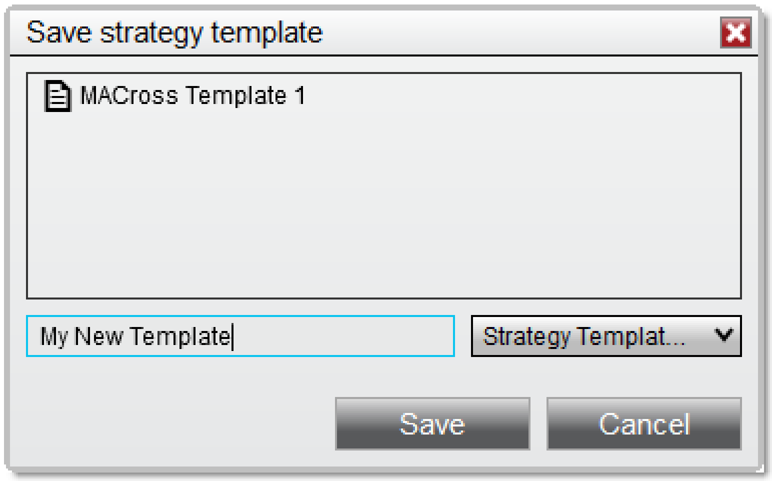
To save custom NinjaScript strategy parameters as default:

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.



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.

11.6.12 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:

- 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.

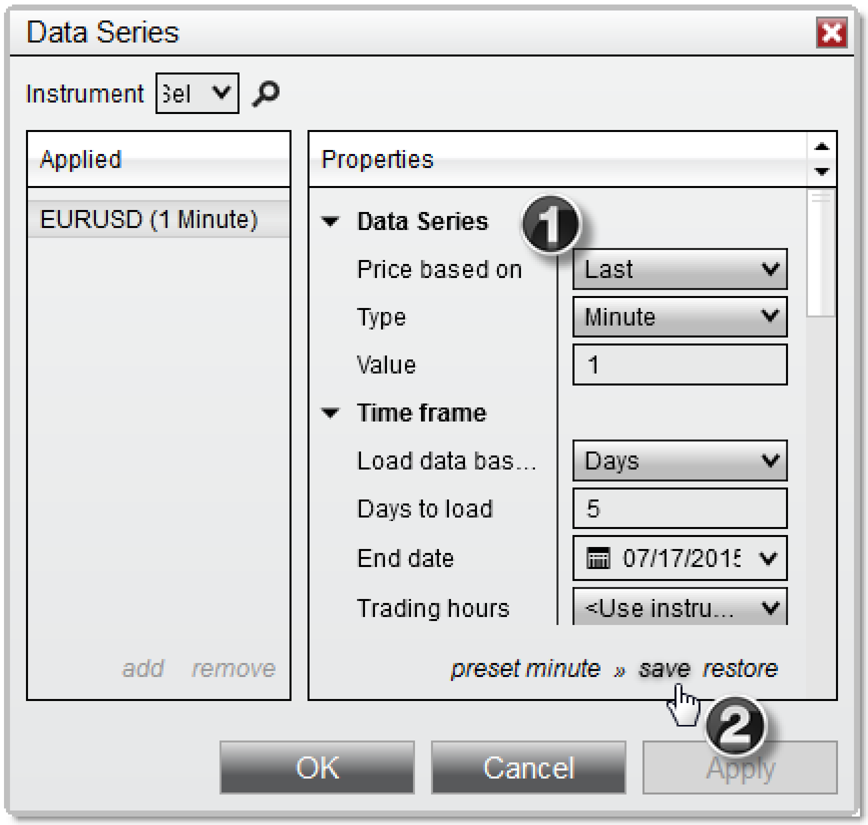
Note: **Period Type** default settings overrule **Chart Style** default settings. When you change the **Period Type**, the parameters saved in that **Period Type**'s default settings will change any parameters which may have already been loaded. However, when a **Chart Style** is changed, any saved defaults 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.

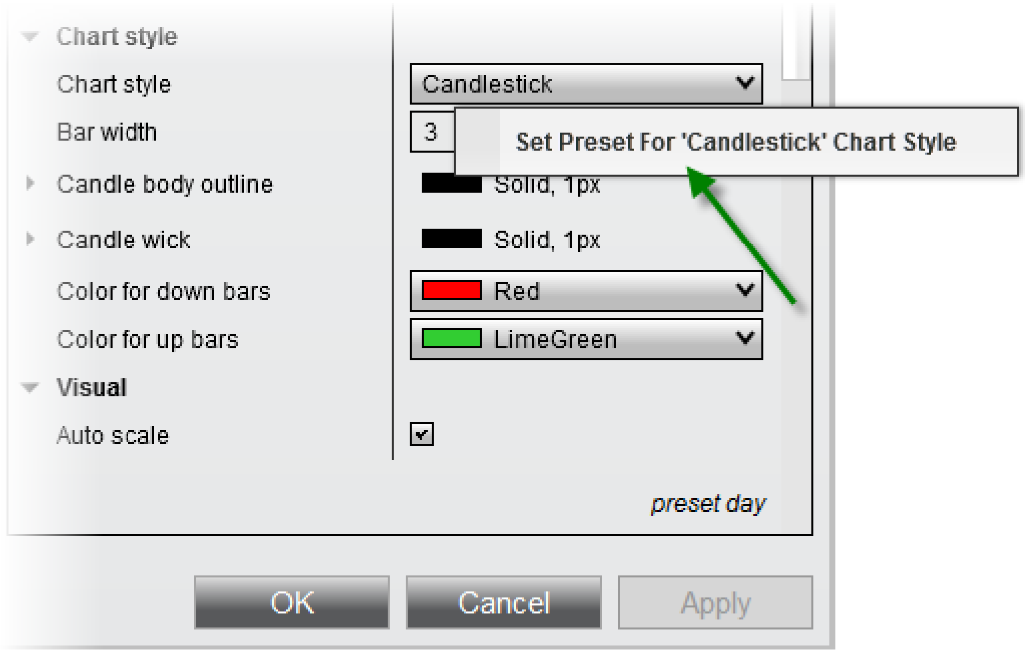


Chart style

Chart style	Candlestick
Bar width	3
Candle body outline	Solid, 1px
Candle wick	Solid, 1px
Color for down bars	Red
Color for up bars	LimeGreen

Visual

Auto scale	✓

Set Preset For 'Candlestick' Chart Style

preset day

Understanding indicator default settings

Each individual indicator's parameters can be saved as default. 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

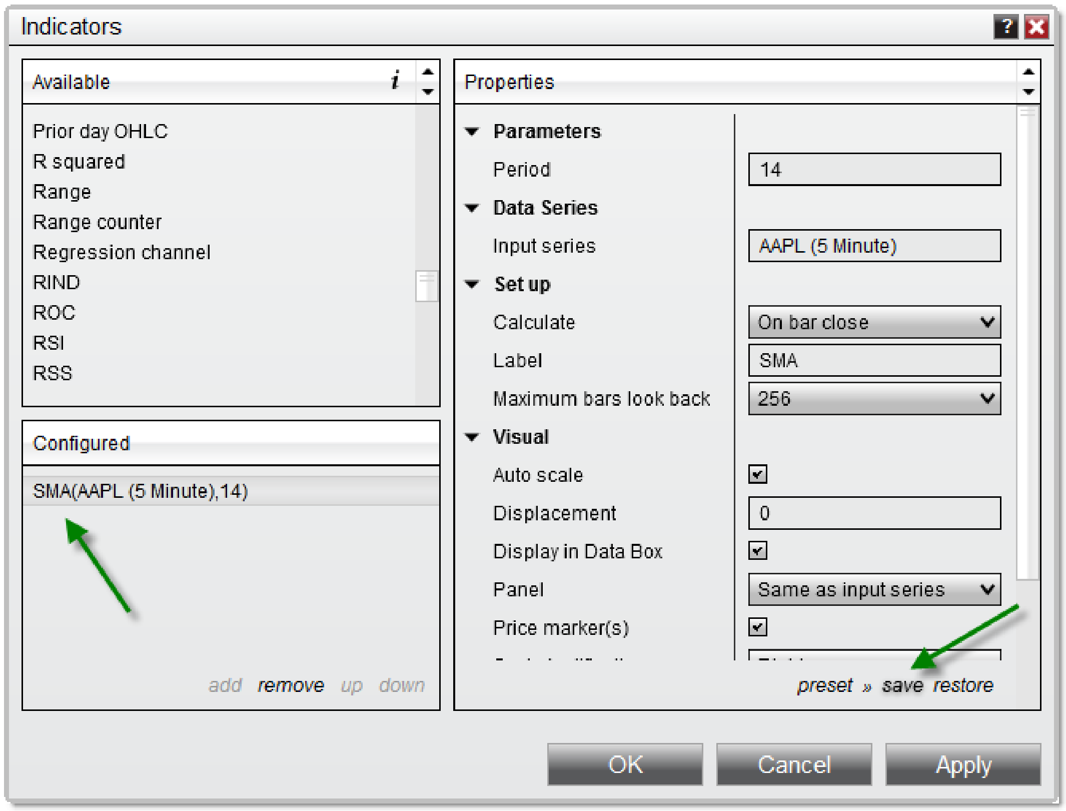
Saving Custom Indicator Settings

To save Indicator parameters as default:

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 apply that indicator to a chart.

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.

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.



SMA(AAPL (5 Minute),14)

Parameters
Period: 14
Data Series
Input series: AAPL (5 Minute)
Set up
Calculate: On bar close
Label: SMA
Maximum bars look back: 256
Visual
Auto scale: ✓
Displacement: 0
Display in Data Box: ✓
Panel: Same as input series
Price marker(s): ✓

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.

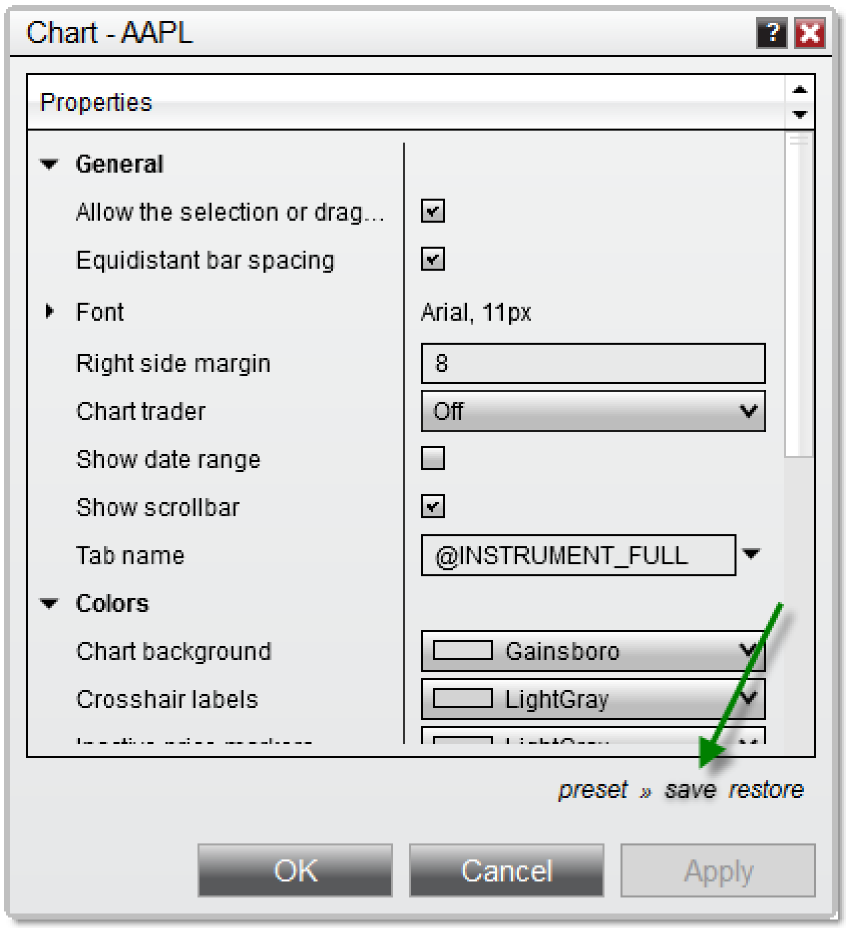


Chart - AAPL

Properties

General

Allow the selection or drag... ✓

Equidistant bar spacing ✓

Font Arial, 11px

Right side margin 8

Chart trader Off

Show date range ☐

Show scrollbar ✓

Tab name @INSTRUMENT_FULL

Colors

Chart background Gainsboro

Crosshair labels LightGray

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

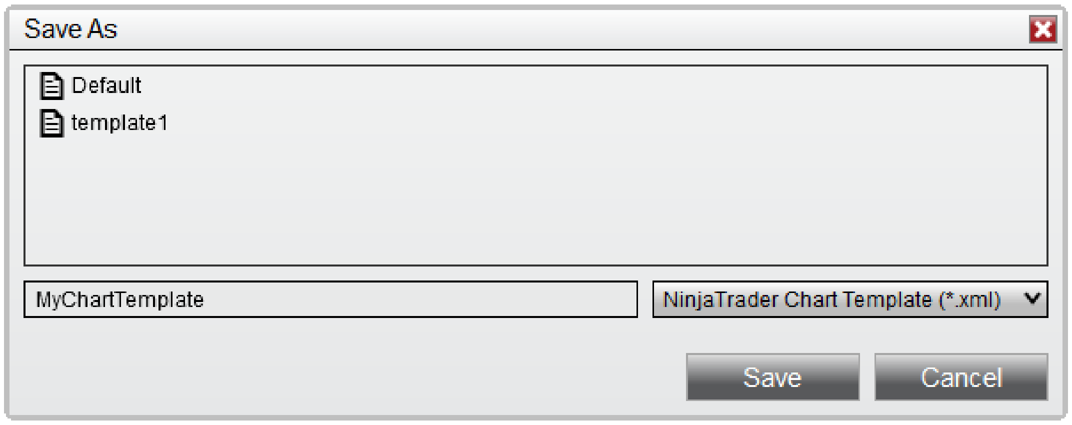
Chart properties, chart panel properties, and indicator settings can be saved as a **Chart Template.** A **Chart Template** can be applied to a new chart or an open chart to load customized chart settings, provided the template and chart share the same number of Data Series objects.

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:

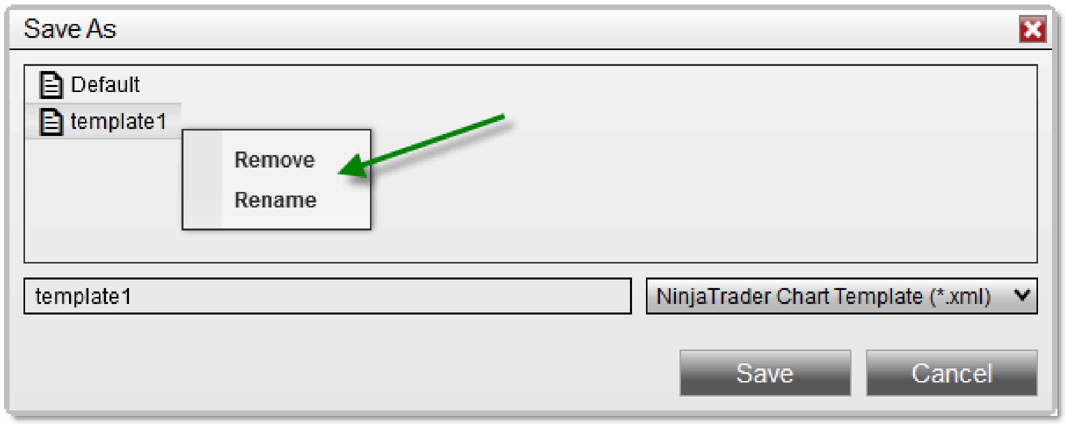
1. Right mouse click within a chart and select the menu item **Templates** followed by either the **Save As** or **Load** menu items
2. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the template for removal from the list of templates, then select 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**.



11.6.13 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.

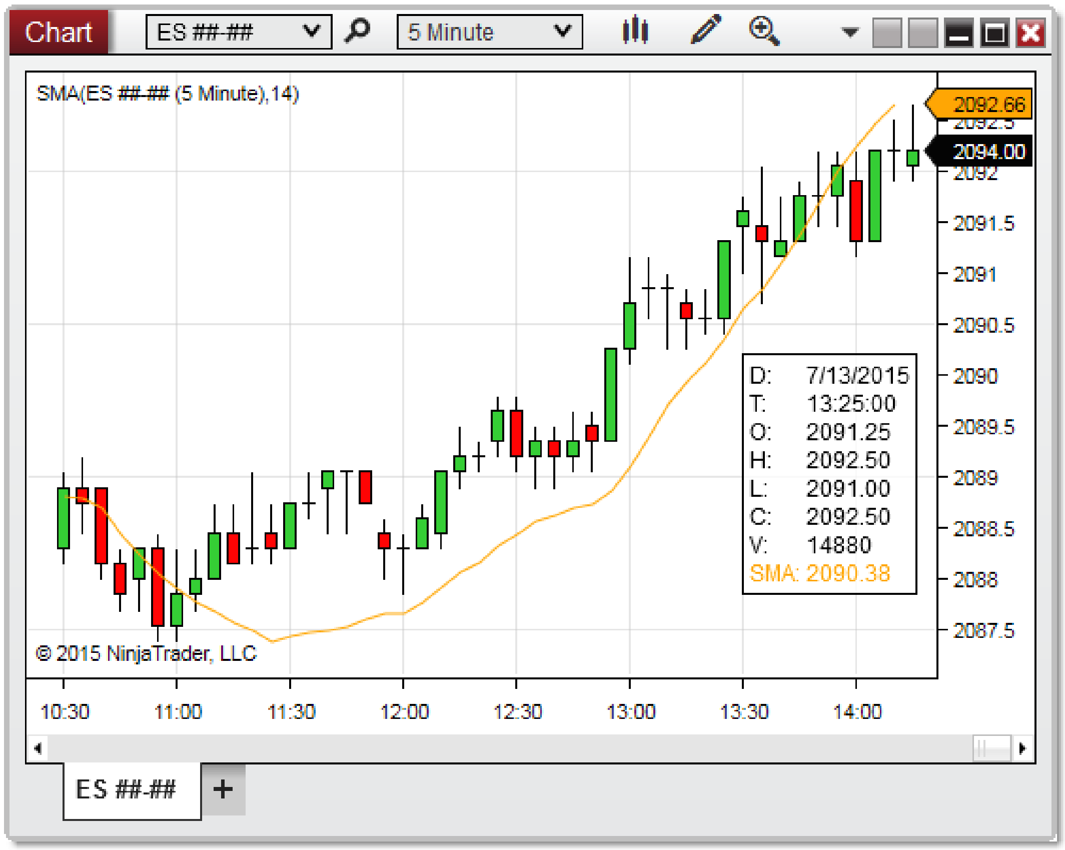
▽ Understanding the Mini Data Box

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.



D: 7/13/2015
T: 13:25:00
O: 2091.25
H: 2092.50
L: 2091.00
C: 2092.50
V: 14880
SMA: 2090.38

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.

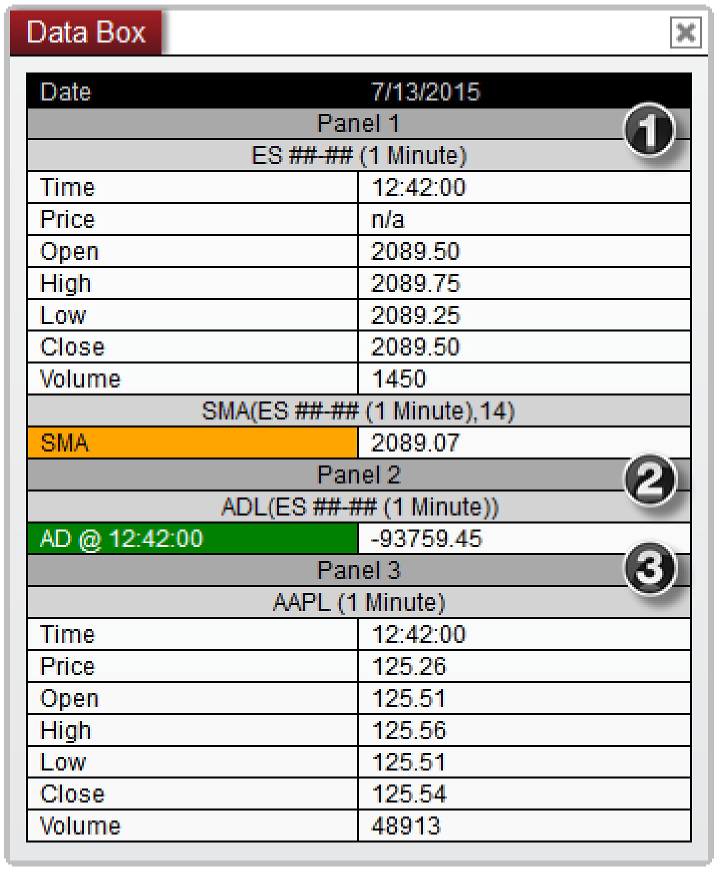
Understanding the Data Box

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.



Date	7/13/2015
Panel 1	
ES ##-## (1 Minute)	
Time	12:42:00
Price	n/a
Open	2089.50
High	2089.75
Low	2089.25
Close	2089.50
Volume	1450
SMA(ES ##-## (1 Minute),14)	
SMA	2089.07
Panel 2	
ADL(ES ##-## (1 Minute))	
AD @ 12:42:00	-93759.45
Panel 3	
AAPL (1 Minute)	
Time	12:42:00
Price	125.26
Open	125.51
High	125.56
Low	125.51
Close	125.54
Volume	48913

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.

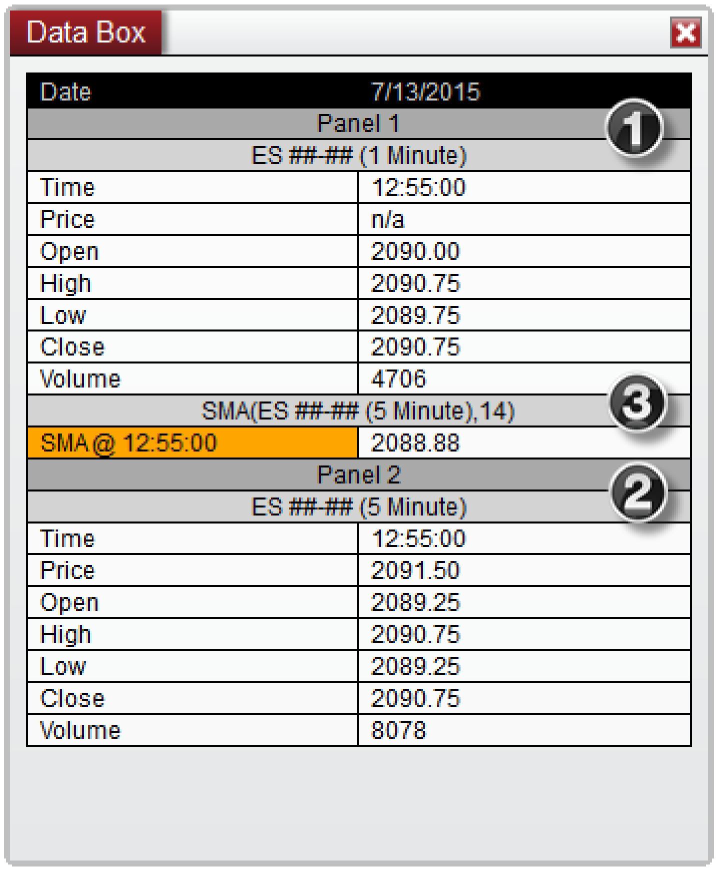
3. Panel 3 includes a second data series, AAPL, with no indicators.

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.



Date	7/13/2015
Panel 1	
ES ##-## (1 Minute)	
Time	12:55:00
Price	n/a
Open	2090.00
High	2090.75
Low	2089.75
Close	2090.75
Volume	4706
SMA(ES ##-## (5 Minute),14)	
SMA @ 12:55:00	2088.88
Panel 2	
ES ##-## (5 Minute)	
Time	12:55:00
Price	2091.50
Open	2089.25
High	2090.75
Low	2089.25
Close	2090.75
Volume	8078

As an example, notice the following about the image above:

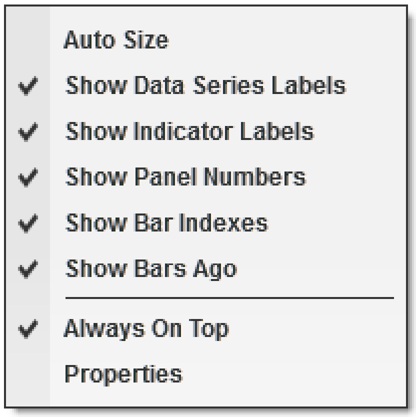
1. Panel 1 contains a 1 minute Data Series.

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.



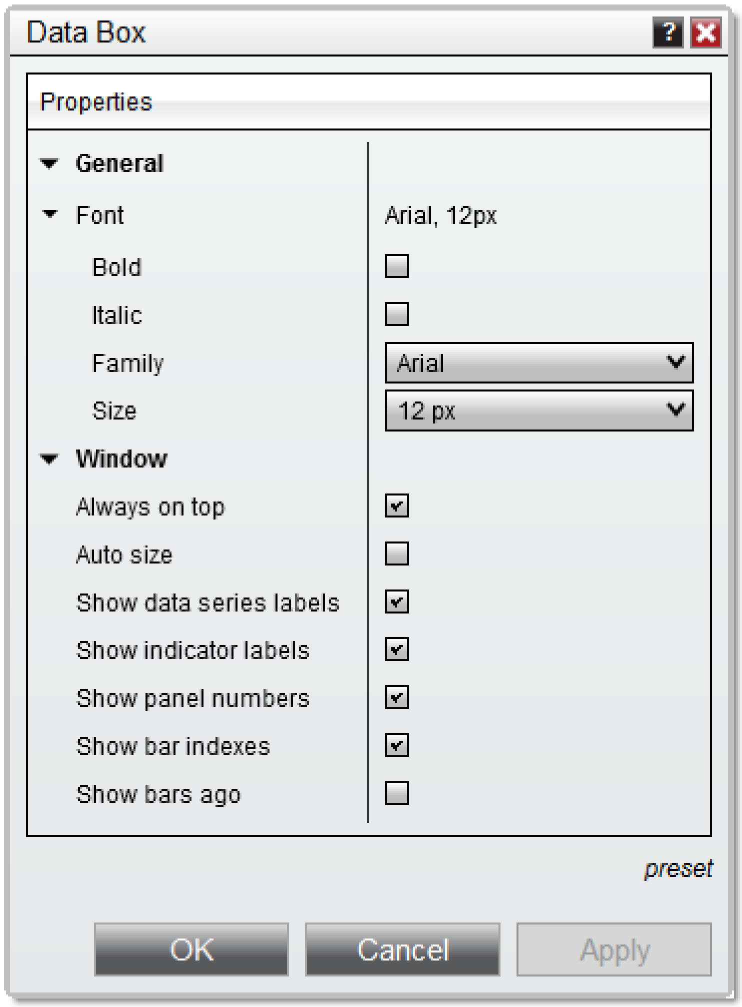
The following options are available:

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:


Data Box

Properties

General

Font: Arial, 12px

Bold ☐

Italic ☐

Family: Arial

Size: 12 px

Window

Always on top ☑

Auto size ☐

Show data series labels ☑

Show indicator labels ☑

Show panel numbers ☑

Show bar indexes ☑

Show bars ago ☐

preset

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.

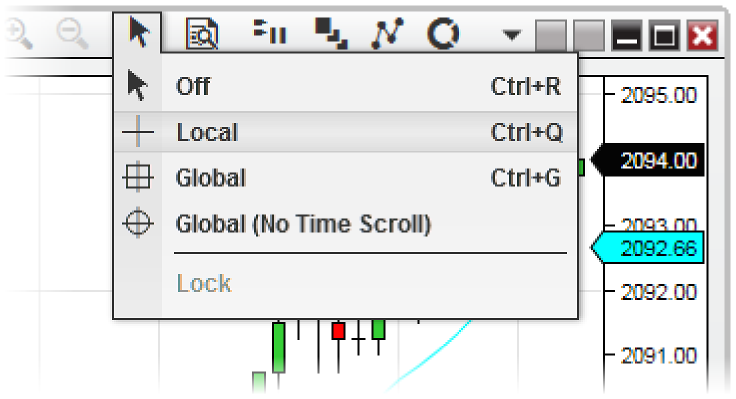
11.6.14 Cross Hair

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:



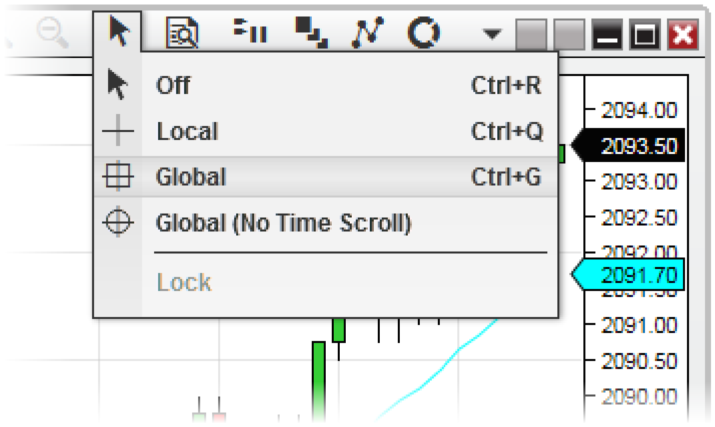
- Left mouse click on the **Cursor** icon in the chart toolbar and select the **Local** menu item.
- Right mouse click within the chart and select the **Crosshair** menu, then select the **Local** menu item.
- Use the default CTRL +Q Hot Key

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.

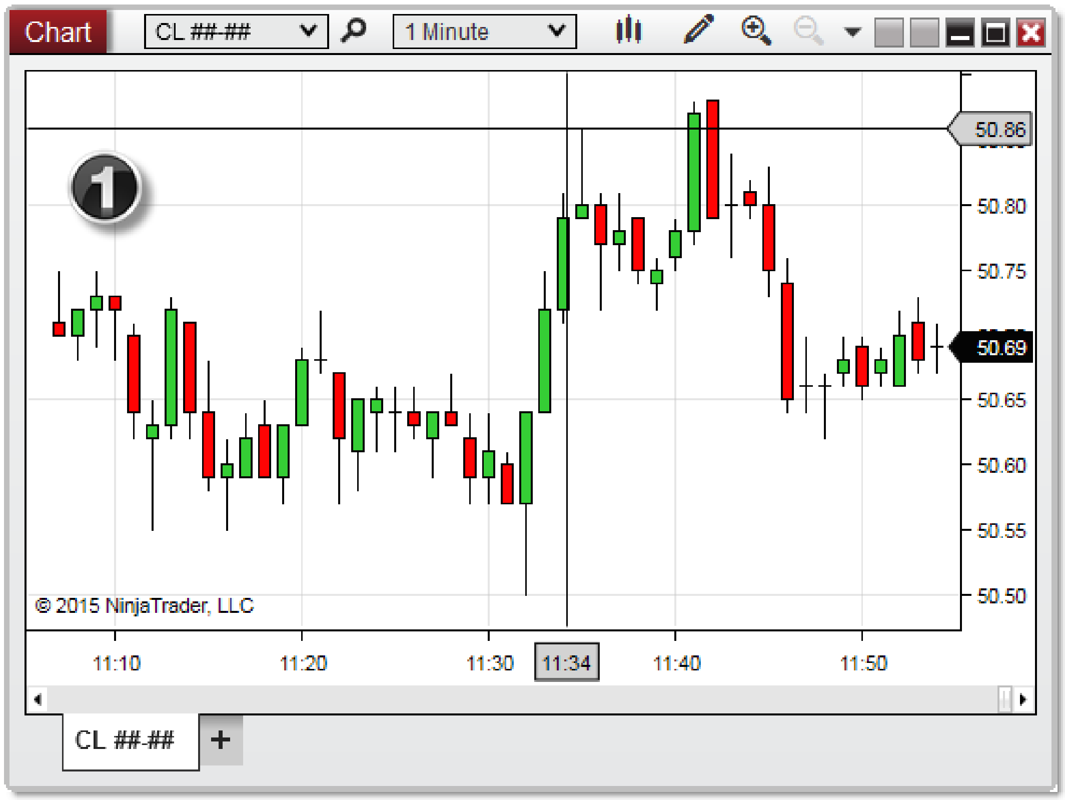
How to use the Global Cross Hair

Using the Global Cross Hair

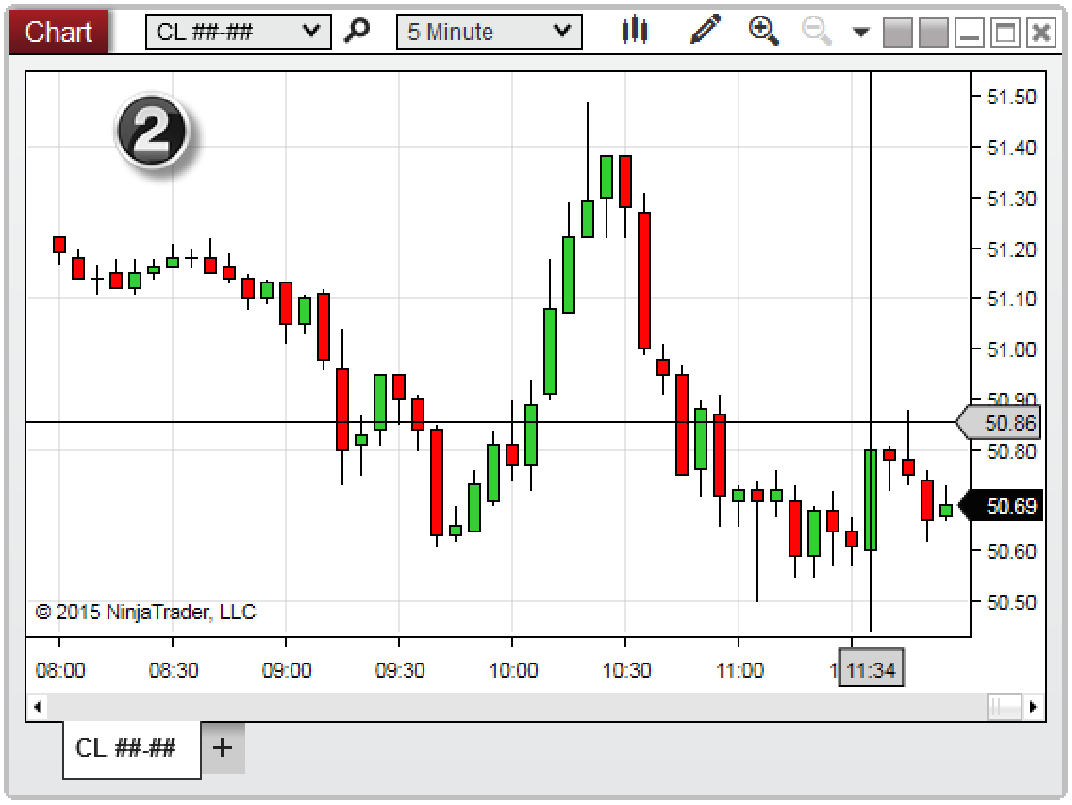
The **Global Cross Hair** must be enabled on more than one chart in order to take full advantage of its functionality.

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

11.6.15 Trading from a Chart

Please see the Chart Trader section under Order Entry section for more information on trading in the chart.

11.6.16 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:

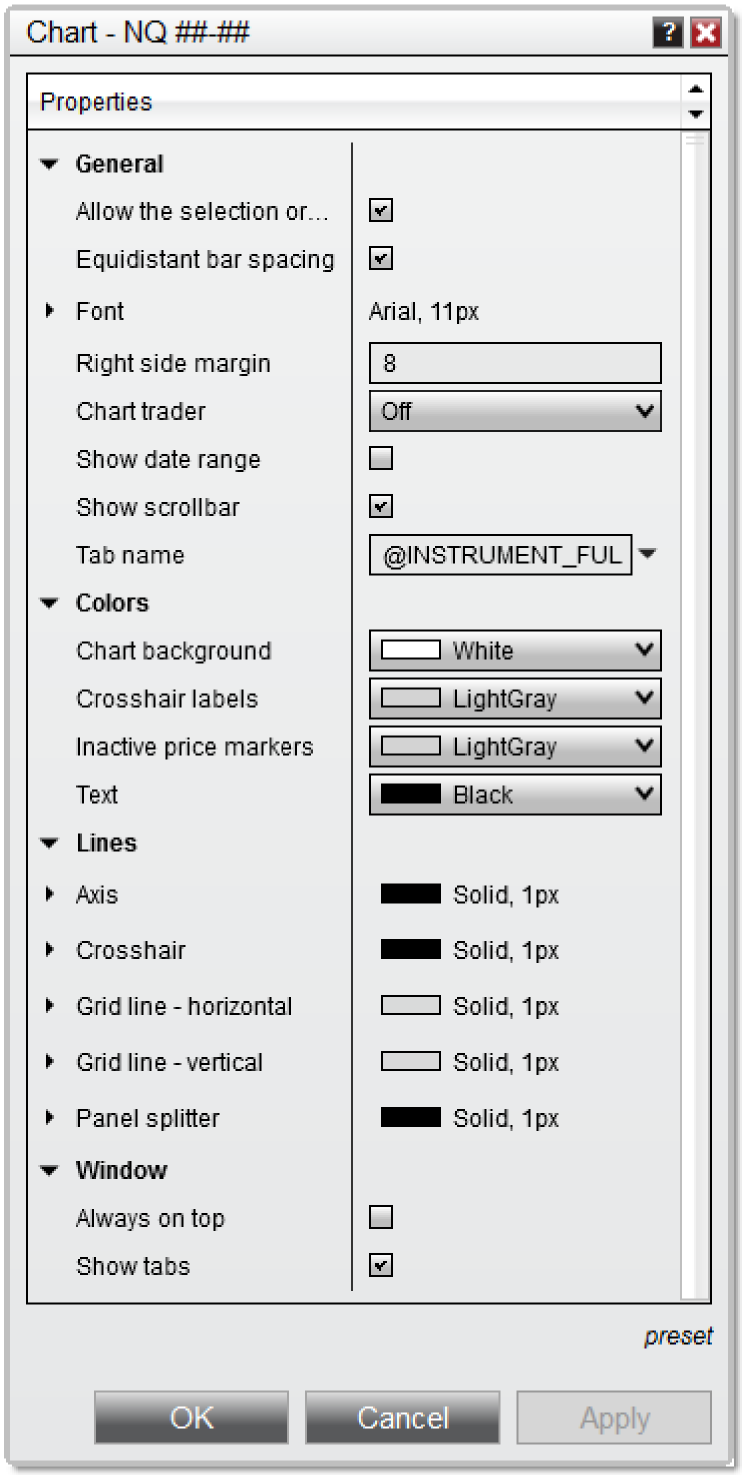


Chart - NQ ##-##

Properties

General

Allow the selection or...	☑
Equidistant bar spacing	☑
Font	Arial, 11px
Right side margin	8
Chart trader	Off
Show date range	☐
Show scrollbar	☑
Tab name	@INSTRUMENT_FUL

Colors

Chart background	⬜ White
Crosshair labels	⬜ LightGray
Inactive price markers	⬜ LightGray
Text	⬛ Black

Lines

Axis	⬛ Solid, 1px
Crosshair	⬛ Solid, 1px
Grid line - horizontal	⬜ Solid, 1px
Grid line - vertical	⬜ Solid, 1px
Panel splitter	⬛ Solid, 1px

Window

Always on top	☐
Show tabs	☑

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 Price Data*" 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 disable showing the date range label in to the top left of chart. The date range reported is 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.
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.

11.6.17 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.

11.6.18 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.

11.6.19 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.

11.6.20 Break at EOD

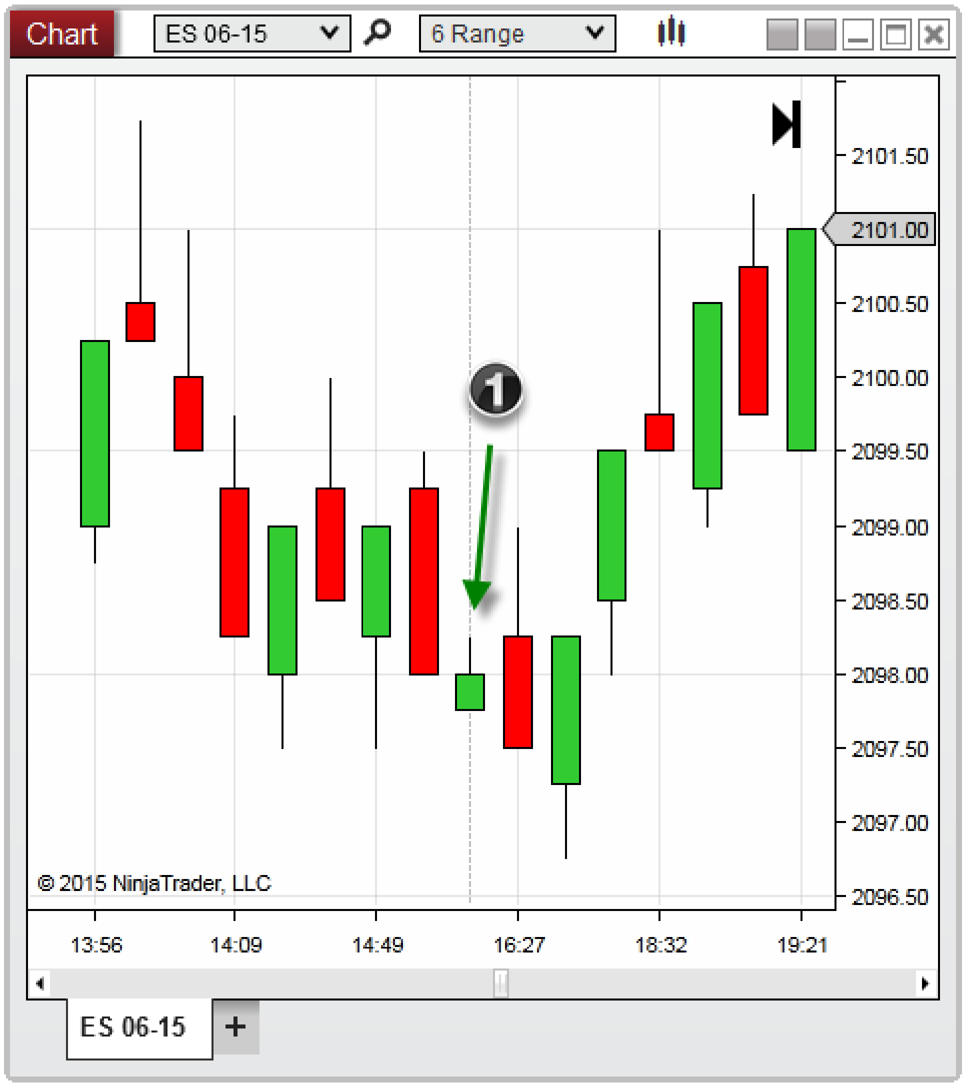
Break at EOD (End Of Day)

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.

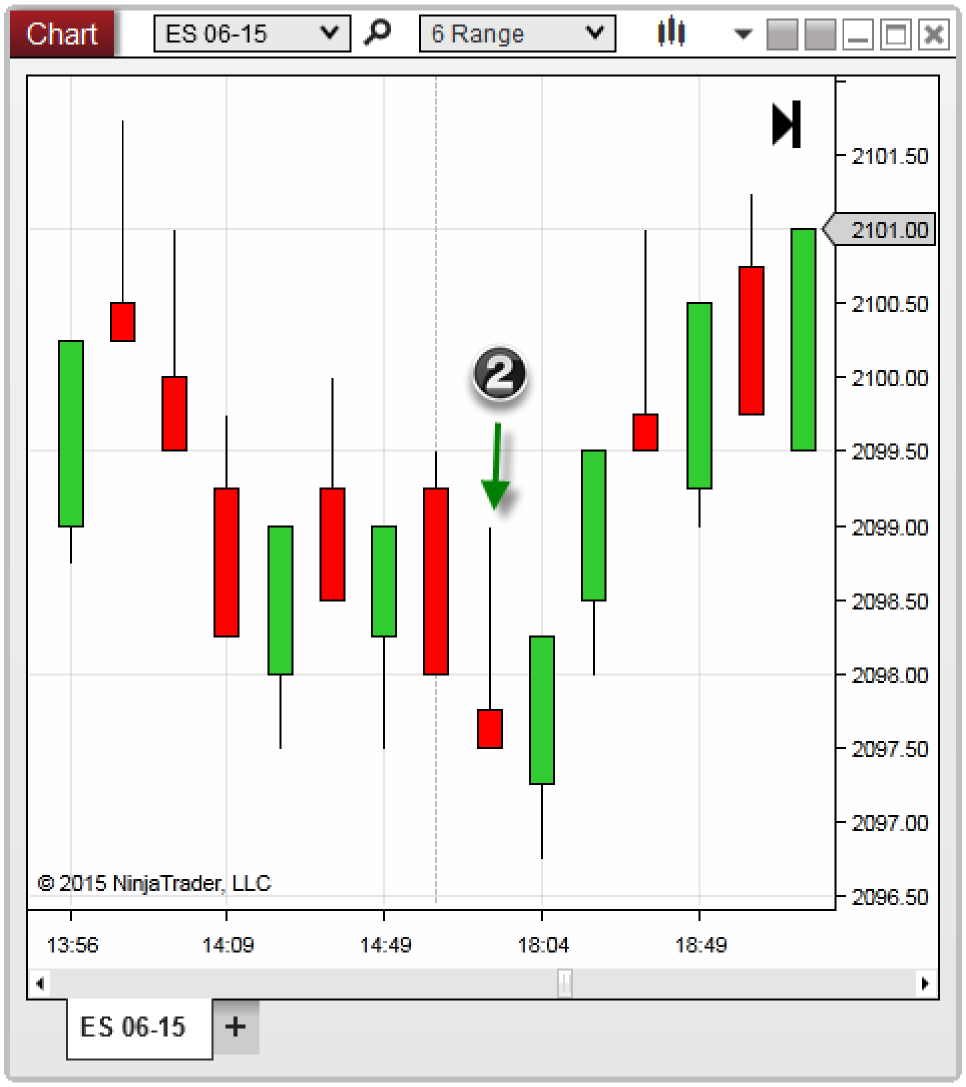
With **Break at EOD** enabled, your bars are ensured to have a known starting point of which to give you a consistent, repeatable chart. This is especially important for bar types that are not based on price such as Volume and Tick based charts. However, for other bar types such as Range, Point and Figure, 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

11.6.21 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**.

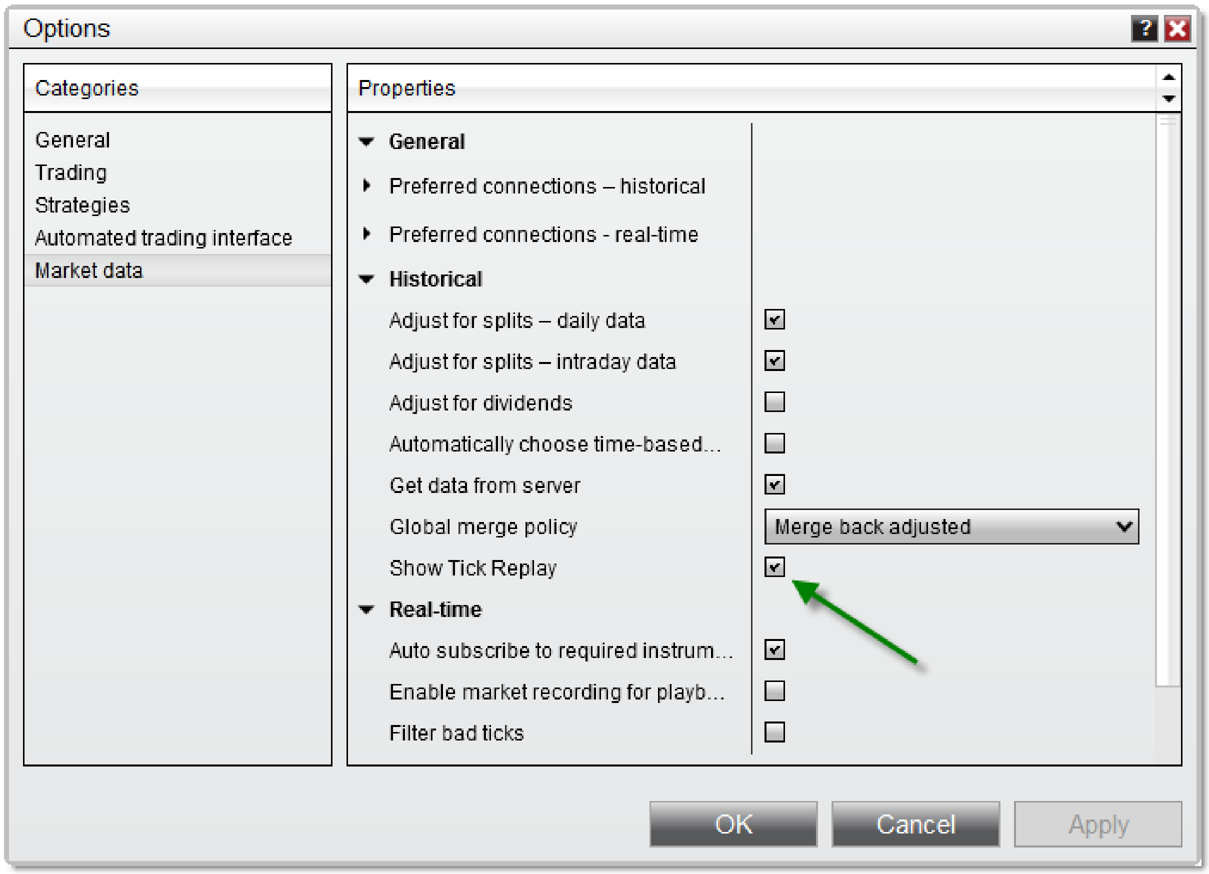
Note: Tick Replay is not intended to function in NinjaScript strategy backtests, and will not provide the same results as running a strategy on live data with Tick Replay enabled. For greater order-fill resolution in strategy backtests, you can use the High Fill Resolution in the Strategy Analyzer.

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**"


Options

Categories

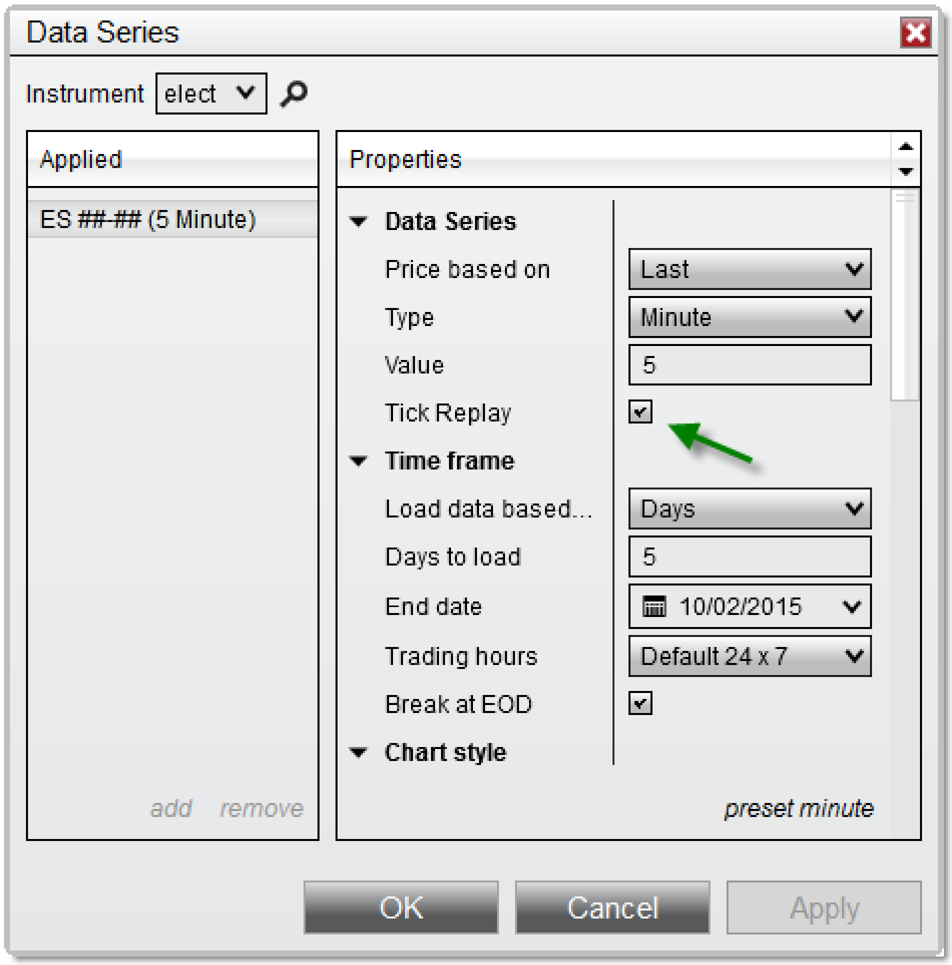
- General
- Trading
- Strategies
- Automated trading interface
- Market data

Properties

General	
Preferred connections – historical	
Preferred connections - real-time	
Historical	
Adjust for splits – daily data	✓
Adjust for splits – intraday data	✓
Adjust for dividends	☐
Automatically choose time-based...	☐
Get data from server	✓
Global merge policy	Merge back adjusted
Show Tick Replay	✓
Real-time	
Auto subscribe to required instrum...	✓
Enable market recording for playb...	☐
Filter bad ticks	☐

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.


Data Series

Instrument elect

Applied

ES ##-## (5 Minute)

Properties

Data Series

Price based on	Last
Type	Minute
Value	5
Tick Replay	✓

Time frame

Load data based...	Days
Days to load	5
End date	10/02/2015
Trading hours	Default 24 x 7
Break at EOD	✓

Chart style

11.6.21.1 Tick Replay Indicators

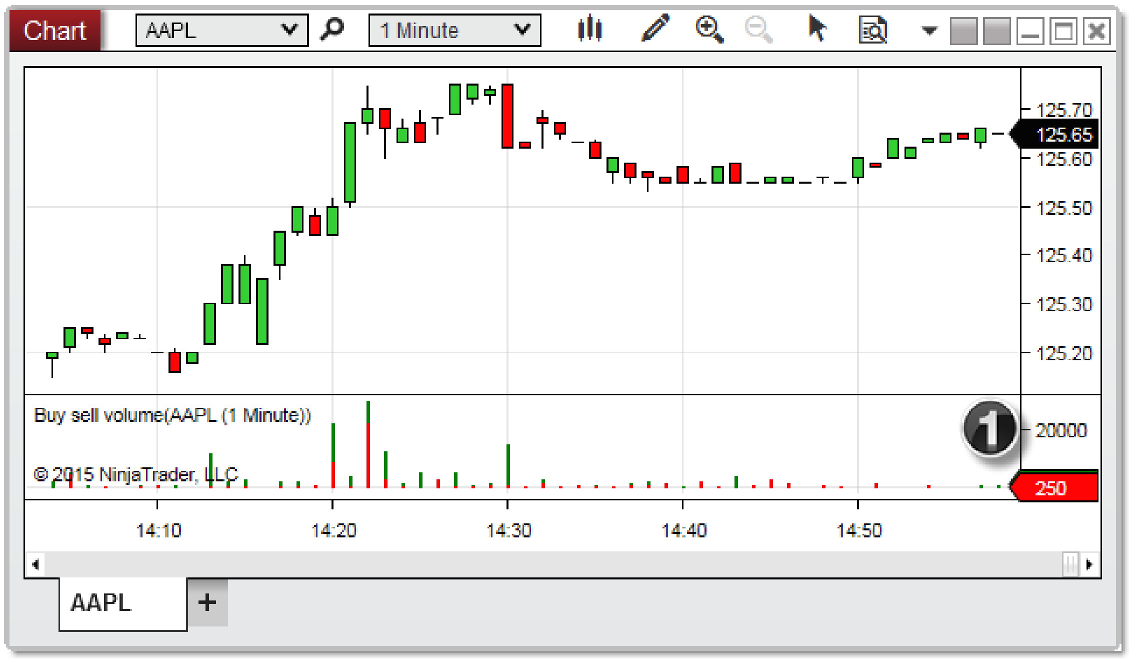
NinjaTrader includes several real-time volume based indicators that are designed to function in real time only. However, Tick Replay allows these indicators to be used on historical data by simulating real-time price movements within historical bars.

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.

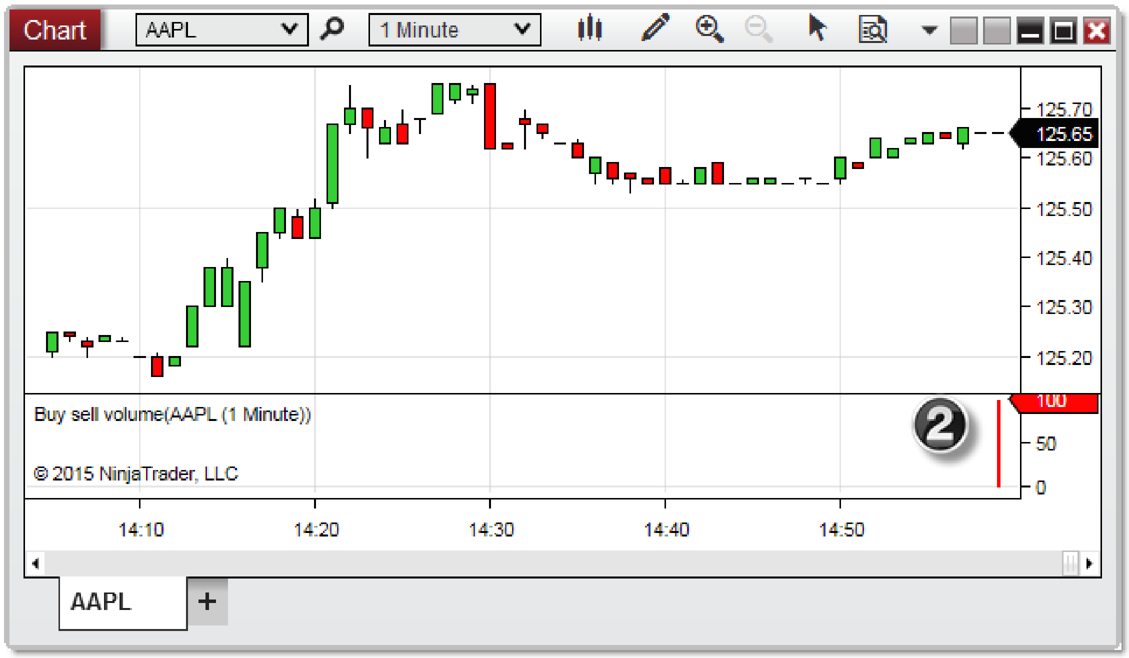
Buy Sell Volume

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.

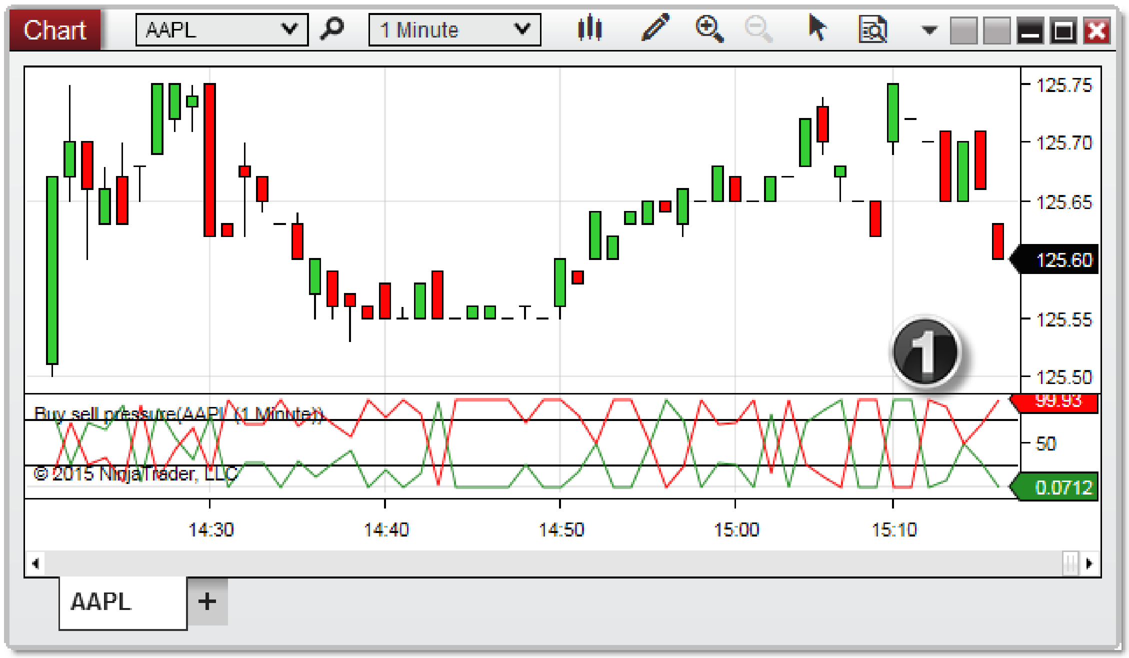


2) In the image above, the **BuySellVolume** indicator has just been applied with **Tick Replay** disabled, limiting it to only displaying what has been calculated in real time.

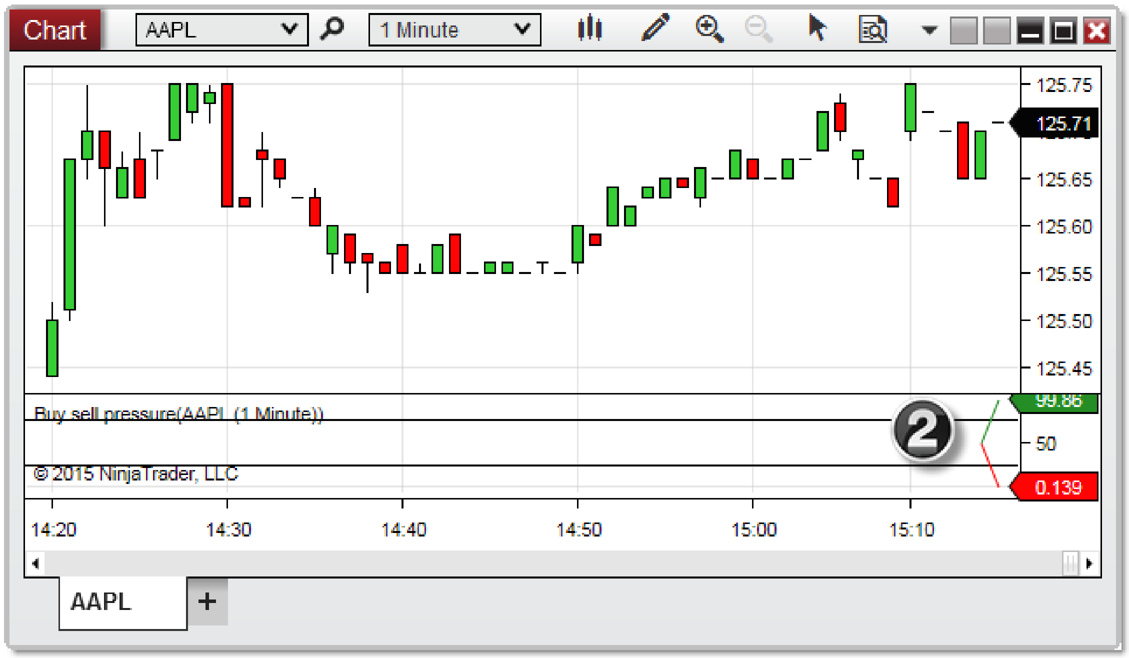
Buy Sell Pressure

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.

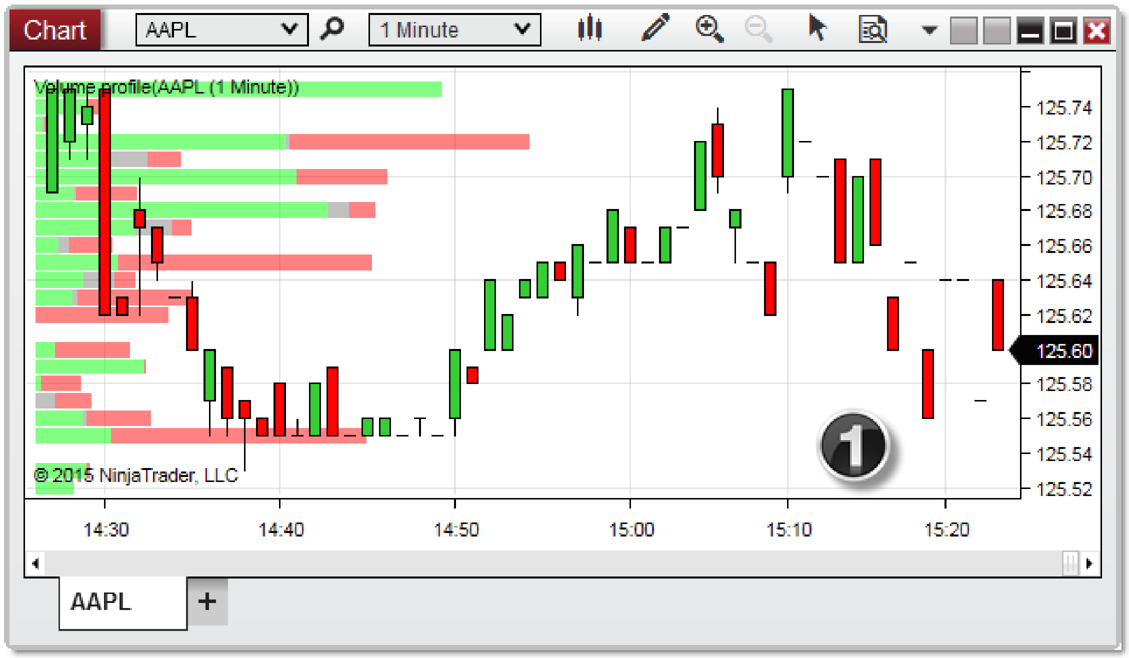


2) In the image above, the **BuySellPressure** indicator has just been applied with **Tick Replay** disabled, limiting it to only displaying what has been calculated in real time.

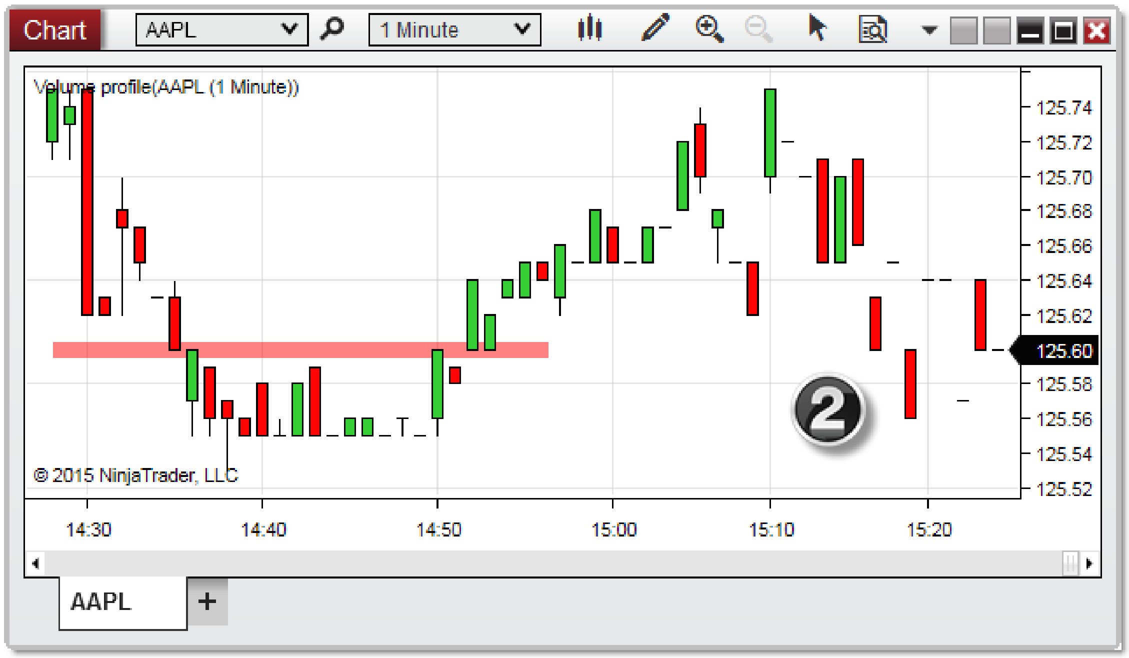
Volume Profile

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.

11.7 Commissions

Commissions Overview

The **Commissions** window allows you to define global and per instrument commission rates to be used on Simulation or Live account. Commissions applied are visible as unrealized PnL while you are in a trade and inside the **Trade Performance** window. Setting up **commissions** in a 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 Realized and Unrealized PnL values provided by brokers.

› Working with Commission Templates
› Applying Commission Templates

11.7.1 Working With Commission Templates

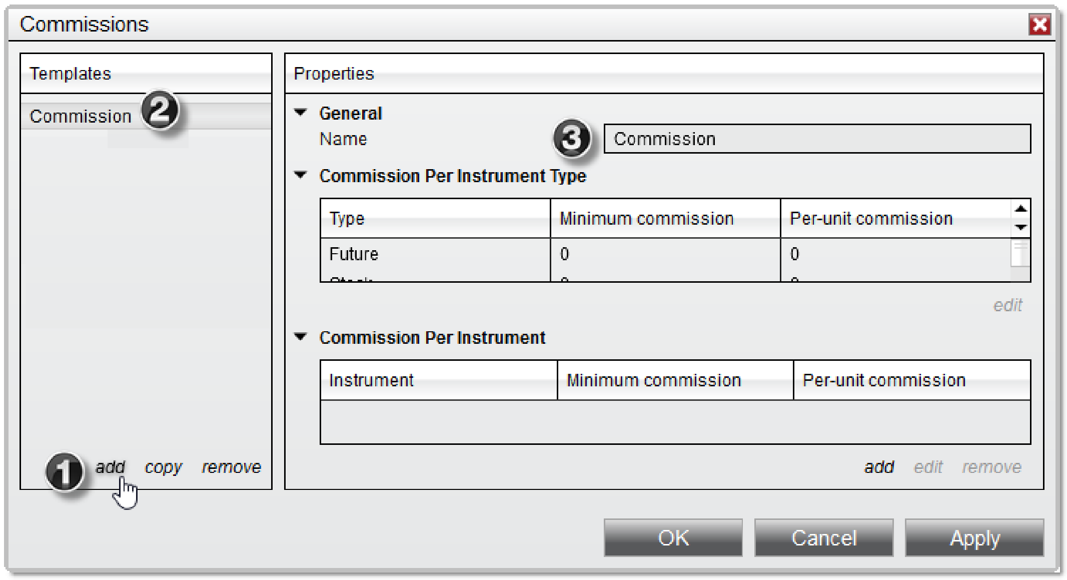
The **Commissions** window allows you to create and manage **Commission Templates** to be applied to different trading accounts configured in NinjaTrader. These templates can be used to set minimum and per-unit commissions for all instruments of a certain type, or to set specific commissions for individual 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.



Name: Commission

Type	Minimum commission	Per-unit commission
Future	0	0

Instrument	Minimum commission	Per-unit commission

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.

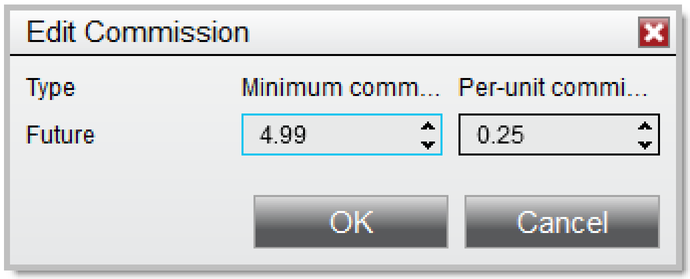
Removing Commission Templates

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**.


Edit Commission

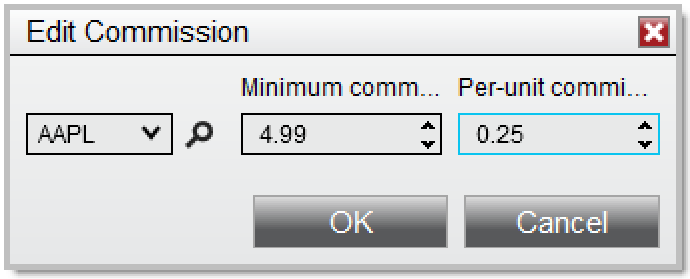
Type	Minimum comm...	Per-unit commi...
Future	4.99	0.25

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**.



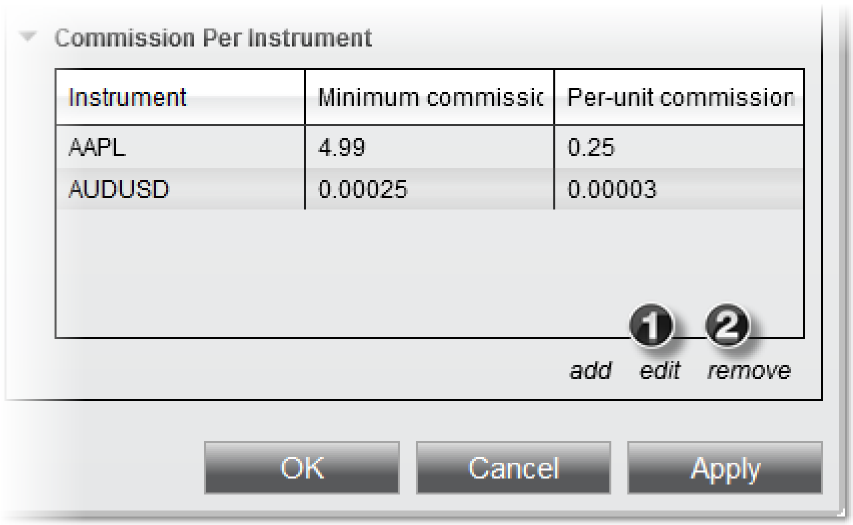
Notes:
1. Commissions entered for specific instruments will override settings for that instrument's type. For example, setting instrument-specific commissions for the E-Mini S&P 500 futures contract will override any commissions set for all Futures instruments.
2. "Minimum Commission" is applied per 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.



Instrument	Minimum commissic	Per-unit commission
AAPL	4.99	0.25
AUDUSD	0.00025	0.00003

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.

11.7.2 Applying Commission Templates

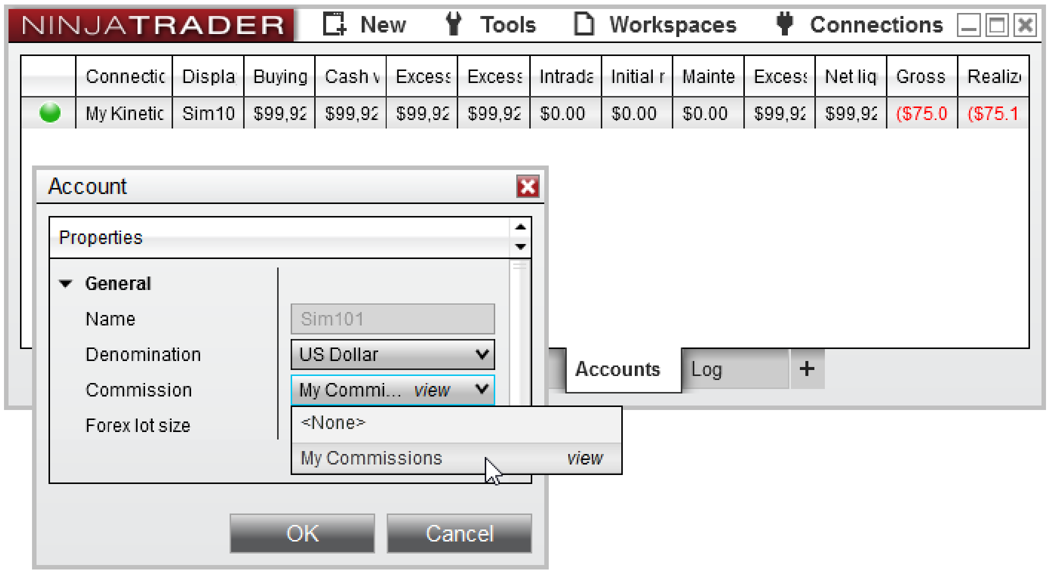
Once a Commission Template has been created, it must be applied to an account, whether it be a live brokerage account connected via a data provider, or a simulation account used for paper trading.

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.



	Connectic	Displa	Buying	Cash v	Excess	Excess	Intrada	Initial r	Mainte	Excess	Net liq	Gross	Realiz
●	My Kinetic	Sim10	$99,92	$99,92	$99,92	$99,92	$0.00	$0.00	$0.00	$99,92	$99,92	($75.0	($75.1

Account

Properties

General

Name	Sim101
Denomination	US Dollar
Commission	My Commi... view
Forex lot size	

<None>

My Commissions view

11.8 Control Center

Play Video

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.

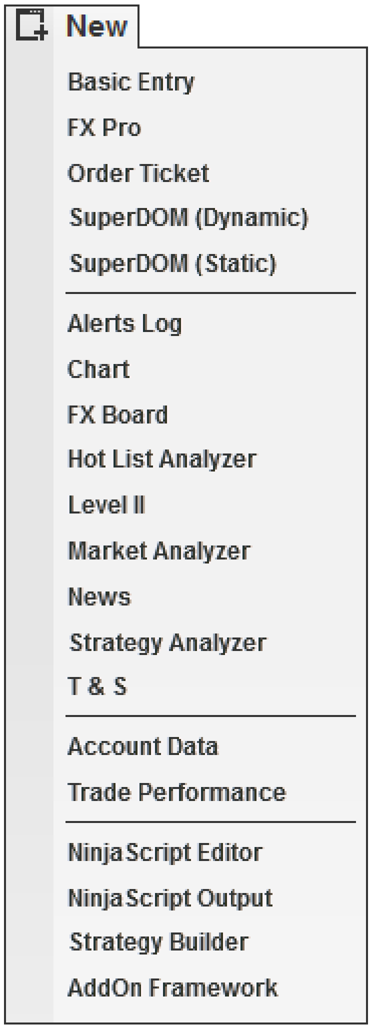
The NinjaTrader Control Center provides a centralized view of account, execution, order, historical log, and position information. It also provides access to all of the various NinjaTrader function windows and enables/disables global application features and commands.

Menu System

- New Menu
- Tools Menu
- Workspaces Menu
- Connections Menu
- Orders Tab
- Strategies Tab
- Executions Tab
- Executions Tab
- Positions Tab
- Accounts Tab
- Log Tab
- Connection Status

11.8.1 New Menu

The following menus and items are available via the **New** menu of the NinjaTrader Control Center.

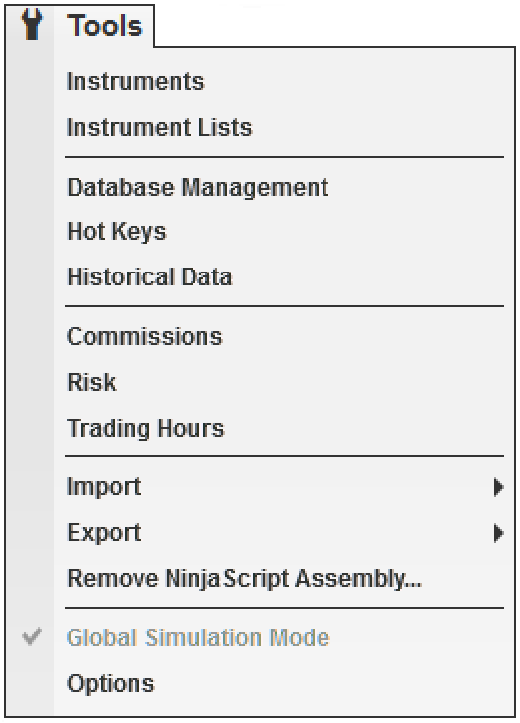


Basic Entry	Creates new Basic Entry window.
FX Pro	Creates a new FX Pro window
Order Ticket	Creates a new Order Ticket window
SuperDOM (Dynamic)	Creates a new SuperDOM (Dynamic) window
SuperDOM (Static)	Creates a new SuperDOM (Static) window

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
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
Strategy Builder	Creates a new Strategy Builder window
AddOn Framework	This is an example for a custom NinjaScript AddOn installed

11.8.2 Tools Menu

The following menus and items are available via the **Tools** menu of the NinjaTrader Control Center.

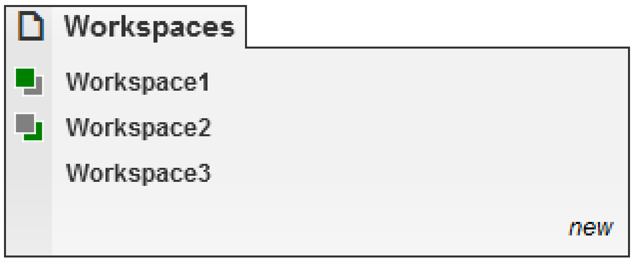


Instruments...	Opens the Instruments window
Instrument Lists...	Opens the Instrument Lists window
Database Management...	Opens the Database Management window
Hot Keys..	Opens the Hot Keys window
Historical Data..	Opens the Historical Data window
Commissions...	Opens the Commissions window
Risk...	Opens the Risk window

Trading Hours...	Opens the Trading Hours window
Vendor Licensing...	Opens the Vendor Licensing window
Import...	Opens the Import Sub Menu
Export...	Opens the Export Sub Menu
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)
Options...	Opens the Options window

11.8.3 Workspaces Menu

The following menus and items are available via the **Workspaces** menu of the NinjaTrader Control Center.

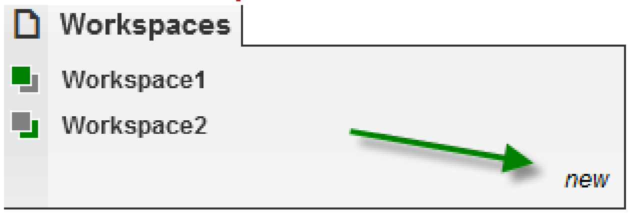


- A workspace named "Untitled1" will load automatically
- You can have multiple workspaces open simultaneously
- Open workspaces are indicated by the double square icon, if there is no icon then the workspace is closed
- The currently active workspace has a filled green square front most. You can only 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

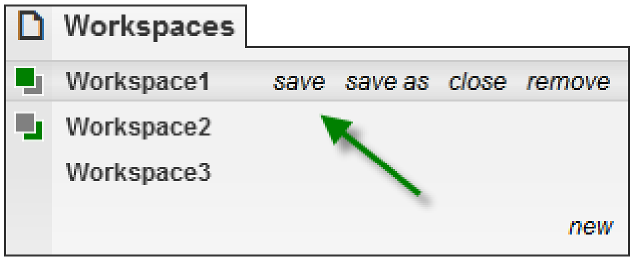
Create a Workspace



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.

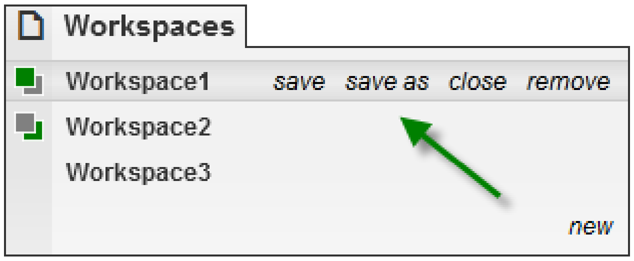
Saving a workspace

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

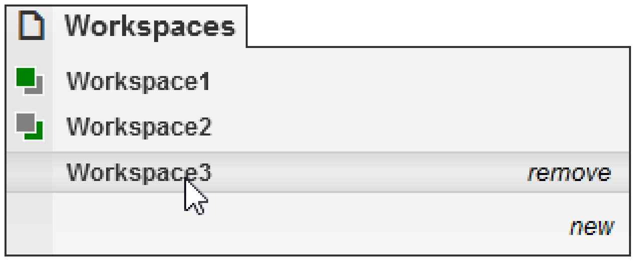


1. From the NinjaTrader Control Center select the menu **Workspaces**

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

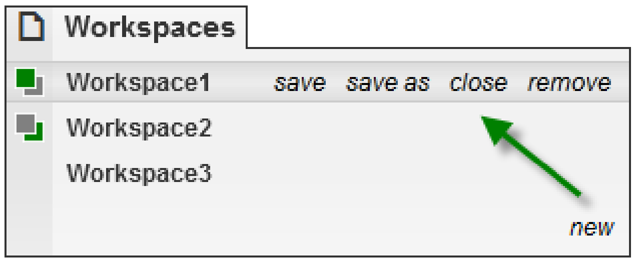
Open a Workspace



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

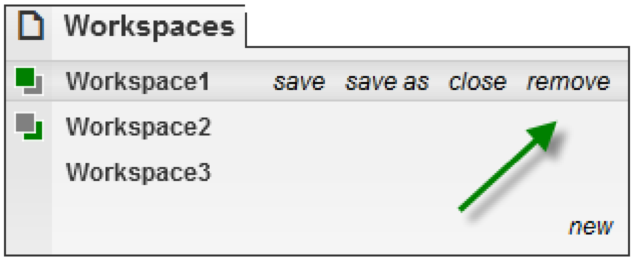
Close a Workspace



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

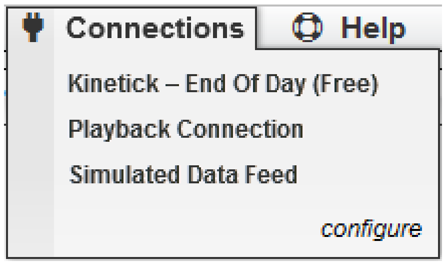
▿ How to quickly switch between Workspaces

Quickly Switching Between Workspaces

Pressing SHIFT+F3 keys together will cycle to the next open workspace.

11.8.4 Connections Menu

The following menus and items are available via the **Tools** menu of the NinjaTrader Control Center.



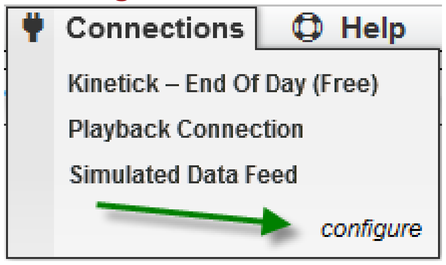
- You cannot remove the predefined constant connections: **Kinetick - End Of Day (Free)**, **Playback Connection**, **Simulated Data Feed**.
- You can connect to multiple connections simultaneously.
- The connection status is reported to the left of the connection name in the connections

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 sales@ninjatrader.com.

Creating a new connection

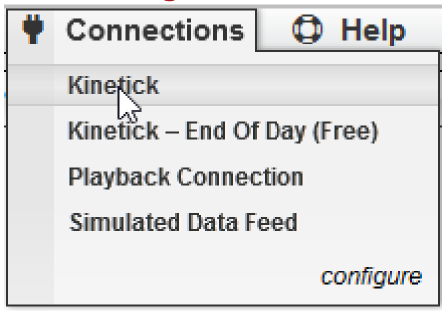
Creating a new connection



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

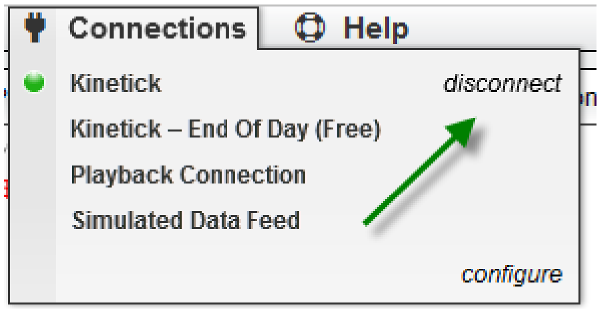
Connecting to a connection



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.

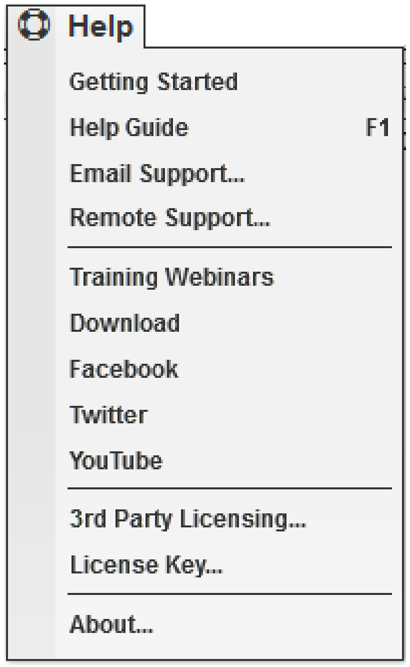
▽ Understanding the Pre-Built Connections

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)

11.8.5 Help Menu

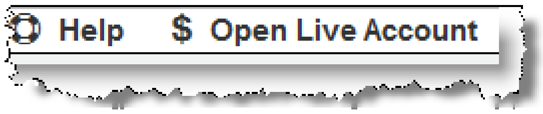
The following menus and items are available via the **Help** menu of the NinjaTrader Control Center.



Getting Started	Opens the Getting Started wizard interface displaying helpful tips to get operational quickly for new users
Help Guide	Opens the application help guide (or press F1)
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)
Training Webinars	Opens up the platform training page with educational webinars
Download	Download the latest version of NinjaTrader
Facebook	Opens your browser and directs you to the NinjaTrader Facebook webpage

Twitter	Opens your browser and directs you to the NinjaTrader Twitter webpage
YouTube	Opens your browser and directs you to the NinjaTrader YouTube webpage
3rd party licensing	Verification for 3rd party add on products
License Key...	Sets your NinjaTrader license key
About...	About NinjaTrader

Executing Trades to a live brokerage account



On the right next to the Help menu, you will find a direct link to purchase or lease your NinjaTrader live license key to execute trades to a live brokerage account

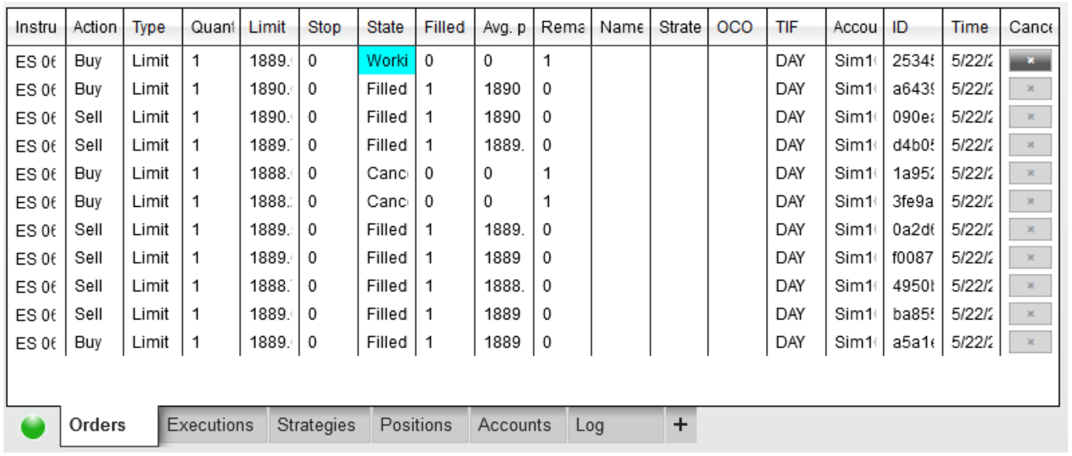
11.8.6 Orders Tab

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.



Instru	Action	Type	Quant	Limit	Stop	State	Filled	Avg. p	Rema	Name	Strate	OCO	TIF	Accou	ID	Time	Cance
ES 06	Buy	Limit	1	1889.	0	Worki	0	0	1				DAY	Sim1	2534	5/22/	×
ES 06	Buy	Limit	1	1890.	0	Filled	1	1890	0				DAY	Sim1	a643	5/22/	×
ES 06	Sell	Limit	1	1890.	0	Filled	1	1890	0				DAY	Sim1	090e	5/22/	×
ES 06	Sell	Limit	1	1889.	0	Filled	1	1889.	0				DAY	Sim1	d4b0	5/22/	×
ES 06	Buy	Limit	1	1888.	0	Canc	0	0	1				DAY	Sim1	1a95	5/22/	×
ES 06	Buy	Limit	1	1888.	0	Canc	0	0	1				DAY	Sim1	3fe9a	5/22/	×
ES 06	Sell	Limit	1	1889.	0	Filled	1	1889.	0				DAY	Sim1	0a2d	5/22/	×
ES 06	Sell	Limit	1	1889.	0	Filled	1	1889	0				DAY	Sim1	f0087	5/22/	×
ES 06	Sell	Limit	1	1888.	0	Filled	1	1888.	0				DAY	Sim1	4950	5/22/	×
ES 06	Sell	Limit	1	1889.	0	Filled	1	1889	0				DAY	Sim1	ba85	5/22/	×
ES 06	Buy	Limit	1	1889.	0	Filled	1	1889	0				DAY	Sim1	a5a1	5/22/	×

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
Remaini ng	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

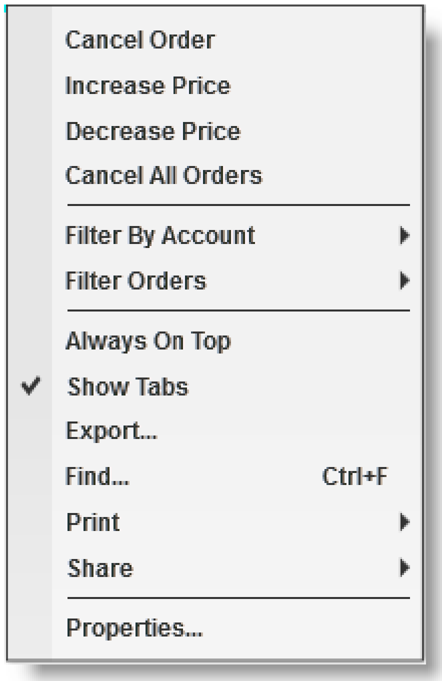
Cancel	Contains buttons which can be used to cancel resting orders.

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:

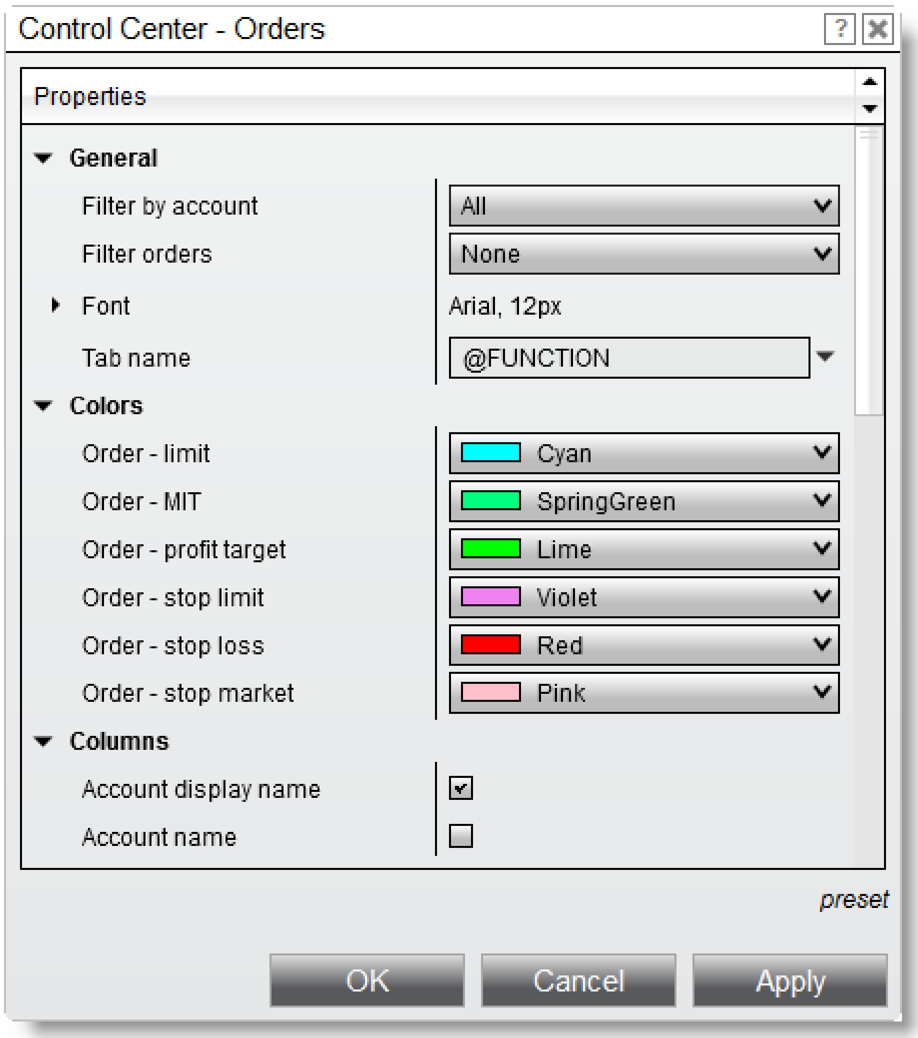


Cancel Order	Cancels the selected order
Increase Price	Increases the price of the order by 1 tick
Decrease Price	Decreases the price of the order by 1 tick
Cancel All Orders	Cancels all working orders
Filter By Account	Filters orders by selected account
Filter Orders	Filters shown orders, possible choices 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


Control Center - Orders

Properties

General	
Filter by account	All
Filter orders	None
Font	Arial, 12px
Tab name	@FUNCTION
Colors	
Order - limit	Cyan
Order - MIT	SpringGreen
Order - profit target	Lime
Order - stop limit	Violet
Order - stop loss	Red
Order - stop market	Pink
Columns	
Account display name	✓
Account name	☐

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

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.

11.8.7 Strategies Tab

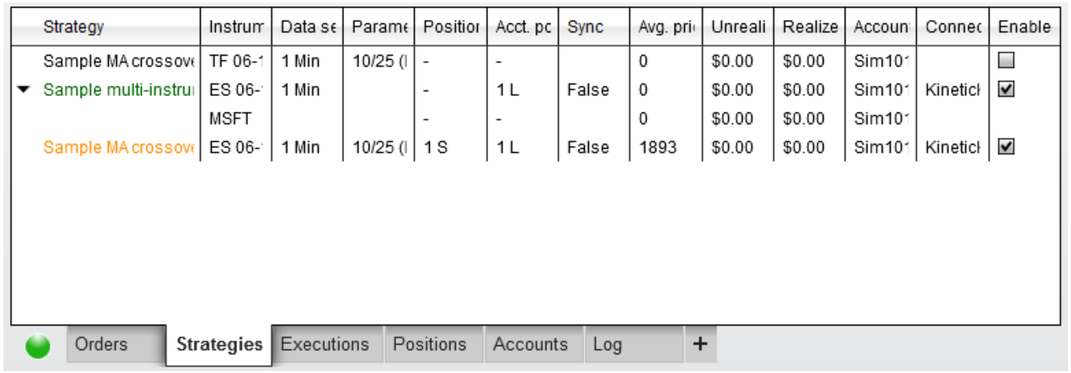
The Strategies tab displays running and terminated strategies in a data grid.

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.



Strategy	Instrum	Data se	Parame	Positior	Acct. pc	Sync	Avg. pri	Unreali	Realize	Accoun	Connec	Enable
Sample MA crossove	TF 06-1	1 Min	10/25 (l	-	-		0	$0.00	$0.00	Sim101		☐
Sample multi-instru	ES 06-1	1 Min		-	1 L	False	0	$0.00	$0.00	Sim101	Kineticl	☑
	MSFT			-	-		0	$0.00	$0.00	Sim101		
Sample MA crossove	ES 06-1	1 Min	10/25 (l	1 S	1 L	False	1893	$0.00	$0.00	Sim101	Kineticl	☑

- 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 options Strategies Tab 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
Instrum ent	The instrument on which the strategy is enabled
Data Series	The interval type and value associated with the strategy's instrument
Paramet ers	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. **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
Unrealiz	Any unrealized profit or loss of an open position

ed	entered by the strategy
Realize d	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
Connect ion	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.

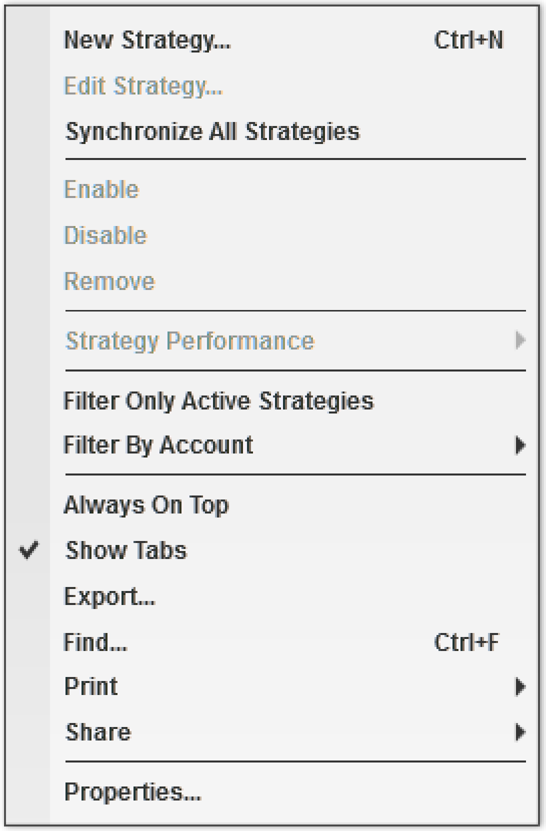
The following additional columns can be applied through the grid's **Properties** window:

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:



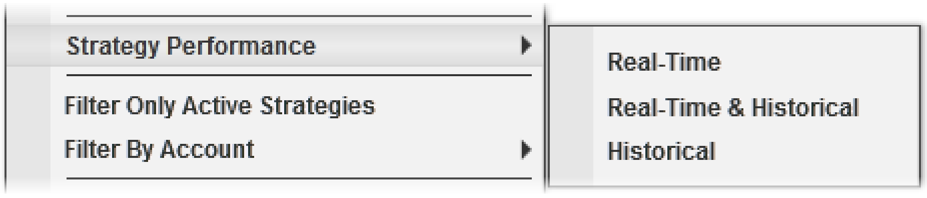
New Strategy...	Run a new automated NinjaScript strategy
Edit Strategy	Brings up the Edit Strategy window to edit the strategy parameters for the selected strategy. (Only disabled strategies can be edited)
Synchronize All Strategies	Will aggregate all strategy positions and syncs aggregate value to the 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	Generates a performance report for the

Performanc e	selected strategy (See the "*How to view strategy performance*" section below)
Filter Only Active Strategies	Displays only active strategies
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

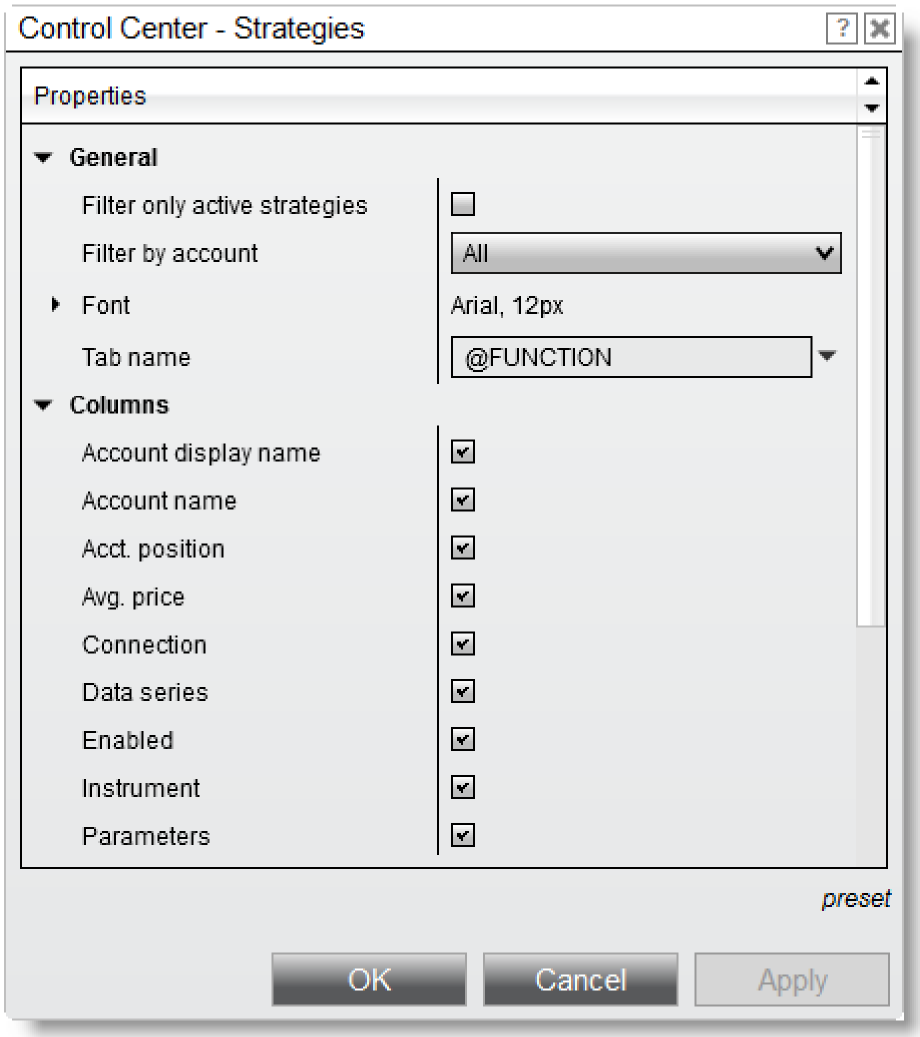
Strategy Performance

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


Control Center - Strategies

Properties

General

Filter only active strategies ☐

Filter by account: All

Font: Arial, 12px

Tab name: @FUNCTION

Columns

Account display name ✓

Account name ✓

Acct. position ✓

Avg. price ✓

Connection ✓

Data series ✓

Enabled ✓

Instrument ✓

Parameters ✓

Strategy Tab Properties

General	
Filter only active strategies	Displays only active strategies
Filter by account	Displays only strategies running on the selected account
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.

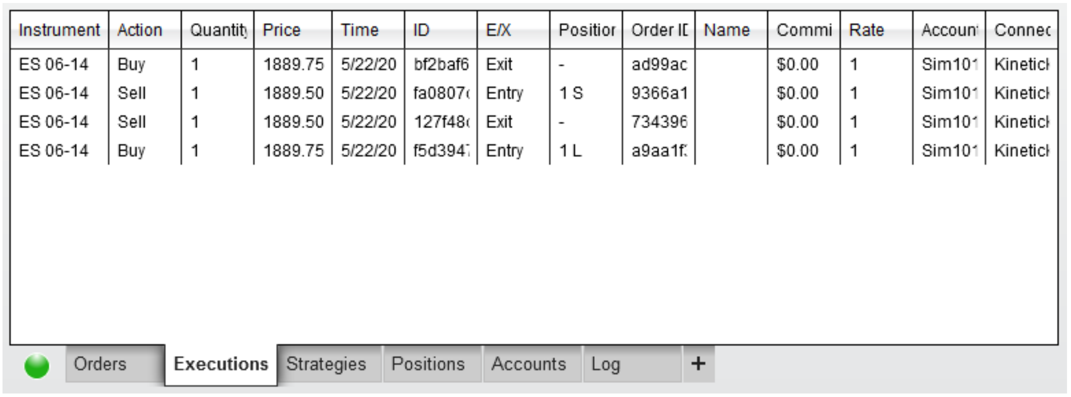
11.8.8 Executions Tab

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.



Instrument	Action	Quantity	Price	Time	ID	E/X	Positior	Order ID	Name	Commi	Rate	Accoun	Connec
ES 06-14	Buy	1	1889.75	5/22/20	bf2baf6	Exit	-	ad99ac		$0.00	1	Sim101	Kineticl
ES 06-14	Sell	1	1889.50	5/22/20	fa0807	Entry	1 S	9366a1		$0.00	1	Sim101	Kineticl
ES 06-14	Sell	1	1889.50	5/22/20	127f48	Exit	-	734396		$0.00	1	Sim101	Kineticl
ES 06-14	Buy	1	1889.75	5/22/20	f5d394	Entry	1 L	a9aa1f		$0.00	1	Sim101	Kineticl

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

© 2017 NinjaTrader, LLC

Connec tion	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:

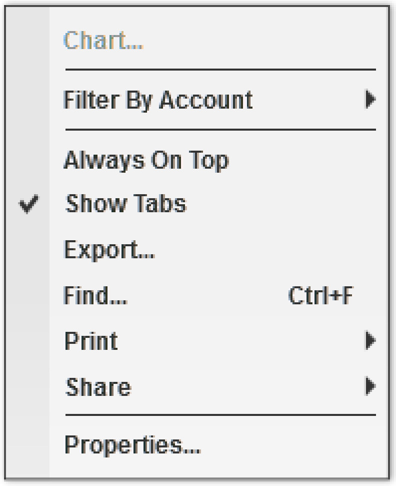


Chart	Opens a chart of the instrument at the time of the selected execution
Filter By Account	Sets which executions to display by account
Export	Exports the grid contents to "CSV" or "Excel" file format
Always	Sets if the window will be always on top of other

On Top	windows
Show Tabs	Sets if the window should allow for tabs
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.

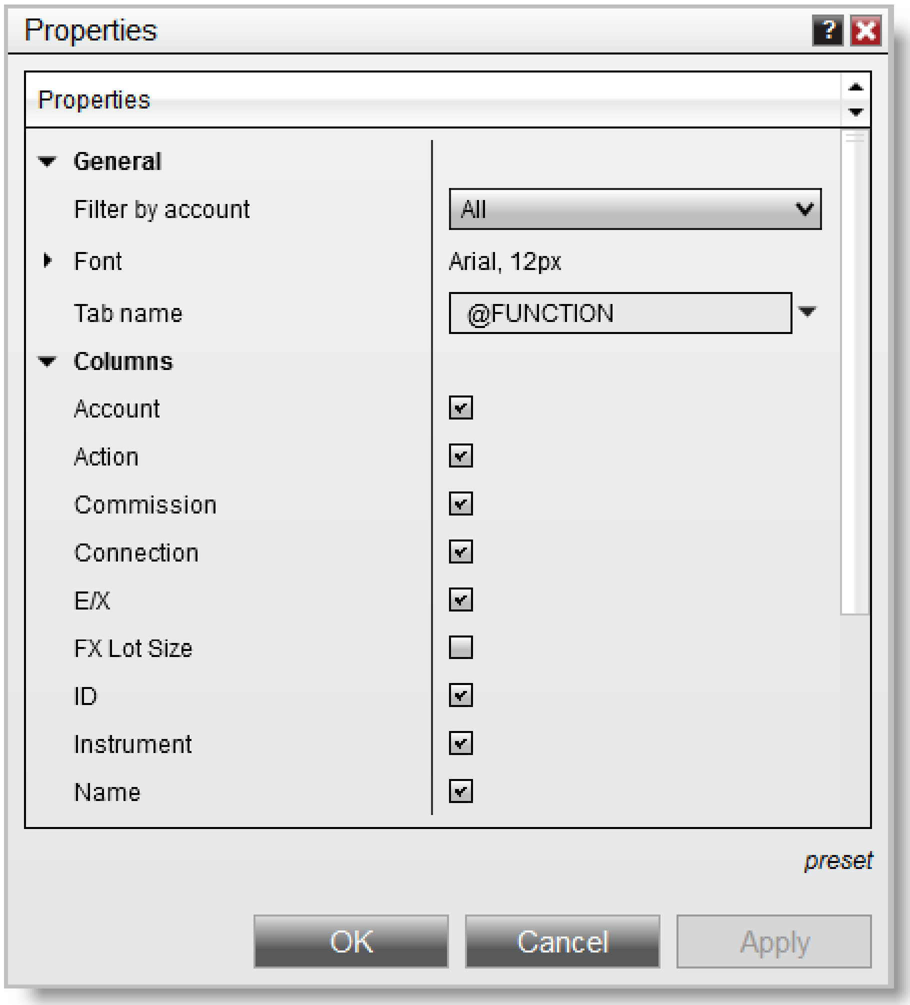
Note: NinjaTrader connection setting "Auto subscribe to required instruments" must be enabled when creating your forex connection for NinjaTrader to automatically get the currency rates needed for PnL conversion when the trade execution occurs. For more information please see the connection guide for 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


Properties

Properties

General

Filter by account: All

Font: Arial, 12px

Tab name: @FUNCTION

Columns

Account: ✓

Action: ✓

Commission: ✓

Connection: ✓

E/X: ✓

FX Lot Size: ☐

ID: ✓

Instrument: ✓

Name: ✓

Executions Tab Properties

General	
Filter by	Filters orders by selected account

account	
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.

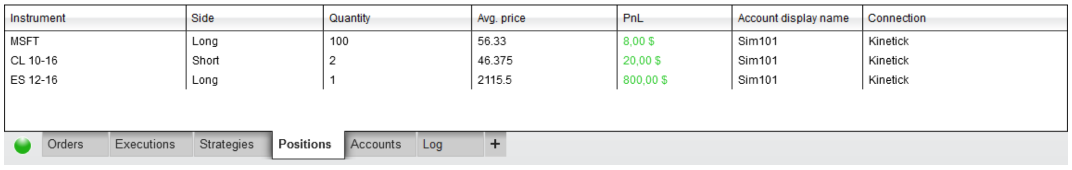
11.8.9 Positions Tab

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.



Instrument	Side	Quantity	Avg. price	PnL	Account display name	Connection
MSFT	Long	100	56.33	8,00 $	Sim101	Kinetick
CL 10-16	Short	2	46.375	20,00 $	Sim101	Kinetick
ES 12-16	Long	1	2115.5	800,00 $	Sim101	Kinetick

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

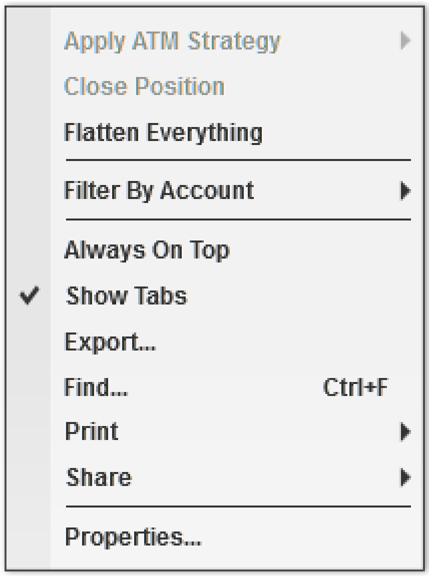
The following additional columns can be applied through the grid's **Properties** window:

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:

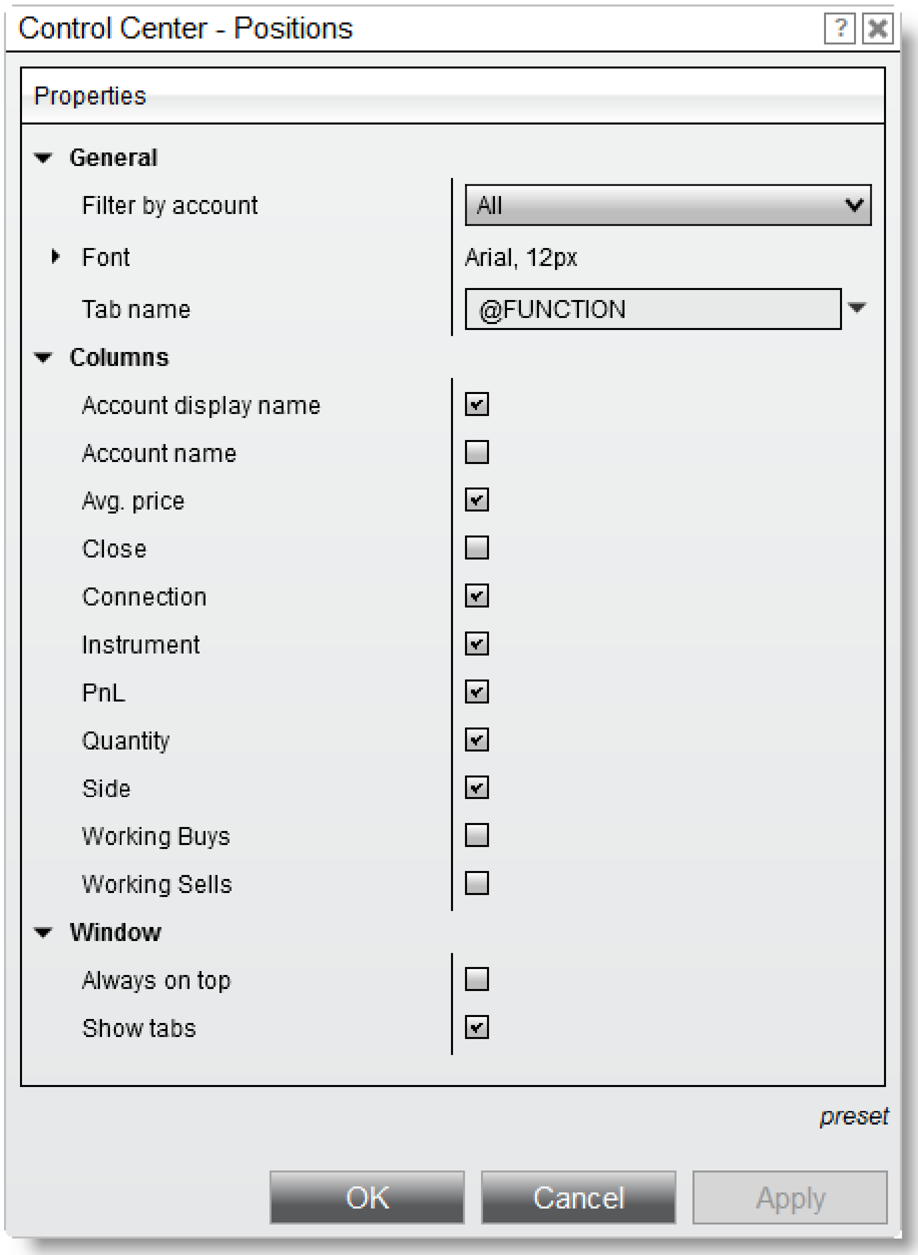


Apply ATM Strategy	Allows you to apply a predefined ATM Strategy Template to an open position using the current market price as the entry price.
Close Position*	Flattens the currently selected position in the grid and cancels any working orders associated to the position's instrument
Flatten Everything*	Flattens all open positions and cancels all working orders
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



General
Filter by account: All
Font: Arial, 12px
Tab name: @FUNCTION
Columns
Account display name ✓
Account name
Avg. price ✓
Close
Connection ✓
Instrument ✓
PnL ✓
Quantity ✓
Side ✓
Working Buys
Working Sells
Window
Always on top
Show tabs ✓

preset

Position Tab Properties

General	
Filter by account	Filters orders by selected account
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.

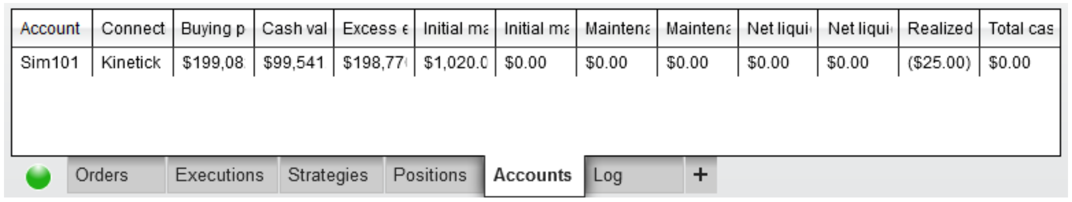
11.8.10 Accounts Tab

The Accounts tab displays current account information in a data grid. The account values that are displayed is dependant on your connectivity provider. Not all connectivity providers transmit complete account data.

Play Video

Understanding the accounts tab

Understanding the Accounts Tab



Account	Connect	Buying p	Cash val	Excess e	Initial ma	Initial ma	Maintena	Maintena	Net liqui	Net liqui	Realized	Total cas
Sim101	Kinetick	$199,08	$99,541	$198,77	$1,020.0	$0.00	$0.00	$0.00	$0.00	$0.00	($25.00)	$0.00

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

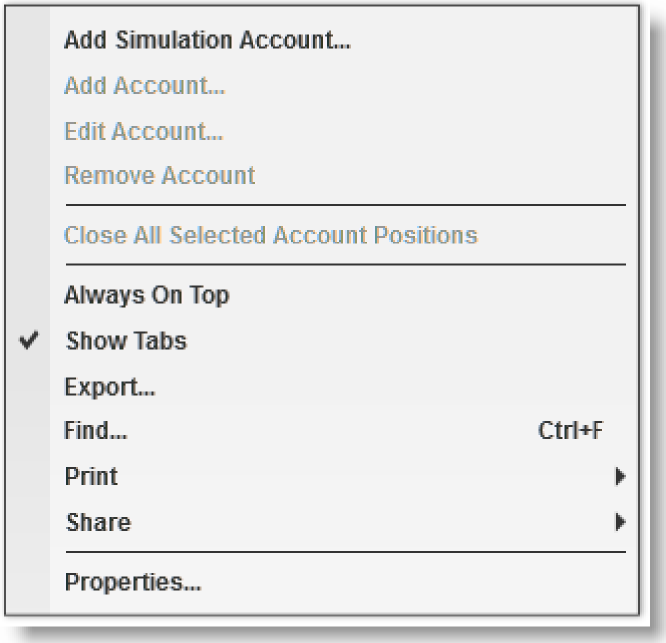
The following additional columns can be applied through the grid's **Properties** window:

Close	Contains a button which will allow you to close the position
Commissi on 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 Maintenan ce 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 Commissi ons	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
Add Account	Opens the Account window to configure a new account
Edit Account	Opens the Account window to edit the selected account (**Note**: The Playback101 account cannot be edited, it inherits its settings from the Sim101)

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

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.

Connectiv	**Buying Power**	**Cash Value**	**Excess Intraday Mar**	**Excess Equ**	**Initial Margin**	**Intraday Margin**	**Overnight Margin**	**Maintenance Margi**	**Maintenance Margi**	**Net Liquidation**	**Net Liquidation By C**	**Realized PnL**	**Total Cash Balance**

ityProvider			gin	ityOvernight				nIntraday	nOvernight		urrency		
NinjaTraderSimulationAccoun	✓	✓	✓			✓						✓	

ts (values are all NinjaTrader calculated since its a simula													

tion account)													
BarChart.com (values **exclude** commissio	✓	✓	✓			✓							

ns)													
Continuum (values **exclude** commissions)		✓										✓	
CQG (va		✓										✓	

lues **exclude** commissions)													
FXCM (values **exclude** commis		✓										✓	

sions)													
InteractiveBrokers (values **include** commiss	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓

ions)												
Rithmic (values **exclude** commissions)		✓									✓	
TDAmer	✓	✓									✓ (NinjaTrader cal	

itrade (values **exclude** commissions)												culated)	

Understanding Currency Conversion

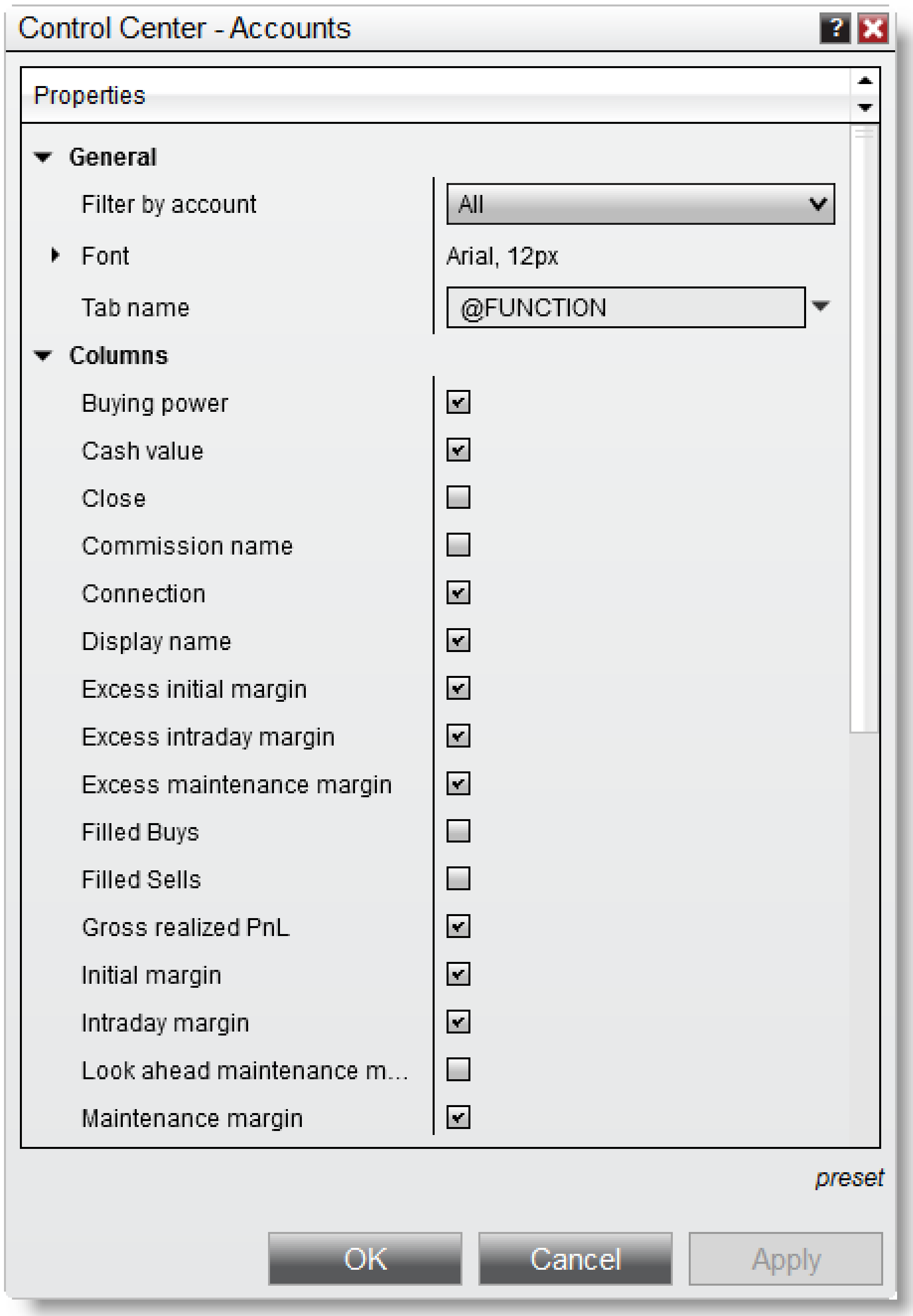
NinjaTrader will attempt to convert currency for forex and futures trader.

- 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


Control Center - Accounts

Properties

General

Filter by account	All
Font	Arial, 12px
Tab name	@FUNCTION

Columns

Buying power	✓
Cash value	✓
Close	☐
Commission name	☐
Connection	✓
Display name	✓
Excess initial margin	✓
Excess intraday margin	✓
Excess maintenance margin	✓
Filled Buys	☐
Filled Sells	☐
Gross realized PnL	✓
Initial margin	✓
Intraday margin	✓
Look ahead maintenance m...	☐
Maintenance margin	✓

Accounts Tab Properties

General	
Filter by account	Filters accounts display by selected account
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.

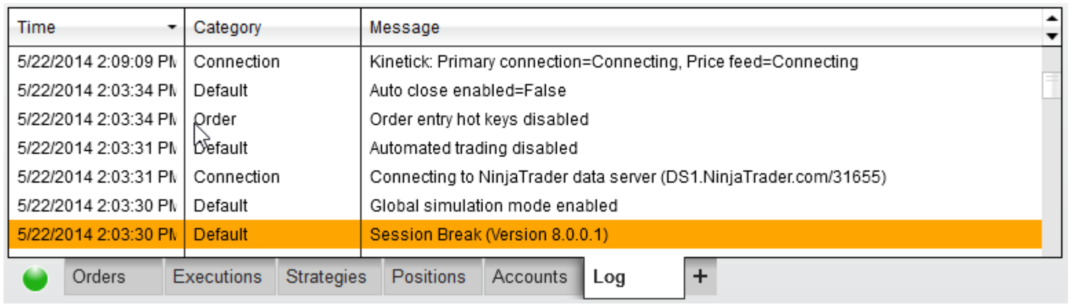
11.8.11 Log Tab

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.

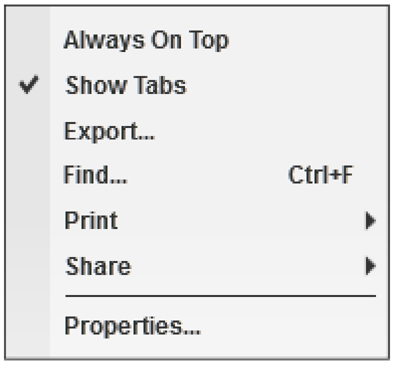


```
Time                 Category     Message
5/22/2014 2:09:09 PM Connection   Kinetick: Primary connection=Connecting, Price feed=Connecting
5/22/2014 2:03:34 PM Default      Auto close enabled=False
5/22/2014 2:03:34 PM Order        Order entry hot keys disabled
5/22/2014 2:03:31 PM Default      Automated trading disabled
5/22/2014 2:03:31 PM Connection   Connecting to NinjaTrader data server (DS1.NinjaTrader.com/31655)
5/22/2014 2:03:30 PM Default      Global simulation mode enabled
5/22/2014 2:03:30 PM Default      Session Break (Version 8.0.0.1)
```

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

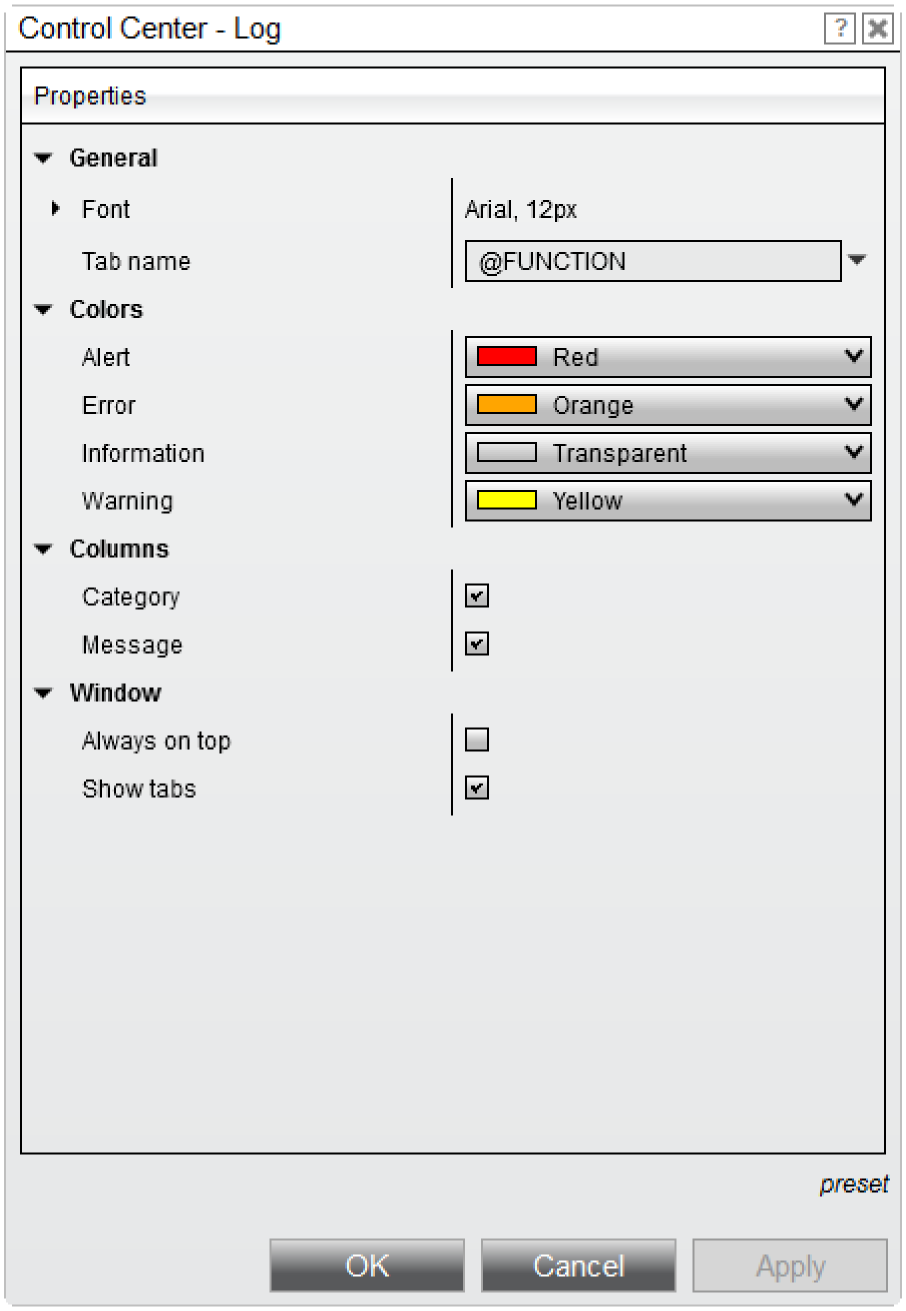
Right mouse clicking within the log display section opens the following menu:



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.
Propertie s...	Configure the positions grid properties

Log tab properties



Control Center - Log

Properties

General

Font	Arial, 12px
Tab name	@FUNCTION

Colors

Alert	🟥 Red
Error	🟧 Orange
Information	⬜ Transparent
Warning	🟨 Yellow

Columns

Category	✓
Message	✓

Window

Always on top	
Show tabs	✓

Log Tab Properties

General	
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
Columns	Sets that columns are enabled or disabled in the log 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.

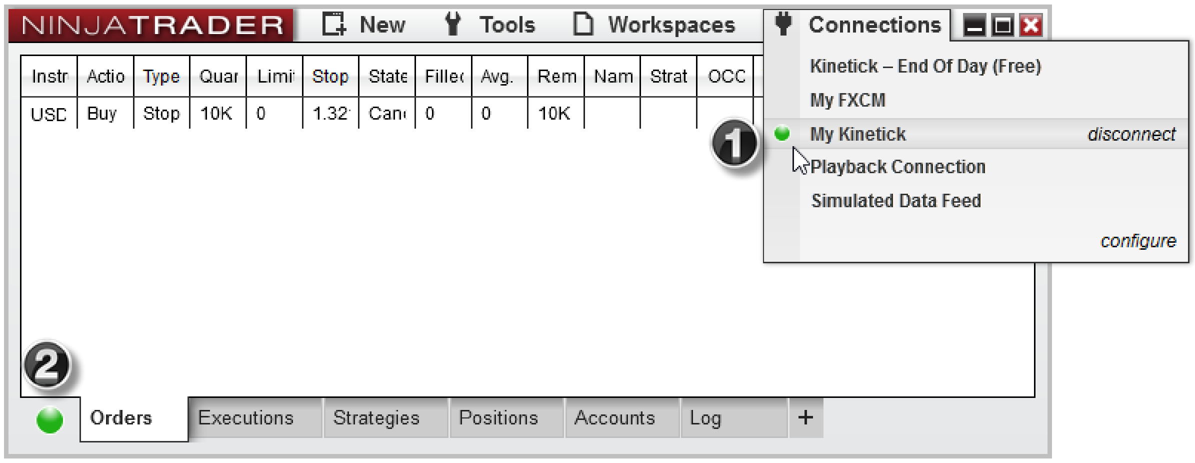
If you change your settings and later wish to go back to the original factory settings, you can left mouse click on the **preset** text and select the option to **restore** to return to the original factory settings - please note though that you cannot save a custom default to restore to.

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.

11.8.12 Connection Status

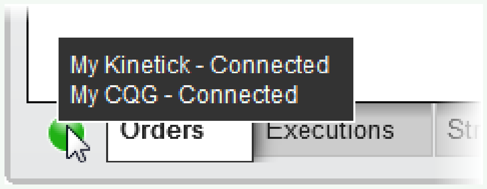
Connection Status

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:

●	Connected	Indicates that NinjaTrader is fully connected

🟡	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

11.9 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

11.9.1 Working with Data Grids

All data grids found throughout NinjaTrader are customizable.

Data Grids

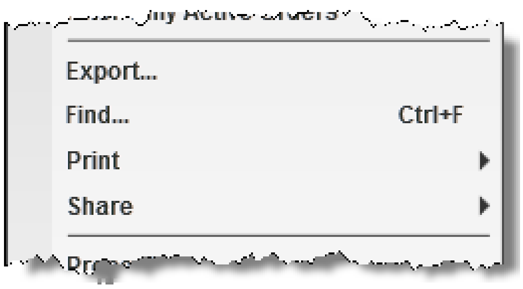
With a data grid you can:

- 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

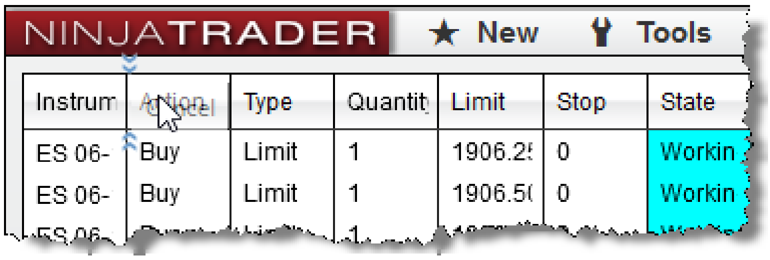
Right Click Menu

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

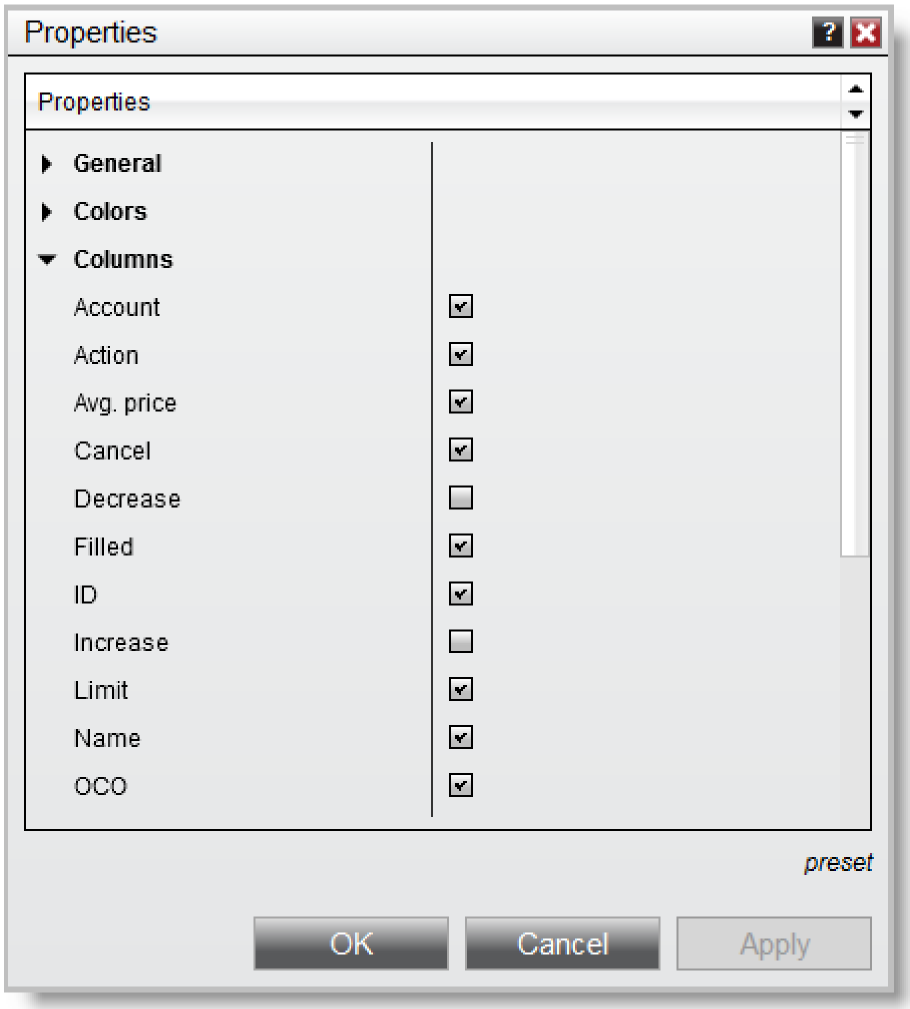
To adjust the order of columns within a data gird, left mouse click and hold on the column header and drag it to the location you wish to place the selected column. Two blue arrows will appear above and below the location in the grid you will be moving the column to (see image below). Release the mouse button to place the column in the new location.



Instrum	Action	Type	Quantit	Limit	Stop	State
ES 06-	Buy	Limit	1	1906.2!	0	Workin
ES 06-	Buy	Limit	1	1906.5(	0	Workin

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.


Properties

Properties

General	
Colors	
Columns	
Account	✓
Action	✓
Avg. price	✓
Cancel	✓
Decrease	☐
Filled	✓
ID	✓
Increase	☐
Limit	✓
Name	✓
OCO	✓

11.10 Database

Database Window Overview

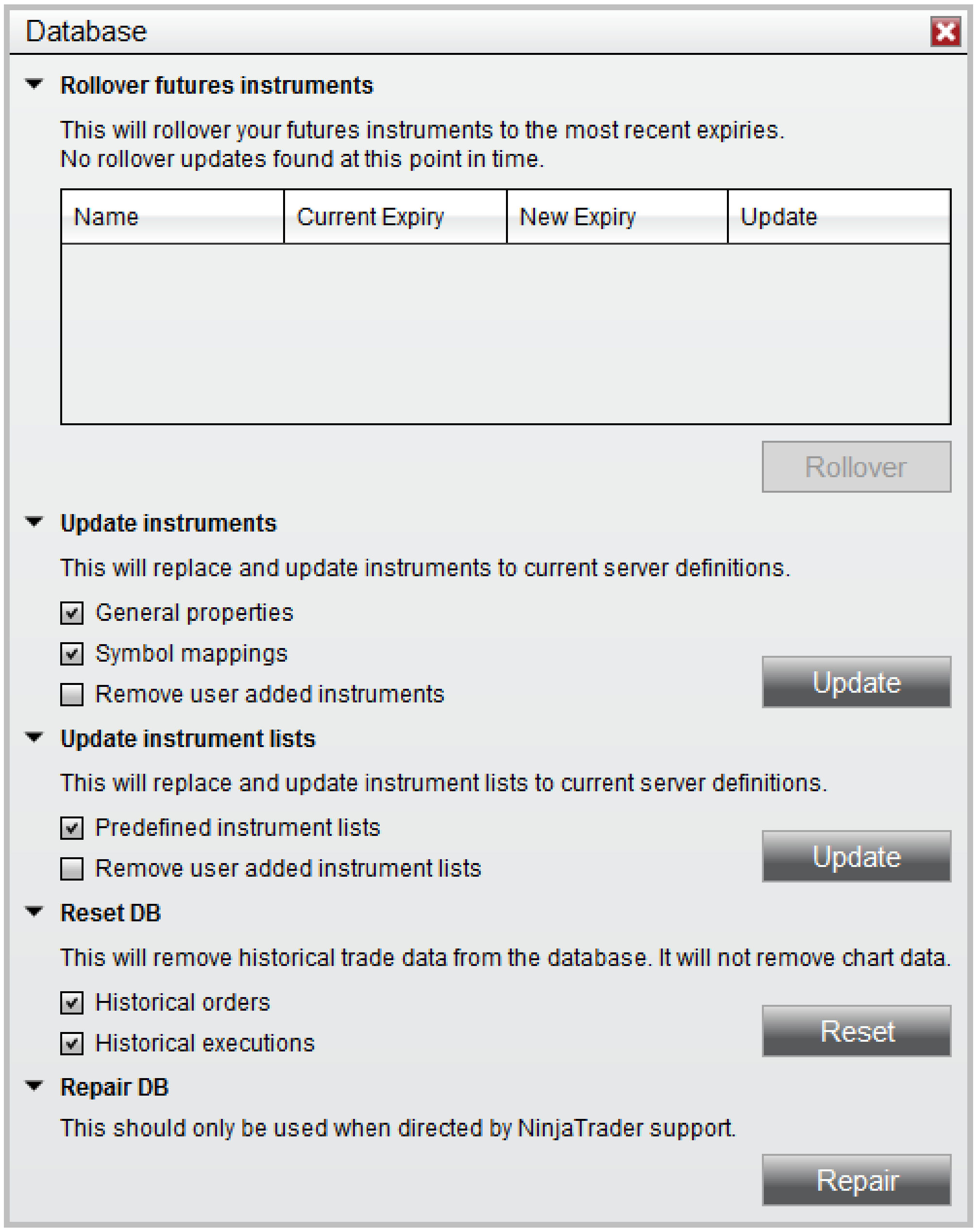
You can access the **Database** window by left mouse clicking in the **Tools** menu within the NinjaTrader Control Center and selecting the menu item **Database...**

The **Database** window allows for the centralized management of all database related functions.

› Database Operations

11.10.1 Database Operations

Various database operations can be performed from the **Database** window.



Database

Rollover futures instruments

This will rollover your futures instruments to the most recent expiries.
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
- ☑ 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

Reset DB

This will remove historical trade data from the database. It will not remove chart data.

- ☑ Historical orders
- ☑ Historical executions

Reset

Repair DB

This should only be used when directed by NinjaTrader support.

Repair

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
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.

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.

11.11 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

11.11.1 Loading Historical Data

When does NinjaTrader load 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
2. The **End date** parameter of the **Data Series** window contains the last day of data available in your data repository
3. The data repository contains no data within 3 days of the first day loaded in the chart

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 data repository/ 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 data repository/ cache for other dates
3. Chart of 1/2/14 to 1/4/14 -> use data stored in data repository/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.

The NinjaTrader cache and speeding up the loading of data

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 internal cache contains data for your instrument of interest:

1. Open and maintain a chart of any time frame containing the instrument
2. Load the instrument into a Market Analyzer window along with an indicator column

The **Market Analyzer** option is not as viable since it only maintains a 100 bar cache as a default setting that can be changed. Since 99% of all charting requests consist of much more than 100 bars, they will most likely incur a data load.

Note: The internal cache is managed by the .NET framework so it is unpredictable when the .NET runtime will clear it. In most cases, as long as you have at least one chart of your instrument open, the instrument’s data cache will persist.

11.11.2 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:

Connectivity Provider	Real-Time Data	Historical Tick Data	Historical Bid / Ask Minute Dat	Historical Bid / Ask Daily Dat	Historical Bid / Ask Tick Dat	Historical Minute Data	Historical Daily Data	Real-Time Tim	Instruments Su	Real-Time Ne	Tick Replay	Daily Bars Tradin	Settlement adj

			a	a	a			estamp	pported	ws		g Hours	usted Close Price For Daily Bars
Kinetick www.kinetick.com	YES (subscription only)	YES (subscription only)	NO	NO	YES	YES (subscription only)	YES	Native	E, F, FX, I	YES	YES	Symbol Map Specific	YES
Bar Chart	YES	YES	NO	NO	NO	YES	YES	Native	E, F, FX, I	NO	YES	Symbol Map Specific	YES

Continuum	YES	YES	YES	YES	YES	YES	YES	Native	F, I	NO	YES	Extended Trading Hours	YES
CQG	YES	YES	YES	YES	YES	YES	YES	Native	F, I	NO	YES	Extended Trading Hours	YES
eSignal	YES	YES	NO	NO	YES	YES	YES	Native	E, F, FX, I	YES	YES	Symbol Map Specific	YES
FXCM	YES	YES	YES	YES	YES	YES	YES	Native	FX	NO	YES	Forex	N/A
Google	NO	NO	NO	NO	NO	NO	YES	---	E	NO	NO	Regular	N/A

												Trading Hours	
Interactive Brokers	YES	NO	YES	YES	NO	YES (live account only)	YES (live account only)	Local	E, F, FX, I	NO	NO	Extended Trading Hours	NO
IQFeed	YES	YES	NO	NO	YES	YES	YES	Native	E, F, FX, I	YES	YES	Symbol Map Specific	YES
Rithmic	YES	YES	YES	YES	YES	YES	YES	Native	F	NO	YES	Extended Trading Hours	YES

TD Ameritrade	YES	NO	NO	NO	NO	YES	YES	Local	E, I	YES	NO	Regular Trading Hours	NO
YAHOO	NO	NO	NO	NO	NO	NO	YES	Local	E	NO	NO	Regular Trading Hours	NO

E = Equities
F = Futures
FX = Forex
I = Indexes

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.

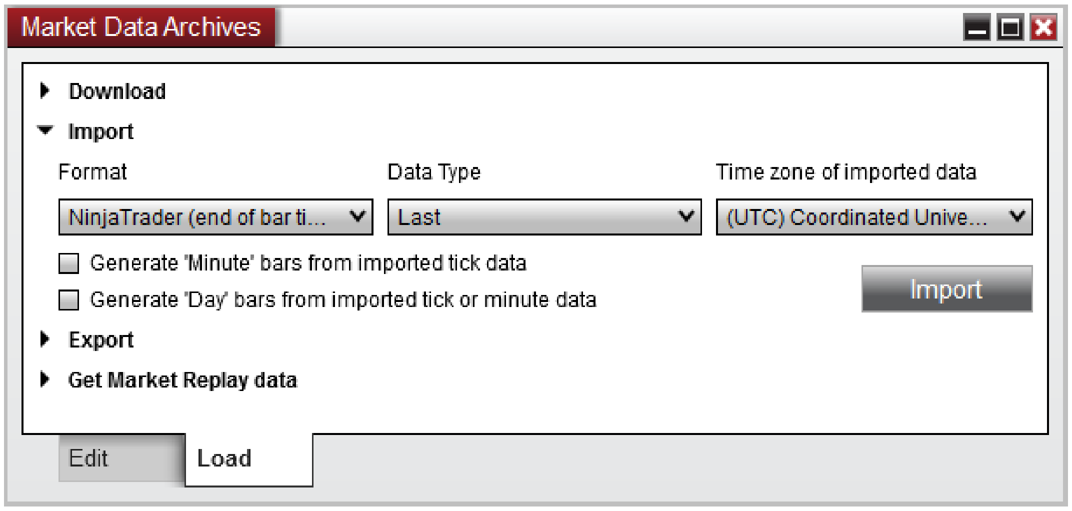
Create a Connection to Yahoo for Free Historical Daily Data

You can create a connection to Yahoo Finance via the Connection Guide. This connection provides access to free historical daily data that can be used for system development and backtesting.

11.11.3 Importing

Historical data can be imported from a text file with a ".txt" extension within the **Load** tab 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


Market Data Archives

Download

Import

Format: NinjaTrader (end of bar ti...

Data Type: Last

Time zone of imported data: (UTC) Coordinated Unive...

☐ Generate 'Minute' bars from imported tick data

☐ Generate 'Day' bars from imported tick or minute data

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. MetaStock - Select this option if importing a MetaStock historical data text file
2. NinjaTrader (timestamps in import file(s) represent end of bar time)
3. NinjaTrader (timestamps in import file(s) represent start of bar time)

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

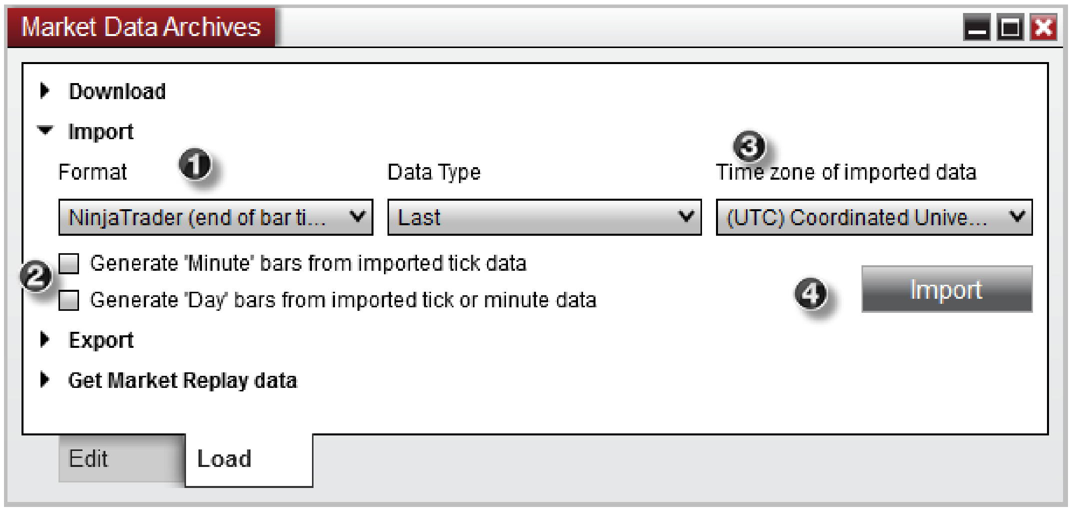
Importing Tips

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:


Market Data Archives

Download

Import

Format: NinjaTrader (end of bar ti...

Data Type: Last

Time zone of imported data: (UTC) Coordinated Unive...

Generate 'Minute' bars from imported tick data

Generate 'Day' bars from imported tick or minute data

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)

❹ Press the Import button

❺ Select the text file from your PC to import and press the "Open" button.

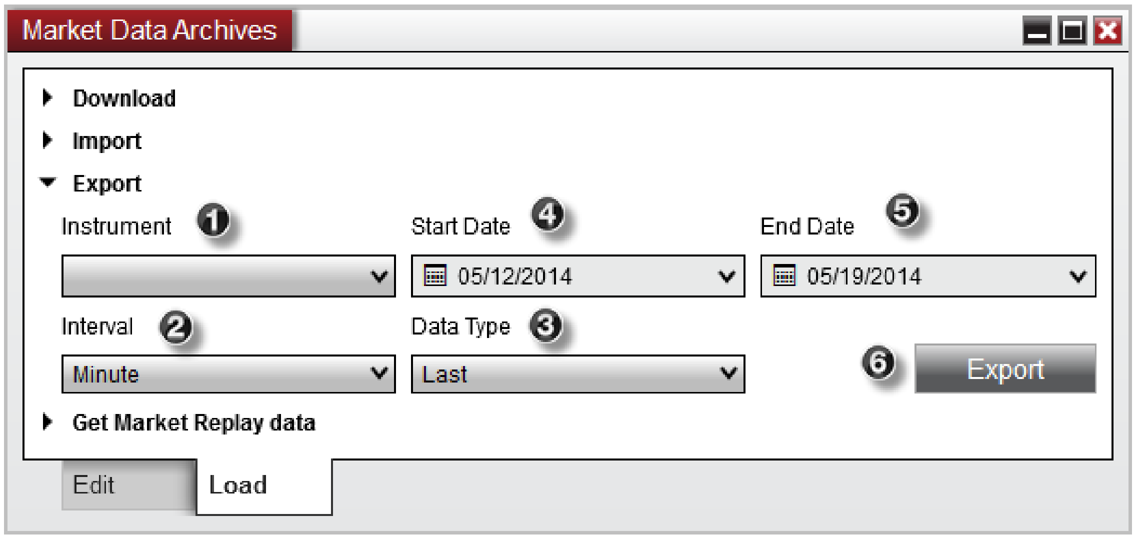
NinjaTrader will attempt to import the text file. If successful, a window will appear confirming this. If unsuccessful, an error window will appear and you should check the Log tab of the Control Center to view the error(s).

11.11.4 Exporting

Historical data stored within NinjaTrader can be exported to a text ".txt" file. This is done within the **Load** tab 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:

❶ Select the instrument to export. The instruments you have data for will be available for selection.

❷ 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.

11.11.5 Editing

Historical data saved in NinjaTrader can be edited via the Edit tab 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**"

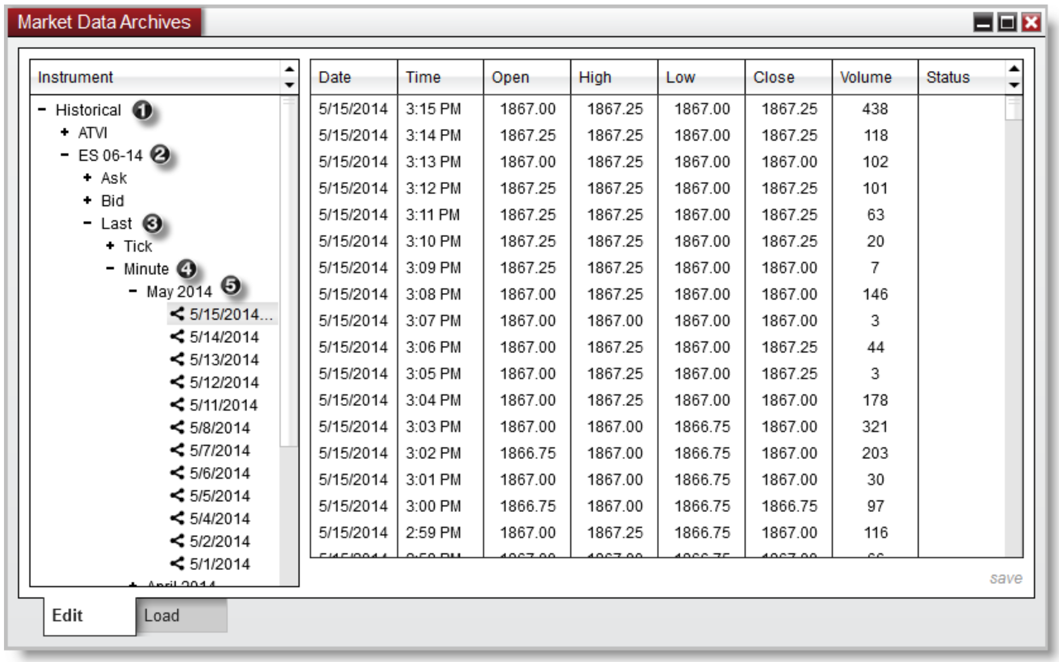
2 Select the "**+**" for the Instrument.

3 Select "Ask," "Bid," or "Last,"data type

4 Select the "**Tick**", "**Minute**", "**Day**" data type

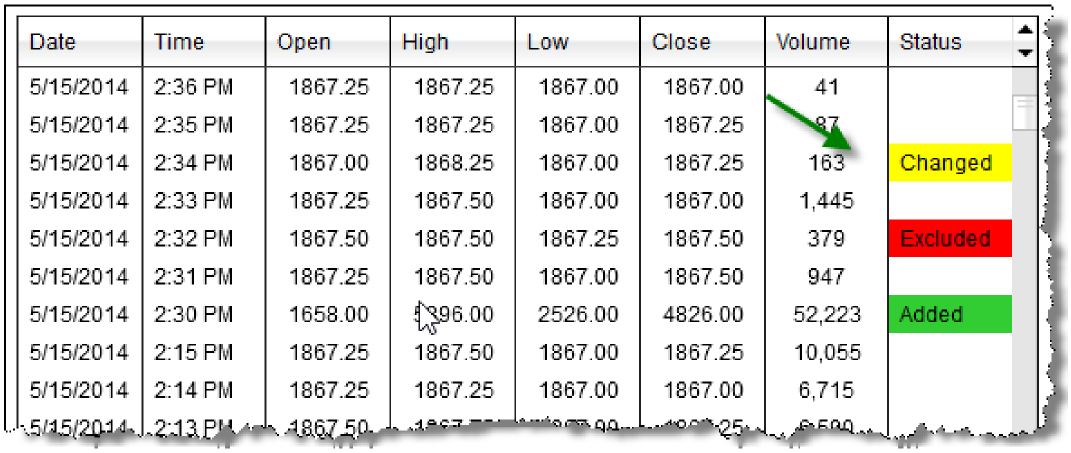
5 Select the **date.**

The data for that data type and date will be shown in the data grid.



Date	Time	Open	High	Low	Close	Volume	Status
5/15/2014	3:15 PM	1867.00	1867.25	1867.00	1867.25	438	
5/15/2014	3:14 PM	1867.25	1867.25	1867.00	1867.25	118	
5/15/2014	3:13 PM	1867.00	1867.25	1867.00	1867.00	102	
5/15/2014	3:12 PM	1867.25	1867.25	1867.00	1867.25	101	
5/15/2014	3:11 PM	1867.25	1867.25	1867.00	1867.25	63	
5/15/2014	3:10 PM	1867.25	1867.25	1867.00	1867.25	20	
5/15/2014	3:09 PM	1867.25	1867.25	1867.00	1867.00	7	
5/15/2014	3:08 PM	1867.00	1867.25	1867.00	1867.00	146	
5/15/2014	3:07 PM	1867.00	1867.00	1867.00	1867.00	3	
5/15/2014	3:06 PM	1867.00	1867.25	1867.00	1867.25	44	
5/15/2014	3:05 PM	1867.00	1867.25	1867.00	1867.25	3	
5/15/2014	3:04 PM	1867.00	1867.25	1867.00	1867.00	178	
5/15/2014	3:03 PM	1867.00	1867.00	1866.75	1867.00	321	
5/15/2014	3:02 PM	1866.75	1867.00	1866.75	1867.00	203	
5/15/2014	3:01 PM	1867.00	1867.00	1866.75	1867.00	30	
5/15/2014	3:00 PM	1866.75	1867.00	1866.75	1866.75	97	
5/15/2014	2:59 PM	1867.00	1867.25	1866.75	1867.00	116	

- 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.



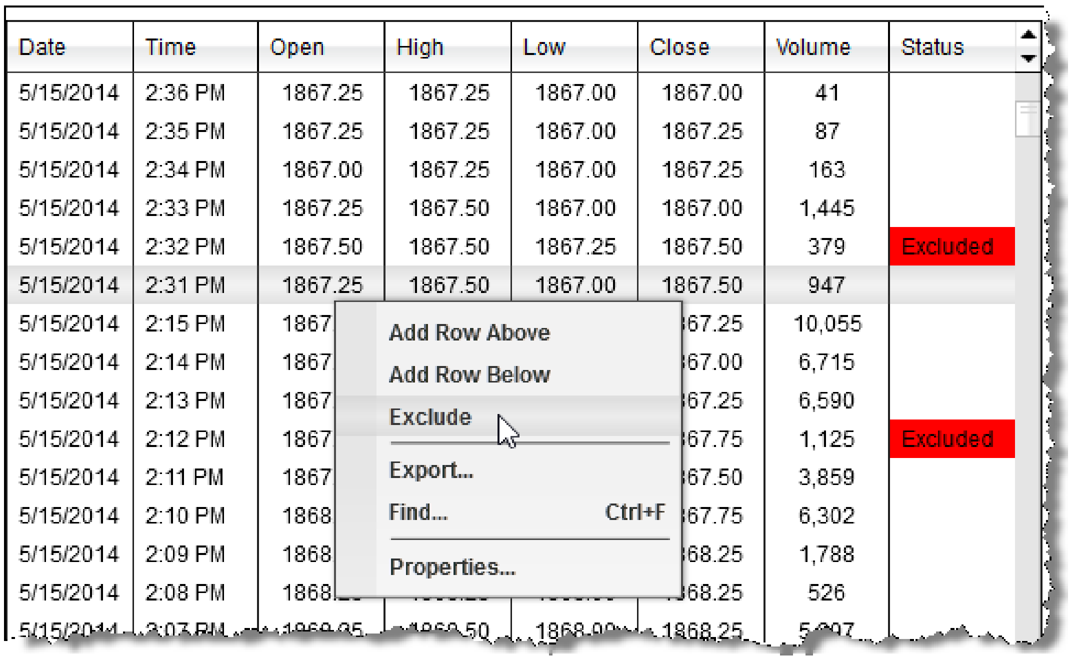
Date	Time	Open	High	Low	Close	Volume	Status
5/15/2014	2:36 PM	1867.25	1867.25	1867.00	1867.00	41	
5/15/2014	2:35 PM	1867.25	1867.25	1867.00	1867.25	87	
5/15/2014	2:34 PM	1867.00	1868.25	1867.00	1867.25	163	Changed
5/15/2014	2:33 PM	1867.25	1867.50	1867.00	1867.00	1,445	
5/15/2014	2:32 PM	1867.50	1867.50	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	5396.00	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.

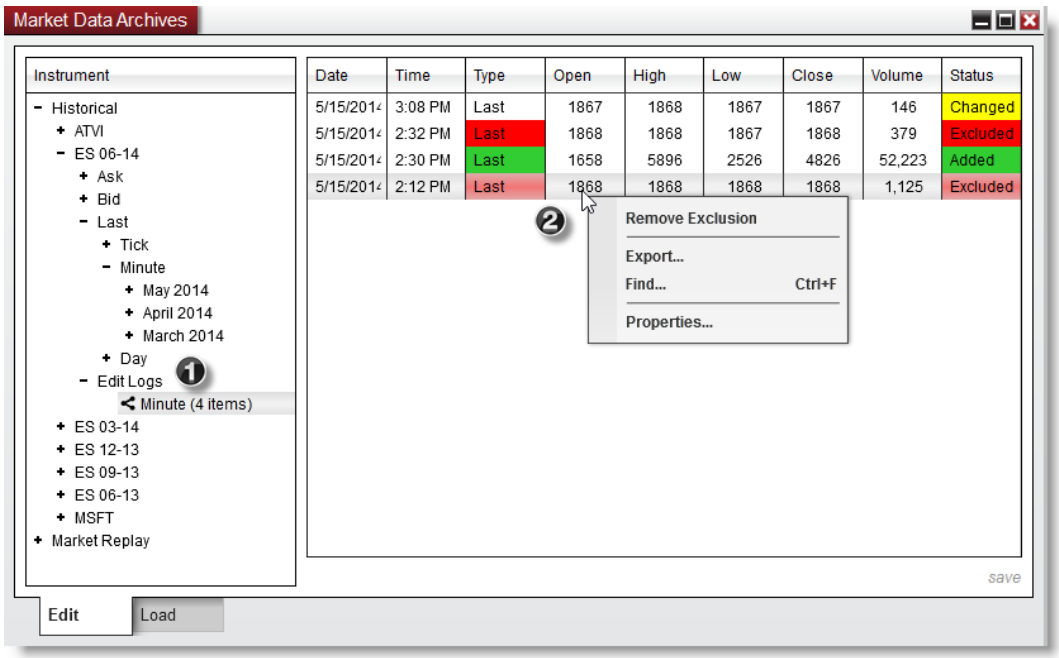


Date	Time	Open	High	Low	Close	Volume	Status
5/15/2014	2:36 PM	1867.25	1867.25	1867.00	1867.00	41	
5/15/2014	2:35 PM	1867.25	1867.25	1867.00	1867.25	87	
5/15/2014	2:34 PM	1867.00	1867.25	1867.00	1867.25	163	
5/15/2014	2:33 PM	1867.25	1867.50	1867.00	1867.00	1,445	
5/15/2014	2:32 PM	1867.50	1867.50	1867.25	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	
5/15/2014	2:07 PM	1868.25	1868.50	1868.00	1868.25	5,207	

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.**



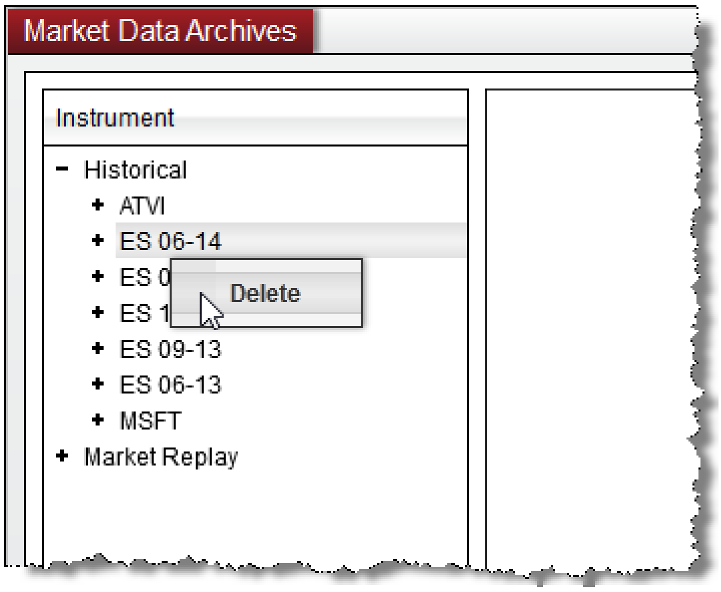
Date	Time	Type	Open	High	Low	Close	Volume	Status
5/15/201	3:08 PM	Last	1867	1868	1867	1867	146	Changed
5/15/201	2:32 PM	Last	1868	1868	1867	1868	379	Excluded
5/15/201	2:30 PM	Last	1658	5896	2526	4826	52,223	Added
5/15/201	2:12 PM	Last	1868	1868	1868	1868	1,125	Excluded

How to delete historical data

Historical data saved in NinjaTrader can be deleted via the Edit tab 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.

11.11.6 Download

Historical data can be downloaded from the data provider via the Download tab.

How to Download Historical Data

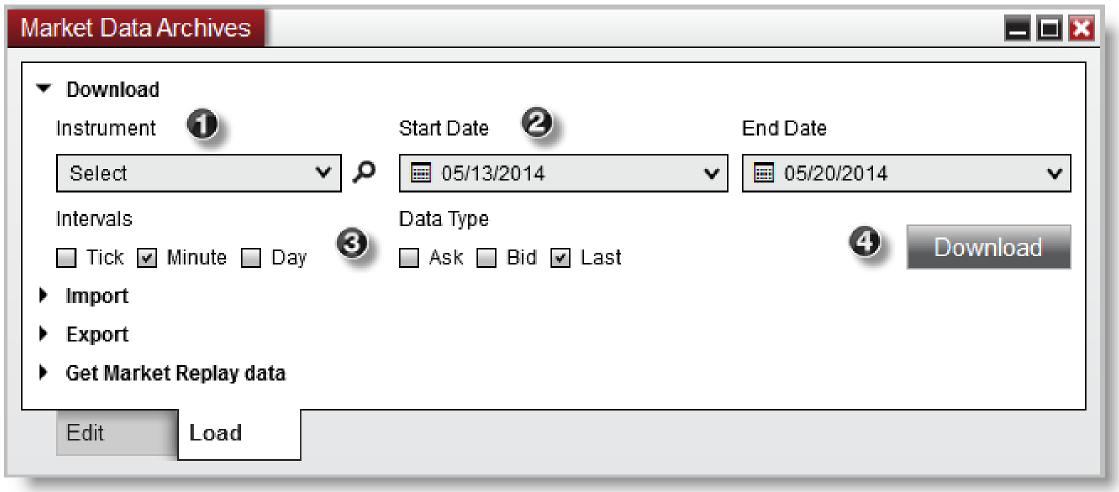
To download historical data first make sure NinjaTrader is connected and historical data is available from your data provider.

❶ Select an instrument for data to be downloaded. (**Tip**: You may also select an instrument list)

❷ Select the desired Start and End date range

❸ Select the desired Intervals and Data Types

❹ 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.

11.12 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...**

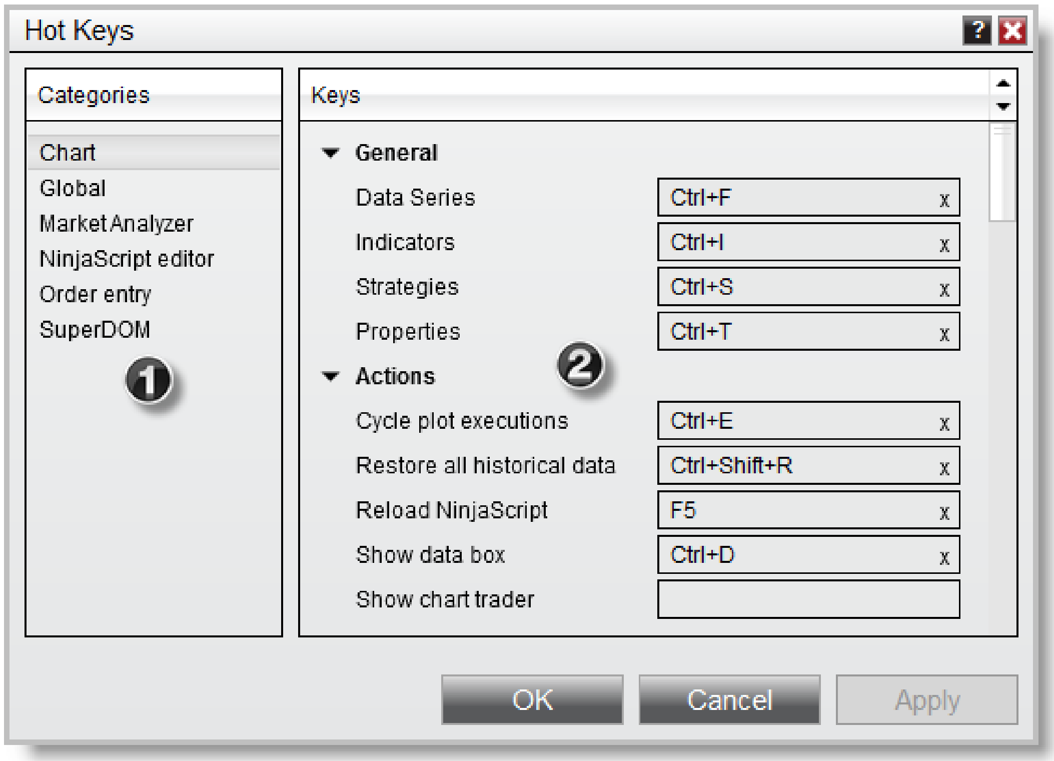
NinjaTrader allows you to assign specific key strokes as **Hot Keys** in order to quickly perform a task. **Hot Key** utilization includes, but is not limited to: opening new windows, performing tasks within open windows, and placing orders in an order entry window. The **Hot Key** window allows you to add and remove **Hot Key** assignments to various application actions.

› Working with Hot Keys
› Trading with Hot Keys

11.12.1 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



General	
Data Series	Ctrl+F
Indicators	Ctrl+I
Strategies	Ctrl+S
Properties	Ctrl+T
Actions	
Cycle plot executions	Ctrl+E
Restore all historical data	Ctrl+Shift+R
Reload NinjaScript	F5
Show data box	Ctrl+D
Show chart trader	

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.

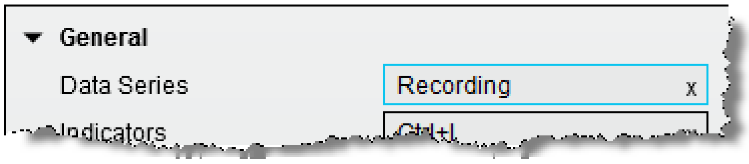
2. **Available Actions and Hot Keys**
The Keys section displays the actions available for Hot Key assignment within the selected active window.

Assigning and Removing Hot Keys

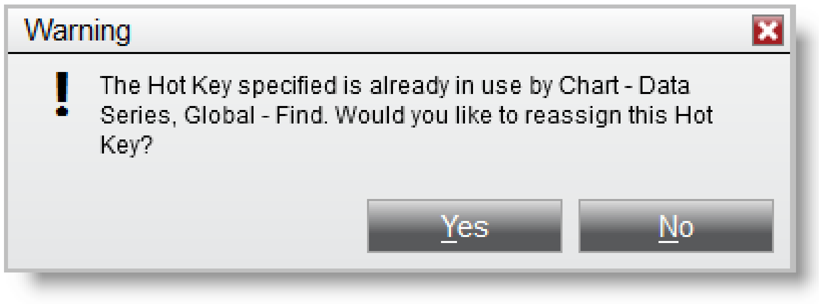
Assigning a Hot Key

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



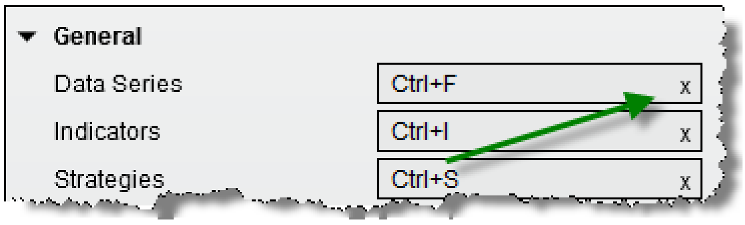
Note: If you try to assign a **Hot Key** that would conflict with an already defined **Hot Key** you will be asked to reassign.


Warning

The Hot Key specified is already in use by Chart - Data Series, Global - Find. Would you like to reassign this Hot Key?

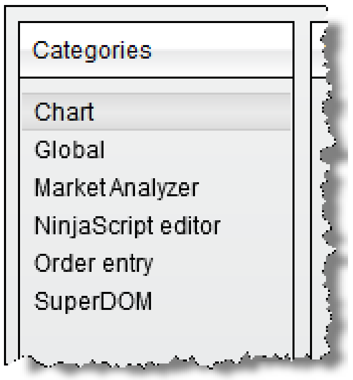
Removing a Hot Key

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

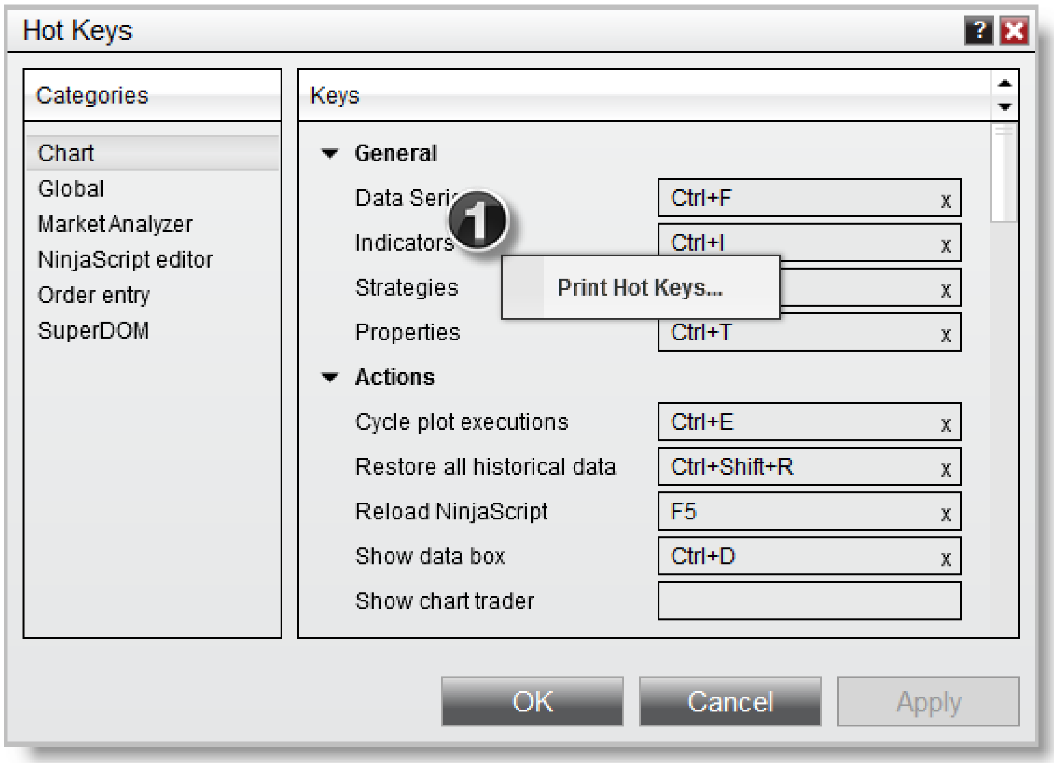
Hot Keys assigned under the Order Entry section are active whenever an order entry window is selected. Please see the Trading with Hot Keys section of the Help Guide for more information on this topic.

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.


Hot Keys

Categories	Keys	
Chart	**General**	
Global	Data Series	Ctrl+F
Market Analyzer	Indicators	Ctrl+I
NinjaScript editor	Strategies	
Order entry	Properties	Ctrl+T
SuperDOM	**Actions**	
	Cycle plot executions	Ctrl+E
	Restore all historical data	Ctrl+Shift+R
	Reload NinjaScript	F5
	Show data box	Ctrl+D
	Show chart trader	

11.12.2 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.

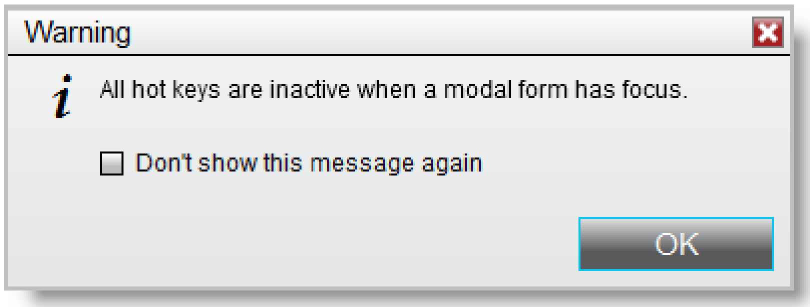
Active Window
You must always be aware of the current active window when using Hot Keys for order entry. Order entry Hot Keys are window sensitive and will only execute an action to the active order entry window. Please see the "*Understanding where Hot Key order entry is active*" section on this page 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.


Warning

All hot keys are inactive when a modal form has focus.

Don't show this message again

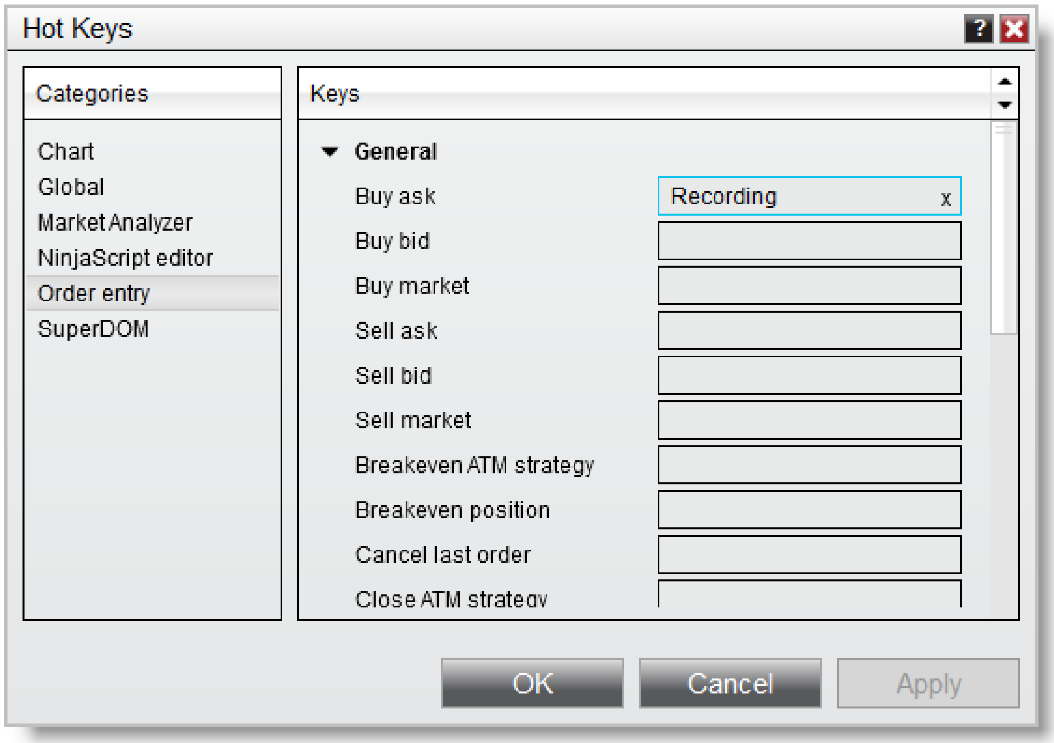
How to enable Hot Key order entry

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


Hot Keys

Categories	Keys	
Chart	**General**	
Global	Buy ask	Recording x
Market Analyzer	Buy bid	
NinjaScript editor	Buy market	
Order entry	Sell ask	
SuperDOM	Sell bid	
	Sell market	
	Breakeven ATM strategy	
	Breakeven position	
	Cancel last order	
	Close ATM strategy	

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.

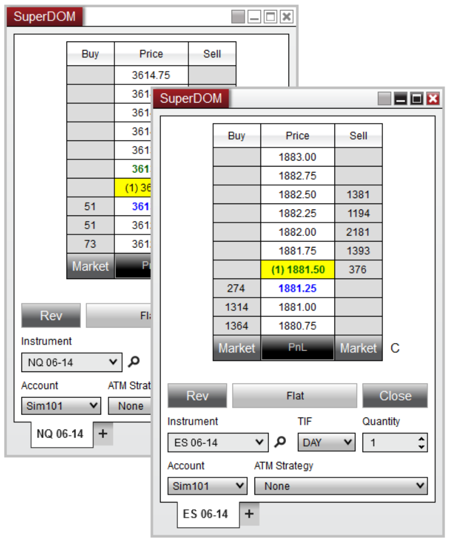
Order Entry Windows

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

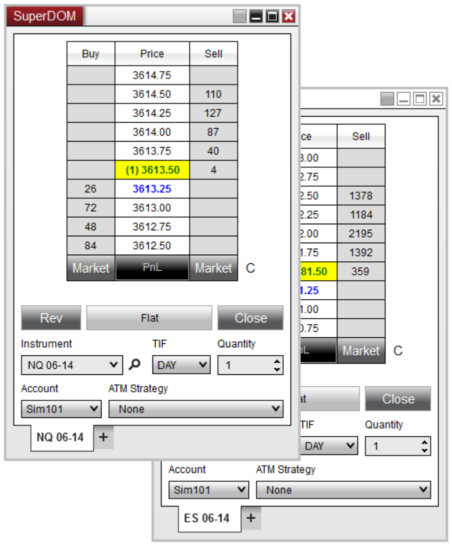
The active window is usually the window that was last clicked on and has the top most view. You will also notice that the active window's close button in the upper right hand corner is red compared to an inactive window that has a grey close button.

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.



Buy	Price	Sell
	1883.00	
	1882.75	
	1882.50	1381
	1882.25	1194
	1882.00	2181
	1881.75	1393
	(1) 1881.50	376
274	1881.25	
1314	1881.00	
1364	1880.75	

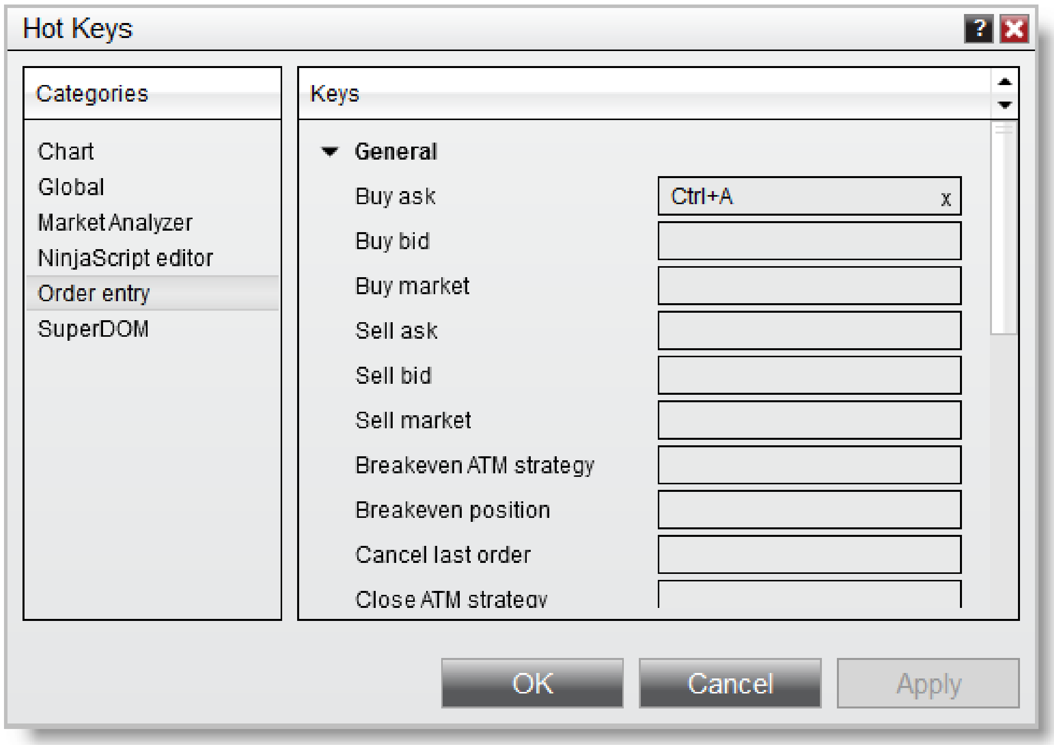
Instrument: ES 06-14 TIF: DAY Quantity: 1
Account: Sim101 ATM Strategy: None



Buy	Price	Sell
	3614.75	
	3614.50	110
	3614.25	127
	3614.00	87
	3613.75	40
	(1) 3613.50	4
26	3613.25	
72	3613.00	
48	3612.75	
84	3612.50	

Instrument: NQ 06-14 TIF: DAY Quantity: 1
Account: Sim101 ATM Strategy: None

Pre-defined order actions and definitions


Hot Keys

Categories
Chart
Global
Market Analyzer
NinjaScript editor
Order entry
SuperDOM

Keys	
General	
Buy ask	Ctrl+A
Buy bid	
Buy market	
Sell ask	
Sell bid	
Sell market	
Breakeven ATM strategy	
Breakeven position	
Cancel last order	
Close ATM strategy	

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
Break	Modifies your stop loss order to your break-even

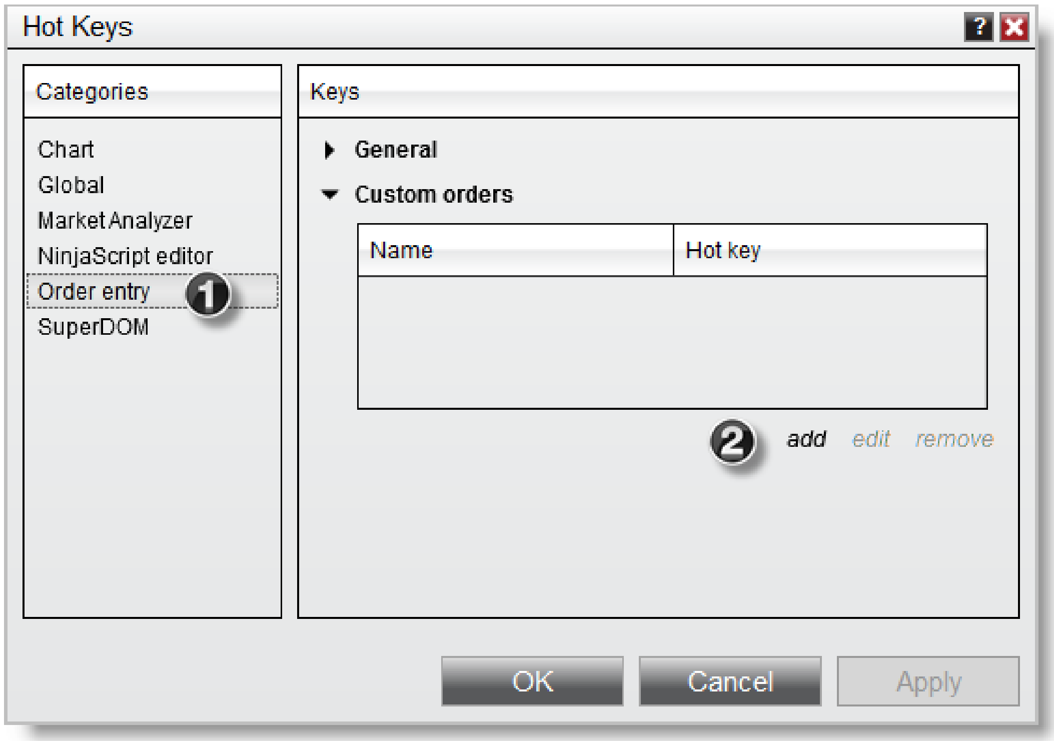
even ATM Strategy	price for the ATM Strategy on the active window
Breakeven Position	Modifies any stops to your break-even price for any open position.
Cancel last Order	Cancels the last submitted order
Close ATM Strategy	Cancels any pending orders and exits any open positions activated by an ATM Strategy
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.
Rever	Closes your open position and any related ATM

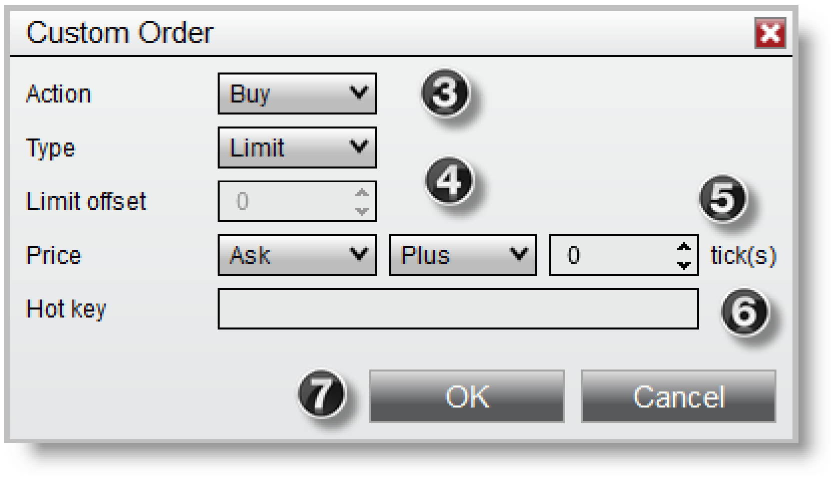
se	orders and submits a market order in the opposite direction to reverse your open position.

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"

11.13 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.

› Using the Hot List Analyzer
› Customizing the Hot List Analyzer
› Hot List Analyzer Properties

11.13.1 Using the Hot List Analyzer

▽ Understanding Hot Lists

What are hot lists?

Hot lists are a unique list of stocks which are constantly being monitor 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:

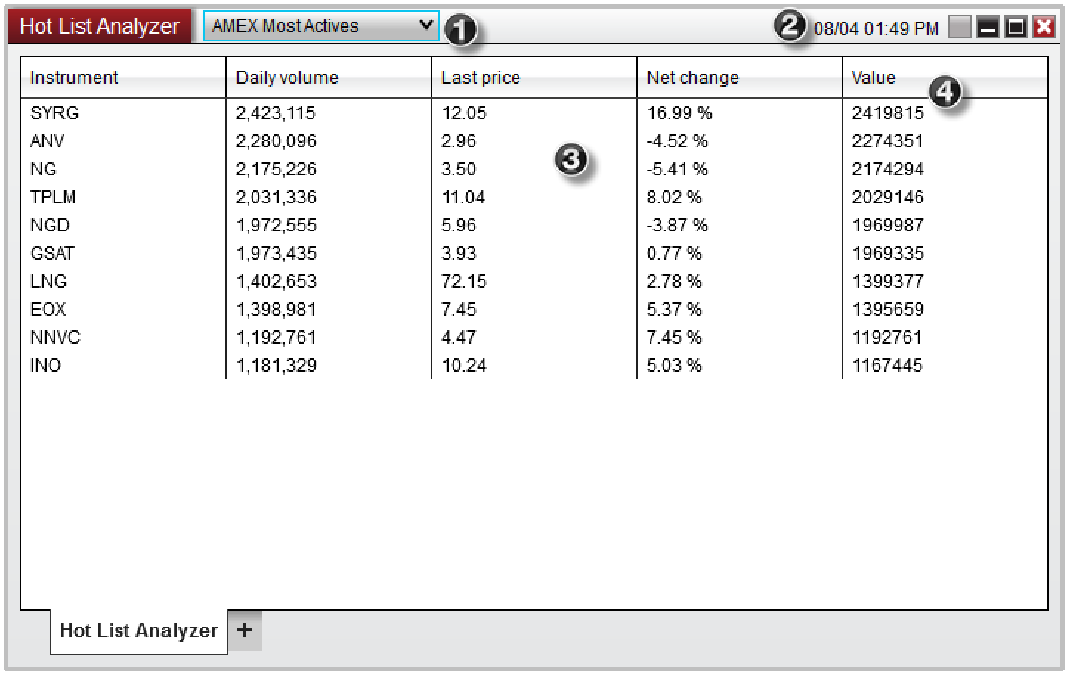
- Kinetick
- IQFeed
- Interactive Brokers

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

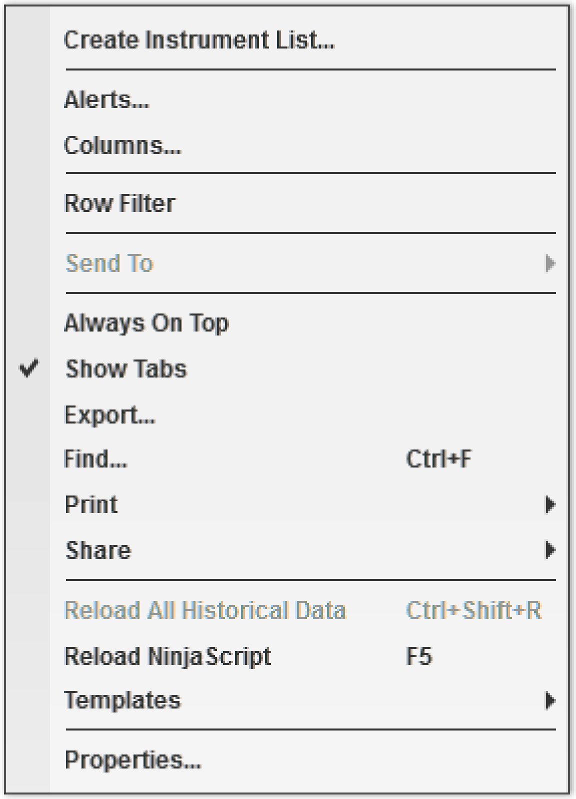


08/04 01:49 PM

Instrument	Daily volume	Last price	Net change	Value
SYRG	2,423,115	12.05	16.99 %	2419815
ANV	2,280,096	2.96	-4.52 %	2274351
NG	2,175,226	3.50	-5.41 %	2174294
TPLM	2,031,336	11.04	8.02 %	2029146
NGD	1,972,555	5.96	-3.87 %	1969987
GSAT	1,973,435	3.93	0.77 %	1969335
LNG	1,402,653	72.15	2.78 %	1399377
EOX	1,398,981	7.45	5.37 %	1395659
NNVC	1,192,761	4.47	7.45 %	1192761
INO	1,181,329	10.24	5.03 %	1167445

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.

11.13.2 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

11.13.3 Hot List Analyzer Properties

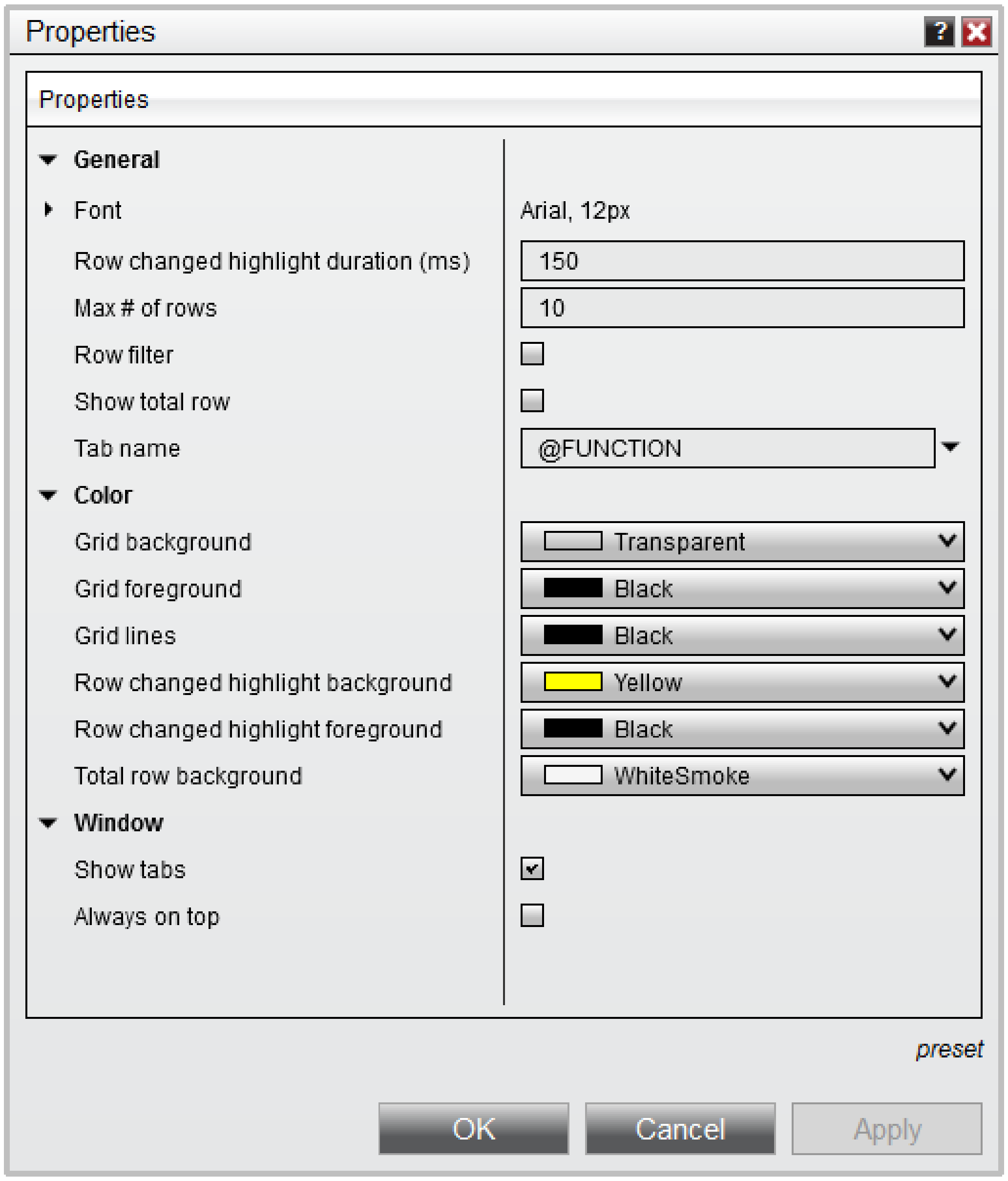
The Hot List Analyzer can be customized to your preferences in the Hot List Analyzer Properties window.

▽ 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



Properties

Properties

General	
Font	Arial, 12px
Row changed highlight duration (ms)	150
Max # of rows	10
Row filter	☐
Show total row	☐
Tab name	@FUNCTION
Color	
Grid background	Transparent
Grid foreground	Black
Grid lines	Black
Row changed highlight background	Yellow
Row changed highlight foreground	Black
Total row background	WhiteSmoke
Window	
Show tabs	✓
Always on top	☐

preset

Property Definitions

General	
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.

11.14 Instrument Lists

Instrument Lists Overview

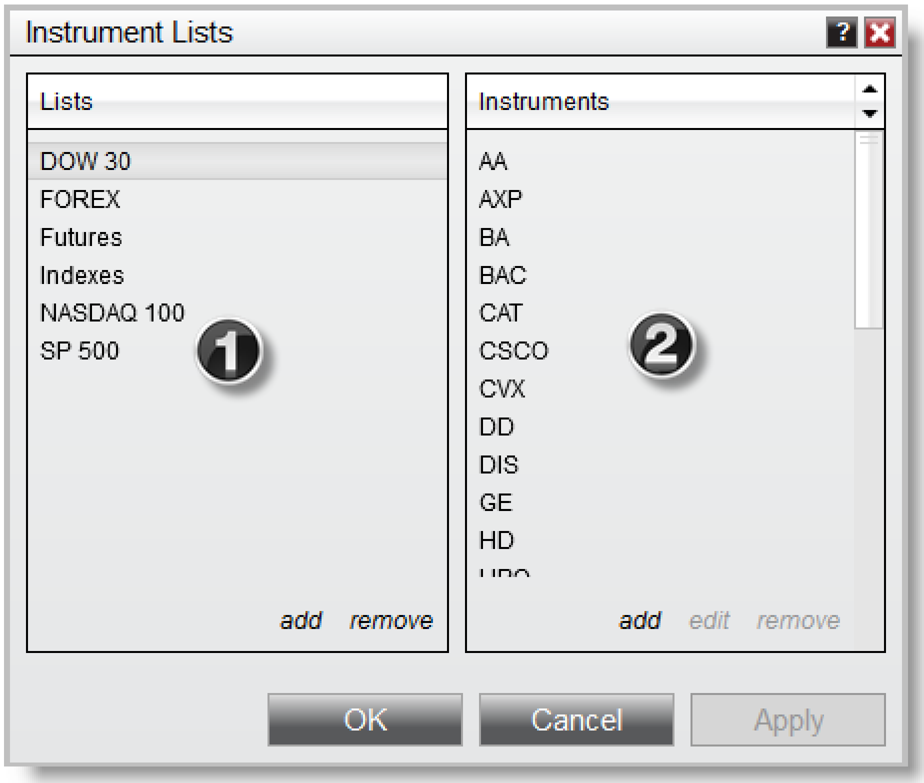
Instrument Lists can be configured by left mouse clicking on the **Tools** menu within the NinjaTrader Control Center and selecting the menu item **Instrument Lists**.
NinjaTrader supports grouping together instruments into easy to access lists. There are several uses for an instrument list:

- 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.

11.14.1 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**

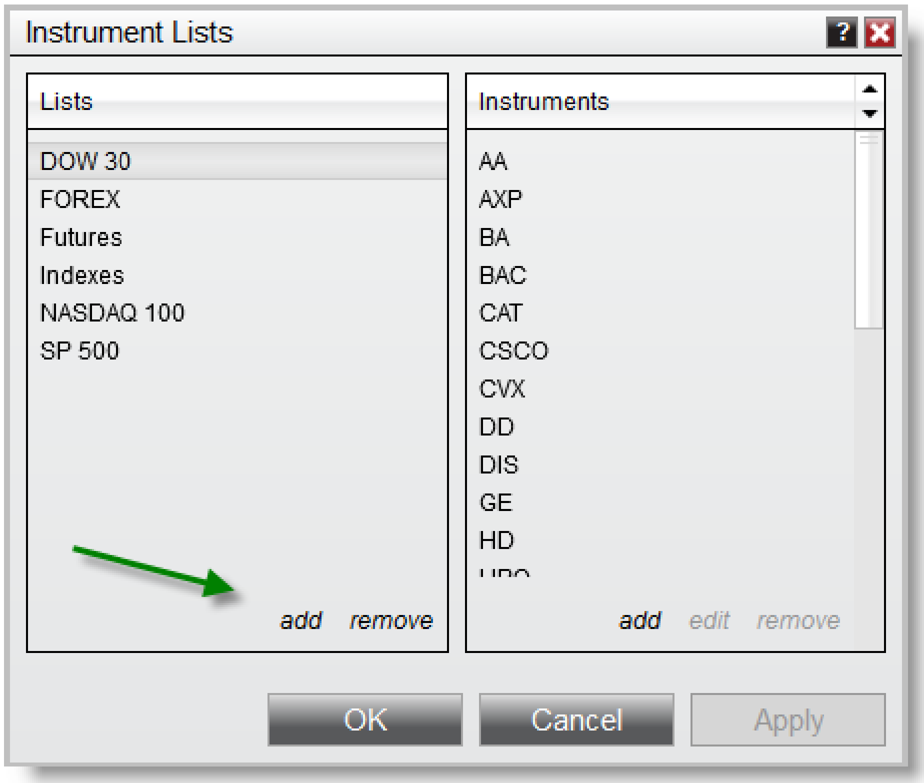
The **Instruments** section displays the selected lists instruments. Please see "*Adding or Removing instruments to a list*" section below for information on how to add and remove instruments to an instrument list.

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**

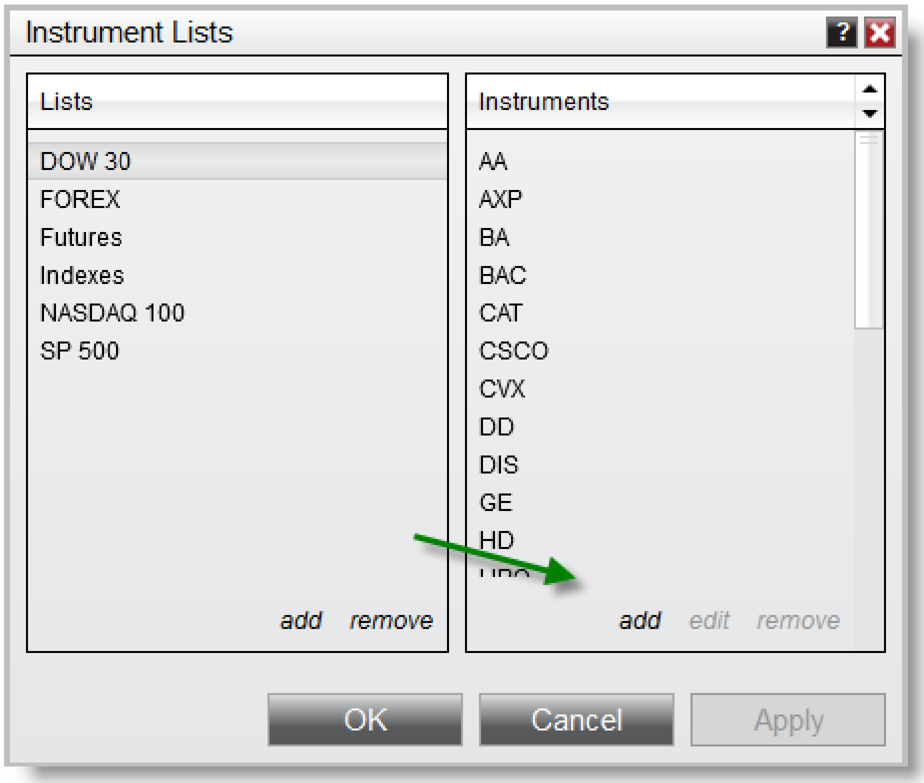
Note: A predefined instrument lists **cannot** be removed.

Adding or Removing instruments to a list

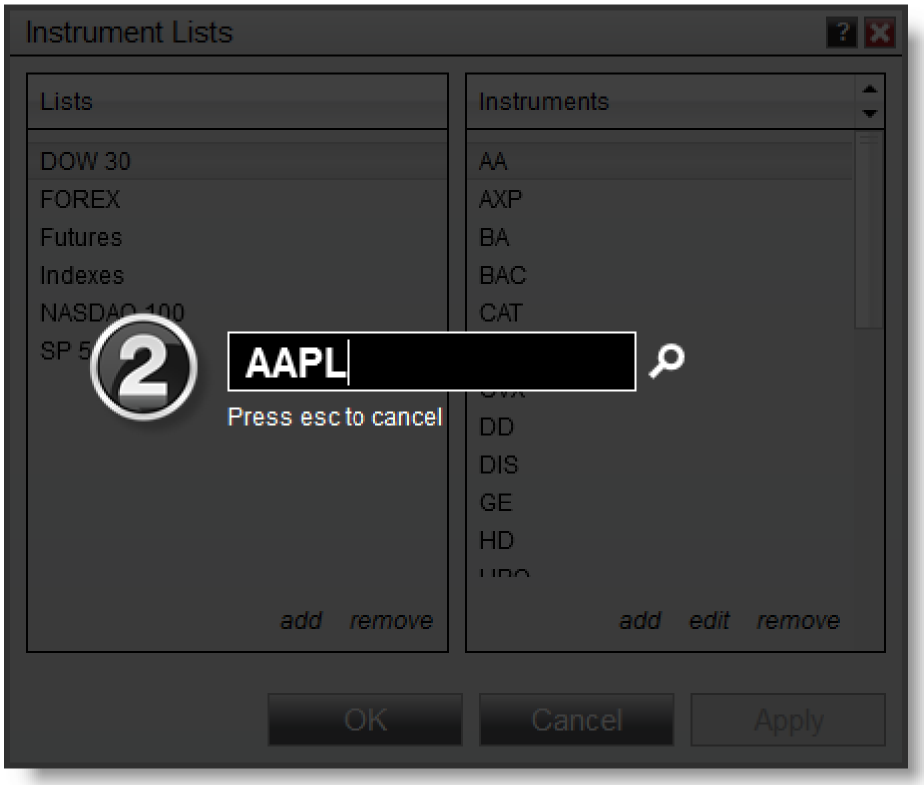
The collection of instruments that are associated to the selected instrument list are displayed in the "**Instruments**" section.

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:

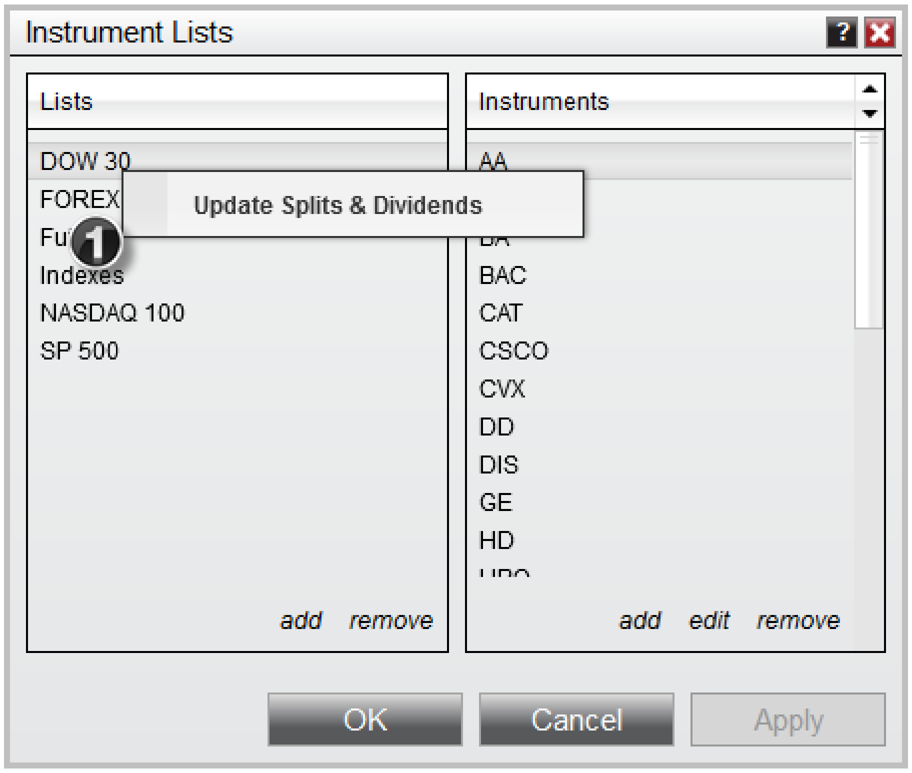
1. Left mouse click on the instrument you wish to remove from the instrument list in the right pane of the **Instrument Lists** window.
2. Press **remove**

11.14.2 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 enableNinjaTrader 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.

11.15 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.

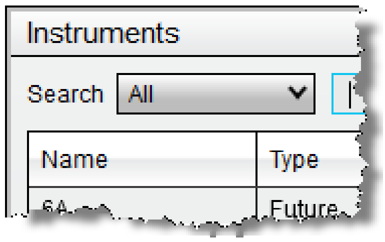
11.15.1 Searching for Instruments

NinjaTrader has a predefined database of commonly supported instruments that you can search through.

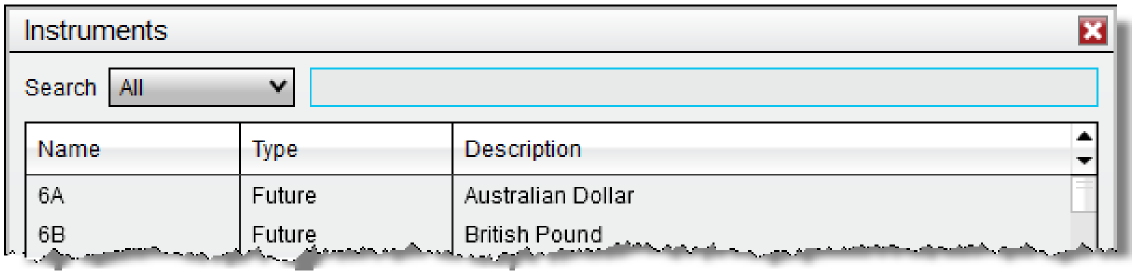
Searching Instruments

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.



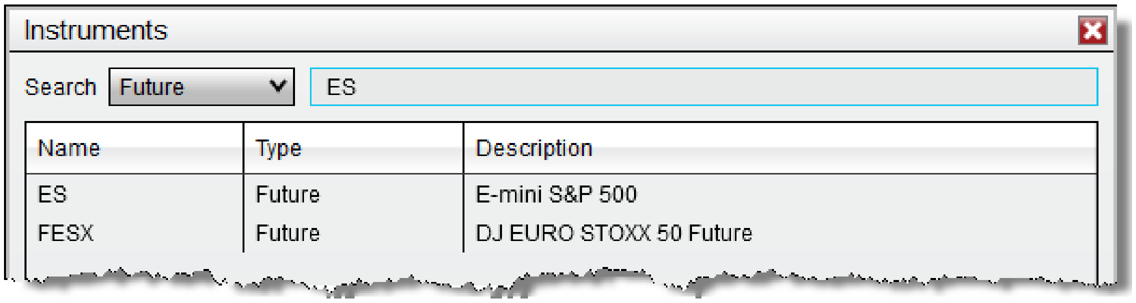
2. Enter any search parameters in either the **Search** field. Typing in capital letters will search for via the instrument name, typing in lowercase letters will search via the instrument description.



Name	Type	Description
6A	Future	Australian Dollar
6B	Future	British Pound

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.



Name	Type	Description
ES	Future	E-mini S&P 500
FESX	Future	DJ EURO STOXX 50 Future

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.

11.15.2 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.

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.

11.15.3 Editing Instruments

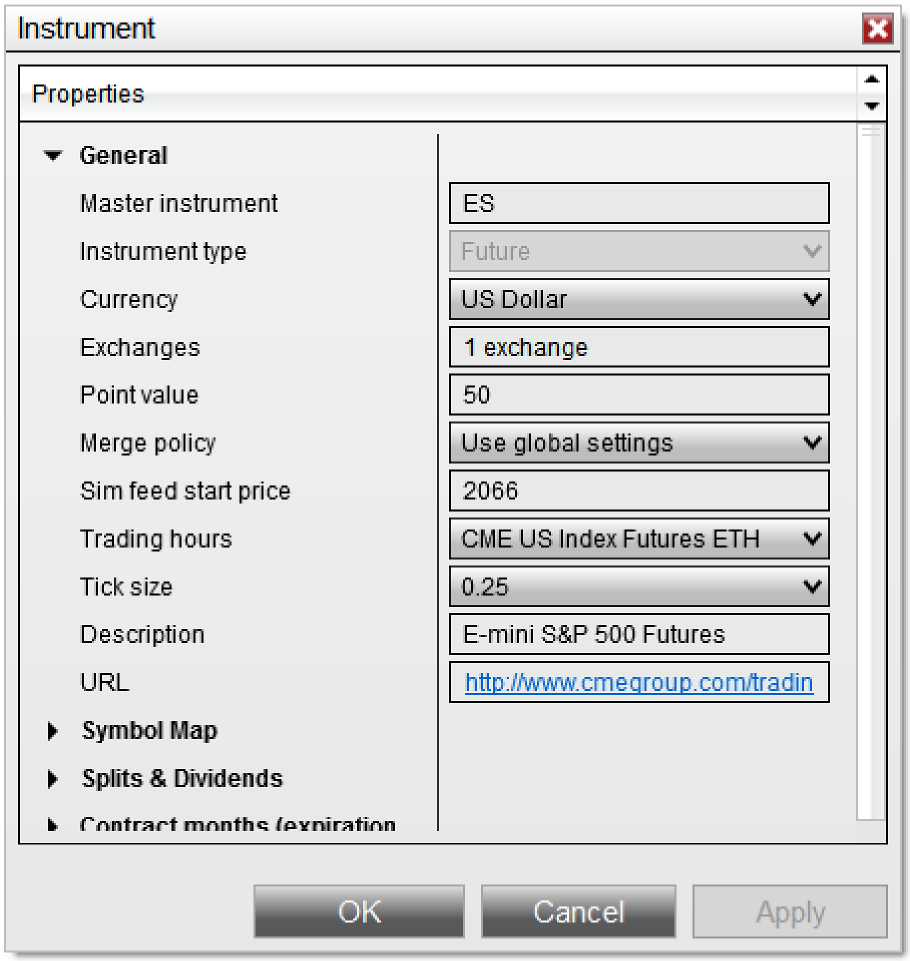
The **Instrument** window displays all parameters that define an instrument including symbol mappings to your connectivity provider and symbol level commission values. The editor allows you to change or add parameters to an instrument's profile. In general, instruments that are predefined in NinjaTrader 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.



- **General**
  - Master instrument: ES
  - Instrument type: Future
  - Currency: US Dollar
  - Exchanges: 1 exchange
  - Point value: 50
  - Merge policy: Use global settings
  - Sim feed start price: 2066
  - Trading hours: CME US Index Futures ETH
  - Tick size: 0.25
  - Description: E-mini S&P 500 Futures
  - URL: http://www.cmegroup.com/tradin
- **Symbol Map**
- **Splits & Dividends**
- **Contract months (expiration**

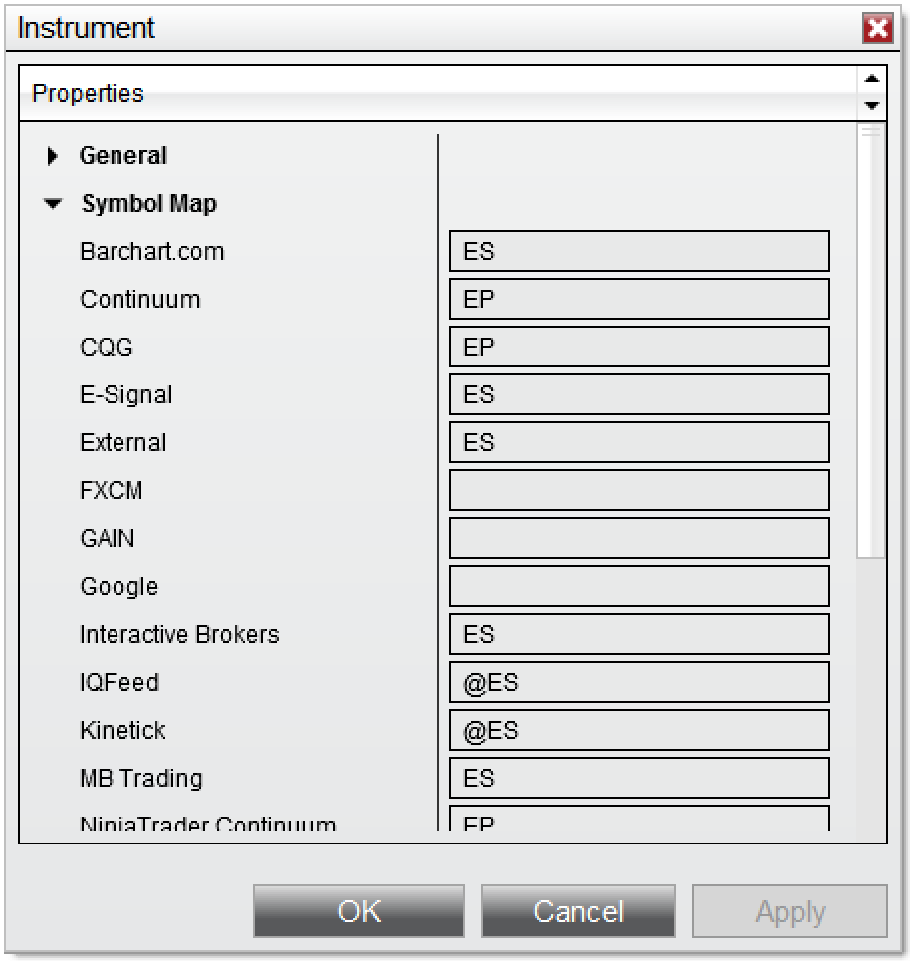
Master instrument	The NinjaTrader master name of the instrument
Instrument type	The instrument type (asset class)
Currency	The currency the instrument trades in
Exchanges	Click to select what exchanges the instrument trades on
Point value	The currency value of 1 point of movement for the instrument

Margin value	The margin required to trade 1 unit of the instrument for back testing and simulation trading
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
Descripti on	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.



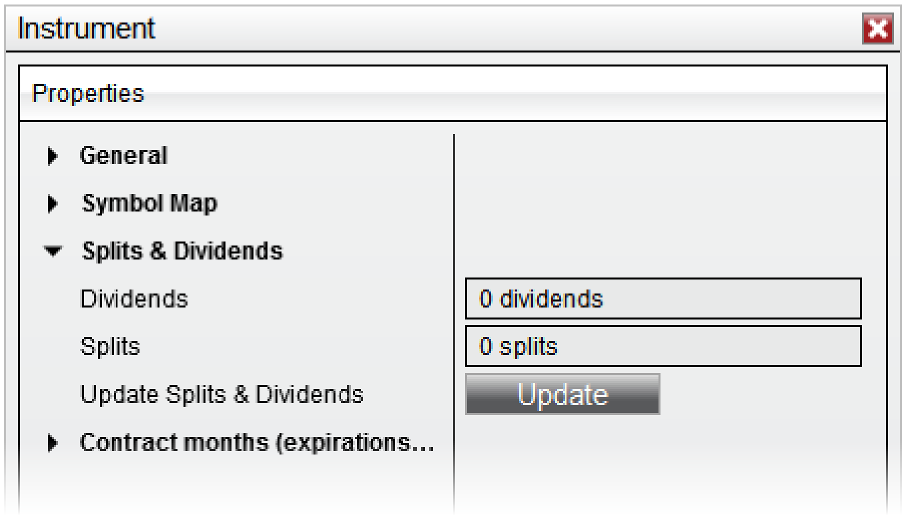
Barchart.com	ES
Continuum	EP
CQG	EP
E-Signal	ES
External	ES
FXCM	
GAIN	
Google	
Interactive Brokers	ES
IQFeed	@ES
Kinetick	@ES
MB Trading	ES
NinjaTrader Continuum	EP

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.

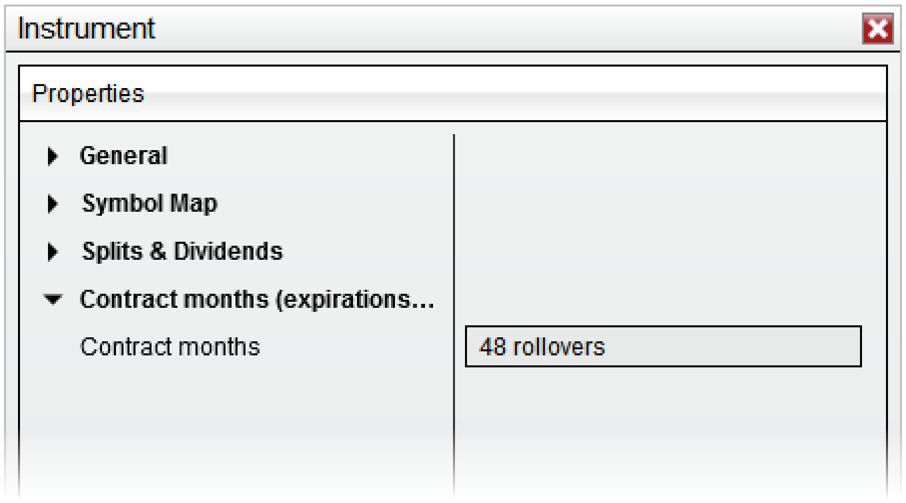


Dividends	0 dividends
Splits	0 splits
Update Splits & Dividends	Update

Understanding the Contract months section

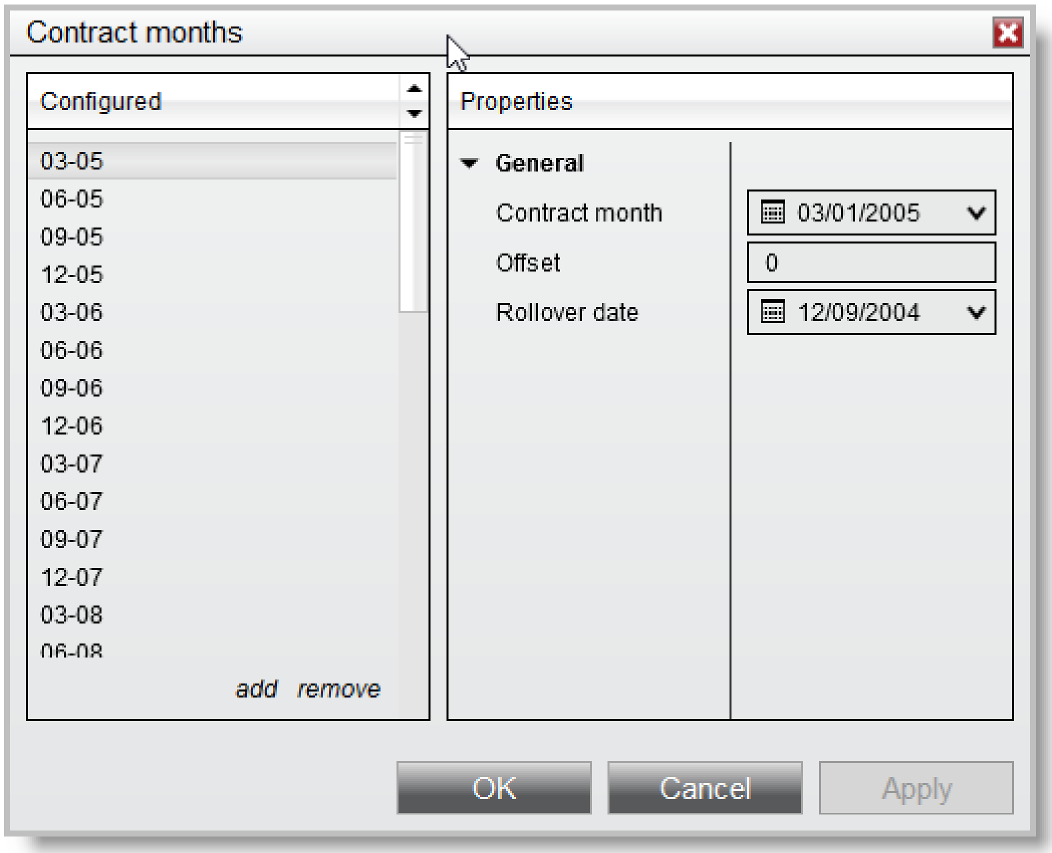
Contract Months

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.



Contract months	48 rollovers

You can open up the defined contract months by left mouse clicking in the **Contract months** field.


Contract months

Configured

03-05
06-05
09-05
12-05
03-06
06-06
09-06
12-06
03-07
06-07
09-07
12-07
03-08
06-08

add *remove*

Properties

General

Contract month	03/01/2005
Offset	0
Rollover date	12/09/2004

You can add and remove contract months by selecting the **add** and **remove** buttons in the bottom of the Configured section.

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:

- Use the old and new expiry's daily price data for calculations
    - If daily data does not exist, use minute data
    - If minute data does not exist, 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.

11.15.4 Rolling Over Futures Contracts

Batch Rollover

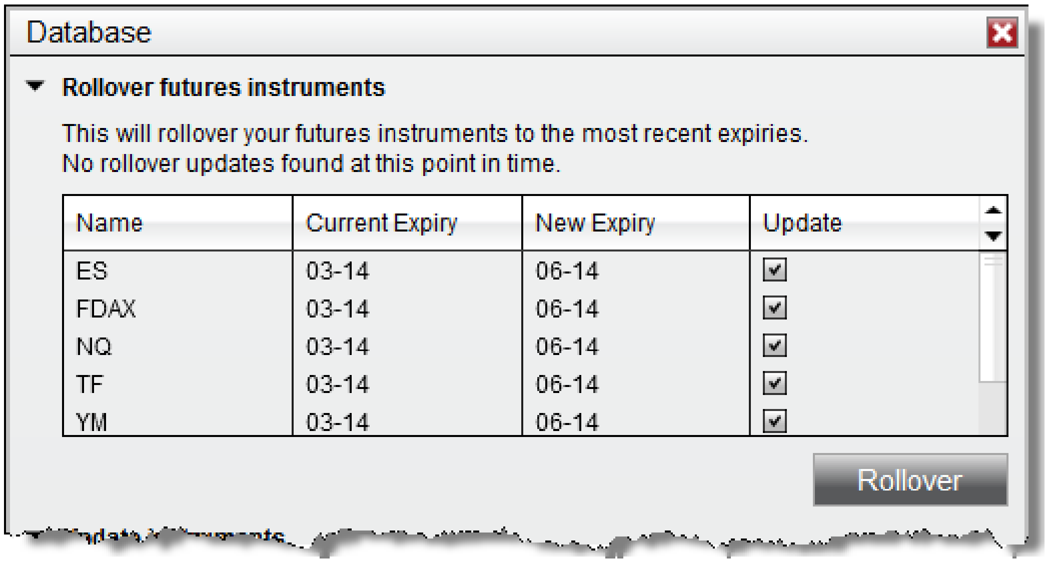
NinjaTrader allows batch rollover of the contract expiry of all instruments across all instrument lists and windows on all open workspaces. To perform this batch rollover please see the steps below.

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.



This will rollover your futures instruments to the most recent expiries.
No rollover updates found at this point in time.

Name	Current Expiry	New Expiry	Update
ES	03-14	06-14	✓
FDAX	03-14	06-14	✓
NQ	03-14	06-14	✓
TF	03-14	06-14	✓
YM	03-14	06-14	✓

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".

11.15.5 Adding Splits and Dividends

You can automatically update an instrument with historical split and dividend adjustment data from within the **Instrument** window. You can choose to update split and dividend 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 and Dividends via the instruments window

To automatically update an instrument with historical split and dividend 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 and dividend 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 & Dividends**.

NinjaTrader will now request historical splits and dividend information from your provider and populate the information in your local database.

Note: The **Update Splits & Dividends** menu item is only enabled when you are connected to one of the providers mentioned above.

Adding Splits and Dividends 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 & Dividends**.

11.15.6 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) and/or using the TradeStation's market data through the External Data Feed Connection for simulation.

Note: Mapping is NOT required for stocks or Forex symbols.

How to map an individual futures contract

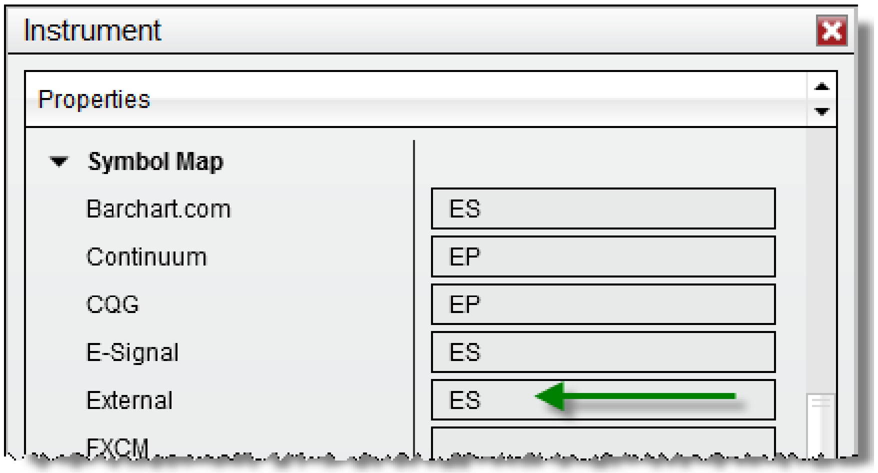
Mapping an Individual Futures Contract

To send either market data or 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. (Please see the Connection Guide for more information on how to connect to a TradeStation data connection.)

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.



Barchart.com	ES
Continuum	EP
CQG	EP
E-Signal	ES
External	ES
FXCM	

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:

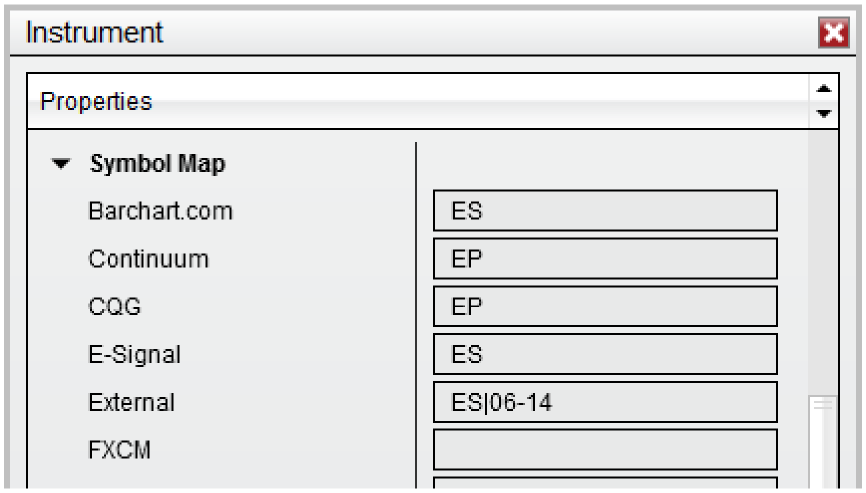
- Automatically map to the next closest expiration date
- User defined contract mapping

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 or want to use the NTExternalFeed strategy to drive NinjaTrader Simulation Edition with a TradeStation continuous contract, 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.



Symbol Map

Barchart.com	ES
Continuum	EP
CQG	EP
E-Signal	ES
External	ES|06-14
FXCM	

Automated Trading Interface - Orders generated for "@ES" will now be routed to the NinjaTrader "ES 06-14" contract.

NTExternalFeed strategy - Data from your "@ES" chart will be sent to the NinjaTrader "ES 06-14" contract.

Please remember to change this when the contract rolls over.

11.15.7 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...**

❶Press 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

❷The symbols for import are listed in the editor

❸Select the exchange the instruments are traded on

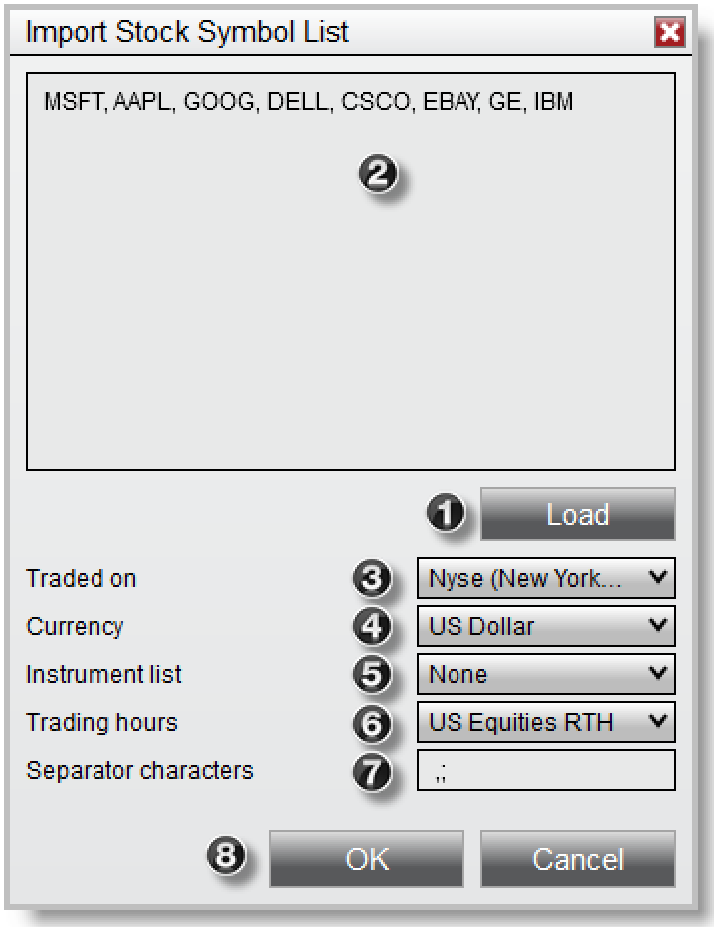
❹Select the currency the instruments are traded in

❺Optionally add the instruments to an Instrument List (optionally create a new one by selecting "**New**" in the combo box.)

❻Select a Session template for the instruments

❼Enter any user defined separator characters

❽Press the **OK** button to import



MSFT, AAPL, GOOG, DELL, CSCO, EBAY, GE, IBM

Traded on: Nyse (New York...
Currency: US Dollar
Instrument list: None
Trading hours: US Equities RTH
Separator characters: ,;

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.

11.16 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.

- Using the Level II Window
- Level II Properties
- Window Linking

11.16.1 Using the Level II Window

Play Video

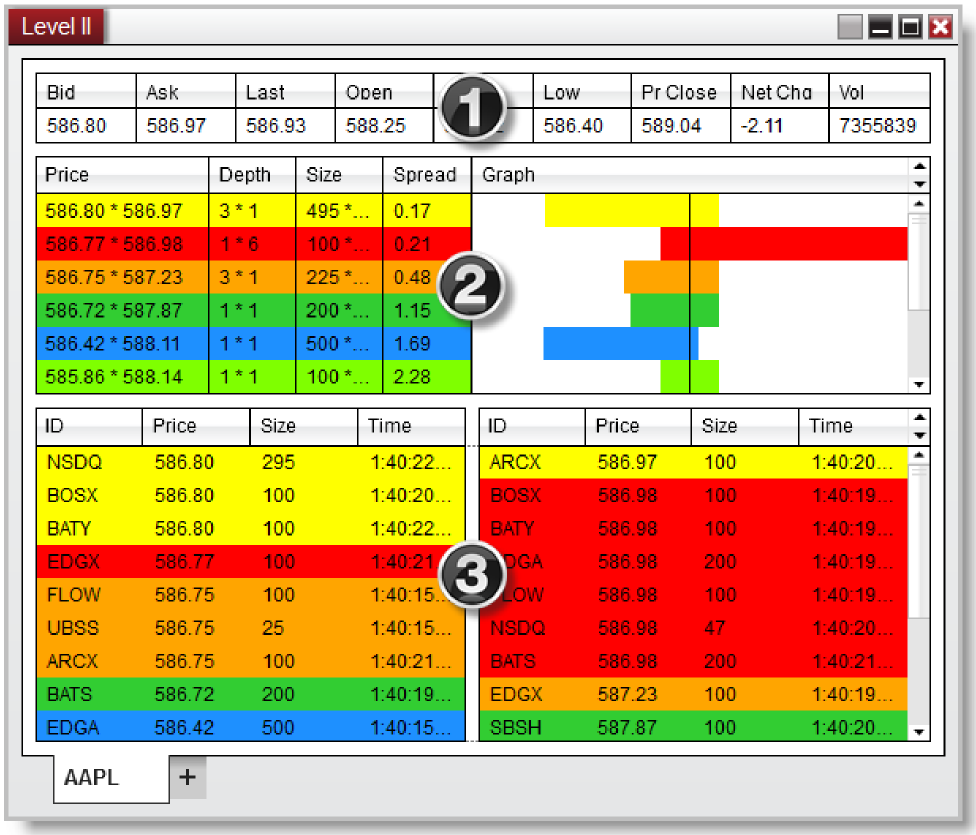
Selecting an Instrument

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



Bid	Ask	Last	Open		Low	Pr Close	Net Chg	Vol
586.80	586.97	586.93	588.25	[illegible]	586.40	589.04	-2.11	7355839

Price	Depth	Size	Spread	Graph
586.80 * 586.97	3 * 1	495 *...	0.17	
586.77 * 586.98	1 * 6	100 *...	0.21	
586.75 * 587.23	3 * 1	225 *...	0.48	
586.72 * 587.87	1 * 1	200 *...	1.15	
586.42 * 588.11	1 * 1	500 *...	1.69	
585.86 * 588.14	1 * 1	100 *...	2.28	

ID	Price	Size	Time
NSDQ	586.80	295	1:40:22...
BOSX	586.80	100	1:40:20...
BATY	586.80	100	1:40:22...
EDGX	586.77	100	1:40:21
FLOW	586.75	100	1:40:15...
UBSS	586.75	25	1:40:15...
ARCX	586.75	100	1:40:21...
BATS	586.72	200	1:40:19...
EDGA	586.42	500	1:40:15...

ID	Price	Size	Time
ARCX	586.97	100	1:40:20...
BOSX	586.98	100	1:40:19...
BATY	586.98	100	1:40:19...
DGA	586.98	200	1:40:19...
LOW	586.98	100	1:40:19...
NSDQ	586.98	47	1:40:20...
BATS	586.98	200	1:40:21...
EDGX	587.23	100	1:40:19...
SBSH	587.87	100	1:40:20...

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**.

❷ 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**.

❸ Details

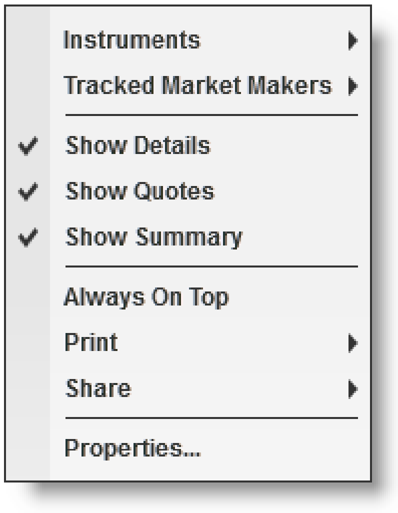
The Details section displays bid data on the left and ask data on the right.

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.

11.16.2 Level II Properties

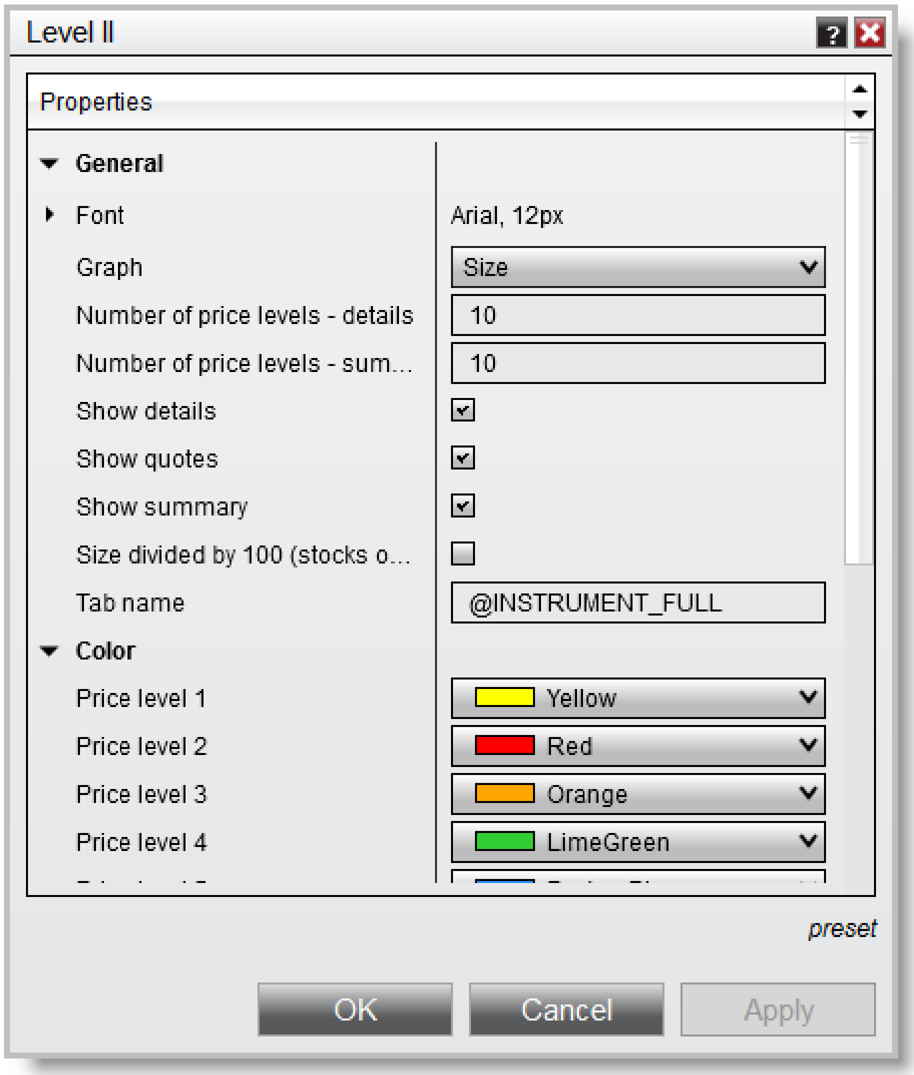
The Level II window can be customized through the **Level II Properties** window.

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:


Level II

Properties

General
- Font: Arial, 12px
- Graph: Size
- Number of price levels - details: 10
- Number of price levels - sum...: 10
- Show details: ✓
- Show quotes: ✓
- Show summary: ✓
- Size divided by 100 (stocks o...: ☐
- Tab name: @INSTRUMENT_FULL

Color
- Price level 1: 🟨 Yellow
- Price level 2: 🟥 Red
- Price level 3: 🟧 Orange
- Price level 4: 🟩 LimeGreen

Property Definitions

General	
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 -	Sets the number of visible

detailed	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.

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.

11.16.3 Window Linking

Please see the Window Linking section of the Help Guide for more information on linking the **Level II** window.

11.17 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

11.17.1 Creating a Market Analyzer Window

Play Video

▽ Understanding the Market Analyzer display

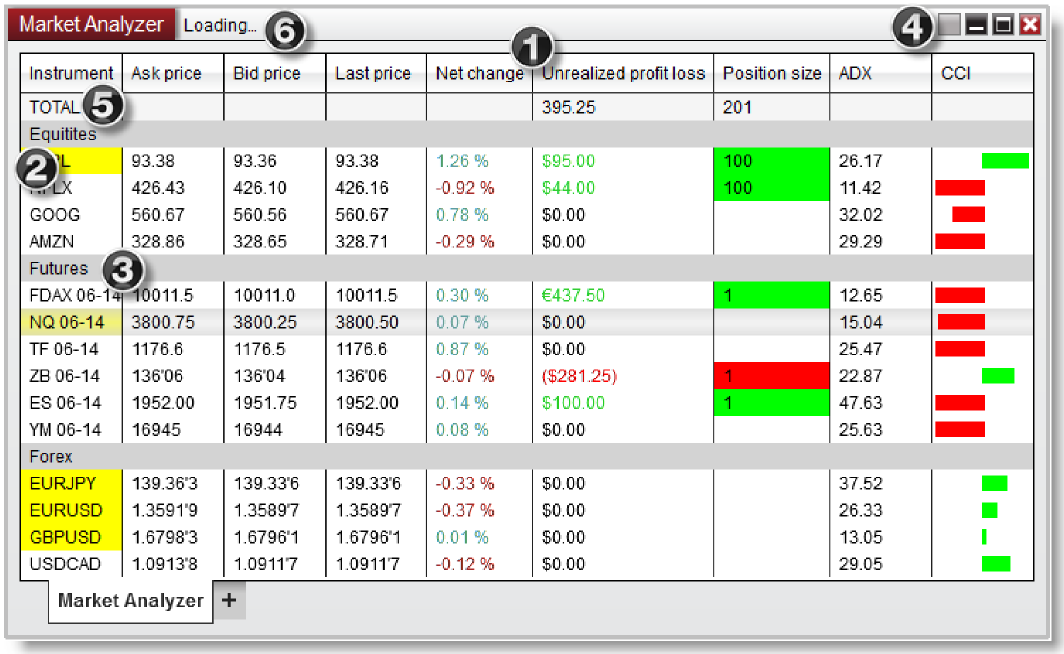
Market Analyzer Display Overview

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

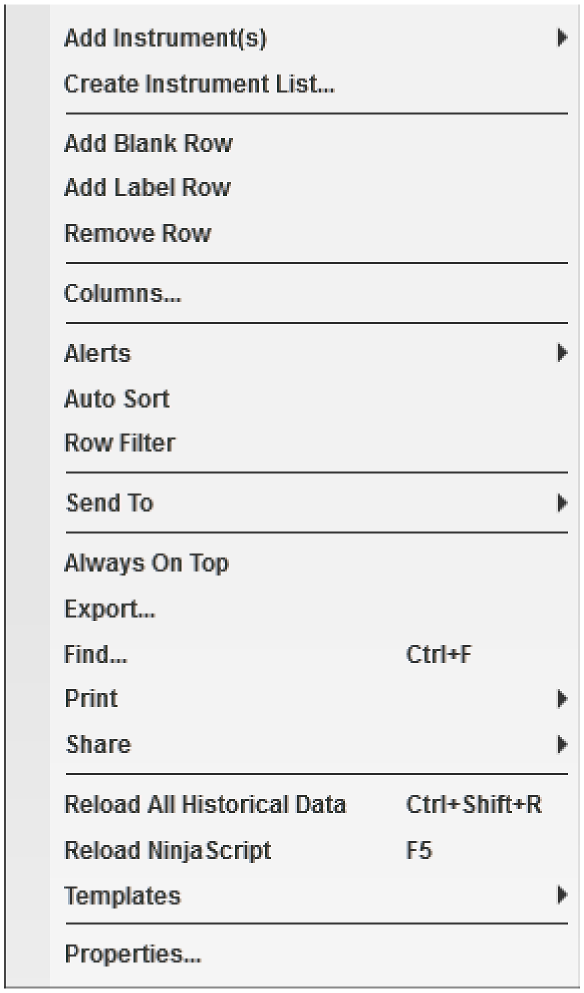
3. Label row	Displays a user defined label row.
4. Link button	Window linking links windows to the same instrument and can be applied to many NinjaTrader windows.
5. Total row	Displays the sum of all rows for a specific column. Can be applied in the Market Analyzer Properties window.
6. Loading dialog	Display a message indicating when an indicator or historical data is being loaded into the Market Analyzer



Instrument	Ask price	Bid price	Last price	Net change	Unrealized profit loss	Position size	ADX	CCI
TOTAL					395.25	201		
Equitites								
L	93.38	93.36	93.38	1.26 %	$95.00	100	26.17	🟩
LX	426.43	426.10	426.16	-0.92 %	$44.00	100	11.42	🟥
GOOG	560.67	560.56	560.67	0.78 %	$0.00		32.02	🟥
AMZN	328.86	328.65	328.71	-0.29 %	$0.00		29.29	🟥
Futures								
FDAX 06-14	10011.5	10011.0	10011.5	0.30 %	€437.50	1	12.65	🟥
NQ 06-14	3800.75	3800.25	3800.50	0.07 %	$0.00		15.04	🟥
TF 06-14	1176.6	1176.5	1176.6	0.87 %	$0.00		25.47	🟥
ZB 06-14	136'06	136'04	136'06	-0.07 %	($281.25)	1	22.87	🟩
ES 06-14	1952.00	1951.75	1952.00	0.14 %	$100.00	1	47.63	🟥
YM 06-14	16945	16944	16945	0.08 %	$0.00		25.63	🟥
Forex								
EURJPY	139.36'3	139.33'6	139.33'6	-0.33 %	$0.00		37.52	🟩
EURUSD	1.3591'9	1.3589'7	1.3589'7	-0.37 %	$0.00		26.33	🟩
GBPUSD	1.6798'3	1.6796'1	1.6796'1	0.01 %	$0.00		13.05	🟩
USDCAD	1.0913'8	1.0911'7	1.0911'7	-0.12 %	$0.00		29.05	🟩

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

11.17.2 Working with Instrument Rows

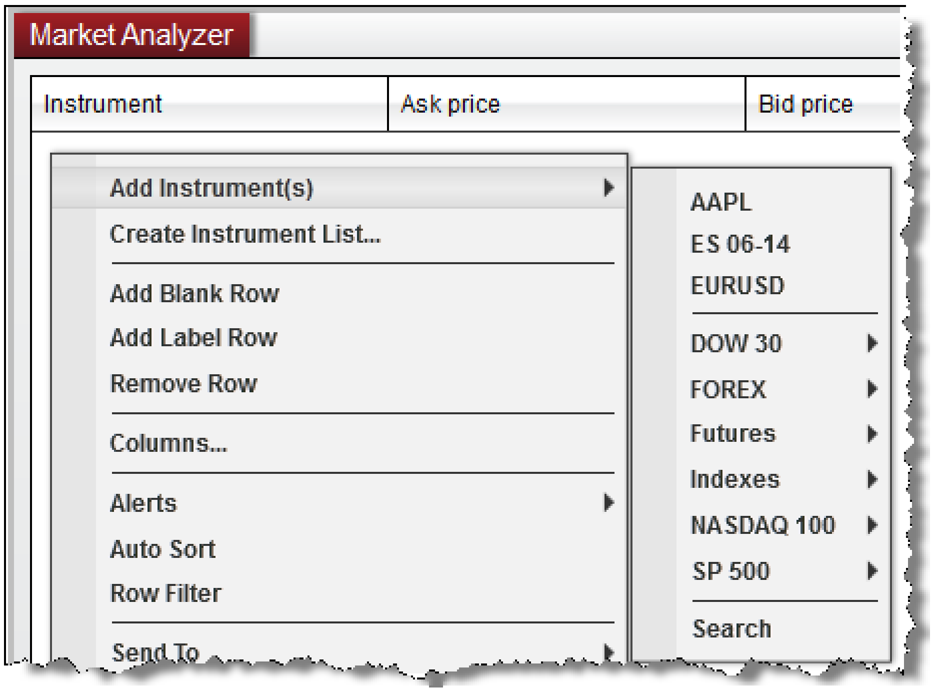
The Market Analyzer window allows you to display a variety of real-time quotes, indicator values, and position information on multiple instruments. You can add, remove, and organize individual instrument rows, Instrument Lists, Label rows, Blank rows, and a Total row with the instructions listed below.

▽ How to add instruments

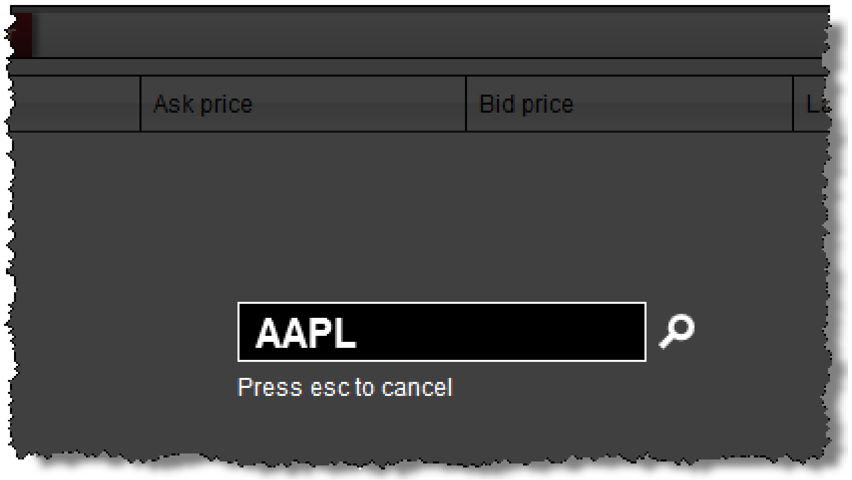
Adding an Instrument

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.

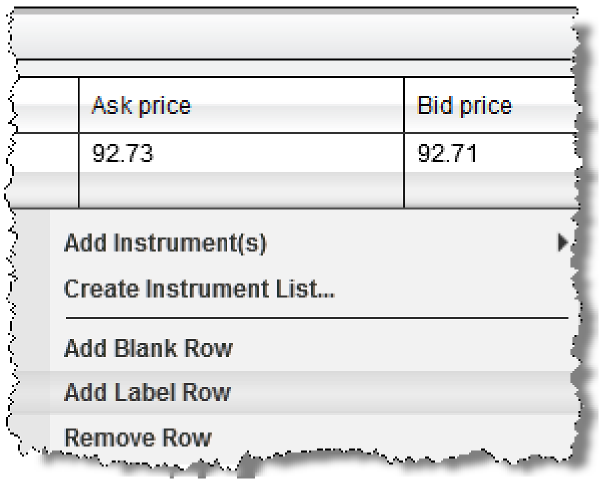
You will now be able to access this list from other features of NinjaTrader using the Instrument Selector. You can further edit this list by using the Instrument Lists window

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.

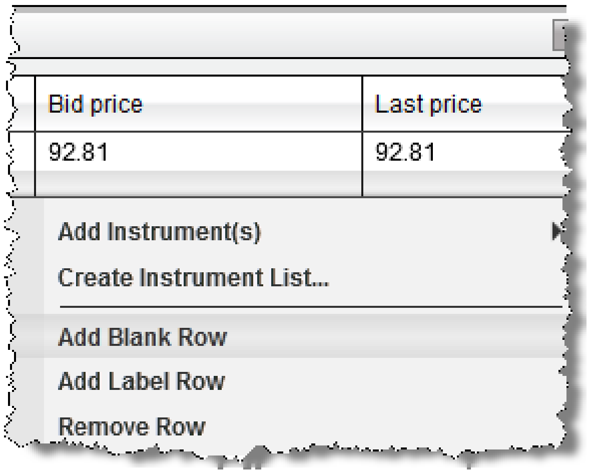
Aligning a Label Row

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.



Bid price	Last price
92.81	92.81

How to move Instrument, Label and Blank rows

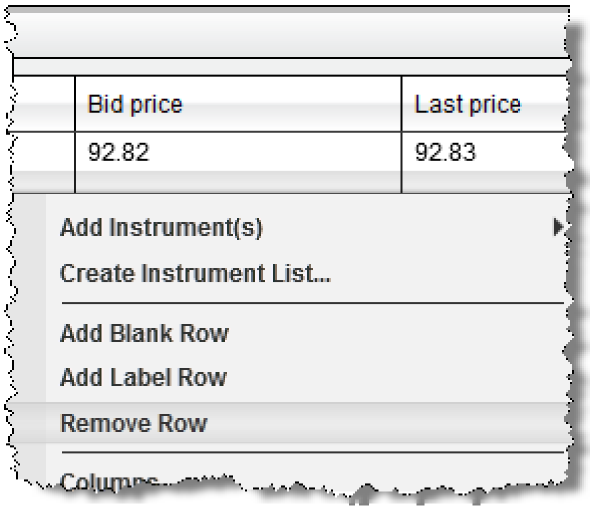
Moving Rows in the Market Analyzer Window

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

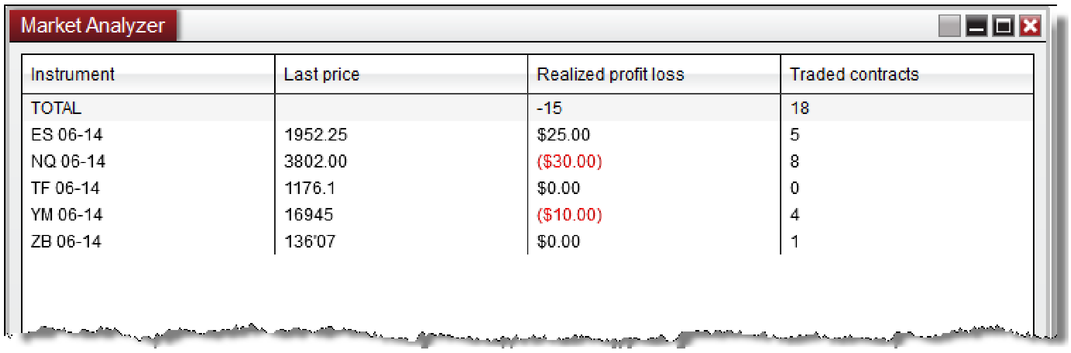
Removing Instrument, Label and Blank Rows

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.

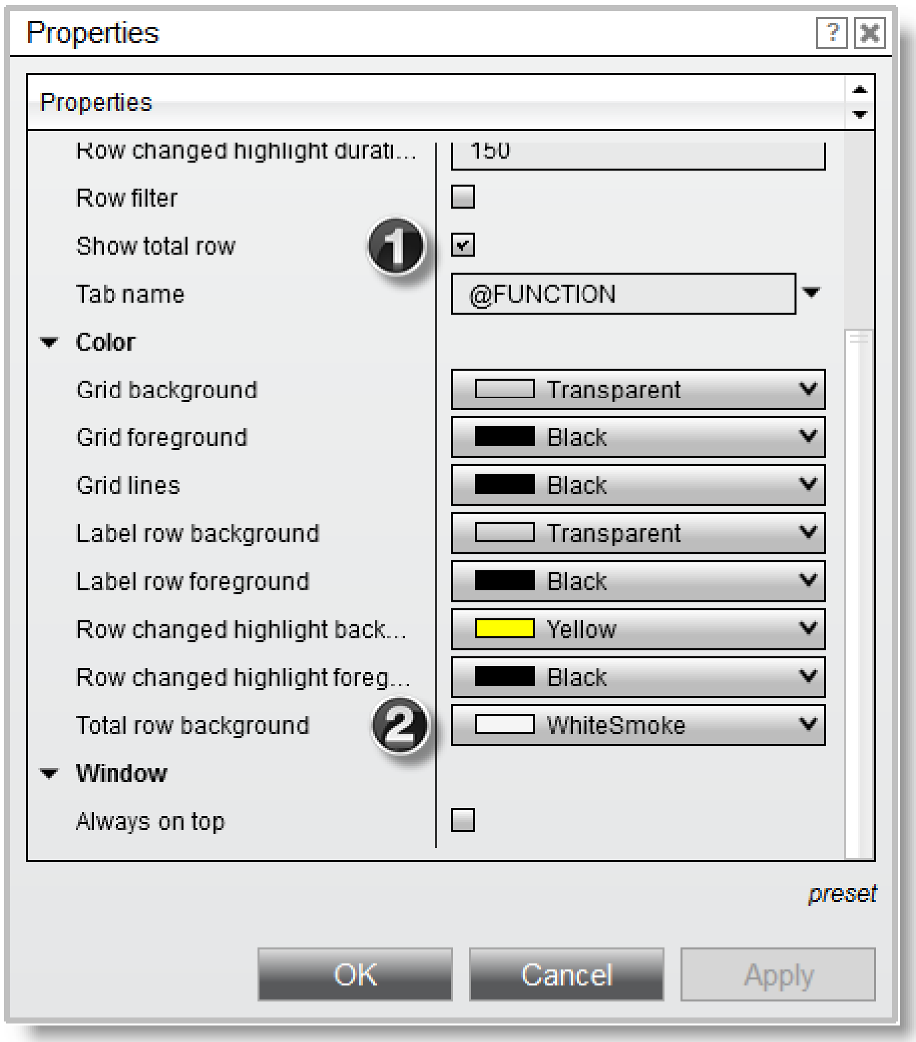


Instrument	Last price	Realized profit loss	Traded contracts
TOTAL		-15	18
ES 06-14	1952.25	$25.00	5
NQ 06-14	3802.00	($30.00)	8
TF 06-14	1176.1	$0.00	0
YM 06-14	16945	($10.00)	4
ZB 06-14	136'07	$0.00	1

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:

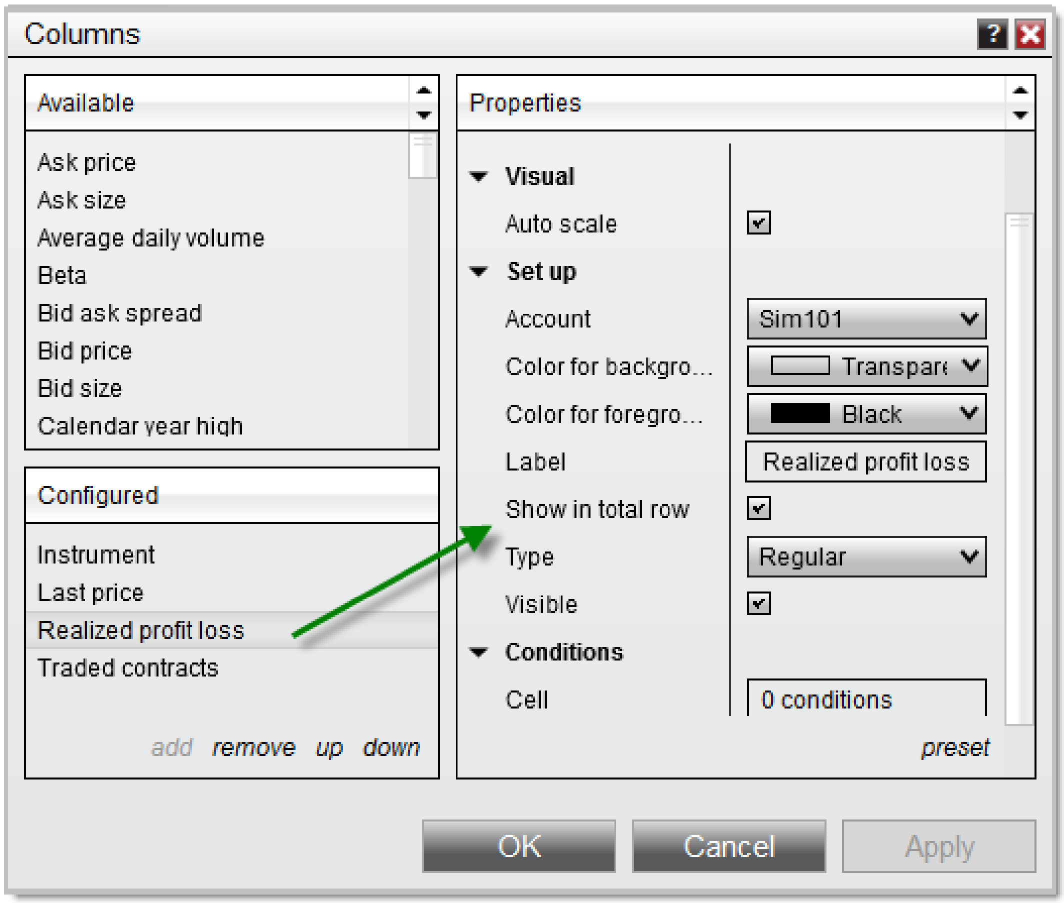
1. Press down on your right mouse button in the Market Analyzer window and select the menu **Properties**.
2. In the Properties menu scroll down to the ❶ Total Row section and ***check*** the box to enable. You can also choose to customize the color of this row with the ❷ **Total row background** property.
3. Press the Apply button to apply the changes or press the OK button to apply the changes and exit the Properties menu.



Properties

Properties	
Row changed highlight durati...	150
Row filter	☐
Show total row	✓
Tab name	@FUNCTION
Color	
Grid background	Transparent
Grid foreground	Black
Grid lines	Black
Label row background	Transparent
Label row foreground	Black
Row changed highlight back...	Yellow
Row changed highlight foreg...	Black
Total row background	WhiteSmoke
Window	
Always on top	☐

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.


Columns

Available
Ask price
Ask size
Average daily volume
Beta
Bid ask spread
Bid price
Bid size
Calendar year high

Configured
Instrument
Last price
Realized profit loss
Traded contracts

add remove up down

Properties	
Visual	
Auto scale	✓
Set up	
Account	Sim101
Color for backgro...	Transpare
Color for foregro...	Black
Label	Realized profit loss
Show in total row	✓
Type	Regular
Visible	✓
Conditions	
Cell	0 conditions

preset

OK Cancel Apply

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.

11.17.3 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.

Understanding the Columns window

The Columns window is used to add, remove, and edit columns within the Market Analyzer window.

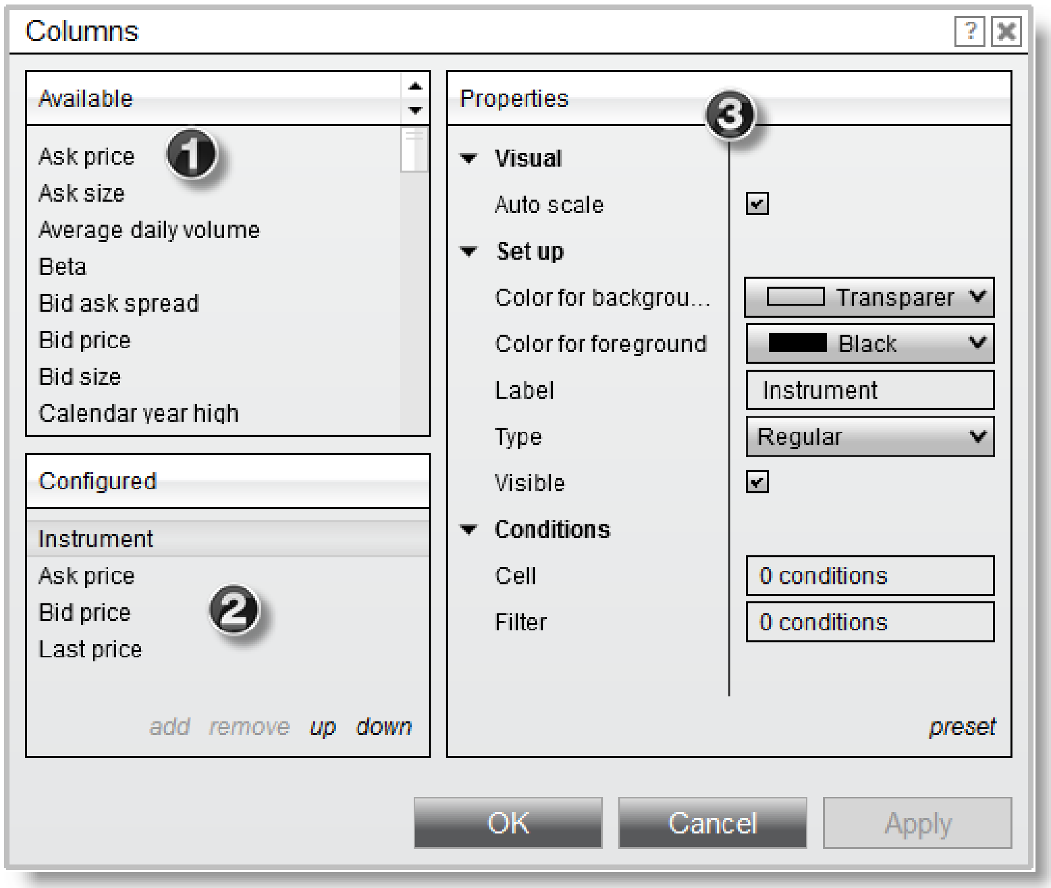
Accessing the Columns Window

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



Columns

Available

- Ask price
- Ask size
- Average daily volume
- Beta
- Bid ask spread
- Bid price
- Bid size
- Calendar year high

Configured

- Instrument
- Ask price
- Bid price
- Last price

Properties

Visual	
Auto scale	✓
Set up	
Color for backgrou...	Transparer
Color for foreground	Black
Label	Instrument
Type	Regular
Visible	✓
Conditions	
Cell	0 conditions
Filter	0 conditions

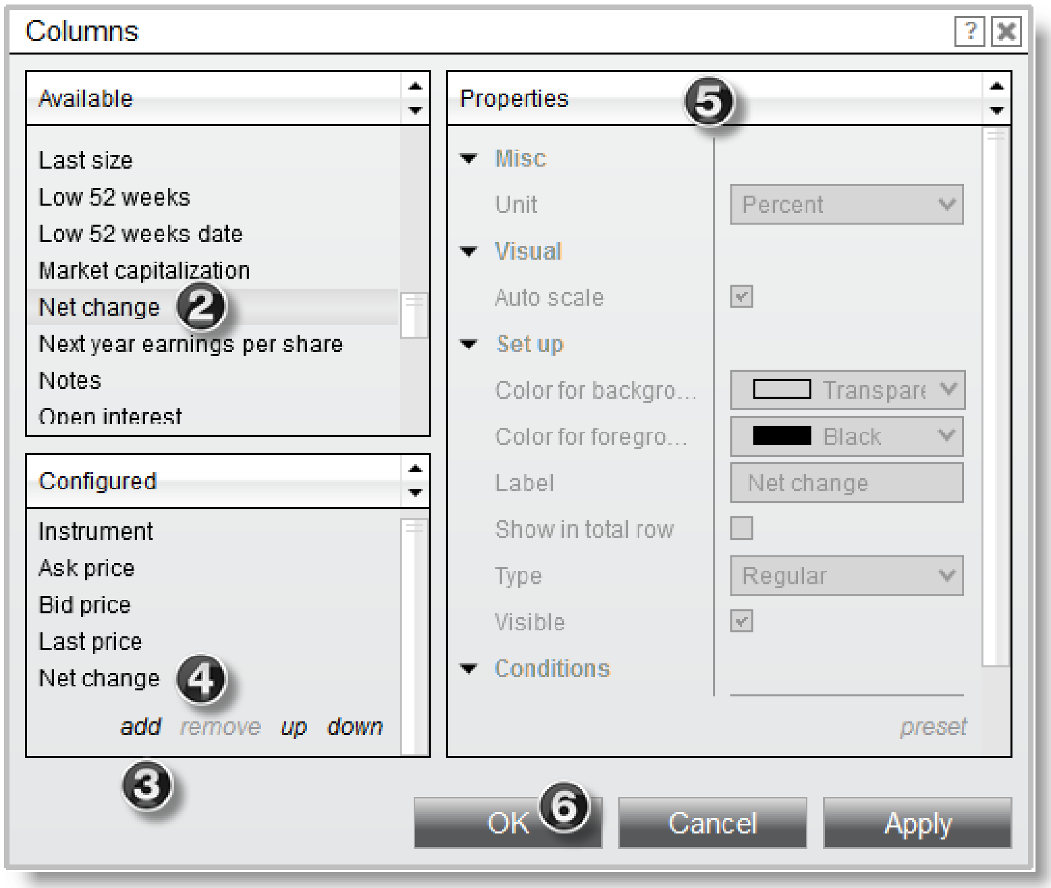
How to add columns

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

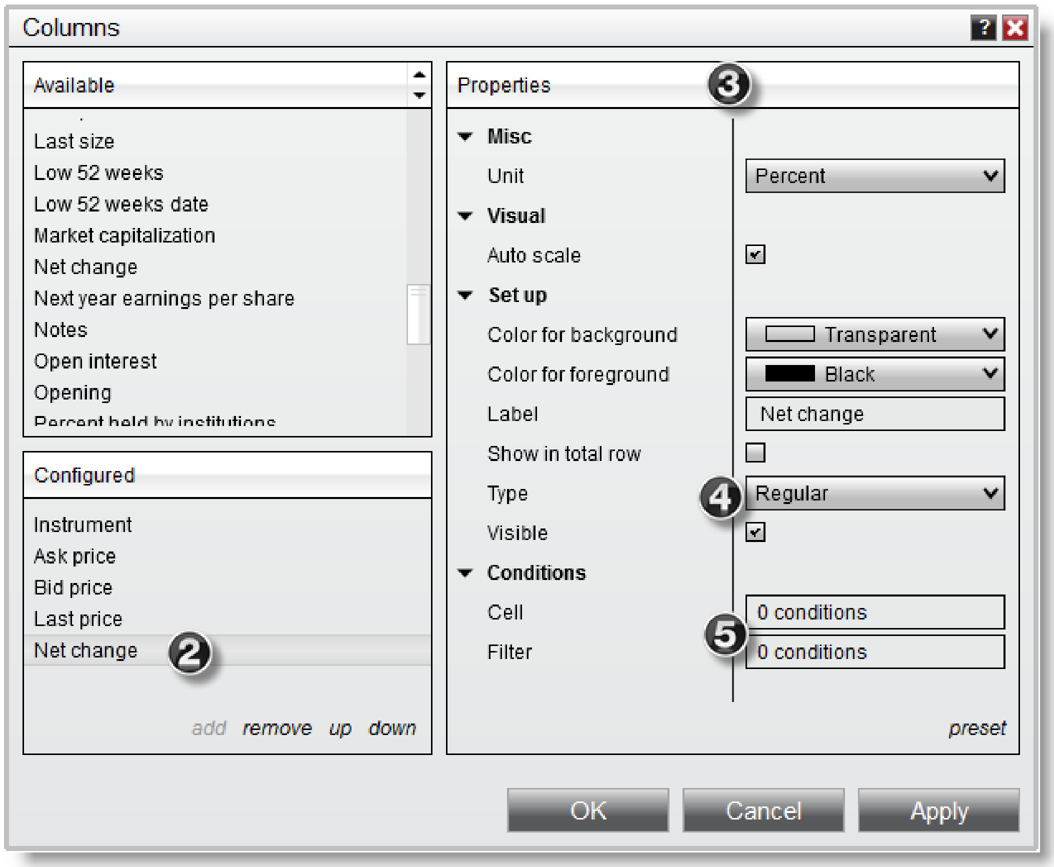
How to customize columns

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

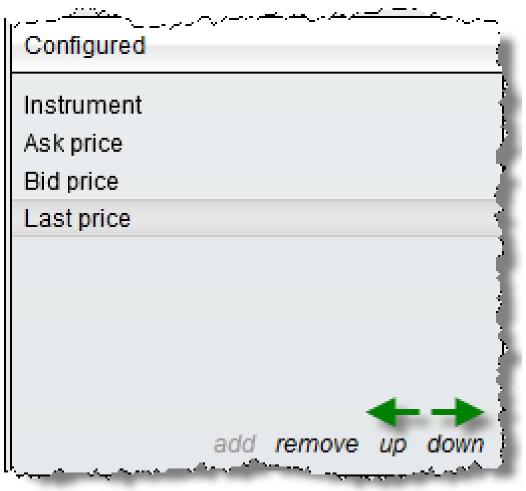
You can customize any column from the Columns window.

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 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.

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.

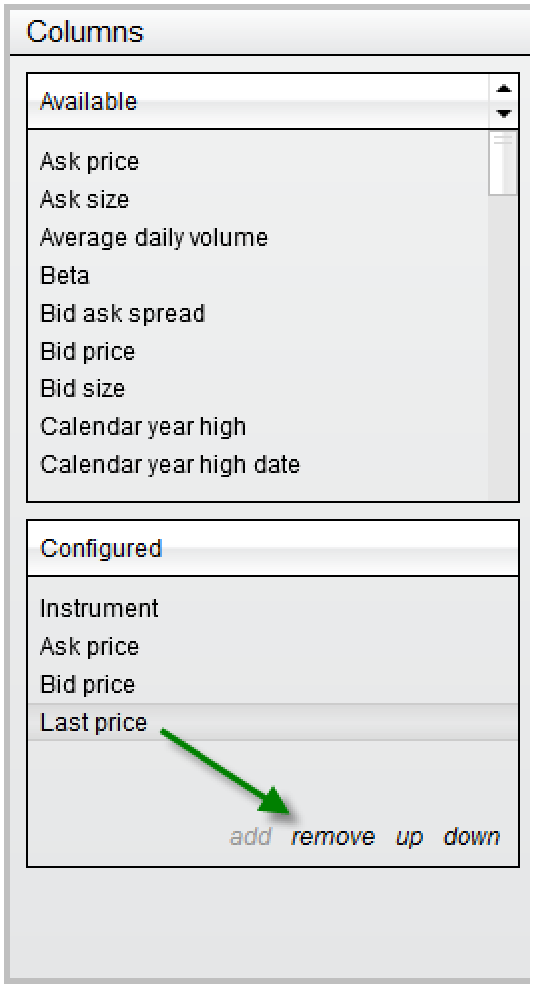
How to remove columns

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.

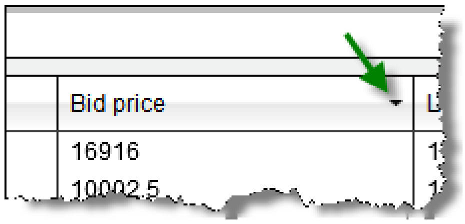
11.17.4 Dynamic Ranking and Sorting

The Market Analyzer window can automatically rank and sort the data rows.

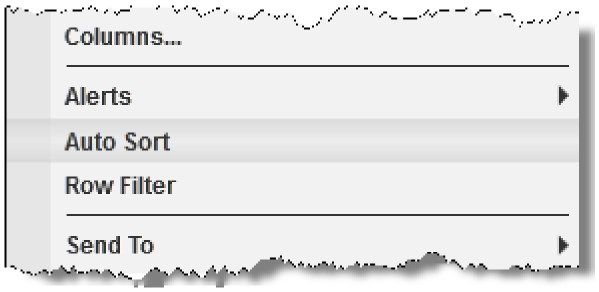
How to Enable Automatic Ranking and Sorting

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.

11.17.5 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

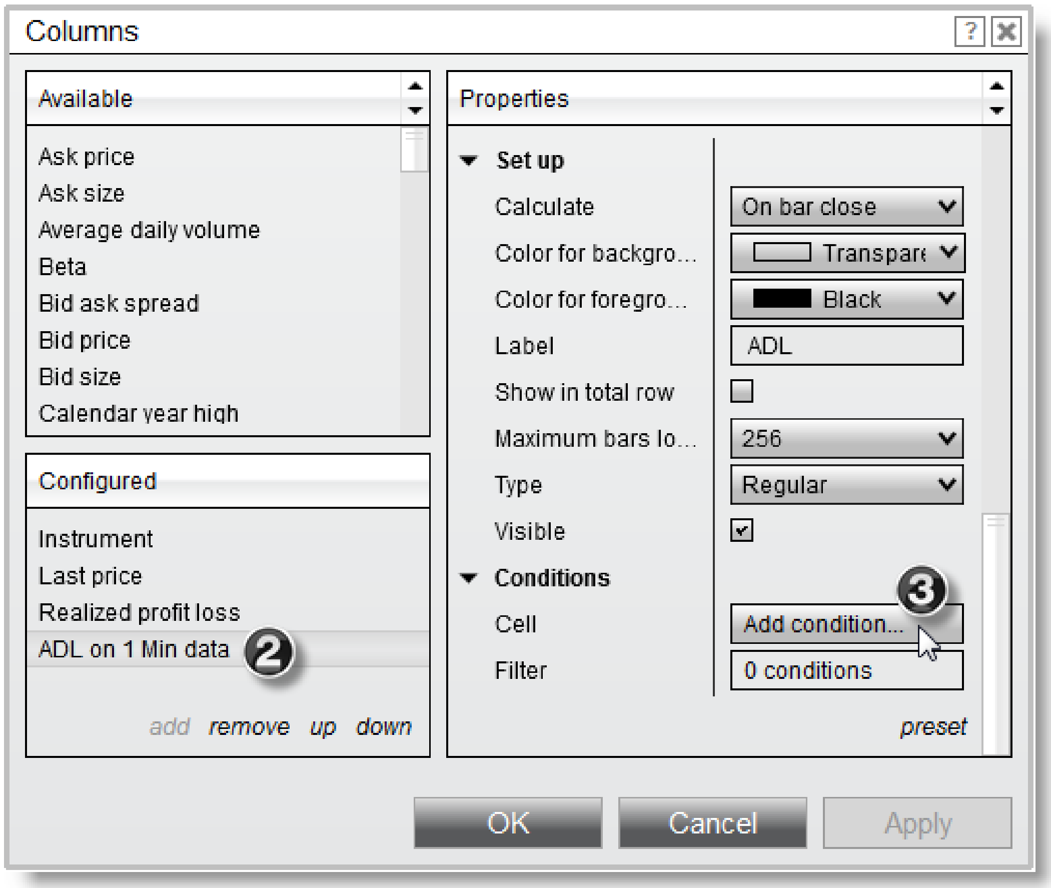
Cell Conditions

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.


Columns

Available
Ask price
Ask size
Average daily volume
Beta
Bid ask spread
Bid price
Bid size
Calendar year high

Configured
Instrument
Last price
Realized profit loss
ADL on 1 Min data

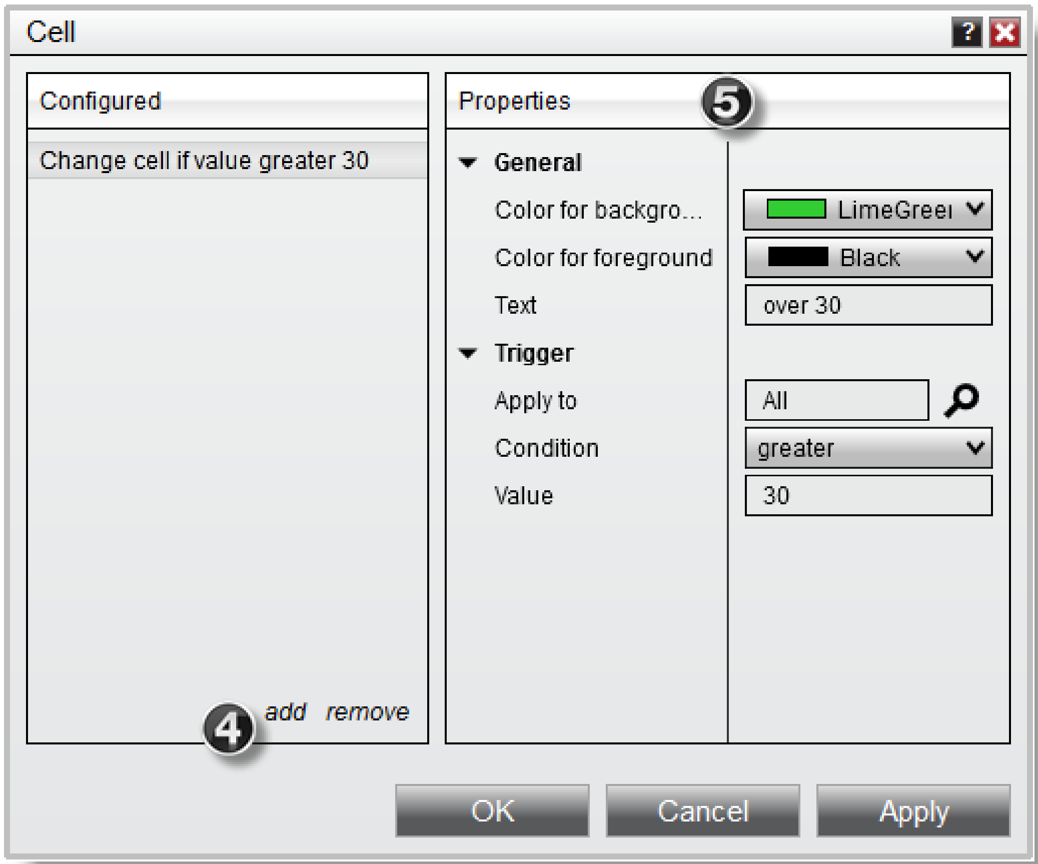
add remove up down

Properties

Set up	
Calculate	On bar close
Color for backgro...	Transpare
Color for foregro...	Black
Label	ADL
Show in total row	☐
Maximum bars lo...	256
Type	Regular
Visible	✓
Conditions	
Cell	Add condition...
Filter	0 conditions

preset

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



Change cell if value greater 30

Properties	
General	
Color for backgro...	🟩 LimeGreen
Color for foreground	■ Black
Text	over 30
Trigger	
Apply to	All
Condition	greater
Value	30

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

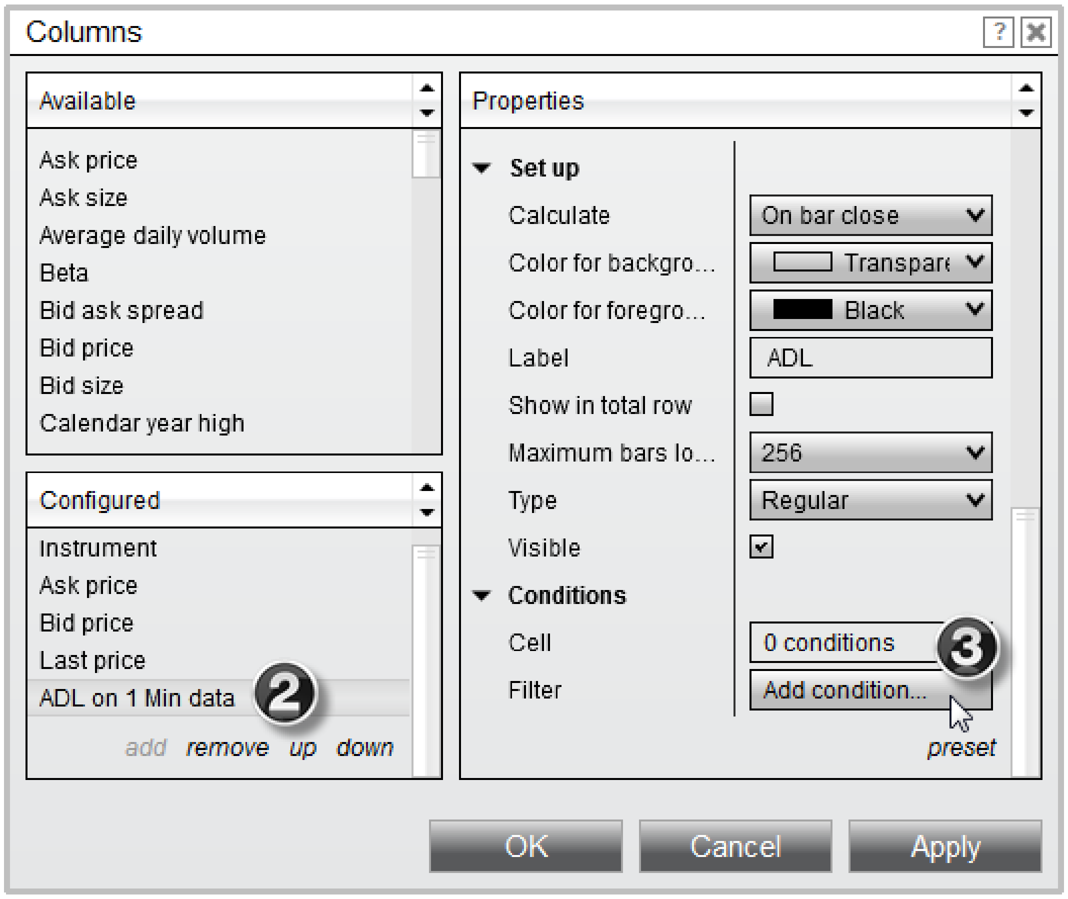
Filter Conditions

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.

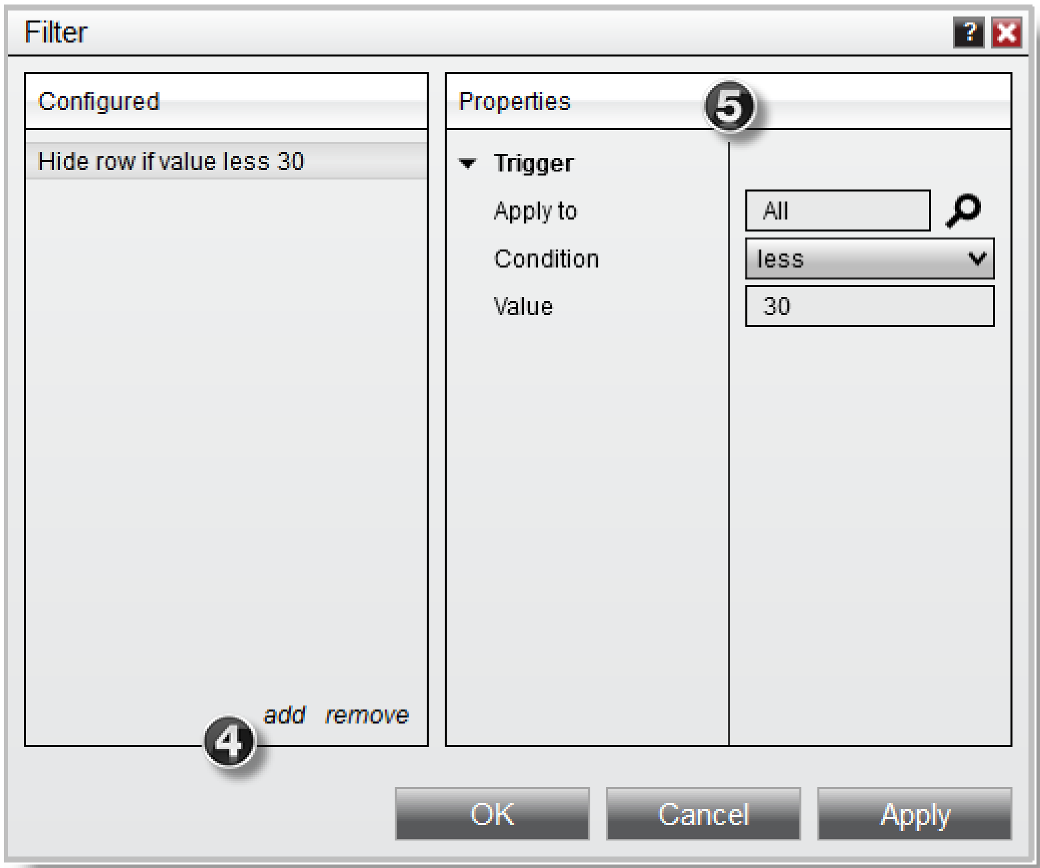

Columns

Available
Ask price
Ask size
Average daily volume
Beta
Bid ask spread
Bid price
Bid size
Calendar year high

Configured
Instrument
Ask price
Bid price
Last price
ADL on 1 Min data

Properties	
Set up	
Calculate	On bar close
Color for backgro...	Transpare
Color for foregro...	Black
Label	ADL
Show in total row	☐
Maximum bars lo...	256
Type	Regular
Visible	☑
Conditions	
Cell	0 conditions
Filter	Add condition...

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



Hide row if value less 30

Trigger

Apply to	All
Condition	less
Value	30

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

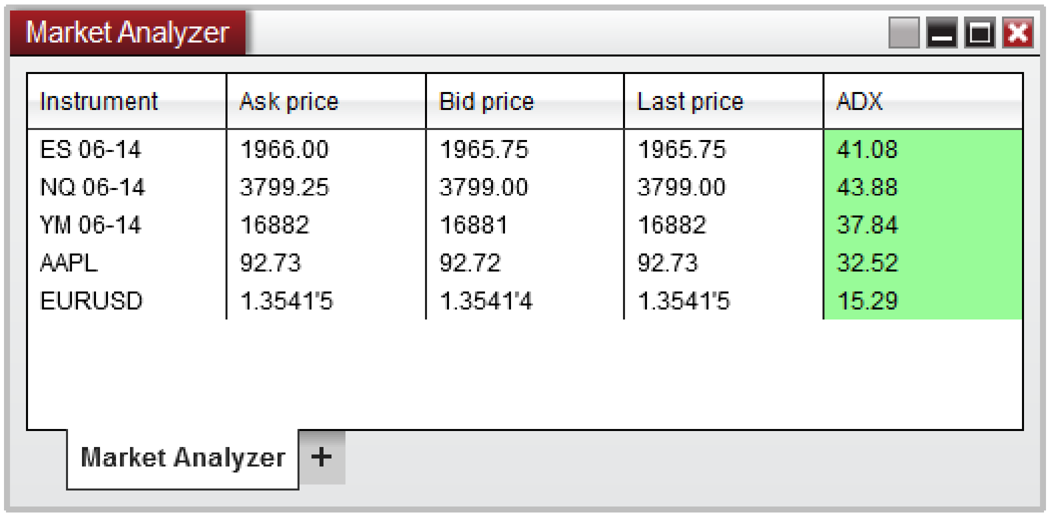
You can remove a Filter Condition by pressing the Remove button.

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

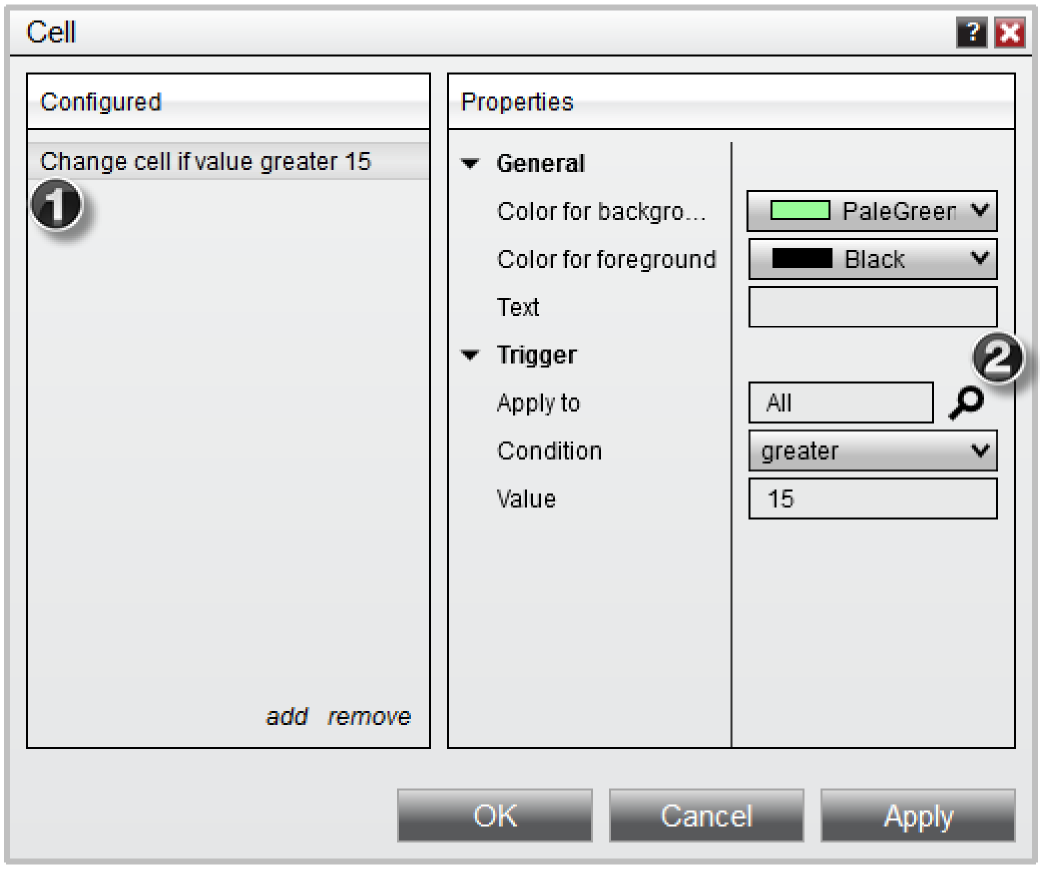
When setting up **Cell** and **Filter conditions**, the default behavior is to apply these conditions to **all** instruments in the **Market Analyzer.**



Instrument	Ask price	Bid price	Last price	ADX
ES 06-14	1966.00	1965.75	1965.75	41.08
NQ 06-14	3799.25	3799.00	3799.00	43.88
YM 06-14	16882	16881	16882	37.84
AAPL	92.73	92.72	92.73	32.52
EURUSD	1.3541'5	1.3541'4	1.3541'5	15.29

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


Cell

Configured

Change cell if value greater 15

add remove

Properties

General

Color for backgro... PaleGreen

Color for foreground Black

Text

Trigger

Apply to All

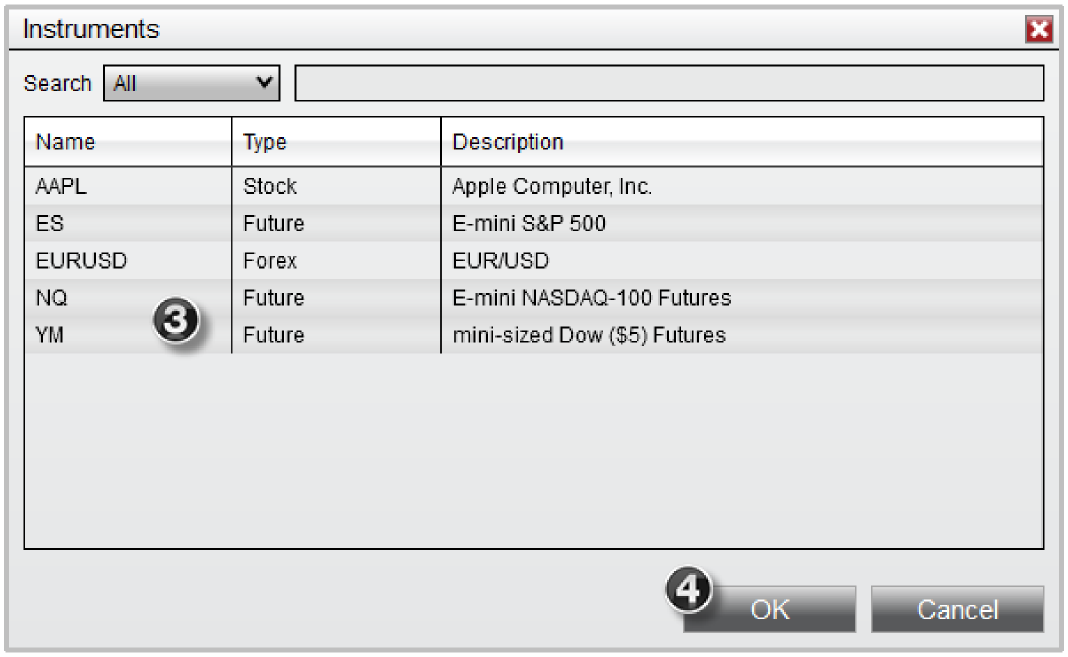
Condition greater

Value 15

3. From the newly opened **Instruments window**, select the instruments you wish to apply the condition

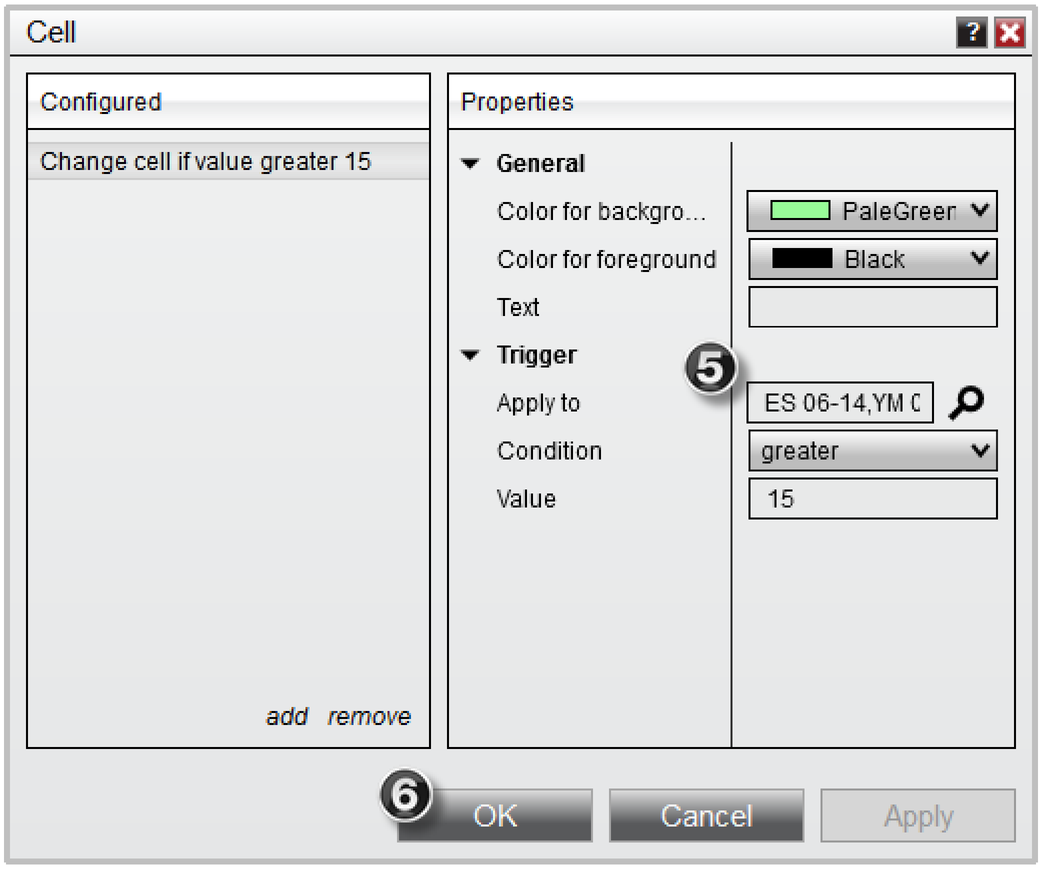
Tip
Multi-select is supported in the Instrument window:
- *To select a consecutive instruments, click the first instrument, press and hold down the Shift key, and then click the last instrument.*
- *To select non-consecutive instruments, press and hold down the Ctrl key, and then click each instrument that you want to select.*

4. Press ***OK*** on the **Instruments window**



Name	Type	Description
AAPL	Stock	Apple Computer, Inc.
ES	Future	E-mini S&P 500
EURUSD	Forex	EUR/USD
NQ	Future	E-mini NASDAQ-100 Futures
YM	Future	mini-sized Dow ($5) Futures

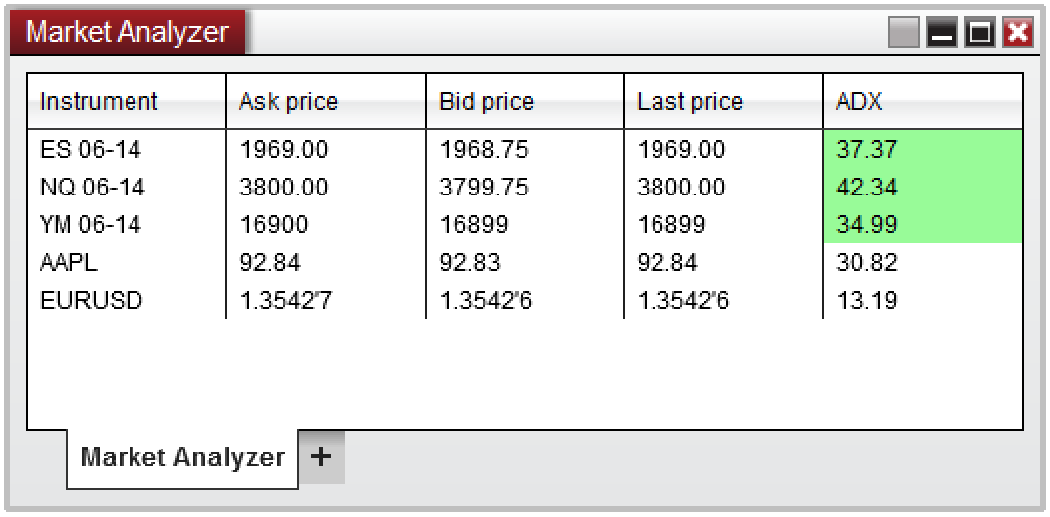
5. Your **Apply to** field will now list the instrument names you selected earlier, indicating that conditions will only be triggered on instruments contained in this list.
6. Press ***OK*** **Conditions** window



Change cell if value greater 15

General	
Color for backgro...	🟩 PaleGreer
Color for foreground	■ Black
Text	
Trigger	
Apply to	ES 06-14,YM (
Condition	greater
Value	15

Your **Market Analyzer** window will now only apply these conditions to the instruments which mach the name you configured



Instrument	Ask price	Bid price	Last price	ADX
ES 06-14	1969.00	1968.75	1969.00	37.37
NQ 06-14	3800.00	3799.75	3800.00	42.34
YM 06-14	16900	16899	16899	34.99
AAPL	92.84	92.83	92.84	30.82
EURUSD	1.3542'7	1.3542'6	1.3542'6	13.19

11.17.6 Market Analyzer Properties

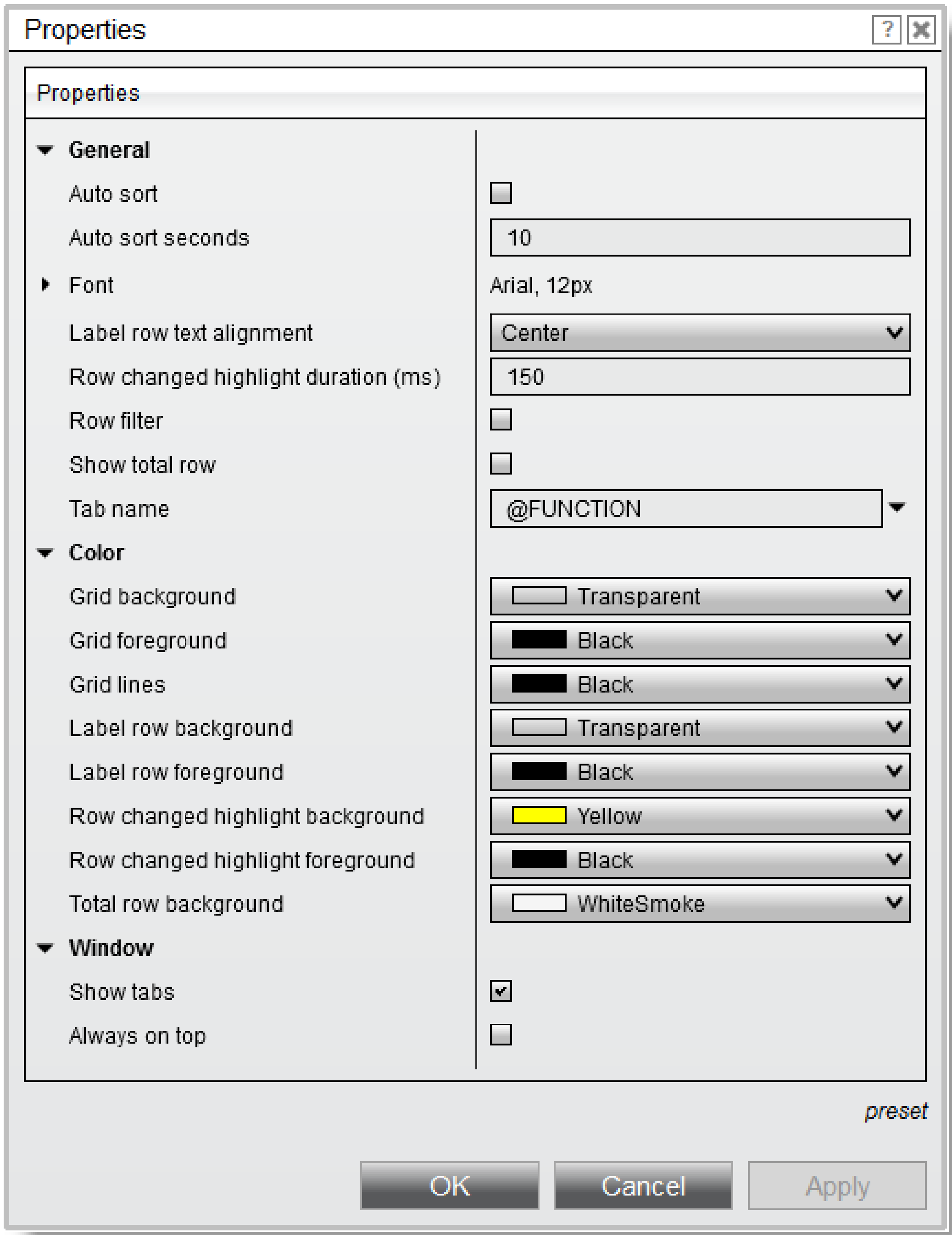
The Market Analyzer can be customized to your preferences in the Market Analyzer Properties window.

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


Properties

Properties

General	
Auto sort	☐
Auto sort seconds	10
Font	Arial, 12px
Label row text alignment	Center
Row changed highlight duration (ms)	150
Row filter	☐
Show total row	☐
Tab name	@FUNCTION
Color	
Grid background	Transparent
Grid foreground	Black
Grid lines	Black
Label row background	Transparent
Label row foreground	Black
Row changed highlight background	Yellow
Row changed highlight foreground	Black
Total row background	WhiteSmoke
Window	
Show tabs	✓
Always on top	☐

preset

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
Font	Sets the font
Label row text alignment	Sets the alignment for the label rows
Row change highlight duration (ms)	Sets the duration (in seconds) the instrument cell will remain highlighted. A value of zero will disable highlighting. **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
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.

Market Analyzer Columns along with custom properites can be saved within a Market Analyzer Template.

11.17.7 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:

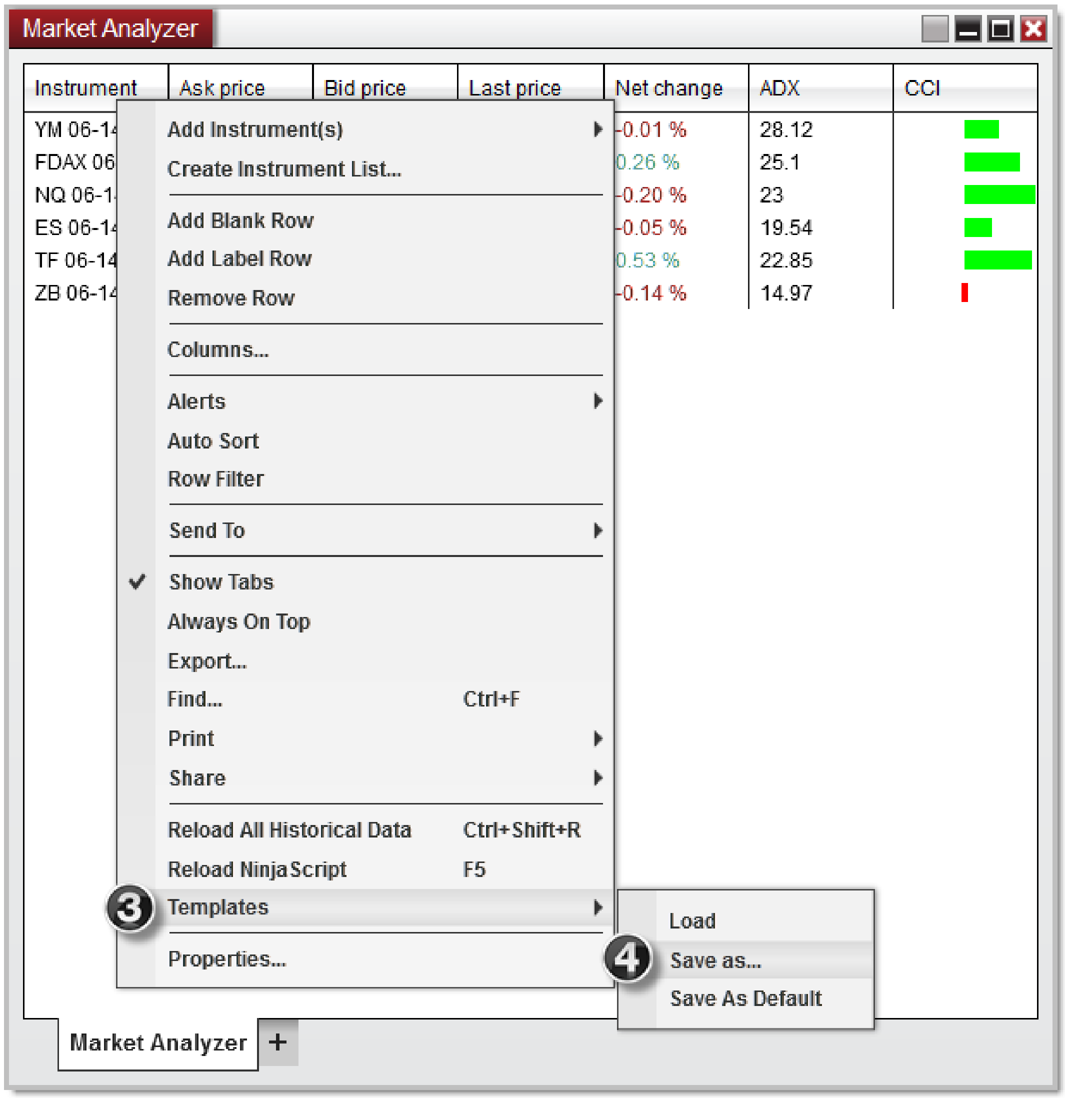
- Column layout
- Column parameters and conditions
- Market Analyzer properties
- Instrument Rows

Saving a Market Analyzer Template

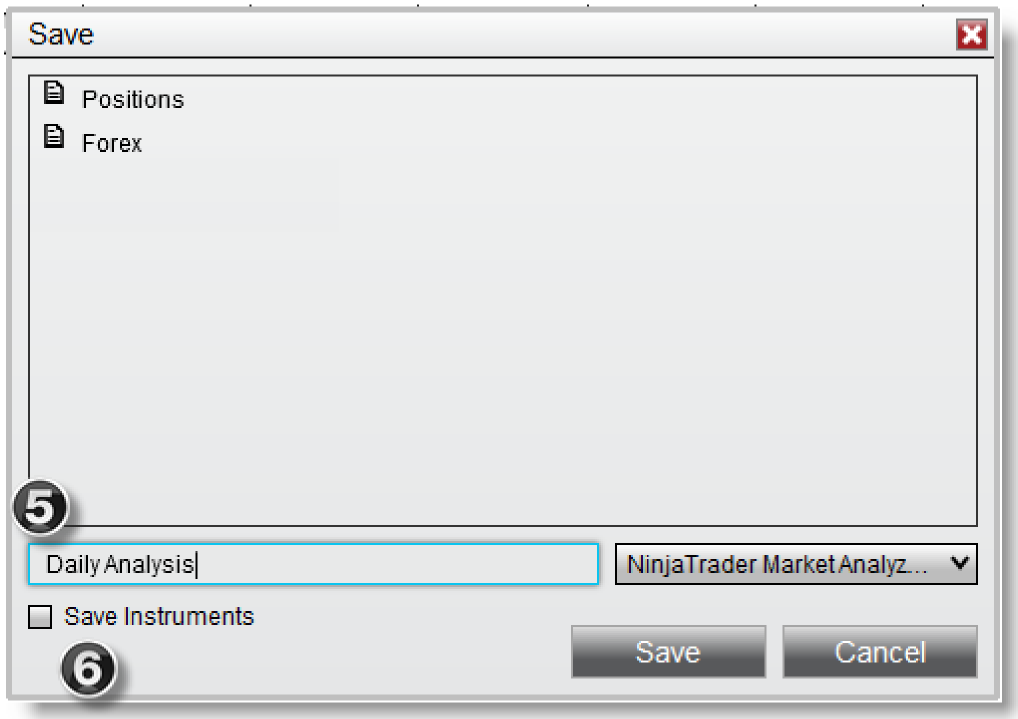
To save a Market Analyzer template (shown in the image below):

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

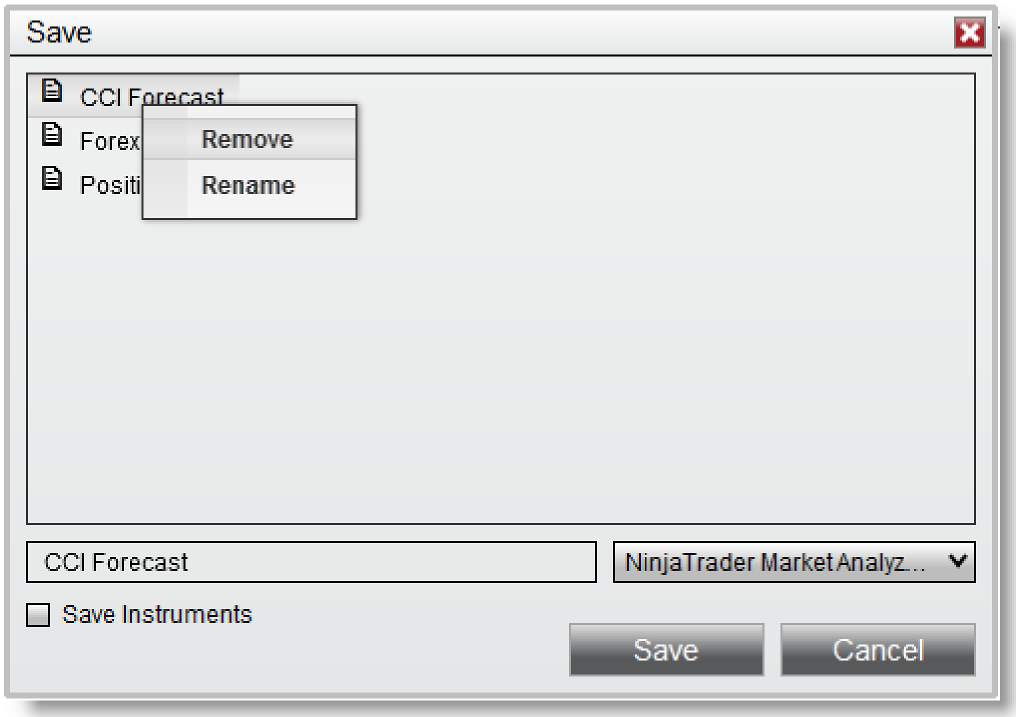
How to remove a Market Analyzer Template

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*

11.17.8 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.

11.17.9 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.

11.17.10 Window Linking

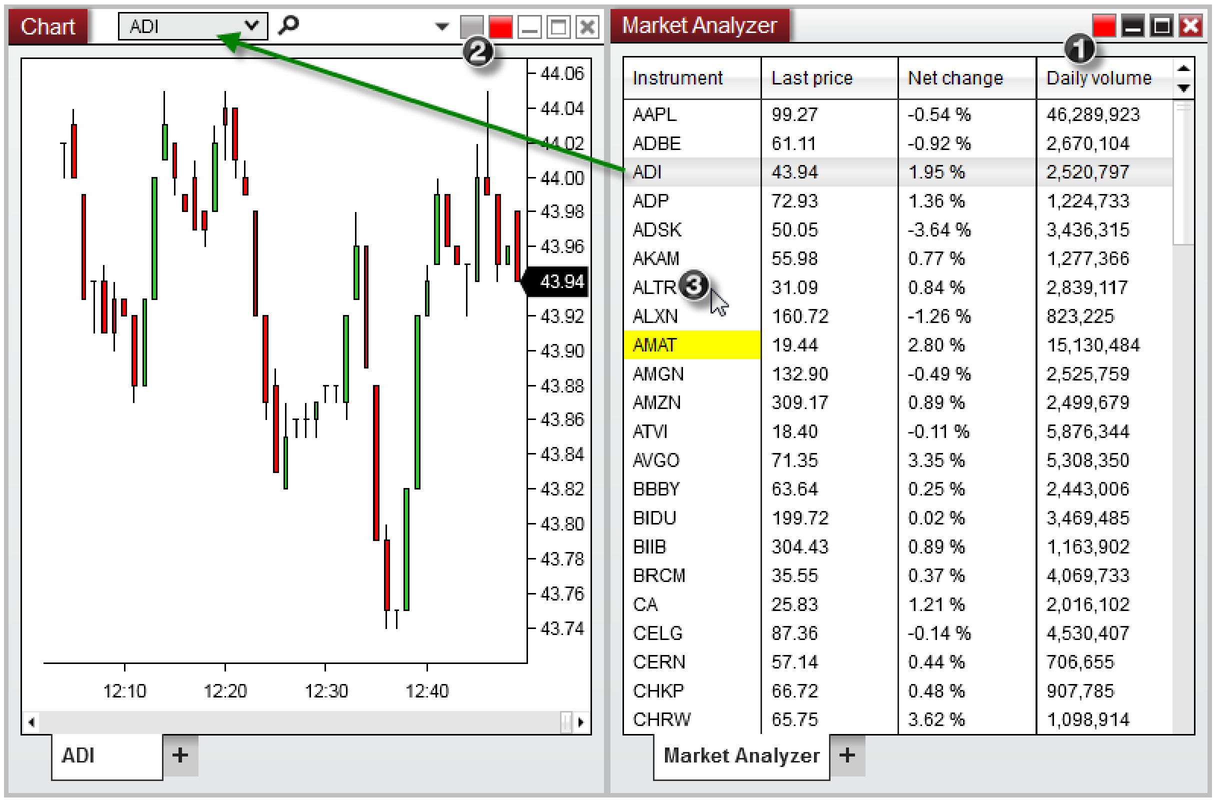
One of the most useful features of the **Market Analyzer** is the ability to link the instruments displayed in the **Market Analyzer** grid to any other window in the NinjaTrader application. This allows you to cycle through a custom list of instruments and quickly load the desired symbols in a **Chart**, **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 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.



Instrument	Last price	Net change	Daily volume
AAPL	99.27	-0.54 %	46,289,923
ADBE	61.11	-0.92 %	2,670,104
ADI	43.94	1.95 %	2,520,797
ADP	72.93	1.36 %	1,224,733
ADSK	50.05	-3.64 %	3,436,315
AKAM	55.98	0.77 %	1,277,366
ALTR	31.09	0.84 %	2,839,117
ALXN	160.72	-1.26 %	823,225
AMAT	19.44	2.80 %	15,130,484
AMGN	132.90	-0.49 %	2,525,759
AMZN	309.17	0.89 %	2,499,679
ATVI	18.40	-0.11 %	5,876,344
AVGO	71.35	3.35 %	5,308,350
BBBY	63.64	0.25 %	2,443,006
BIDU	199.72	0.02 %	3,469,485
BIIB	304.43	0.89 %	1,163,902
BRCM	35.55	0.37 %	4,069,733
CA	25.83	1.21 %	2,016,102
CELG	87.36	-0.14 %	4,530,407
CERN	57.14	0.44 %	706,655
CHKP	66.72	0.48 %	907,785
CHRW	65.75	3.62 %	1,098,914

11.18 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.

› News Window
› News Properties

11.18.1 News Window

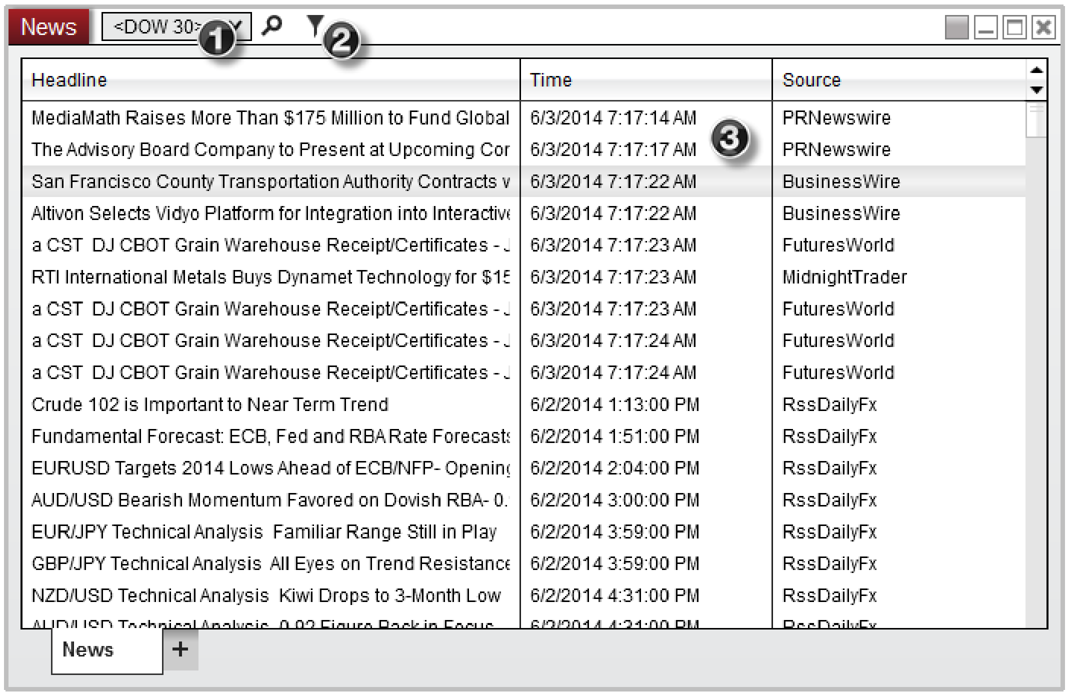
News sent from the connectivity provider is displayed in the **News** window. Alerts and filters can be configured based on keywords in the news headline.

Understanding the News Window

News Window Display

The **News** window give you the ability to:

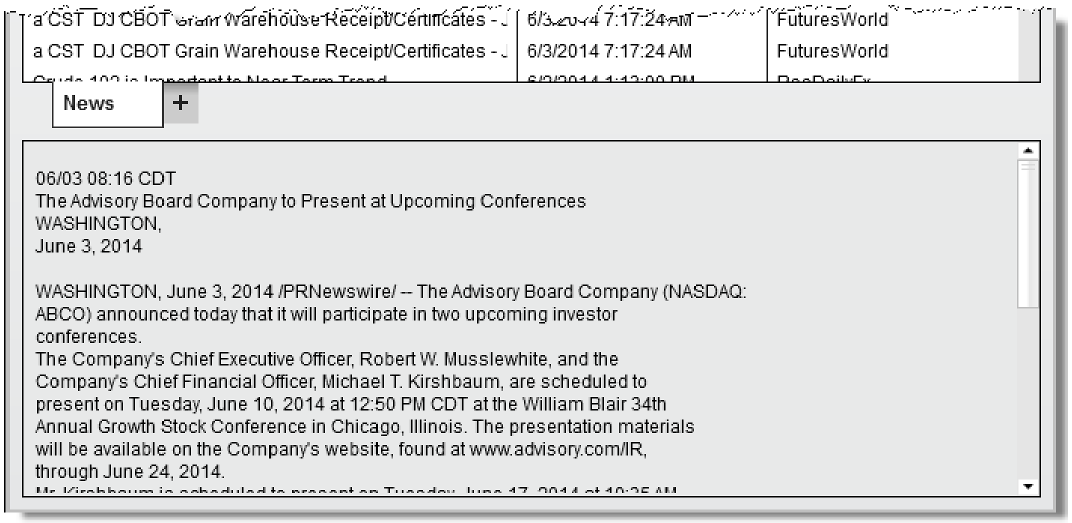
1. Filter news based on individual instruments or instrument list
2. Setup user defined Keyword filters
3. View a list of real-time news headlines



Headline	Time	Source
MediaMath Raises More Than $175 Million to Fund Global	6/3/2014 7:17:14 AM	PRNewswire
The Advisory Board Company to Present at Upcoming Cor	6/3/2014 7:17:17 AM	PRNewswire
San Francisco County Transportation Authority Contracts v	6/3/2014 7:17:22 AM	BusinessWire
Altivon Selects Vidyo Platform for Integration into Interactive	6/3/2014 7:17:22 AM	BusinessWire
a CST DJ CBOT Grain Warehouse Receipt/Certificates -	6/3/2014 7:17:23 AM	FuturesWorld
RTI International Metals Buys Dynamet Technology for $15	6/3/2014 7:17:23 AM	MidnightTrader
a CST DJ CBOT Grain Warehouse Receipt/Certificates -	6/3/2014 7:17:23 AM	FuturesWorld
a CST DJ CBOT Grain Warehouse Receipt/Certificates -	6/3/2014 7:17:24 AM	FuturesWorld
a CST DJ CBOT Grain Warehouse Receipt/Certificates -	6/3/2014 7:17:24 AM	FuturesWorld
Crude 102 is Important to Near Term Trend	6/2/2014 1:13:00 PM	RssDailyFx
Fundamental Forecast: ECB, Fed and RBA Rate Forecasts	6/2/2014 1:51:00 PM	RssDailyFx
EURUSD Targets 2014 Lows Ahead of ECB/NFP- Opening	6/2/2014 2:04:00 PM	RssDailyFx
AUD/USD Bearish Momentum Favored on Dovish RBA- 0.	6/2/2014 3:00:00 PM	RssDailyFx
EUR/JPY Technical Analysis Familiar Range Still in Play	6/2/2014 3:59:00 PM	RssDailyFx
GBP/JPY Technical Analysis All Eyes on Trend Resistance	6/2/2014 3:59:00 PM	RssDailyFx
NZD/USD Technical Analysis Kiwi Drops to 3-Month Low	6/2/2014 4:31:00 PM	RssDailyFx
AUD/USD Technical Analysis 0.92 Figure Back in Focus	6/2/2014 4:31:00 PM	RssDailyFx

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**.

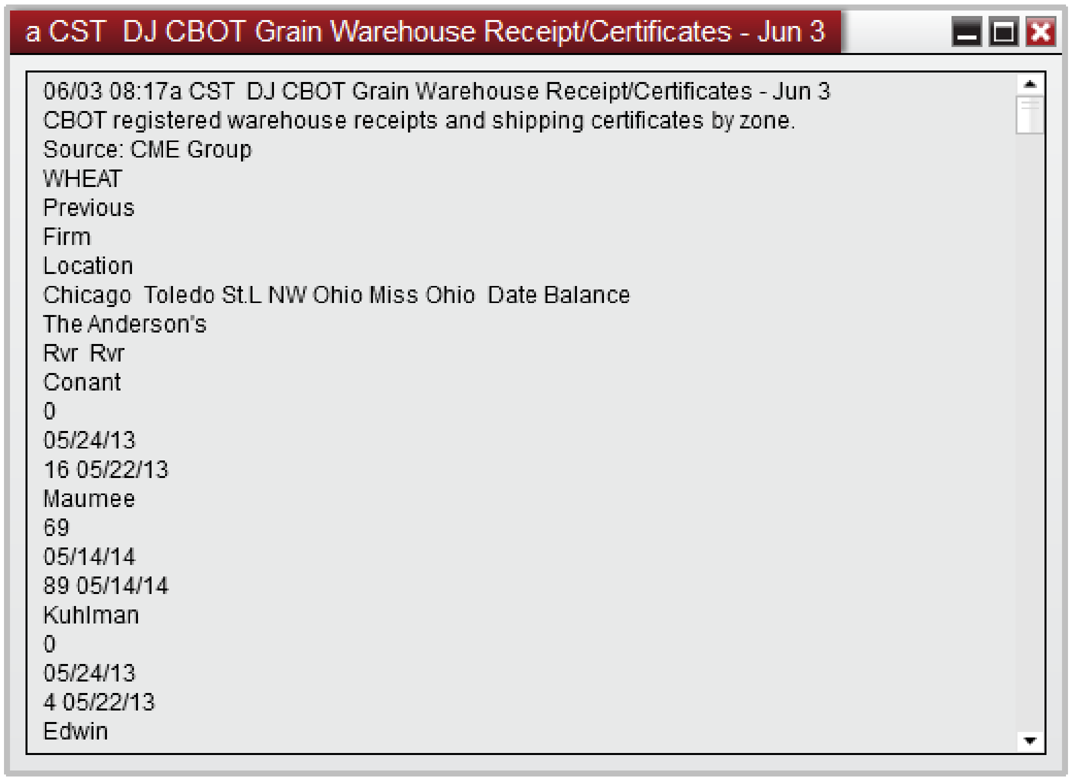

a CST DJ CBOT Grain Warehouse Receipt/Certificates - J
6/3/2014 7:17:24 AM
FuturesWorld
a CST DJ CBOT Grain Warehouse Receipt/Certificates - J
6/3/2014 7:17:24 AM
FuturesWorld
News
+

06/03 08:16 CDT
The Advisory Board Company to Present at Upcoming Conferences
WASHINGTON,
June 3, 2014

WASHINGTON, June 3, 2014 /PRNewswire/ -- The Advisory Board Company (NASDAQ: ABCO) announced today that it will participate in two upcoming investor conferences.
The Company's Chief Executive Officer, Robert W. Musslewhite, and the Company's Chief Financial Officer, Michael T. Kirshbaum, are scheduled to present on Tuesday, June 10, 2014 at 12:50 PM CDT at the William Blair 34th Annual Growth Stock Conference in Chicago, Illinois. The presentation materials will be available on the Company's website, found at www.advisory.com/IR, through June 24, 2014.

Headline Window

Double clicking on a headline will open the Headline Window which will display the content of the news article.



06/03 08:17a CST DJ CBOT Grain Warehouse Receipt/Certificates - Jun 3
CBOT registered warehouse receipts and shipping certificates by zone.
Source: CME Group
WHEAT
Previous
Firm
Location
Chicago Toledo St.L NW Ohio Miss Ohio Date Balance
The Anderson's
Rvr Rvr
Conant
0
05/24/13
16 05/22/13
Maumee
69
05/14/14
89 05/14/14
Kuhlman
0
05/24/13
4 05/22/13
Edwin

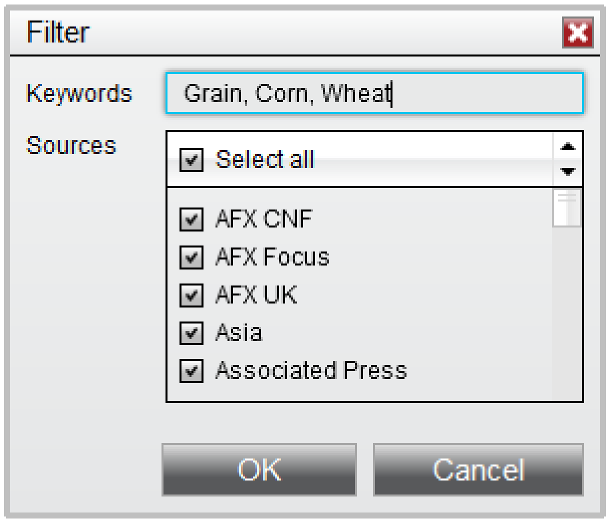
▽ How to create a filter condition

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



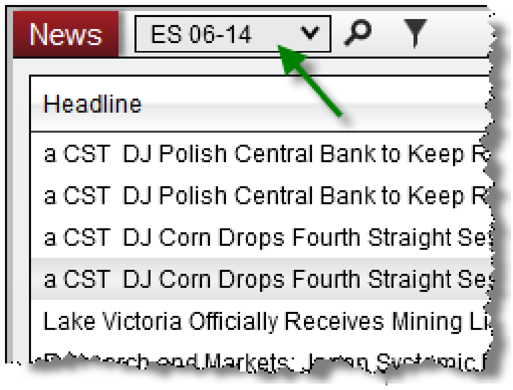
Keywords: Grain, Corn, Wheat

Sources:
- ✓ Select all
- ✓ AFX CNF
- ✓ AFX Focus
- ✓ AFX UK
- ✓ Asia
- ✓ Associated Press

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



a CST DJ Polish Central Bank to Keep R
a CST DJ Polish Central Bank to Keep R
a CST DJ Corn Drops Fourth Straight Se
a CST DJ Corn Drops Fourth Straight Se
Lake Victoria Officially Receives Mining Li

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

11.18.2 News Properties

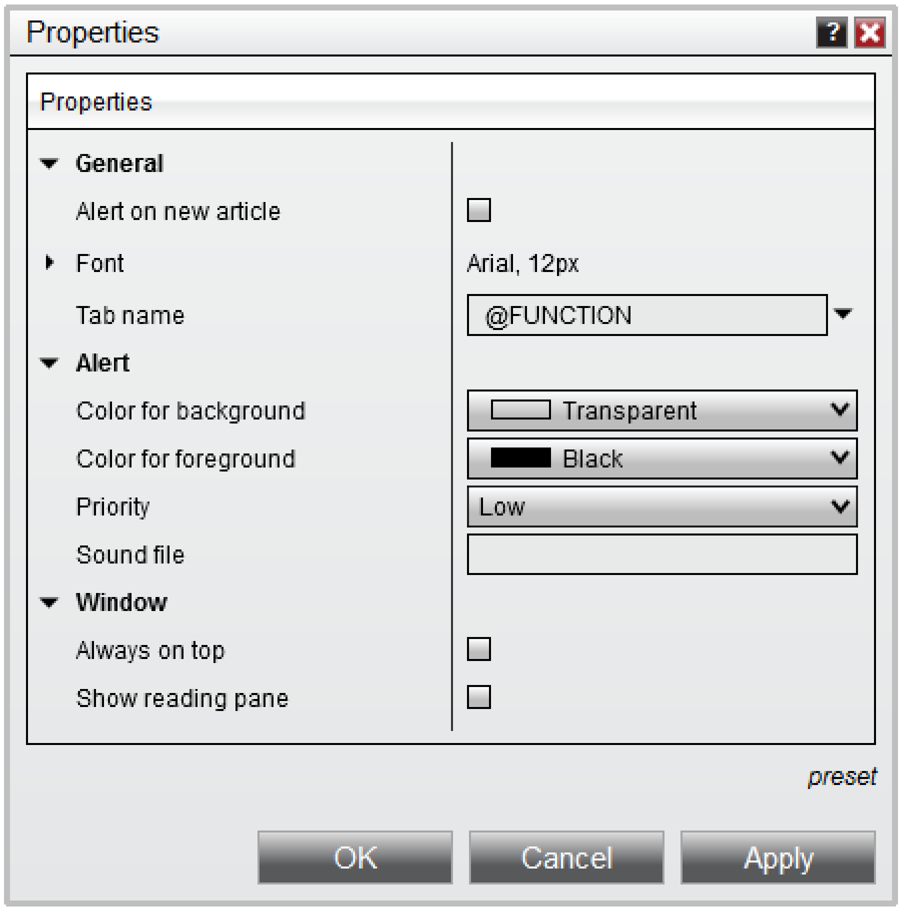
The News window can be customized through the **News Properties** window.

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:



Properties

Properties

General

Alert on new article ☐

Font: Arial, 12px

Tab name: @FUNCTION

Alert

Color for background: Transparent

Color for foreground: Black

Priority: Low

Sound file

Window

Always on top ☐

Show reading pane ☐

preset

Property Definitions

General	
Alert on new article	Sets the option to receive alerts when new article is received
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.

11.19 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 four 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

Note: 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.

.

11.19.1 Attaching Orders To Indicators

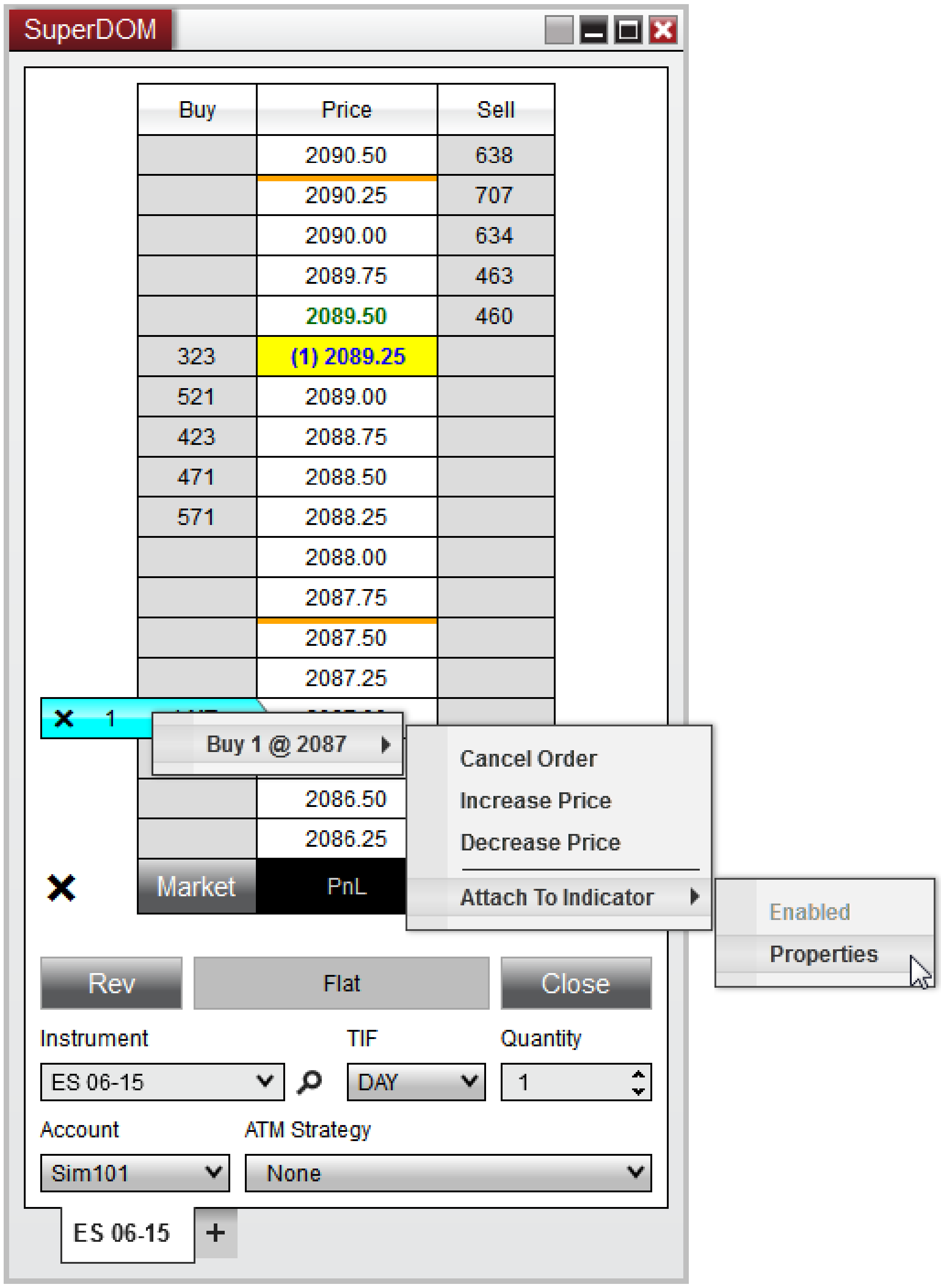
Indicator Tracking

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.



Buy	Price	Sell
	2090.50	638
	2090.25	707
	2090.00	634
	2089.75	463
	2089.50	460
323	(1) 2089.25	
521	2089.00	
423	2088.75	
471	2088.50	
571	2088.25	
	2088.00	
	2087.75	
	2087.50	
	2087.25	
	2086.50	
	2086.25	

Attaching an order to an indicator

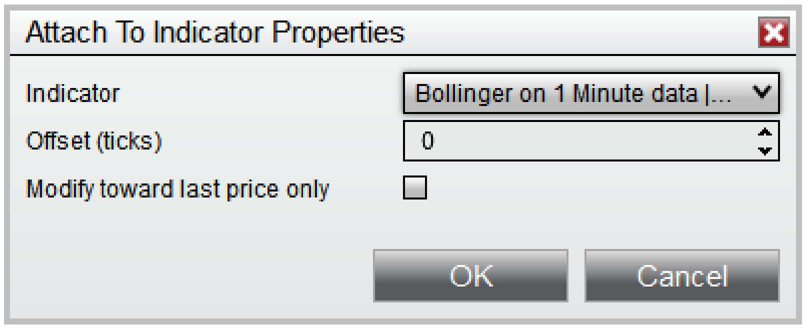
Configuring Attach to Indicator Properties

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

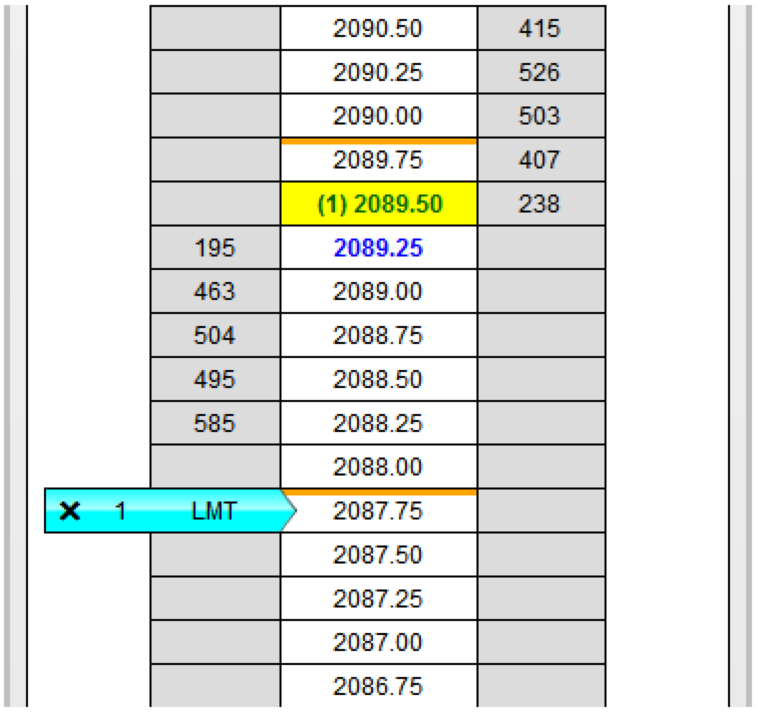
This will open an **Attach to Indicator Properties** window which will allow you to define the following properties:

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.


Attach To Indicator Properties

Indicator	Bollinger on 1 Minute data |...
Offset (ticks)	0
Modify toward last price only	☐

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.



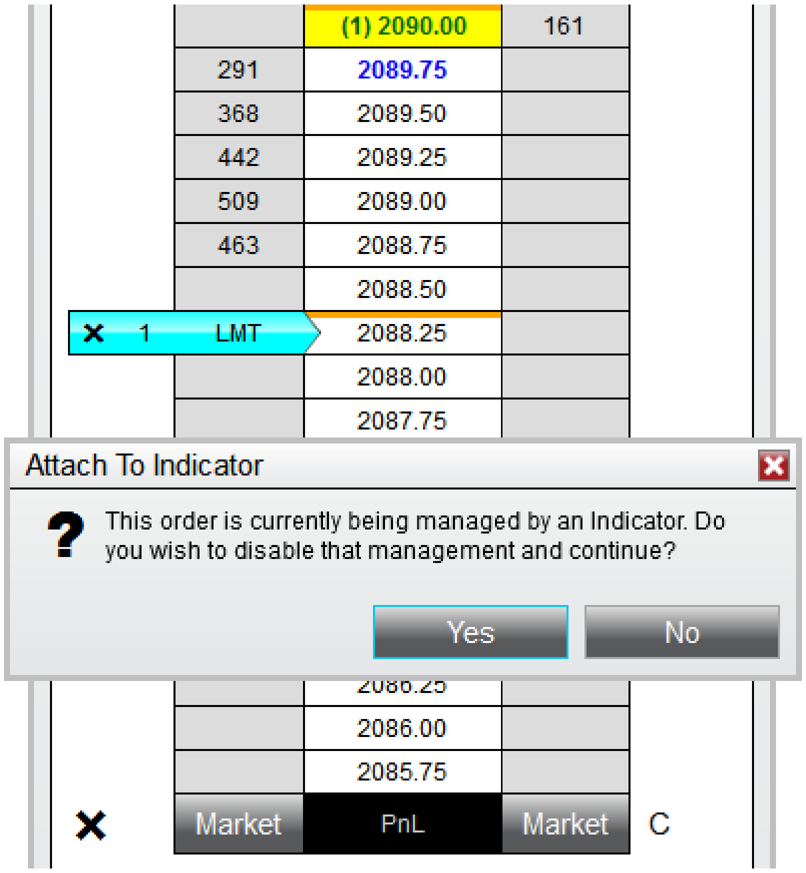
	2090.50	415
	2090.25	526
	2090.00	503
	2089.75	407
	(1) 2089.50	238
195	2089.25	
463	2089.00	
504	2088.75	
495	2088.50	
585	2088.25	
	2088.00	
✕ 1 LMT	2087.75	
	2087.50	
	2087.25	
	2087.00	
	2086.75	

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.



Attach To Indicator

This order is currently being managed by an Indicator. Do you wish to disable that management and continue?

Yes No

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.

11.19.2 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
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 or exchange	No color
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

Note: 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.

11.19.3 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.

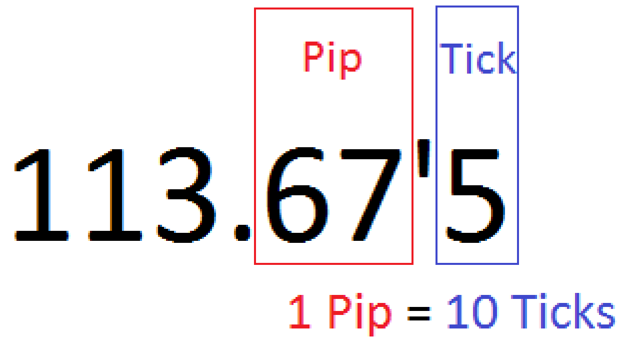
11.19.4 Working With Forex

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

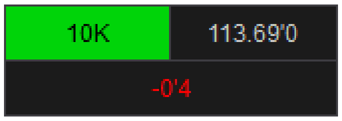
Pips vs. Ticks

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.

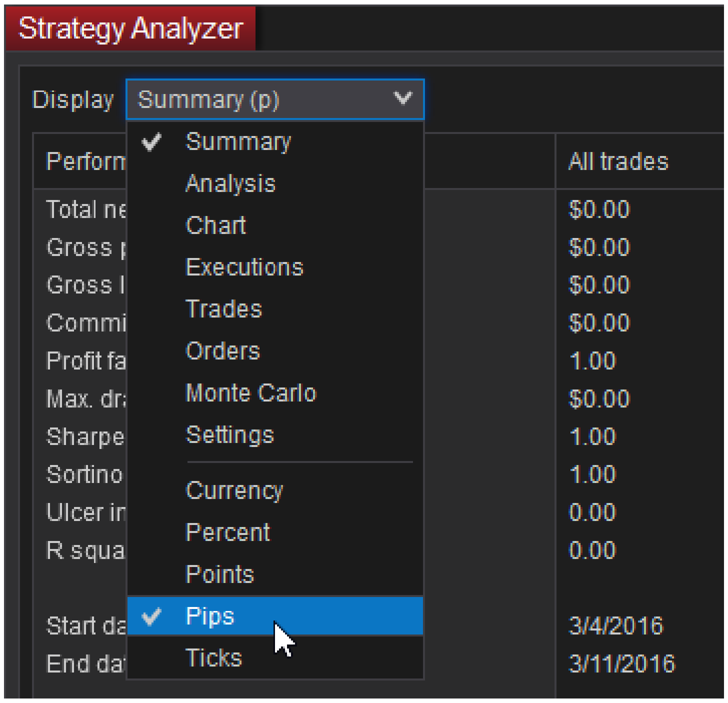


Setting the Pips Calculation Mode

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.



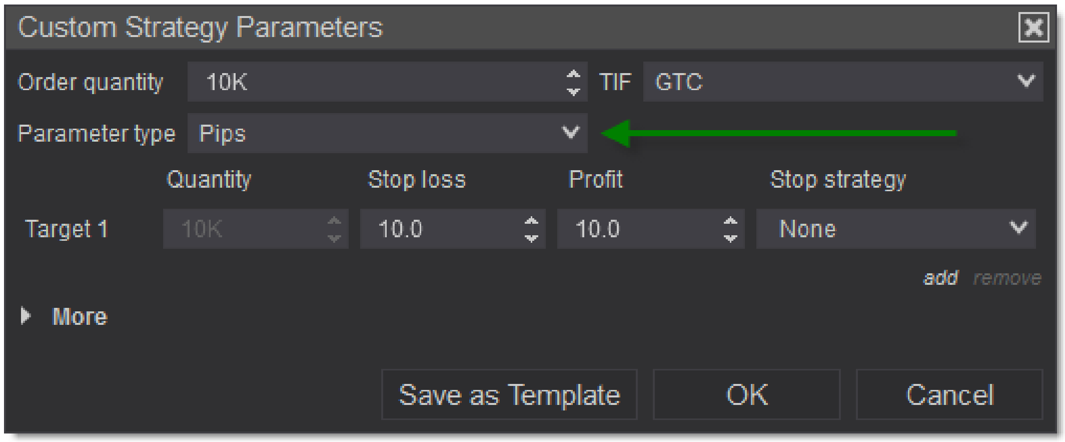
In the Trade Performance window and Strategy Analyzer, the calculation mode can be changed via the **Display** dropdown menu, which affects all relevant statistics.



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.



Order quantity: 10K
TIF: GTC
Parameter type: Pips

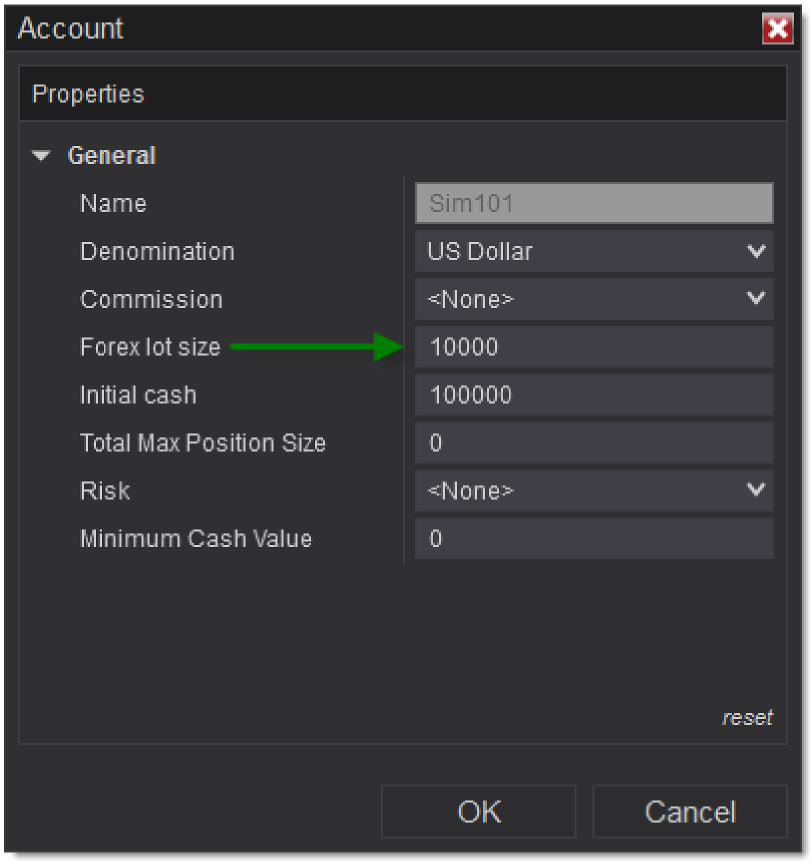
	Quantity	Stop loss	Profit	Stop strategy
Target 1	10K	10.0	10.0	None

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 Simulator 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).


Account

Properties

General

Name	Sim101
Denomination	US Dollar
Commission	<None>
Forex lot size	10000
Initial cash	100000
Total Max Position Size	0
Risk	<None>
Minimum Cash Value	0

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.
- Interactive Brokers users have the ability to set their FX Lot Size manually in their Connection configuration.

Forex Lot Size For Live Accounts
For live trading accounts, the default FX Lot Size is pulled directly from your brokerage account. You can contact your broker to change the default Lot Size on their end, and NinjaTrader will automatically reflect the change.

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.

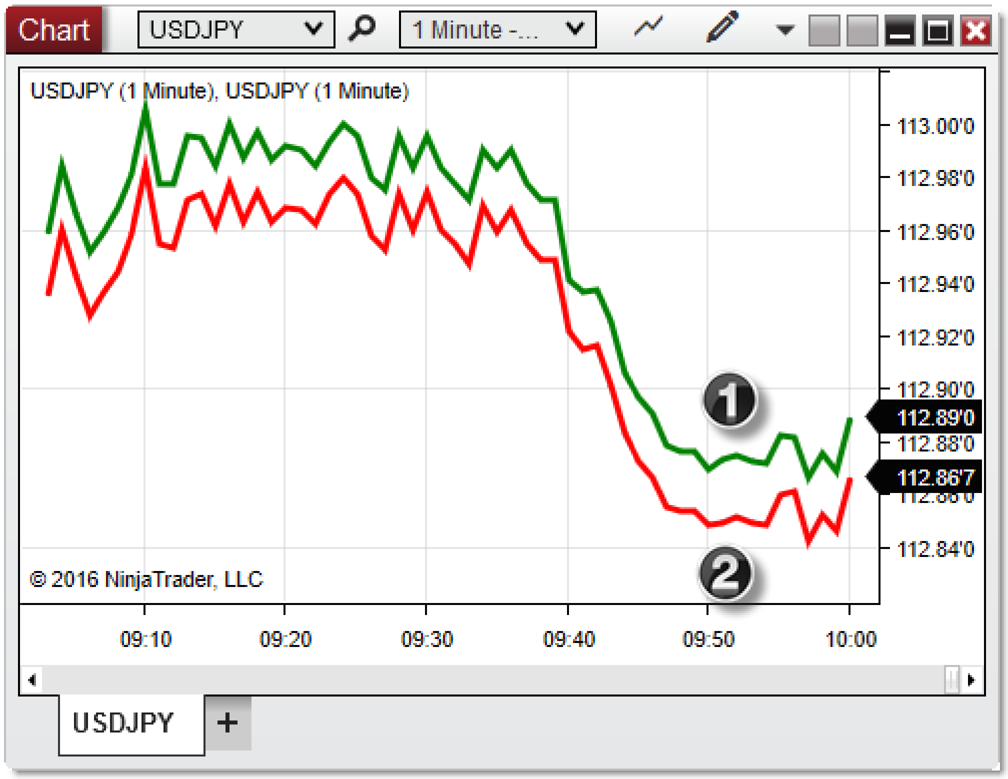
How Bars Are Built and Orders Filled

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



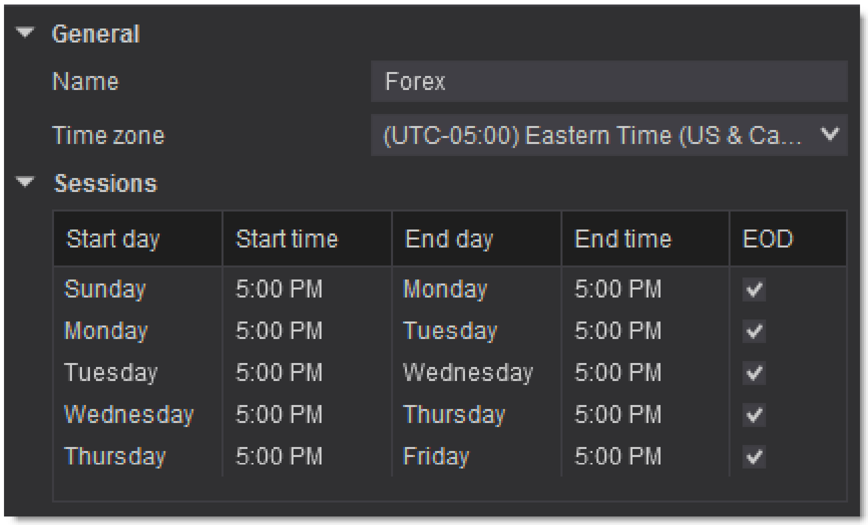
1. Ask Price: All realtime Buy orders are filled at the Ask
2. Bid Price: All realtime Sell orders and all backtest Buys and Sells are filled at the Bid

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.



General

Name	Forex
Time zone	(UTC-05:00) Eastern Time (US & Ca...

Sessions

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	✓

11.19.5 Trade Controls

Trade Controls are located in various Order Entry windows available throughout the product.

Play Video

Trade Controls

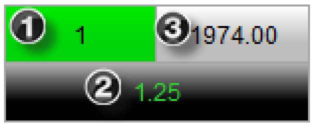
Many of the NinjaTrader **Order Entry** features will have a number of shared controls design to aid you in setting various order parameters such as **Quantity**, **Price**, **TIF**, or used to display account/instrument **Position** information. These controls are designed to behave in the same manner no matter which order entry feature you're using.

› Closing a Position or ATM Strategy
› Position Display
› Price Selector
› Quantity Selector
› TIF Selector

11.19.5.1 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, Points, Ticks* and *None* (hides your PnL). There are two ways to configured 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

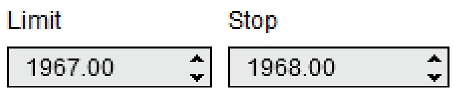
You can also configure the **PnL Display** to show you your selected account's **Realized PnL** for the current trading session when not in a position. To enable this feature, right click on the order entry feature, select **Properties**, and check **Show realized PnL when Flat.**

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. 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.

11.19.5.2 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**:

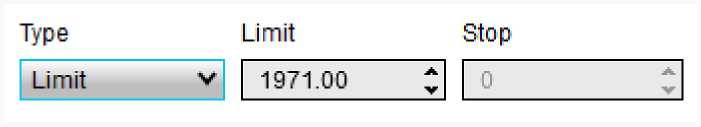
Middle Mouse Click	Sets the current Last Price
Ctrl + Middle Mouse Click	Sets the current Ask Price
Alt + Middle Mouse Click	Sets the current Bid Price
Middle Scroll Up/Down	Adjusts the current price 1 tick
Ctrl + Middle	Adjusts the current price 10 ticks

Scroll Up/ Down	

Order Types and Price Fields

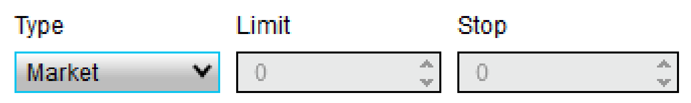
Upon selecting an order type, the relevant fields specific to that type of order will be enabled to allow you to edit the order price before submitting the order. If a field is not relevant to an order type, it will be disabled. Note that a **Stop-Limit** order has both fields enabled which implies that both fields must have a value in order to place this type of order.

Limit Order - Limit Field



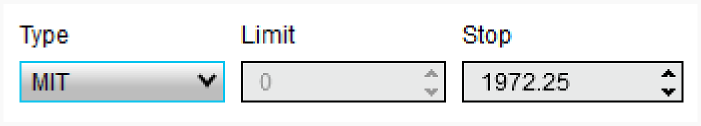
Type	Limit	Stop
Limit	1971.00	0

Market Order - No Fields



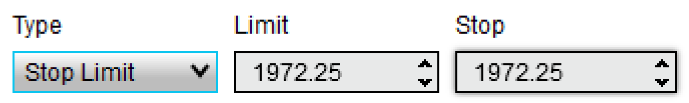
Type	Limit	Stop
Market	0	0

MIT Order - Stop Field



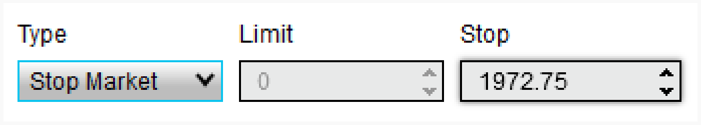
Type	Limit	Stop
MIT	0	1972.25

Stop-Limit Order - Stop and Limit Fields



Type	Limit	Stop
Stop Limit	1972.25	1972.25

Stop-Market Order - Stop Field



Type	Limit	Stop
Stop Market	0	1972.75

11.19.5.3 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 (Standard), 10K (Micro), or 1K (Mini)

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.

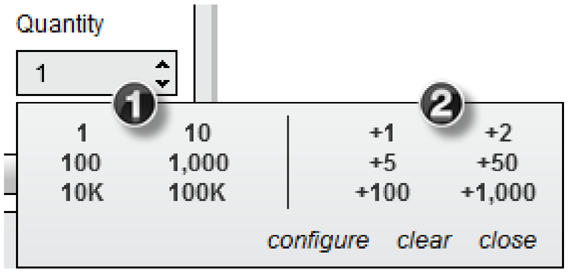
You can also control the quantity using the up/down arrows next to the quantity selector, or by using the scroll wheel on your mouse. These methods will change the quantity depending on the instrument type's minimum values described in the "*Default Order Quantities"* section above.

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.



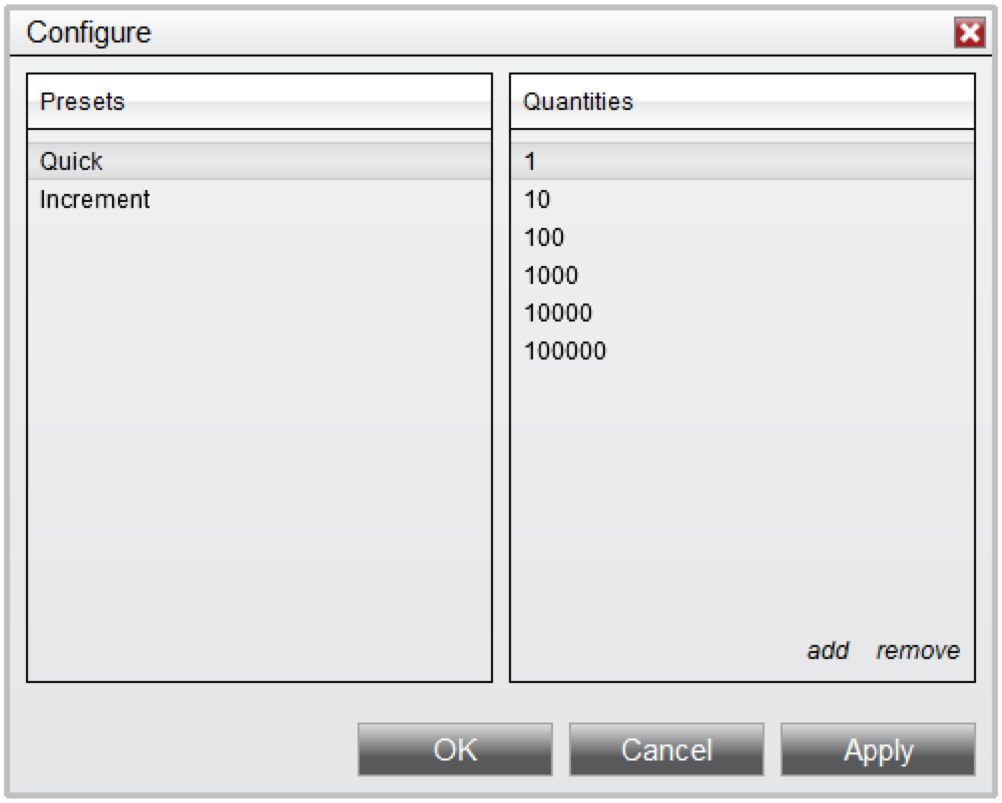
1. Quick	Sets the order quantity used to an exact pre-defined quantity
2. Increment	Increases the current order quantity value by pre-defined value

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

11.19.5.4 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.

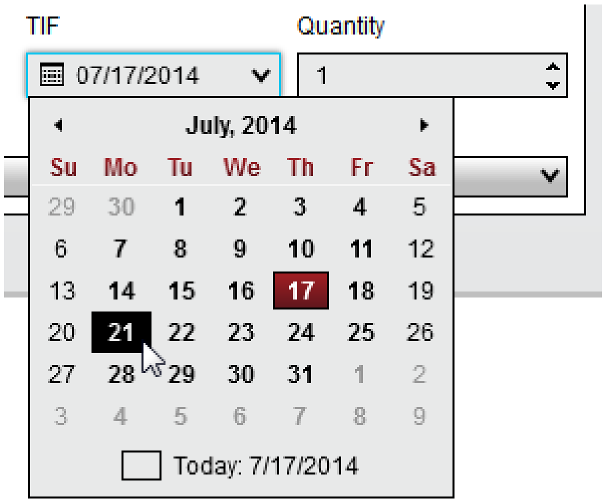
Available TIF Options

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**.

11.19.6 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

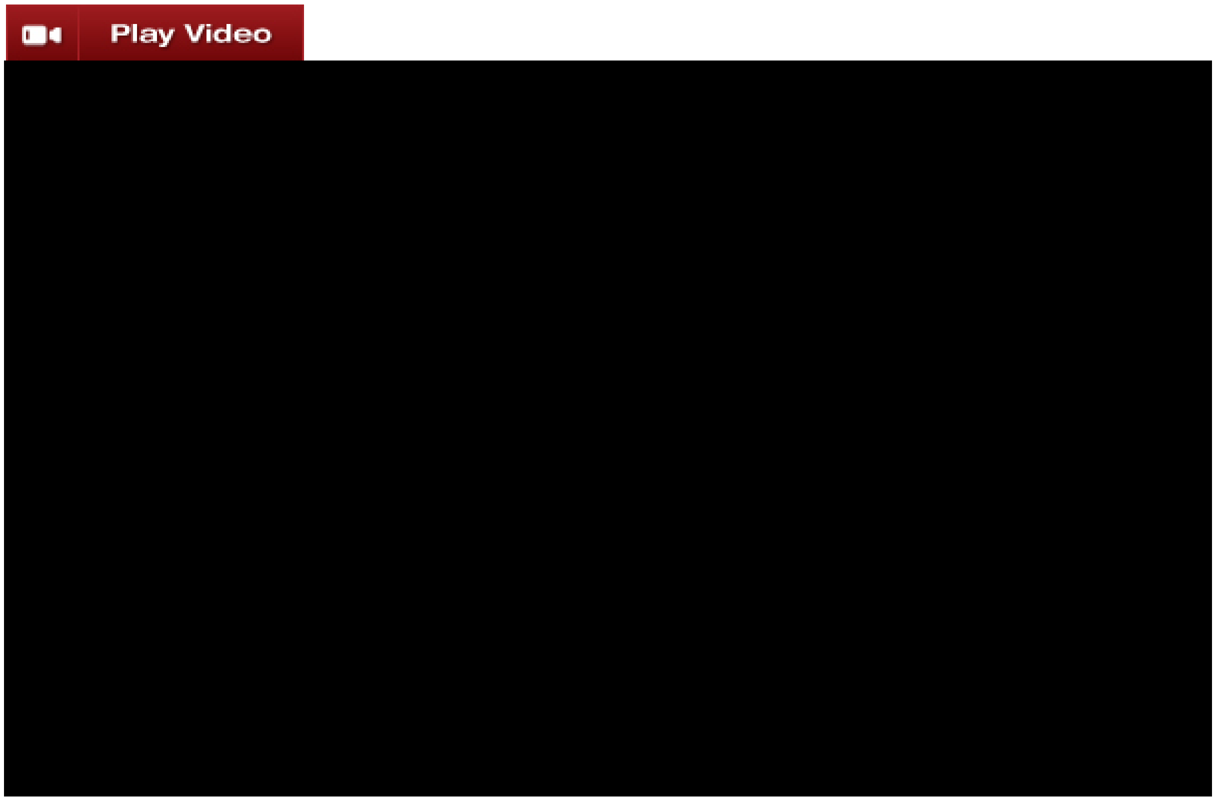
› Properties

Order Management

› Submitting Orders
› Modifying and Cancelling Orders
› Managing Positions

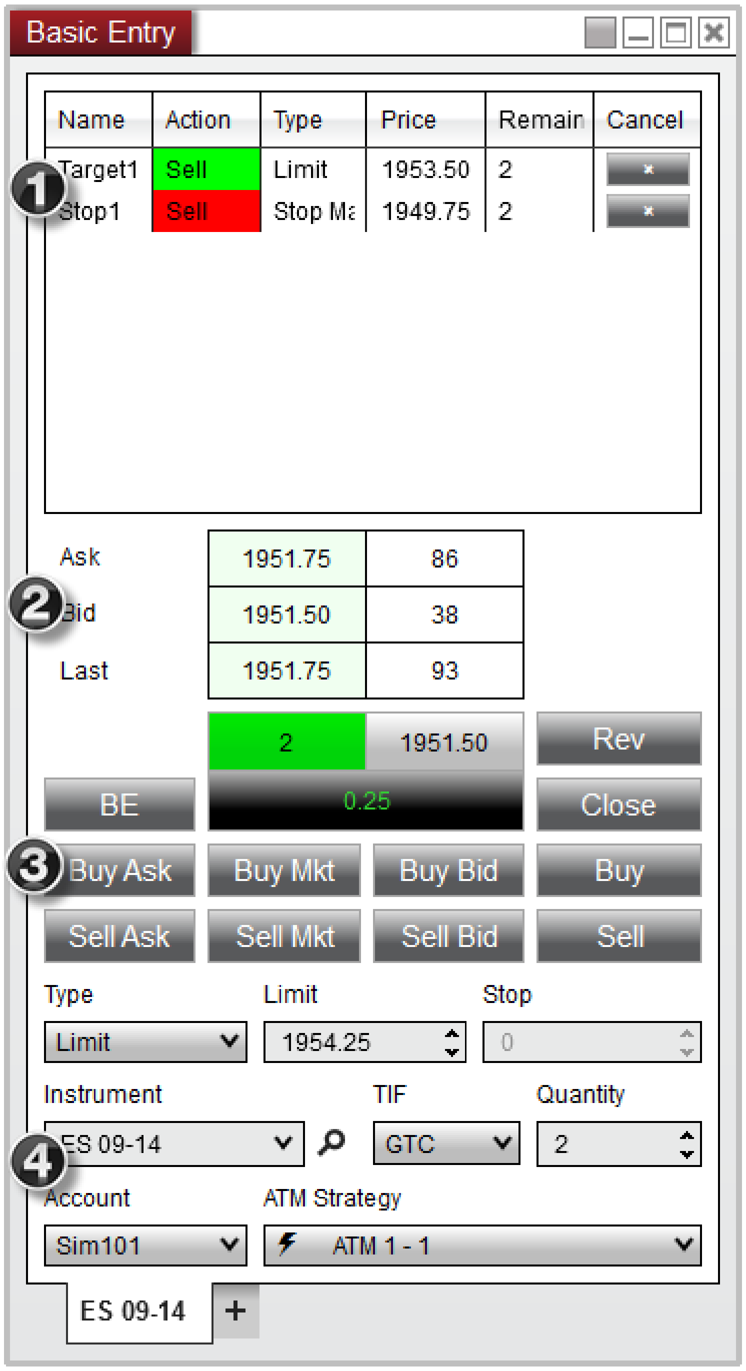
11.19.6.1 Display Overview

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**.



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



Name	Action	Type	Price	Remain	Cancel
Target1	Sell	Limit	1953.50	2	×
Stop1	Sell	Stop Ma	1949.75	2	×

Ask	1951.75	86
Bid	1951.50	38
Last	1951.75	93

2 1951.50

0.25

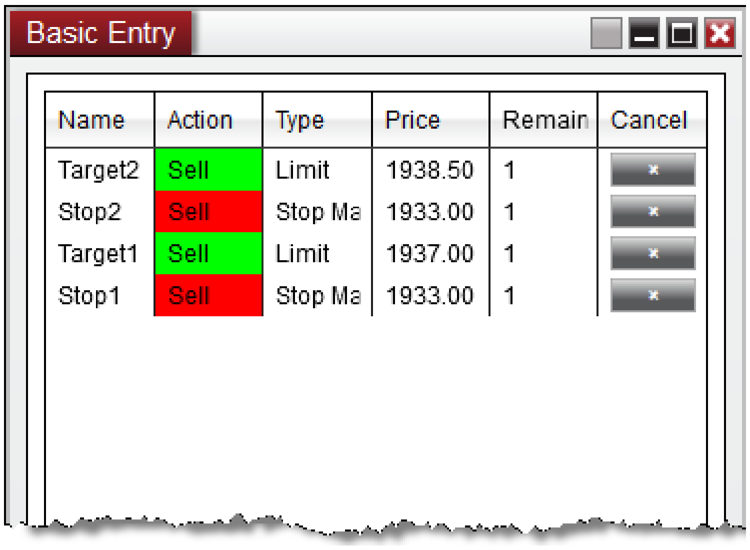
Type: Limit
Limit: 1954.25
Stop: 0
Instrument: ES 09-14
TIF: GTC
Quantity: 2
Account: Sim101
ATM Strategy: ATM 1 - 1

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.



Name	Action	Type	Price	Remain	Cancel
Target2	Sell	Limit	1938.50	1	×
Stop2	Sell	Stop Ma	1933.00	1	×
Target1	Sell	Limit	1937.00	1	×
Stop1	Sell	Stop Ma	1933.00	1	×

Column Definitions

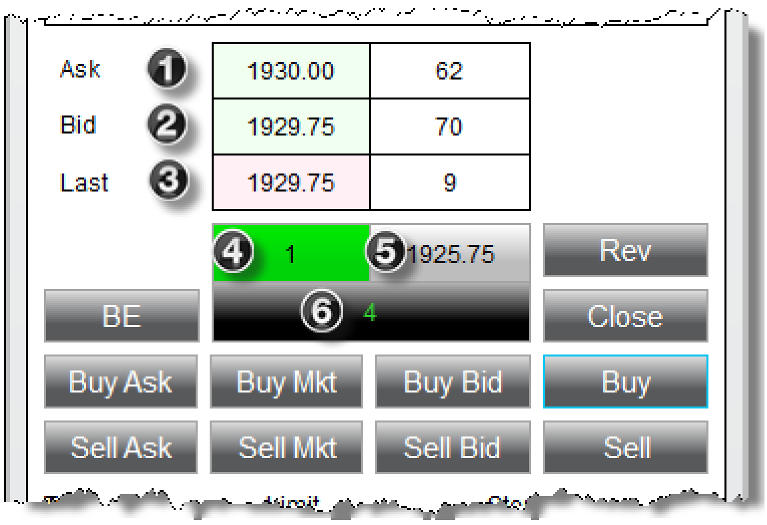
Name	Order name such as Stop1 or Target1
Action	Buy or Sell
Type	Order type
Price	Order price
Remaining	Number of contracts/shares remaining to be filled
Cancel	Cancels the order(s)

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.

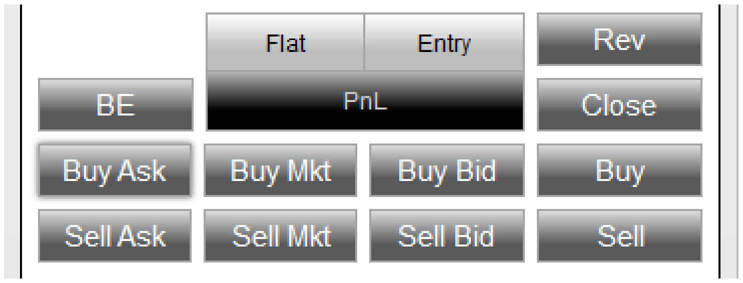
1. Current best ask price and size
2. Current best bid price and size
3. Last traded price and size
4. Market position (FLAT or green background with position size for long, red background for short)
5. Position average entry price
6. Unrealized profit or loss for current position (Clicking on this cell 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	Adjusts any open stop orders opposite of your

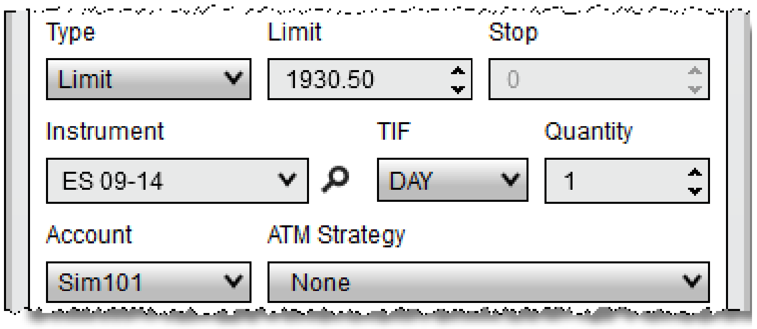
(Break-even)	position to the positions average entry price
Buy Ask	Submits a Buy Limit order at the current ask price
Sell Ask	Submits a Sell Limit order at the current ask price
Buy Mkt (Market)	Submits a Buy Market order at the current market price
Sell Mkt (Market)	Submits a Sell Market order at the 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
Instrume nt	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

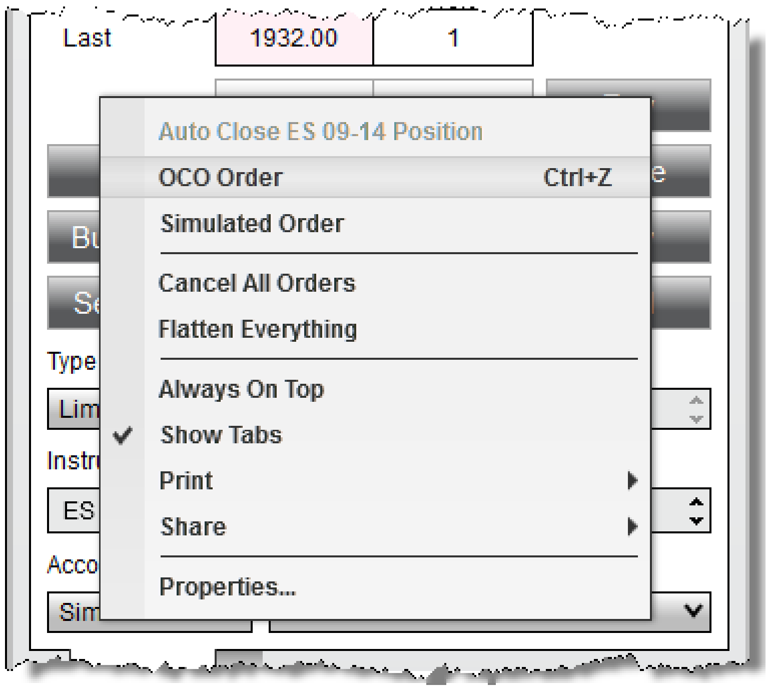
Understanding the right click menu

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**

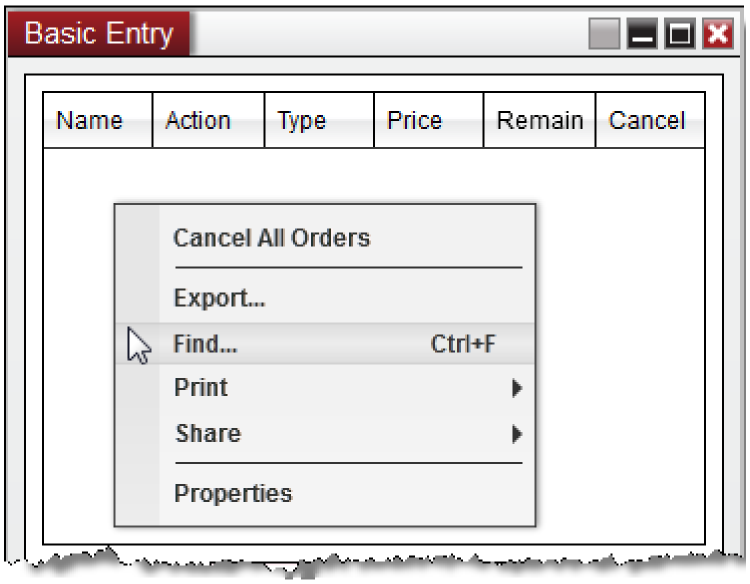


Auto Close Position	Automatically Closes the current instruments position at a specified time
OCO Order	Enables/Disables the OCO (one cancels other) function for a pending order
Simulated Order	Enables/Disables the Simulated Order functionality for a pending order
Cancel All Orders	Cancels all active 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

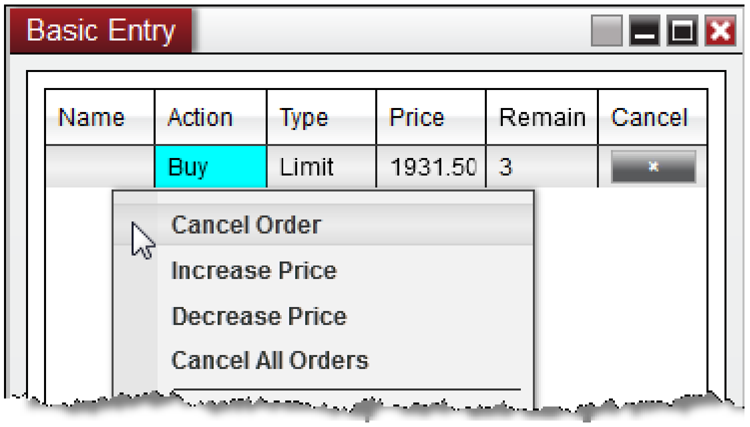
Right clicking in an empty grid will bring up a number of general menu items specific to the **Order Grid**



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.



Name	Action	Type	Price	Remain	Cancel
	Buy	Limit	1931.50	3	×

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

11.19.6.2 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.

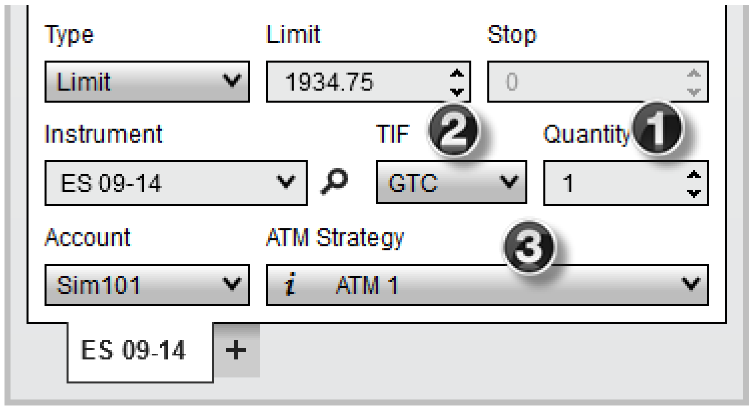
How to Select an Account

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

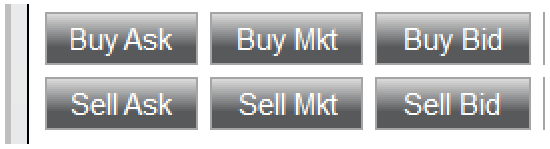


Type	Limit	Stop
Limit	1934.75	0
Instrument	TIF	Quantity
ES 09-14	GTC	1
Account	ATM Strategy	
Sim101	ATM 1	

How to submit orders with quick buttons

Quick Buttons

You can enter orders rapidly by pressing on any one of the quick order buttons.



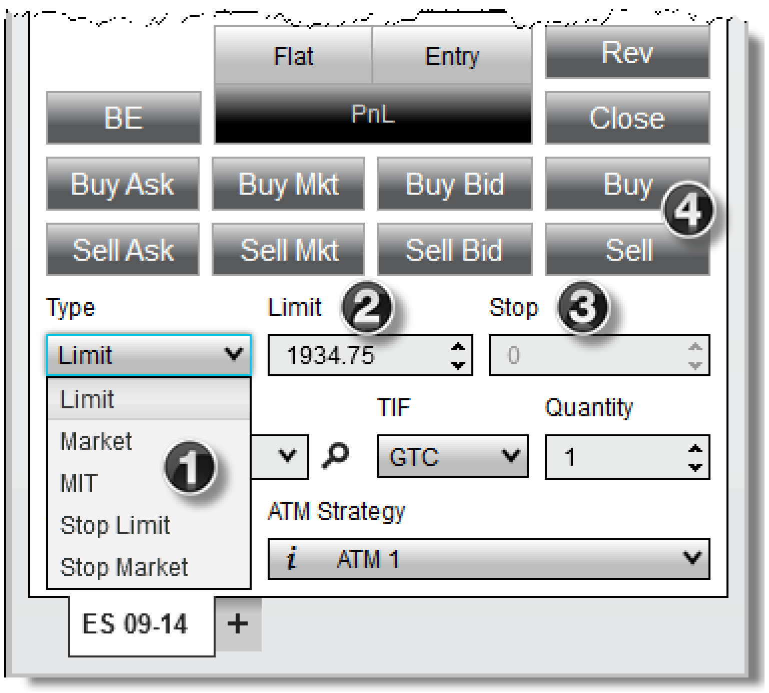
Buy Ask	Submits a Buy Limit order at the current ask price
Sell Ask	Submits a Sell Limit order at the current ask price
Buy Mkt (Market)	Submits a Buy Market order at the current market price
Sell Mkt (Market)	Submits a Sell Market order at the current Market price
Buy Bid	Submits a Buy Limit order at 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.

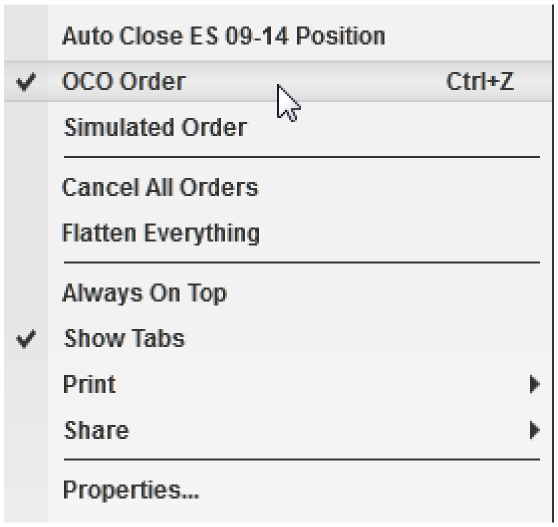
Understanding the OCO (One Cancel Other) function

OCO Orders (One Cancels Other)

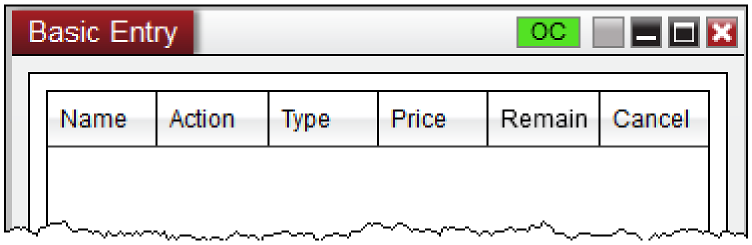
Stop Loss and **Profit Target** orders (submitted automatically via an ATM Strategy) are always sent as **OCO**, however, you can submit entry or exit orders as **OCO** orders as well. Why? The market may be trading in a channel and you wish to sell at resistance or buy at support, whichever comes first 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:

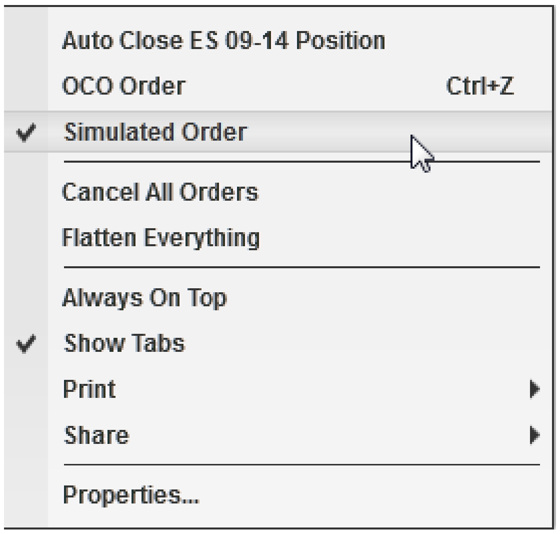
- Right clicking in the **Basic Entry** window and selecting the menu item "**OCO Order**" to enable the OCO function
- For your first order, select the desired option from the "**ATM Strategy**" drop down list
- Submit your stop order to buy above the market
- For your second order, select the desired 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)

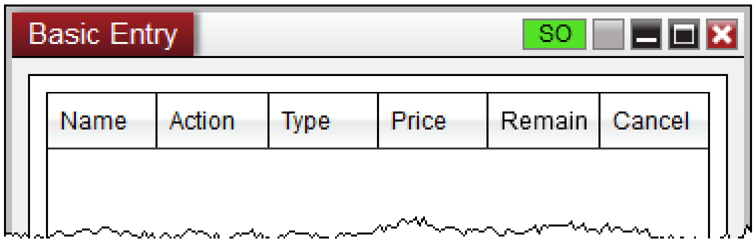
Simulated Stop Orders (Simulated Order)

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.



11.19.6.3 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.

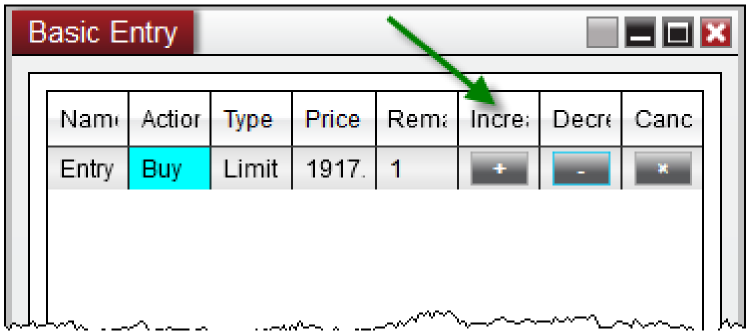
Tip:
Hold down the CTRL key when scrolling in the **price editor** to change the price in steps of 10 tick increments.

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**"



Nam	Actior	Type	Price	Rem	Incre	Decr	Canc
Entry	Buy	Limit	1917.	1	+	-	×

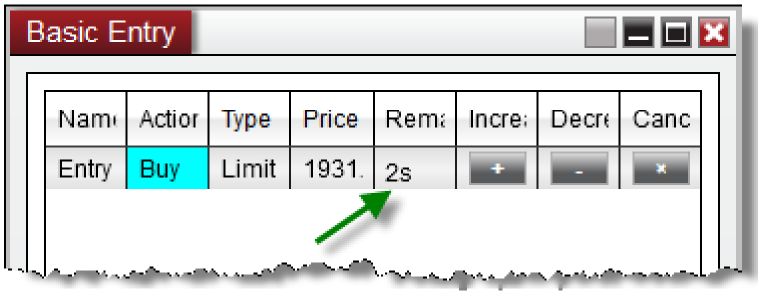
1. You can increase the price of an order in one tick increments by left mouse clicking on the "**+**" button. Holding the CTRL key down while pressing the "**+**" button will modify the order by 5 tick increments, and holding the ALT key will modify the order by 10 tick increments.

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

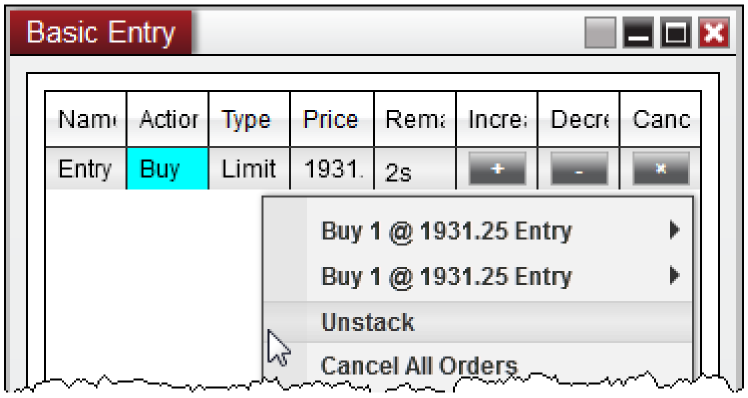
You can change the size of an order by left clicking in the "**Remaining**" column, typing in a new quantity value and pressing the "Enter" key on your keyboard.

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.



Nam	Actior	Type	Price	Rem	Incre	Decr	Canc
Entry	Buy	Limit	1931.	2s	+	-	×

If you would like to break up these orders to manage individually, you can right click on the Order and select "**Unstack**"



Cancelling orders

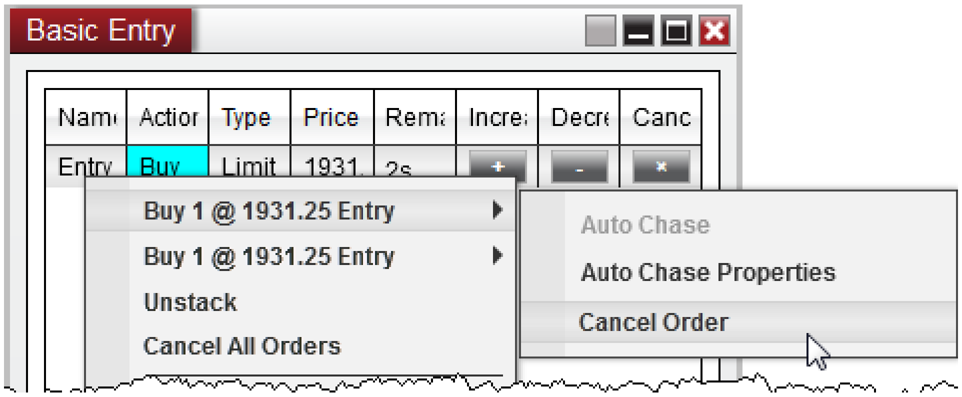
Cancelling an Individual Order

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**"



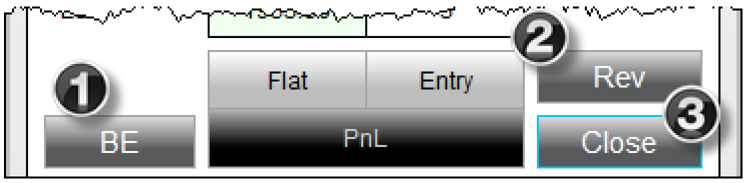
11.19.6.4 Managing Positions

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.

11.19.6.5 Properties

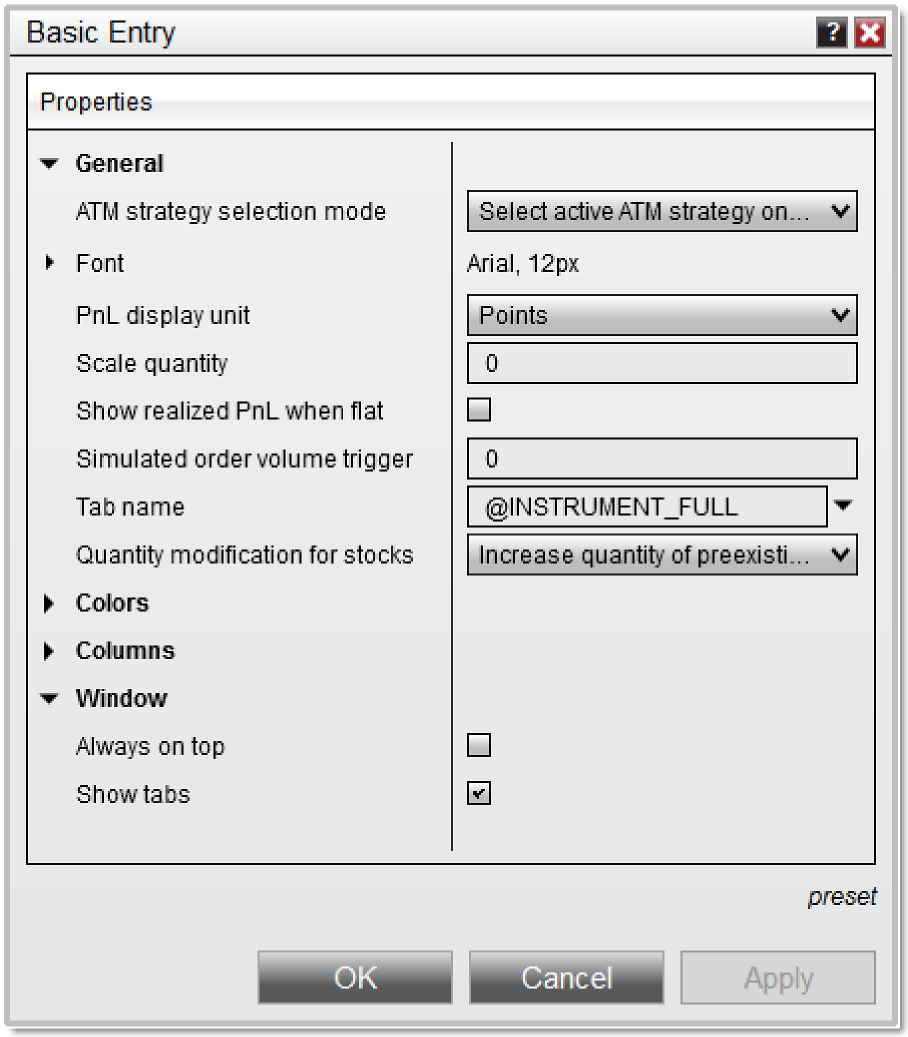
The Basic Entry order entry window is highly efficient by design but can also be customized to your preferences through the Basic Entry Properties menu.

▽ 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:



General
ATM strategy selection mode: Select active ATM strategy on...
Font: Arial, 12px
PnL display unit: Points
Scale quantity: 0
Show realized PnL when flat: ☐
Simulated order volume trigger: 0
Tab name: @INSTRUMENT_FULL
Quantity modification for stocks: Increase quantity of preexisti...
Colors
Columns
Window
Always on top: ☐
Show tabs: ✓

preset

Property Definitions

General	
ATM strategy selection mode	Sets the behavior mode of the price ladder display and strategy selector (see more)
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.

11.19.7 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.

11.19.7.1 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

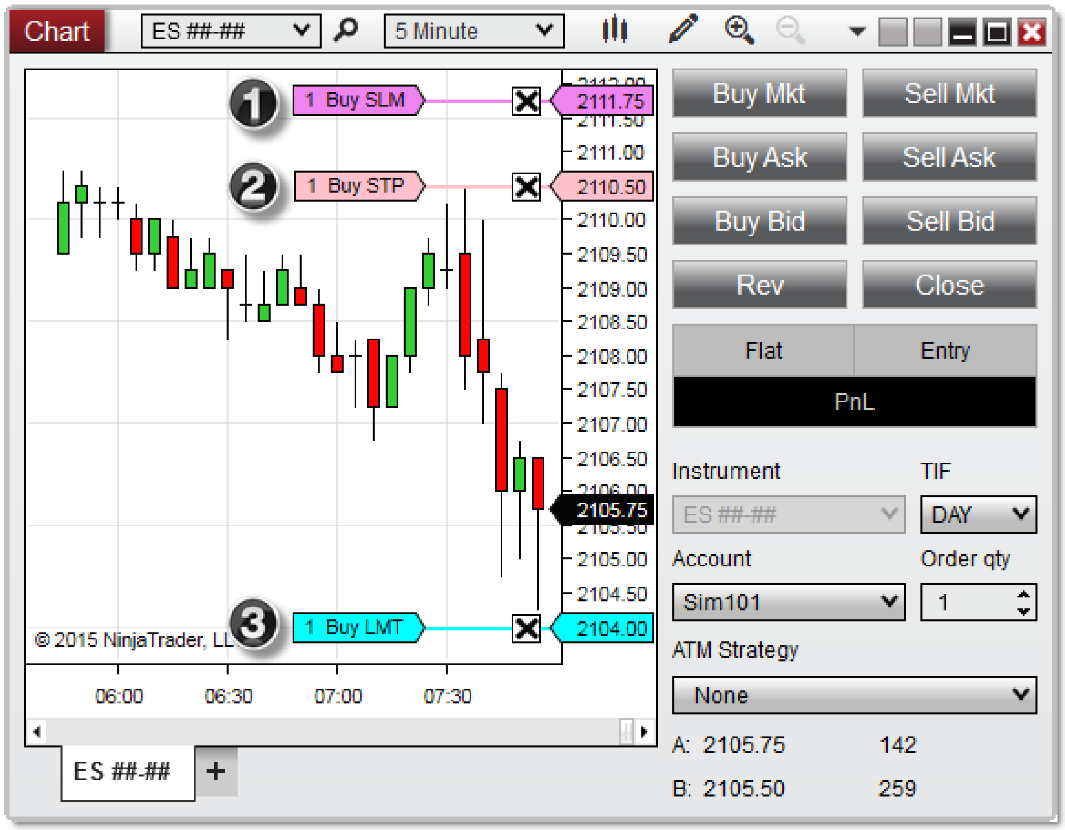
Order Display

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

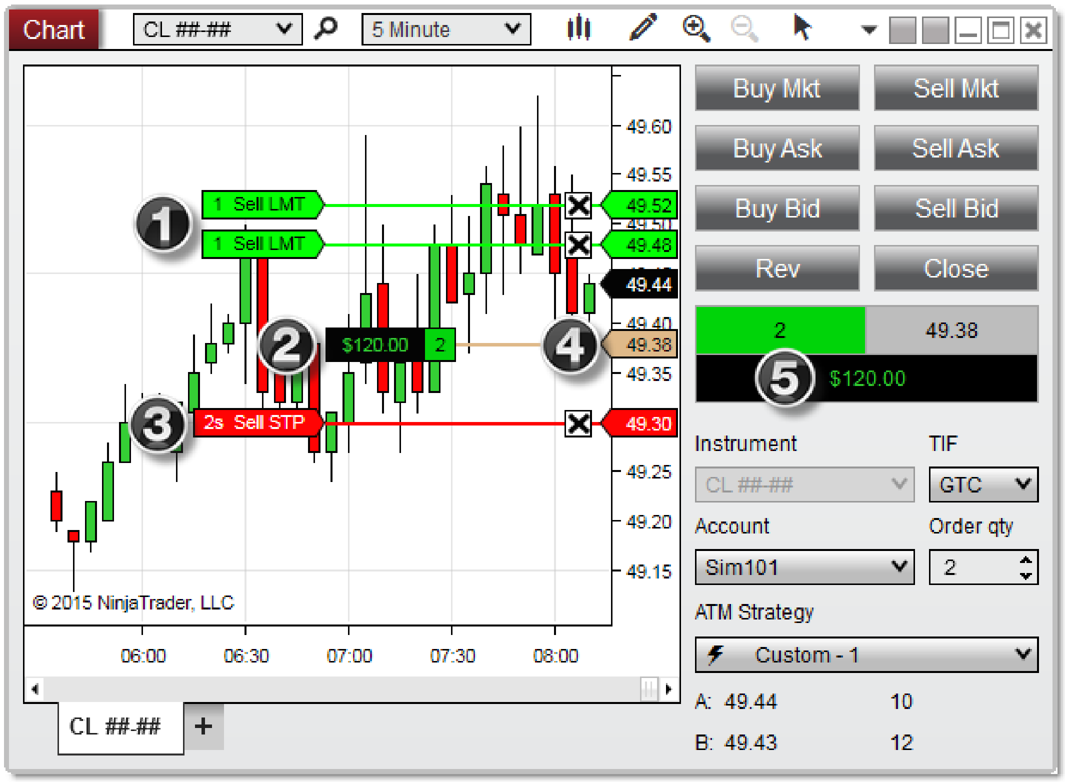
Understanding Position Display

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.



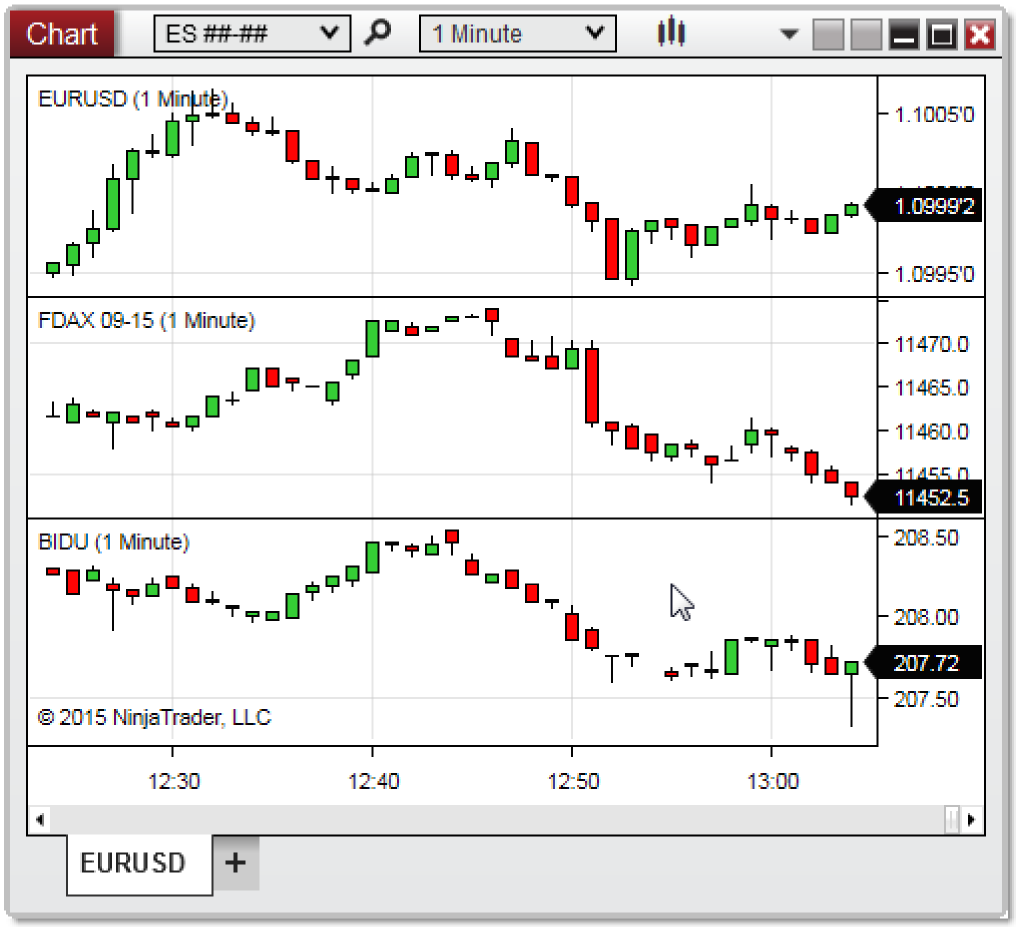
1. Two profit target orders
2. Position size and PnL flag for 2 contracts long
3. Two stop loss orders*
4. Average entry price
5. PnL in **Chart Trader** panel

* The stop loss line and flag represents two orders, as indicated by the letter "s" next to the qty number "2."

Working with multiple instruments

Multi-Series Charts

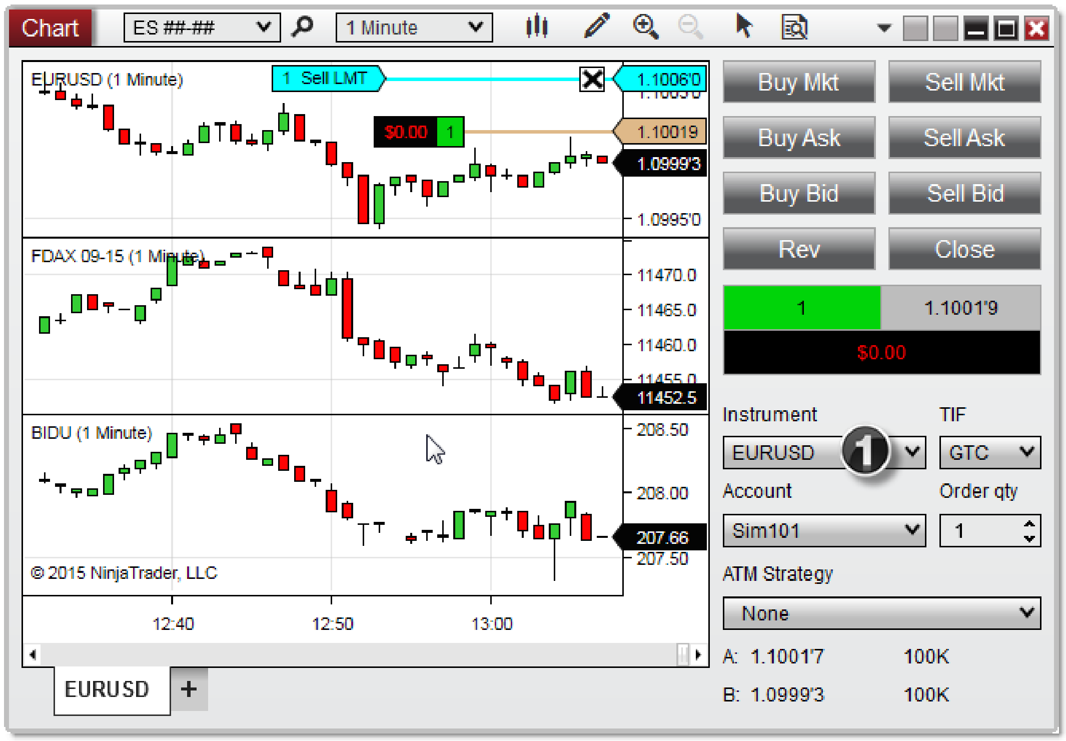
NinjaTrader charts include the ability to plot multiple instruments within a single chart window, and each individual instrument on a chart can be selected and worked with separately using **Chart Trader**. For more information on how to manage instruments on a chart, see the Working with Multiple Data Series page.



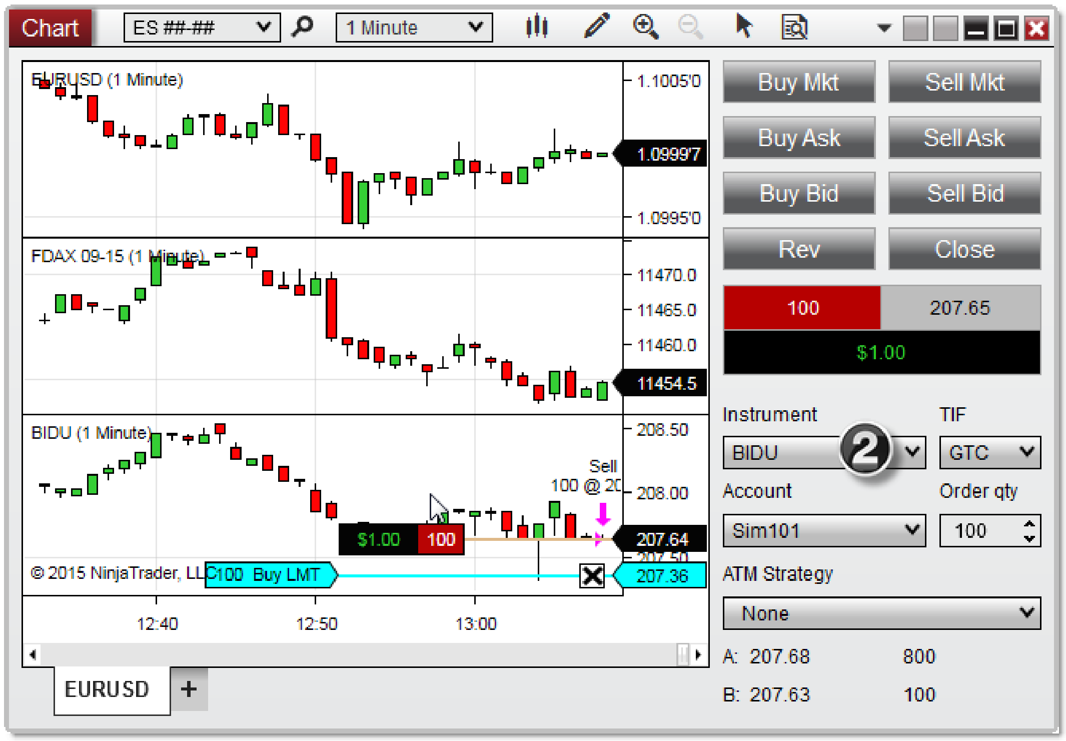
In the image above, we have applied a EURUSD instrument, an FDAX ##-## instrument, and a BIDU instrument in three separate panels of the same chart.

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.



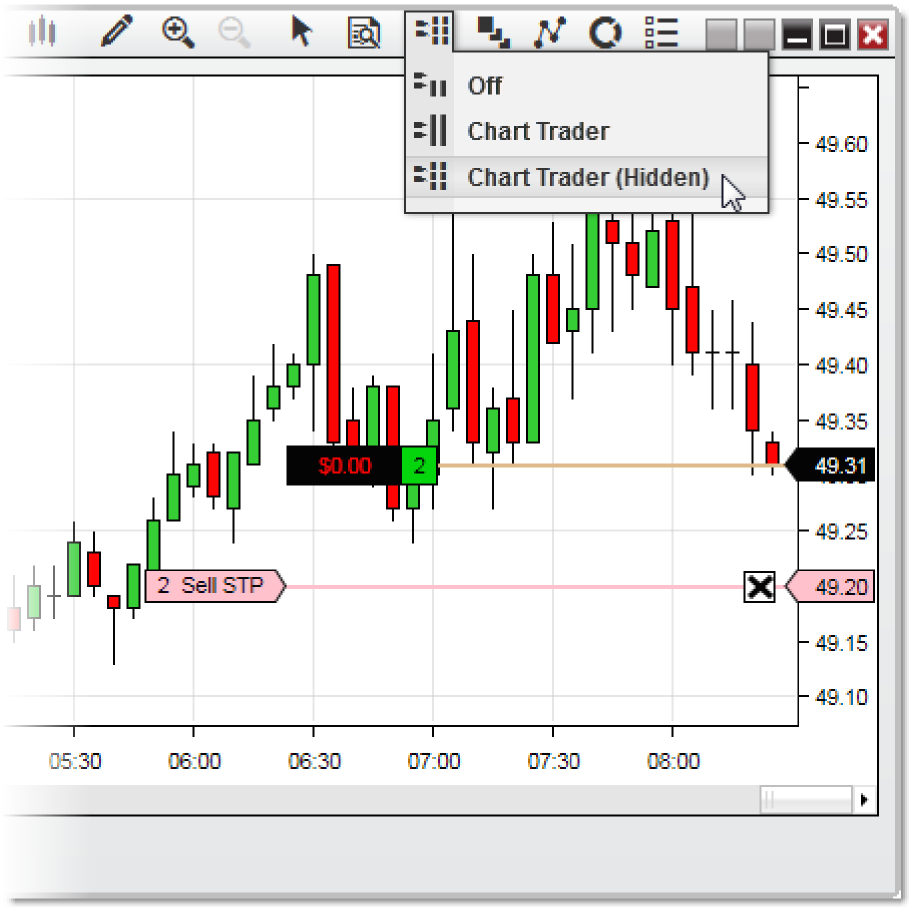
2. With BIDU selected, we see a different set of orders and positions.

11.19.7.2 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.

11.19.7.3 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

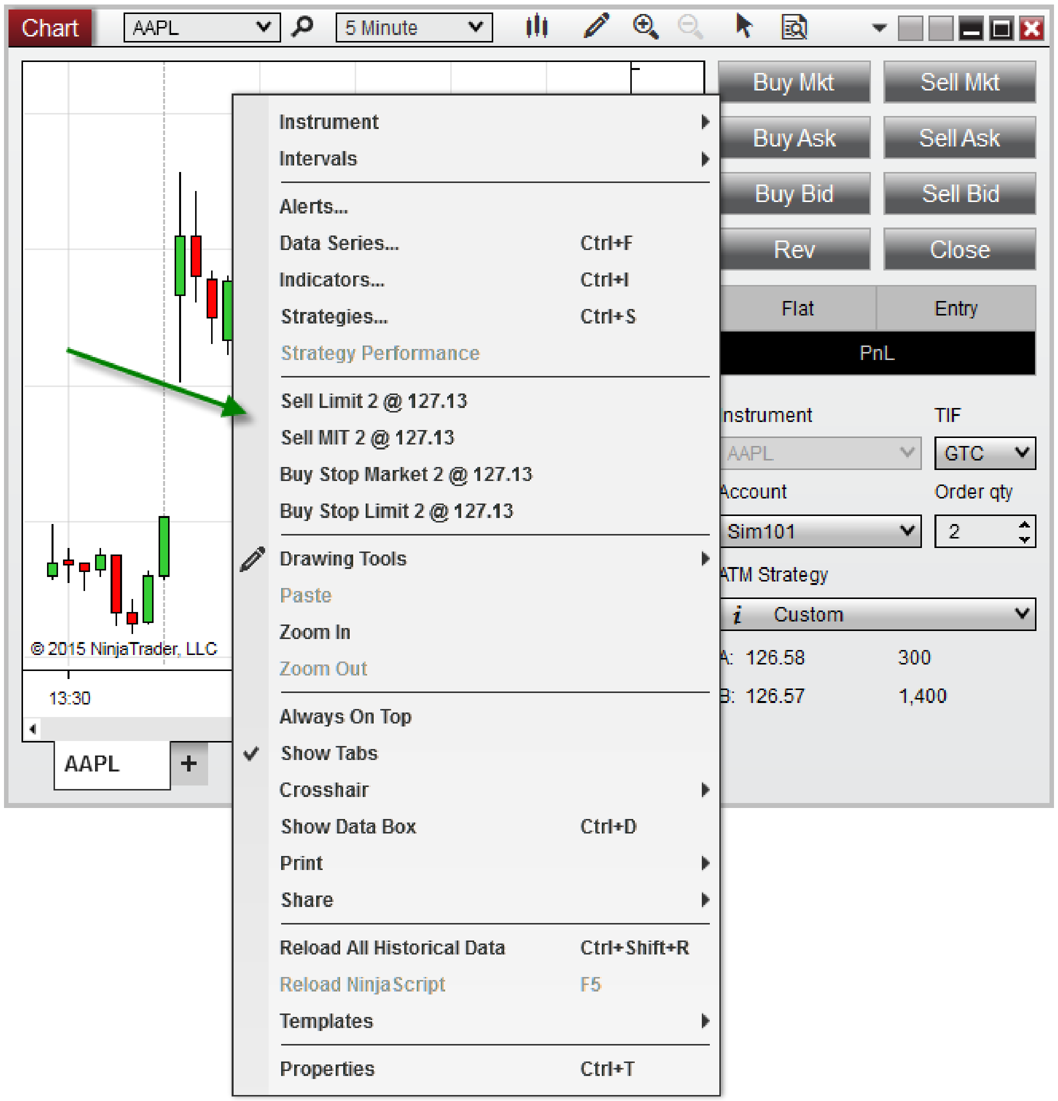
A more thorough explanation of these concepts can be found on the ATM Strategy Parameters page.

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

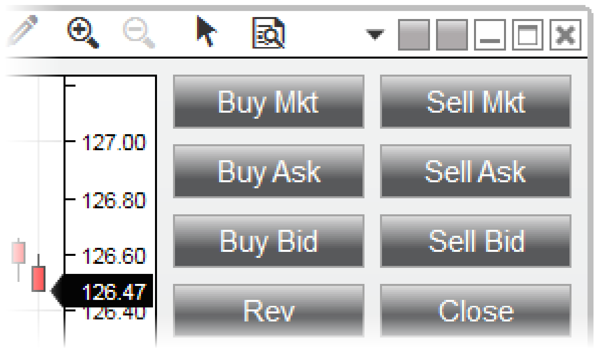
When submitting a stop-limit order, a numeric field will appear, allowing you to set the limit offset of the order (the number of ticks away at which you wish to place the Limit price of the Stop-Limit order). Either by using your mouse scroll wheel or clicking on the up/down arrows in the numeric field, 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 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.

!! images here to show the process?

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)

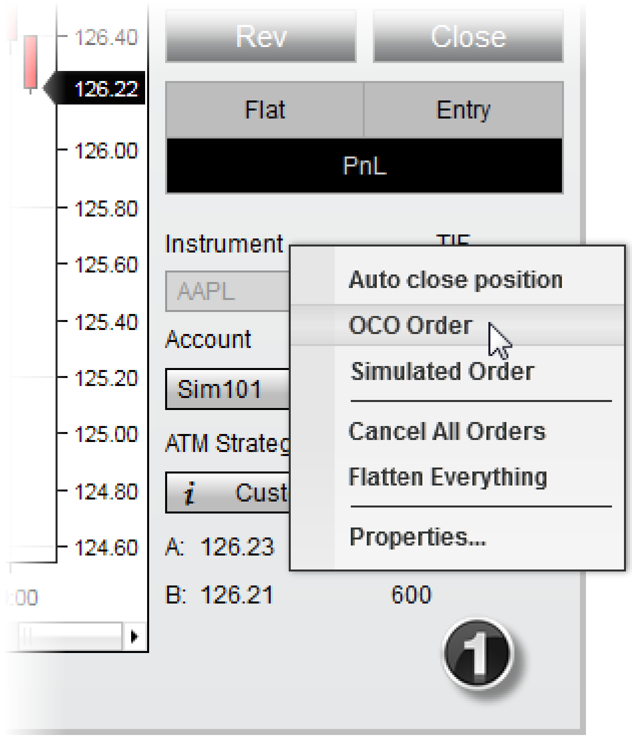
One Cancels Other functionality ties two resting order together, so that when one is canceled or filled, the other will be canceled automatically. This can be ideal for manually placing bracket entry orders, or placing Stop Loss and Profit Target orders. Stop loss and profit target orders submitted automatically 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**.

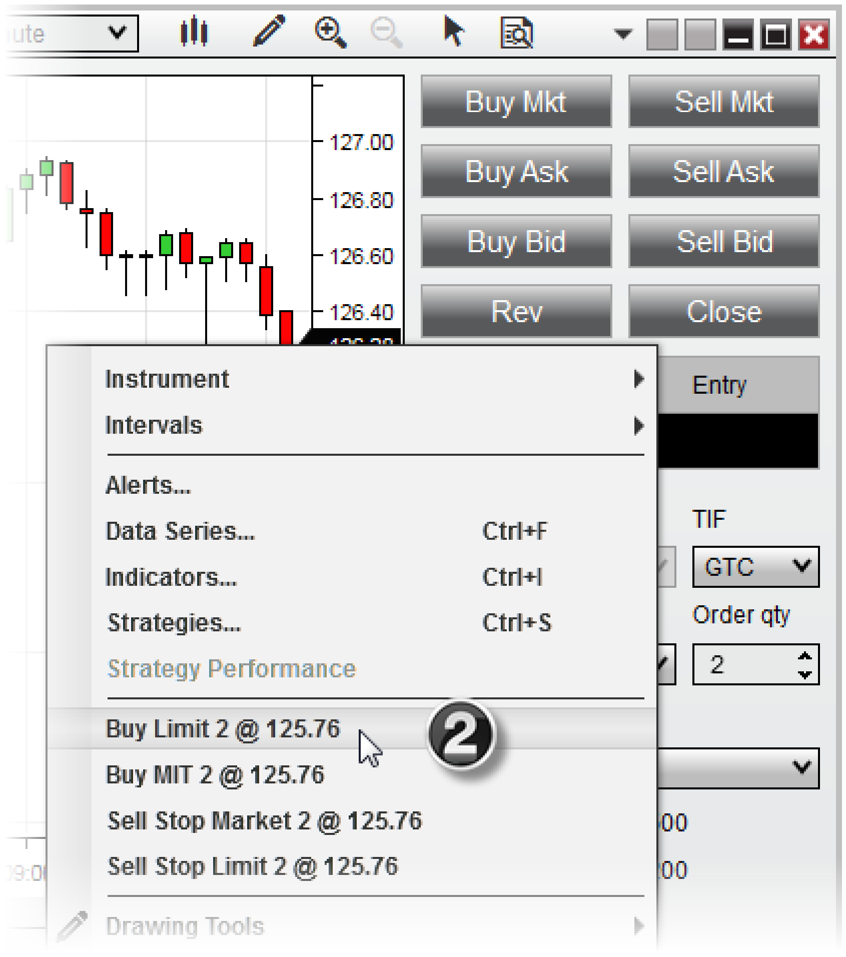
All orders placed while OCO is enabled will be part of the same OCO group. Once any order of this group is filled or cancelled, all other orders that belong to this group will be cancelled. If you want each OCO order to create it's own set of stop loss and profit target brackets, ensure that the ATM 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.

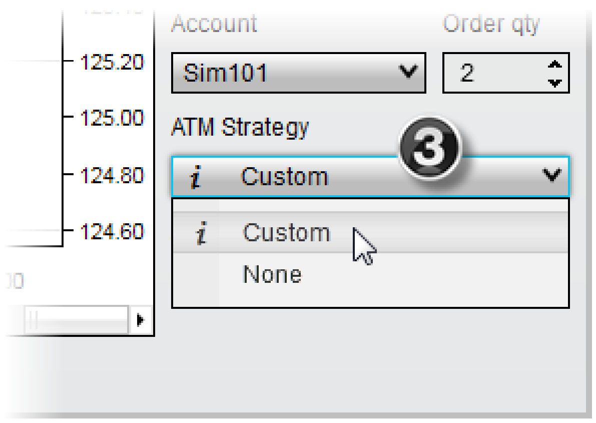
Example:
Below are steps for submitting a Sell Limit and a Buy Limit order via OCO.


126.40
126.22
126.00
125.80
125.60
125.40
125.20
125.00
124.80
124.60
Rev
Close
Flat
Entry
PnL
Instrument
TIF
AAPL
Account
Sim101
ATM Strateg
Cust
A: 126.23
B: 126.21
600
Auto close position
OCO Order
Simulated Order
Cancel All Orders
Flatten Everything
Properties...
1

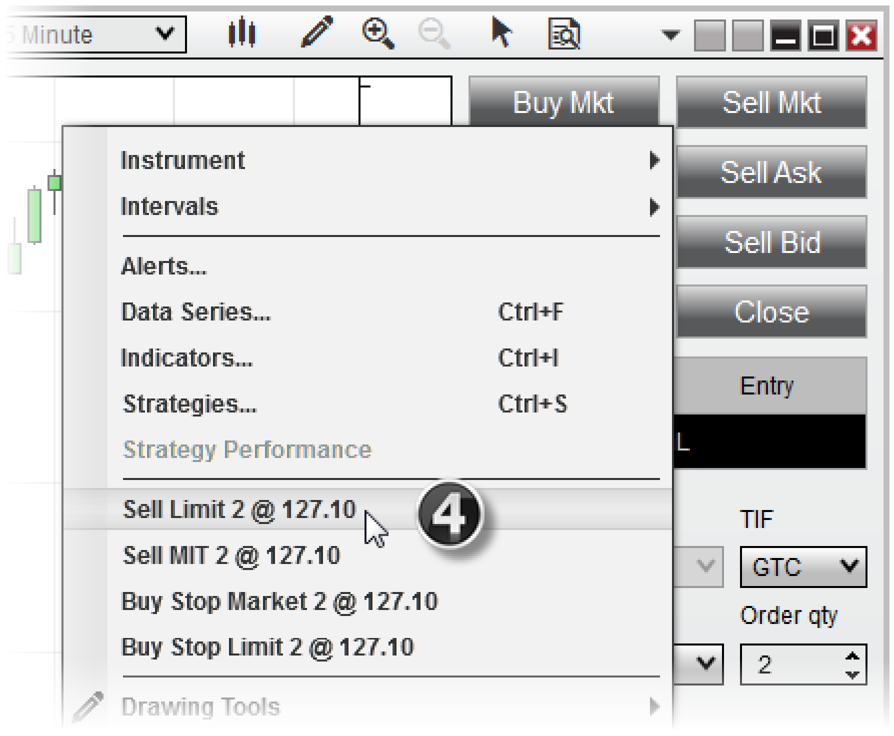
1. Enable OCO



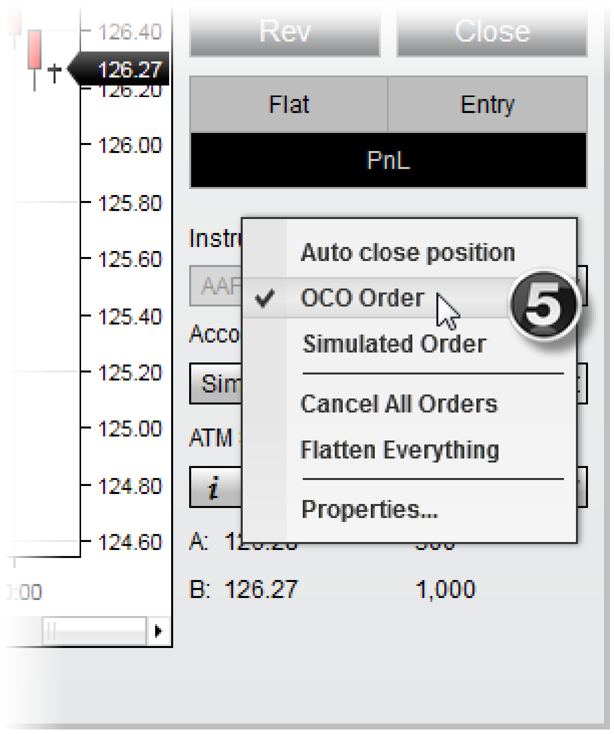
2. Place a Buy Limit order



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.

11.19.7.4 Modifying and Cancelling Orders

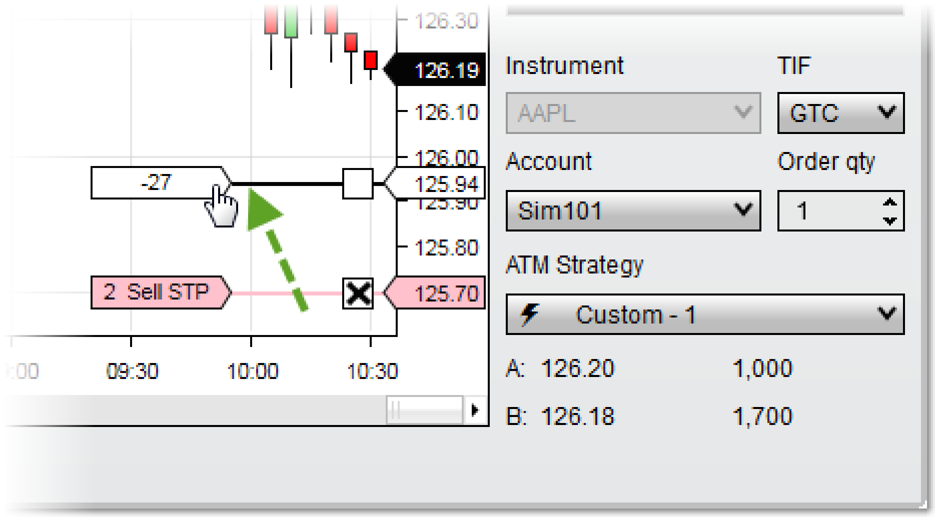
An order can be modified or canceled within a chart when **Chart Trader** is enabled.

▽ 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."


Instrument: AAPL
TIF: GTC
Account: Sim101
Order qty: 1
ATM Strategy: Custom - 1
A: 126.20 1,000
B: 126.18 1,700

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.

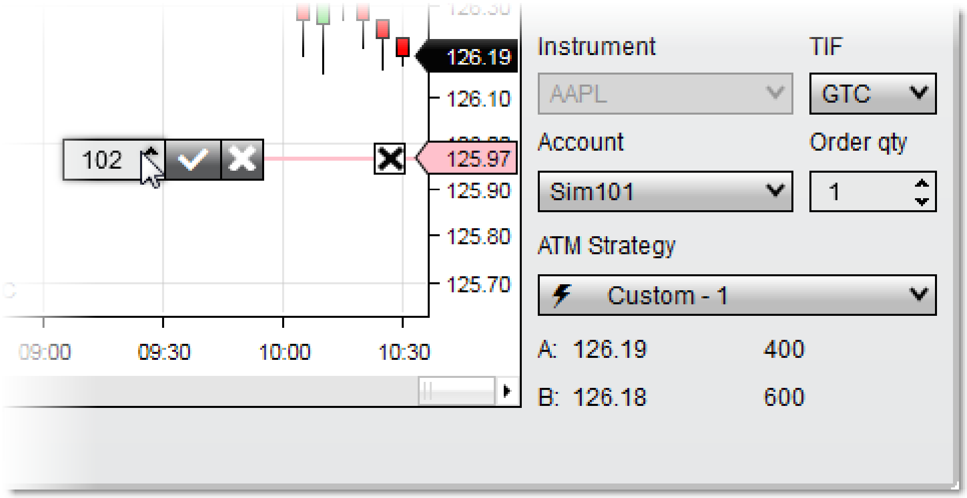
To cancel a pending order price modification, press the "Esc" key on your keyboard.

How to modify an order size

Modifying Order Size

To modify the size of an order:

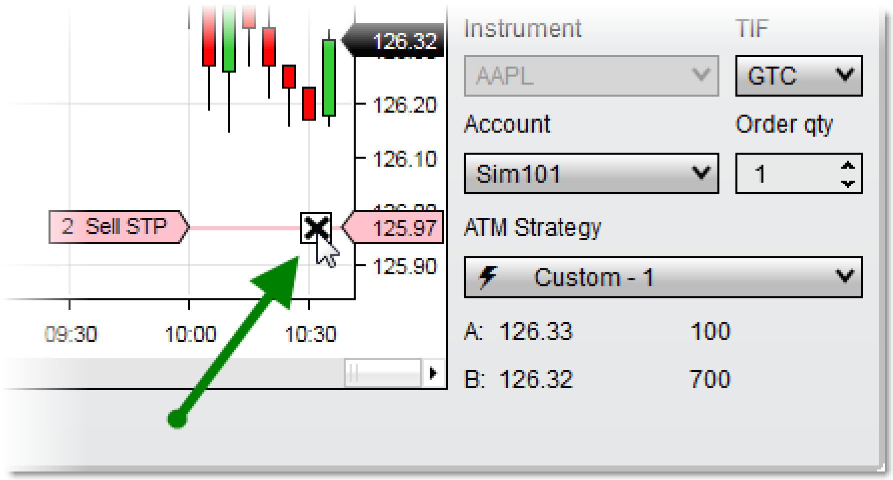
Left mouse click on the area of the order label that displays the order quantity. An order size modification control will appear. Modify the quantity in the quantity field by using either the up/down arrows, the mouse scroll wheel, or by typing the desired quantity. When done, click the check mark to 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.



11.19.7.5 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

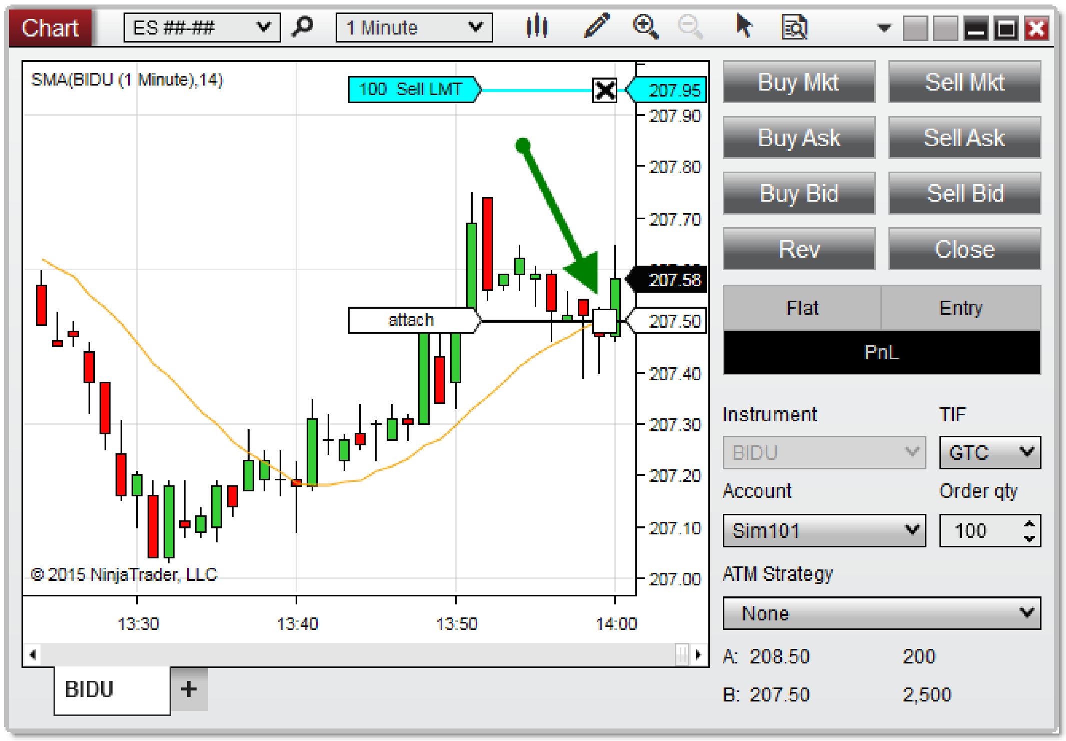
▽ How to attach an order to an indicator

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.

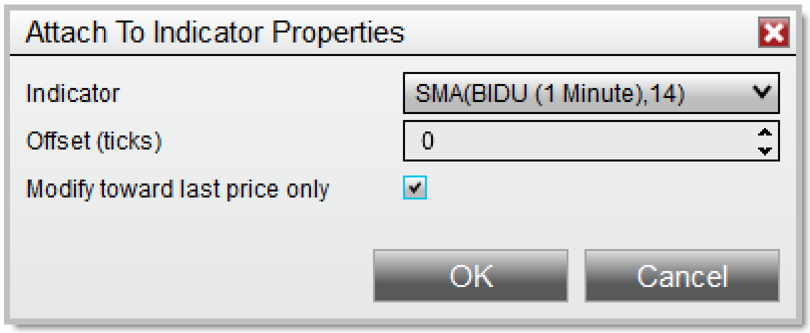
Attach to Indicator properties

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:


Attach To Indicator Properties

Indicator	SMA(BIDU (1 Minute),14)
Offset (ticks)	0
Modify toward last price only	✓

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

11.19.7.6 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

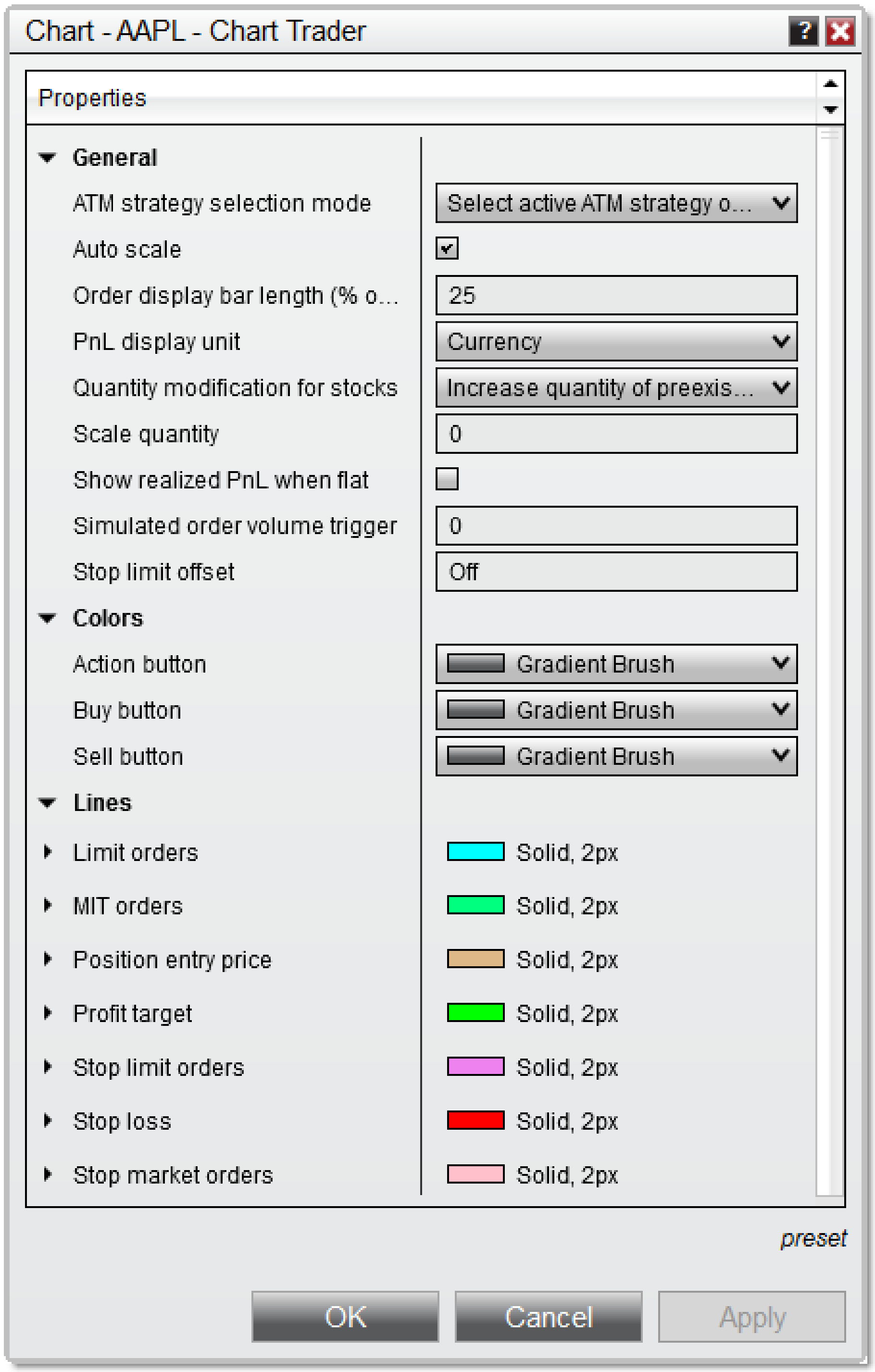
To access the **Chart Trader Properties** window:

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:



General	
ATM strategy selection mode	Select active ATM strategy o...
Auto scale	✓
Order display bar length (% o...	25
PnL display unit	Currency
Quantity modification for stocks	Increase quantity of preexis...
Scale quantity	0
Show realized PnL when flat	☐
Simulated order volume trigger	0
Stop limit offset	Off
Colors	
Action button	Gradient Brush
Buy button	Gradient Brush
Sell button	Gradient Brush
Lines	
Limit orders	Solid, 2px
MIT orders	Solid, 2px
Position entry price	Solid, 2px
Profit target	Solid, 2px
Stop limit orders	Solid, 2px
Stop loss	Solid, 2px
Stop market orders	Solid, 2px

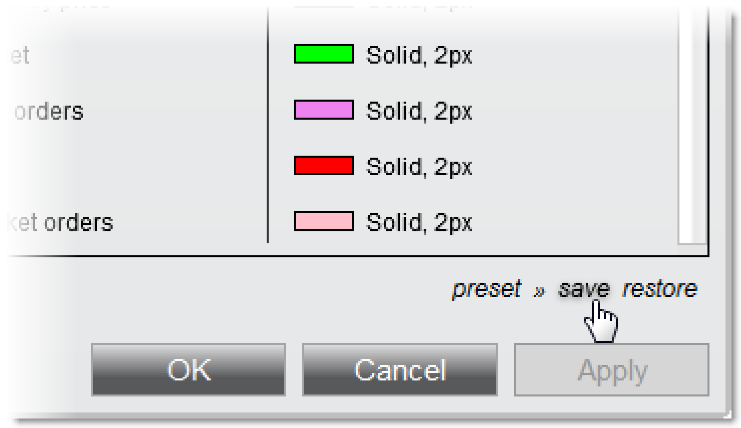
ATM	Sets the behavior of the **ATM Strategy**

Strategy selection mode	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
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)
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.
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.

11.19.8 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

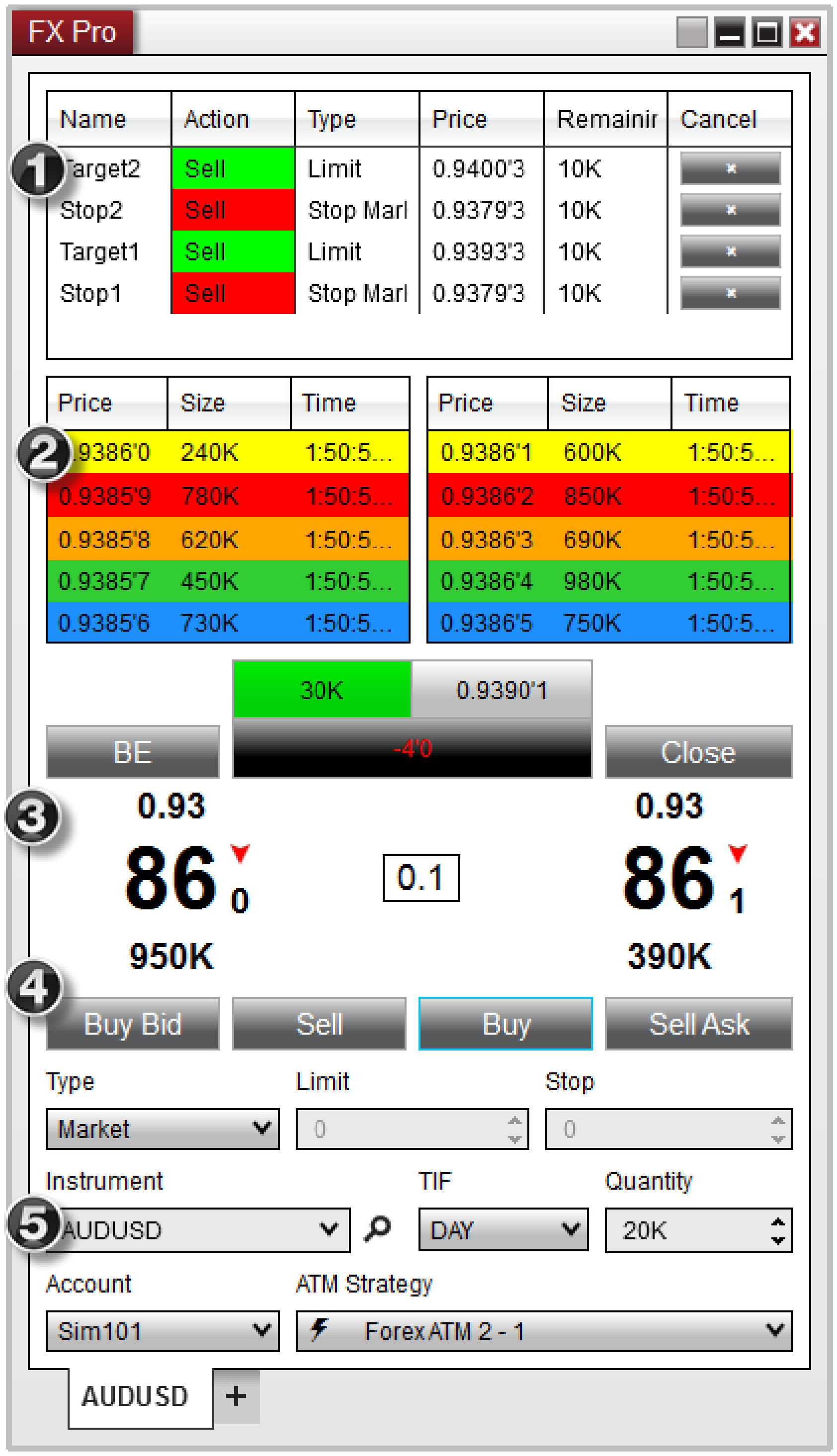
- Submitting Orders
- Modifying and Cancelling Orders
- Managing Positions

11.19.8.1 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



Name	Action	Type	Price	Remainir	Cancel
Target2	Sell	Limit	0.9400'3	10K	×
Stop2	Sell	Stop Marl	0.9379'3	10K	×
Target1	Sell	Limit	0.9393'3	10K	×
Stop1	Sell	Stop Marl	0.9379'3	10K	×

Price	Size	Time	Price	Size	Time
0.9386'0	240K	1:50:5...	0.9386'1	600K	1:50:5...
0.9385'9	780K	1:50:5...	0.9386'2	850K	1:50:5...
0.9385'8	620K	1:50:5...	0.9386'3	690K	1:50:5...
0.9385'7	450K	1:50:5...	0.9386'4	980K	1:50:5...
0.9385'6	730K	1:50:5...	0.9386'5	750K	1:50:5...

30K 0.9390'1
BE -4'0 Close
0.93 86 0 950K
0.1
0.93 86 1 390K
Buy Bid Sell Buy Sell Ask

Type: Market
Limit: 0
Stop: 0
Instrument: AUDUSD
TIF: DAY
Quantity: 20K
Account: Sim101
ATM Strategy: Forex ATM 2 - 1

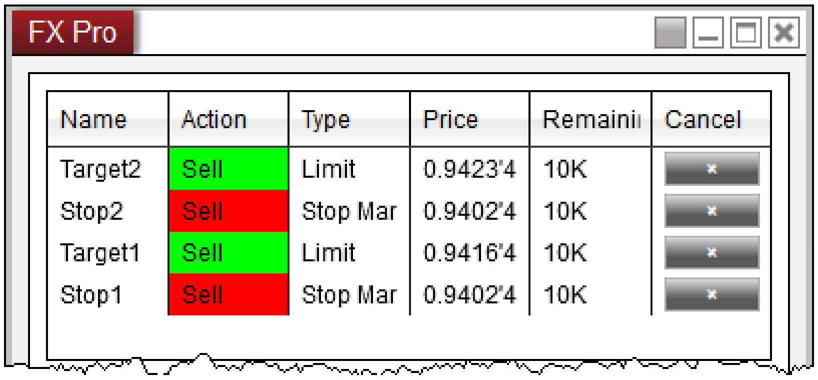
AUDUSD +

Please see the sections below for more information on each of these sections.

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.



Name	Action	Type	Price	Remainii	Cancel
Target2	Sell	Limit	0.9423'4	10K	×
Stop2	Sell	Stop Mar	0.9402'4	10K	×
Target1	Sell	Limit	0.9416'4	10K	×
Stop1	Sell	Stop Mar	0.9402'4	10K	×

Column Definitions

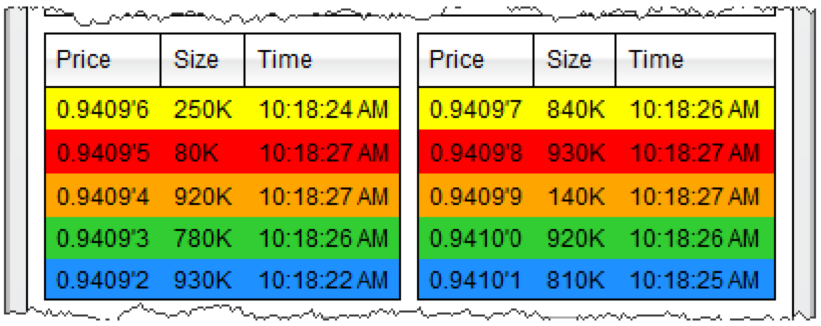
Name	Order name such as Stop1 or Target1
Action	Buy or Sell
Type	Order type
Price	Order price
Remainin g	Number of contracts/shares remaining to be filled
Cancel	Cancels the order(s)

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

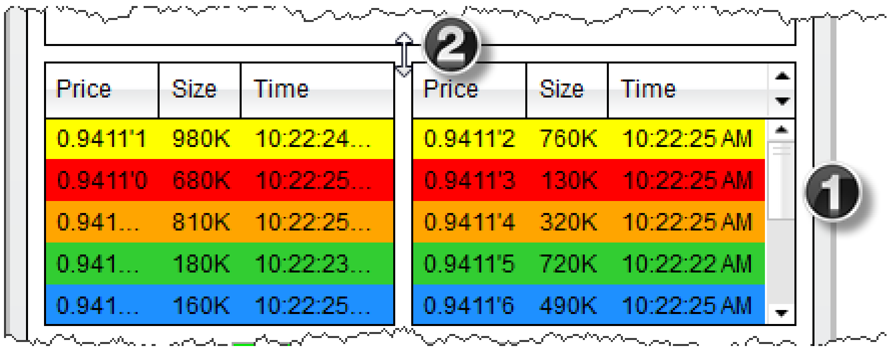
If a price is at a sub pip level, the sub-pip value is displayed as a value after an apostrophe, as in the following example in which the sub-pip value is highlighted in red.

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**



Price	Size	Time	Price	Size	Time
0.9411'1	980K	10:22:24...	0.9411'2	760K	10:22:25 AM
0.9411'0	680K	10:22:25...	0.9411'3	130K	10:22:25 AM
0.941...	810K	10:22:25...	0.9411'4	320K	10:22:25 AM
0.941...	180K	10:22:23...	0.9411'5	720K	10:22:22 AM
0.941...	160K	10:22:25...	0.9411'6	490K	10:22:25 AM

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.

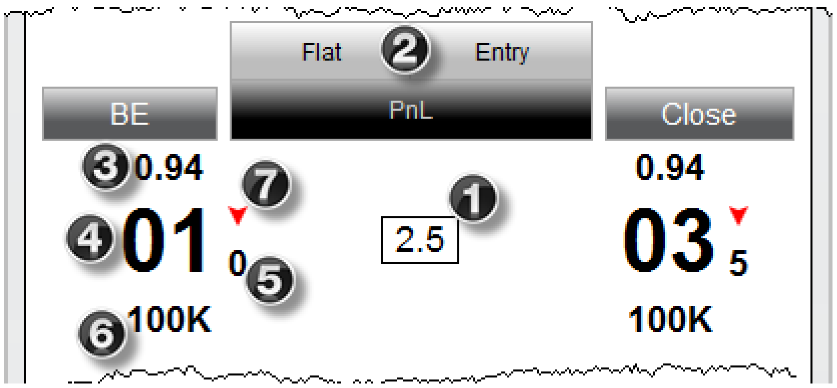
Understanding the market display section

Market Display

The Market Display panel shows the inside bid and ask along with current position information.

Market Display Definitions

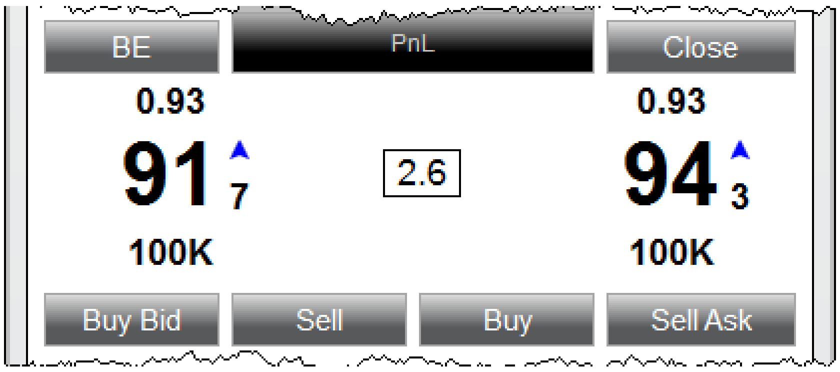
1. The current spread between the best bid and best ask (the image below is showing a spread of 2.5 pips)
2. Position information
3. The handle of the current bid (current ask is on the right side of the spread)
4. The current bid
5. Tenth-pip value. In the image below, the current bid is 0.9401 and 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.

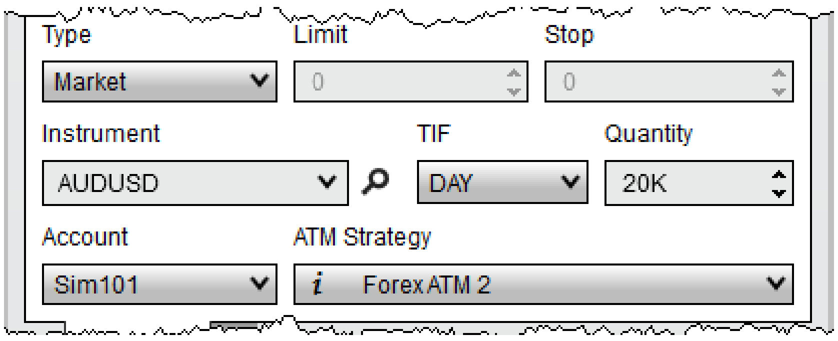
Buy Bid	Submits a Buy Limit order at the best bid price
Sell	Submits a Sell order based on the current Order Controls configured
Buy	Submits a Buy order based on the current Order Controls configured
Sell Ask	Submits a Sell Limit order at the best ask price

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	Market
Limit	0
Stop	0
Instrument	AUDUSD
TIF	DAY
Quantity	20K
Account	Sim101
ATM Strategy	Forex ATM 2

Type	Selects the order Type to be submitted
Limit	Sets the order Limit price, if applicable
Stop	Sets the order Stop Price, if applicable
Instrume nt	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

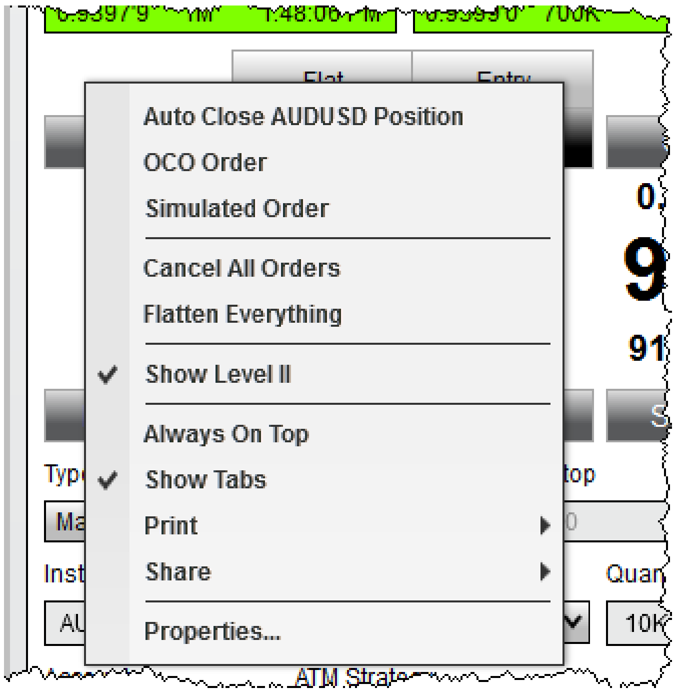
Understanding the right click menu

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**

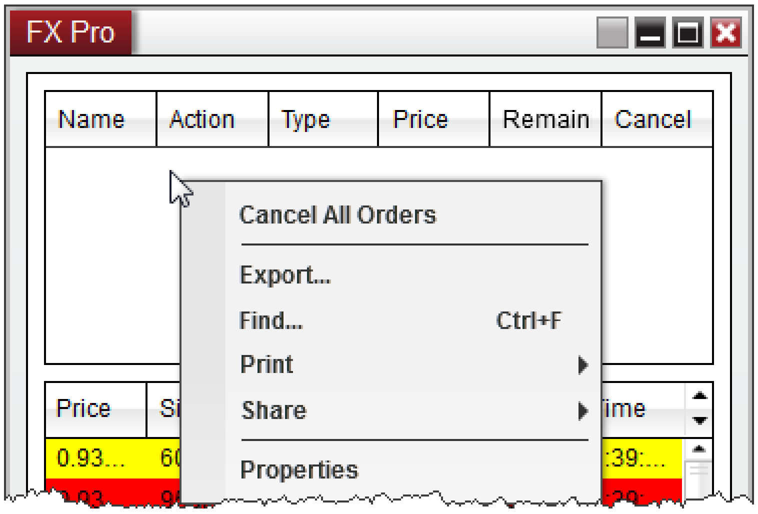


Auto	Automatically Closes the current instruments

Close Position	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

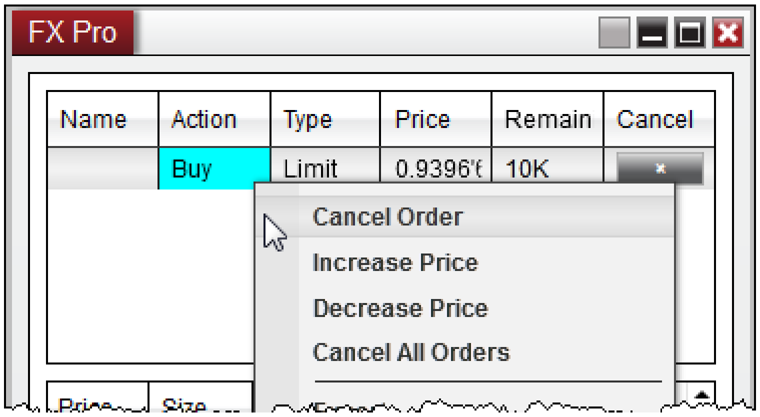
Order Grid Control Right Click Menu

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
Propertie s	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

11.19.8.2 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.

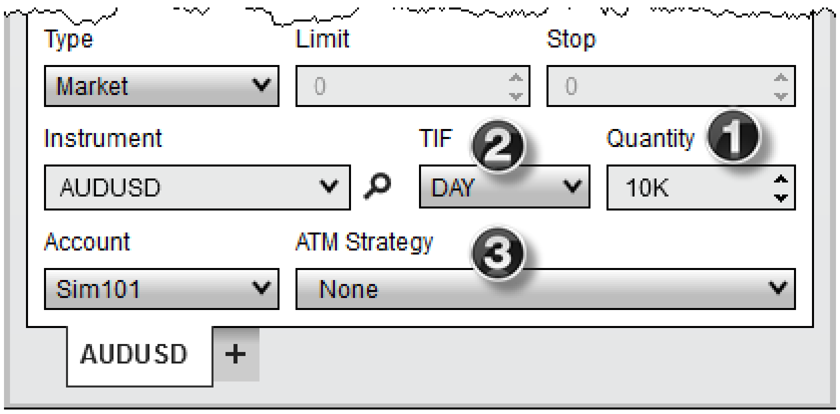
How to Select an Account

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

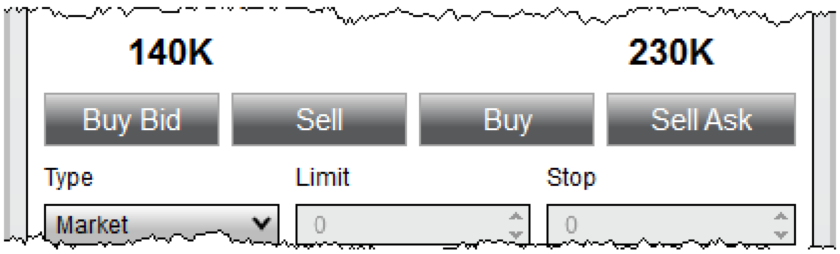
1. Set the order "**Quantity**" field (info)
2. Set the **"TIF"** (Time in Force) field (info)
3. Set the **"ATM Strategy**" (info)
4. Enter an order with any of the methods described below



How to submit orders with quick buttons

Quick Buttons

You can enter orders rapidly by pressing on any one of the quick order buttons.



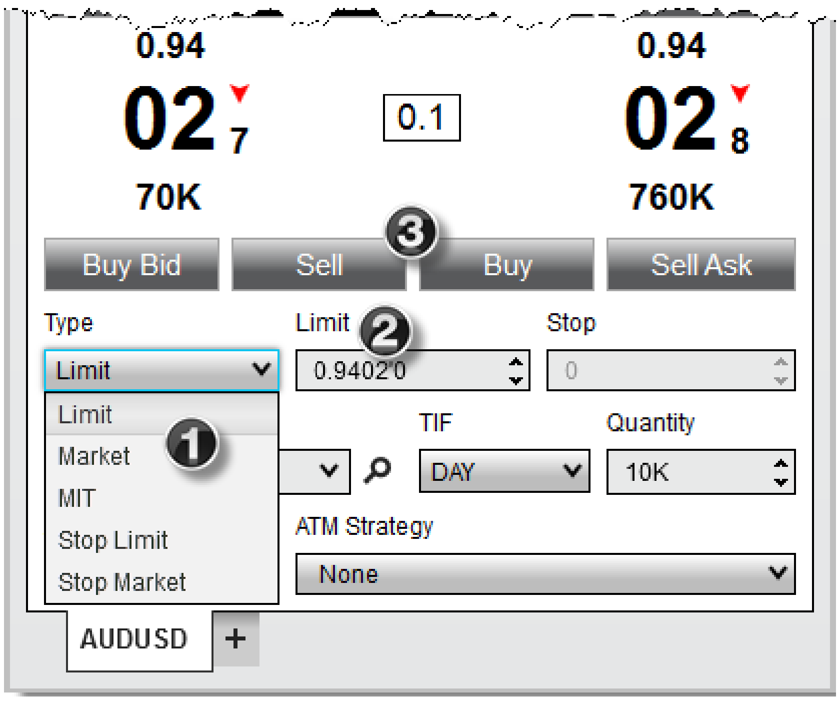
Buy Bid	Submits a Buy Limit order at the best bid price
Sell	Submits a Sell order based on the current **Order Controls** configured
Buy	Submits a Buy order based on the current **Order Controls** configured
Sell Ask	Submits a Sell Limit order at the best Ask price

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

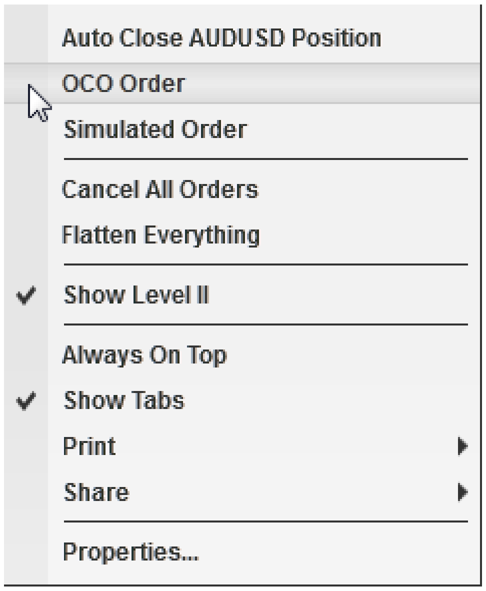
2. Hold down the CTRL key when increasing/decreasing limit/stop prices to change the price in steps of one-pip increments (10 tenth-pips)
3. Left clicking on a price in the Level II panel will load that price into the limit and stop price fields automatically.

Understanding the OCO (One Cancel Other) function

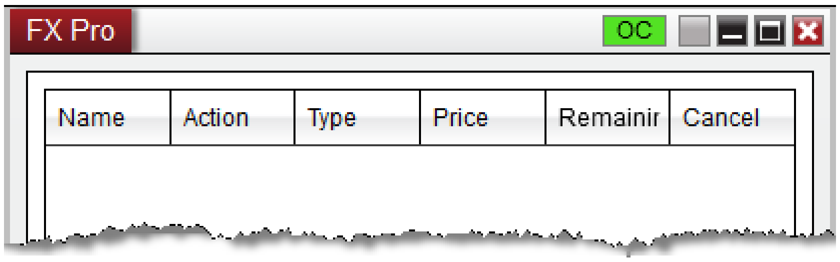
OCO Orders (One Cancels Other)

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:

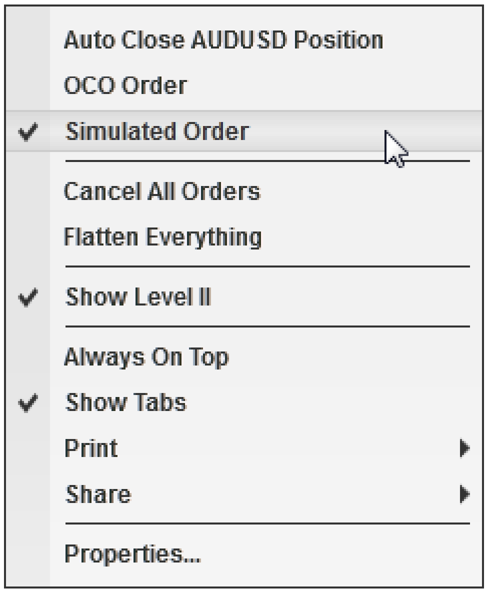
- Right clicking in the **FX Pro** window and selecting the menu item "**OCO Order**" to enable the OCO function
- For your first order, select the desired option from the "**ATM Strategy**" drop down list
- Submit your stop order to buy above the market
- For your second order, select the desired option 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.

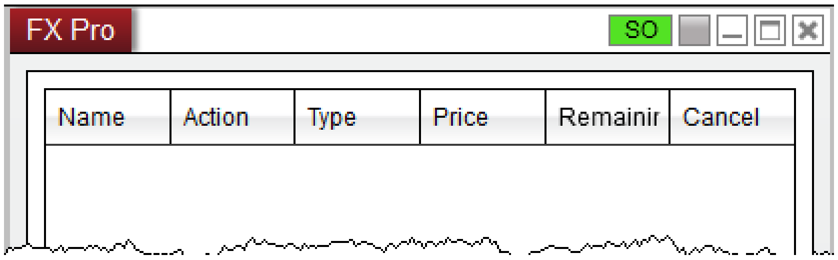
How to submit Simulated Stop Orders (Simulated Order)

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.



11.19.8.3 Modifying and Cancelling Orders

You can modify an existing order's quantity, price, or cancel an order entirely from **Order Grid** display of the **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.

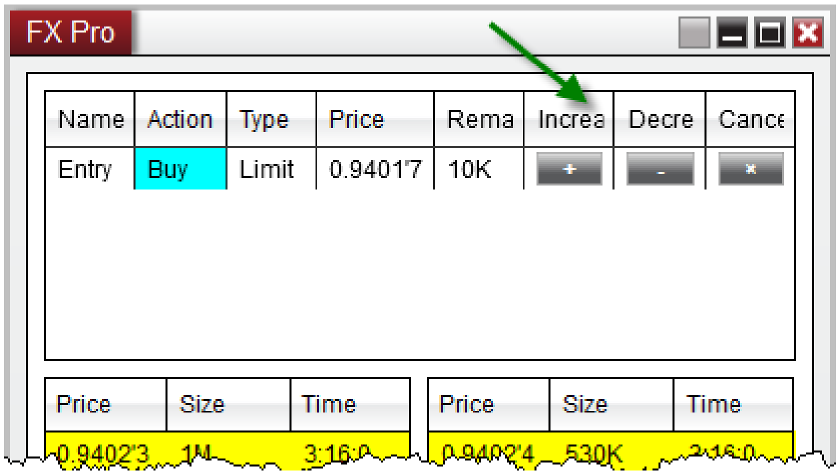
Tip:
Hold down the CTRL key when scrolling in the **price editor** to change the price in half-pip increments.

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**"



FX Pro

Name	Action	Type	Price	Rema	Increa	Decre	Cance
Entry	Buy	Limit	0.94017	10K	+	-	×

Price	Size	Time
0.94023	1M	3:16:0

Price	Size	Time
0.94024	530K	3:16:0

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.

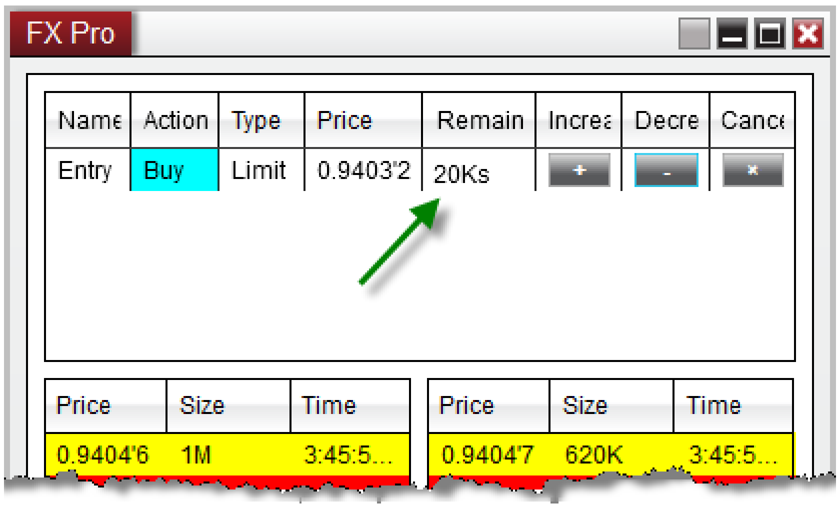
Changing the Quantity of an Order

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.

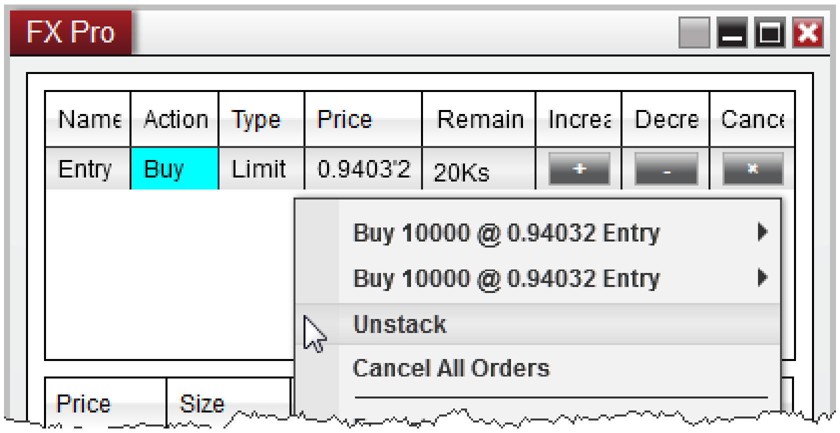


Name	Action	Type	Price	Remain	Increa	Decre	Cance
Entry	Buy	Limit	0.9403'2	20Ks	+	-	×

Price	Size	Time
0.9404'6	1M	3:45:5...

Price	Size	Time
0.9404'7	620K	3:45:5...

If you would like to break up these orders to manage individually, you can right click on the Order and select "**Unstack**"



Cancelling orders

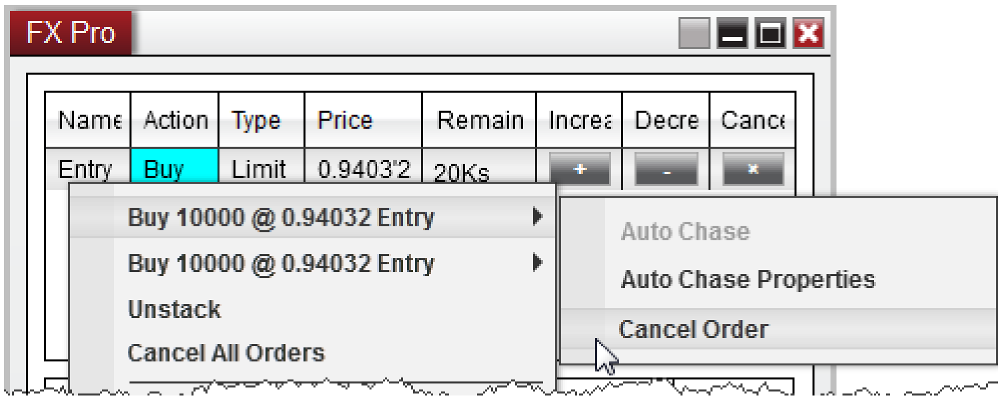
Cancelling an Individual Order

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**"

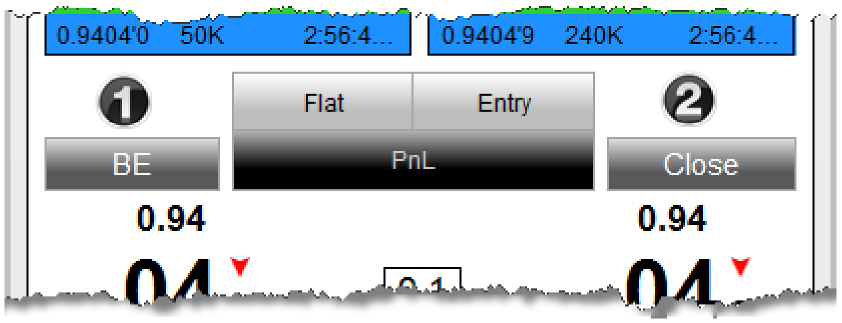


11.19.8.4 Managing Positions

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 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.

11.19.8.5 Properties

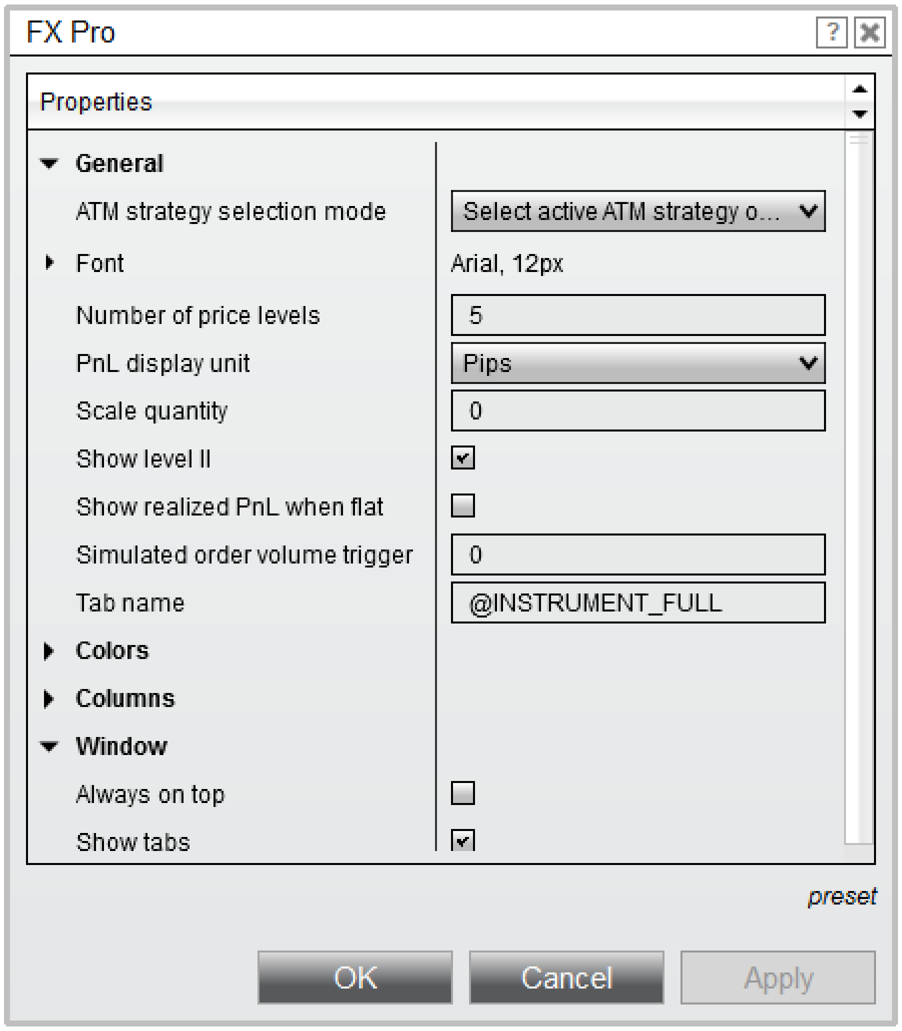
The FX Pro order entry window is highly efficient by design but can also be customized to your preferences through the FX Pro Properties menu.

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:



General	
ATM strategy selection mode	Select active ATM strategy o...
Font	Arial, 12px
Number of price levels	5
PnL display unit	Pips
Scale quantity	0
Show level II	✓
Show realized PnL when flat	☐
Simulated order volume trigger	0
Tab name	@INSTRUMENT_FULL
Colors	
Columns	
Window	
Always on top	☐
Show tabs	✓

Property Definitions

General	
ATM strategy selection mode	Sets the behavior mode of the price ladder display and strategy selector (see more)
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 p anel
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.

11.19.9 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

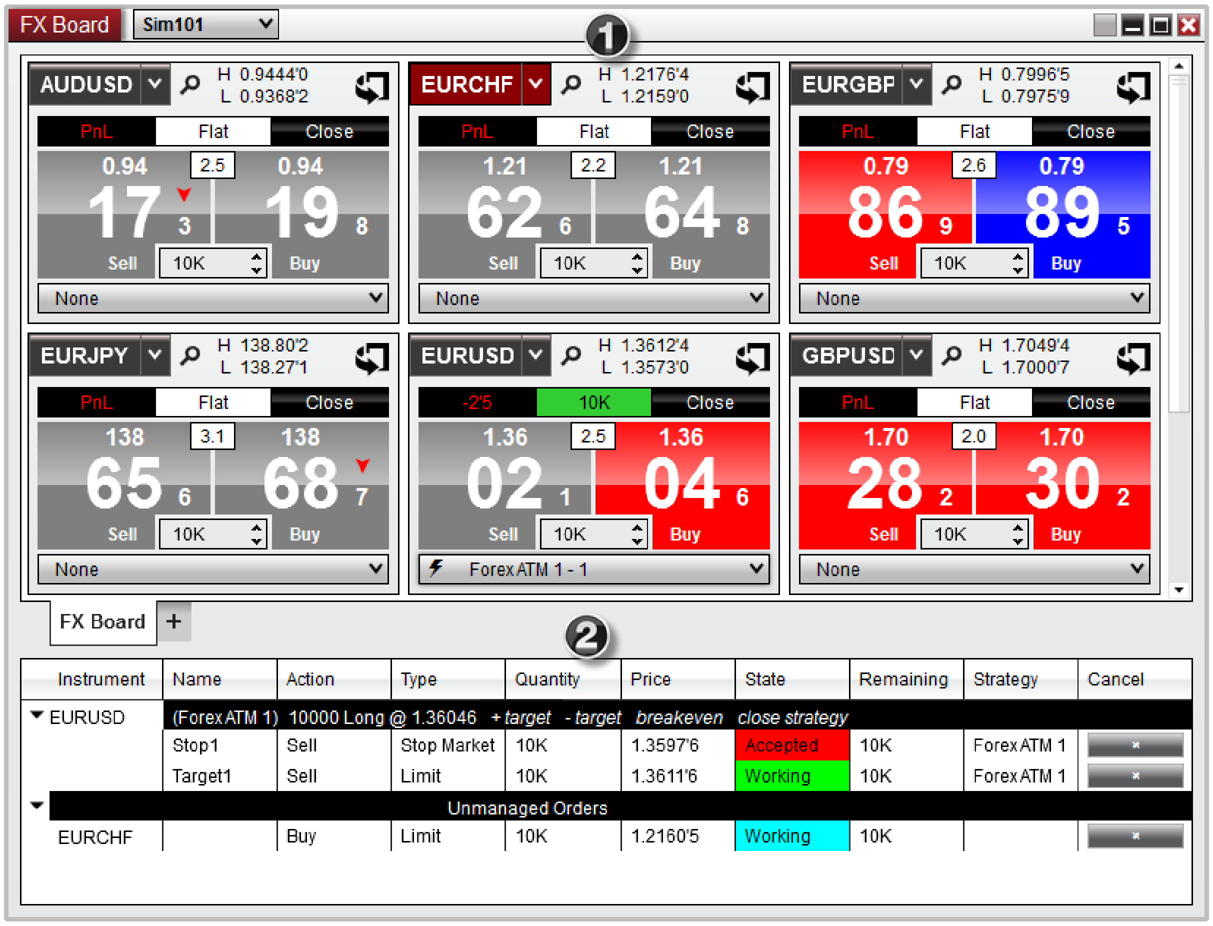
11.19.9.1 Display Overview

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)



Instrument	Name	Action	Type	Quantity	Price	State	Remaining	Strategy	Cancel
EURUSD	(Forex ATM 1) 10000 Long @ 1.36046 *+target -target breakeven close strategy*								
	Stop1	Sell	Stop Market	10K	1.3597'6	Accepted	10K	Forex ATM 1	×
	Target1	Sell	Limit	10K	1.3611'6	Working	10K	Forex ATM 1	×
	Unmanaged Orders								
EURCHF		Buy	Limit	10K	1.2160'5	Working	10K		×

Please see the sections below for more information on: **Market Display** and **Order Grid**

Understanding the market data section

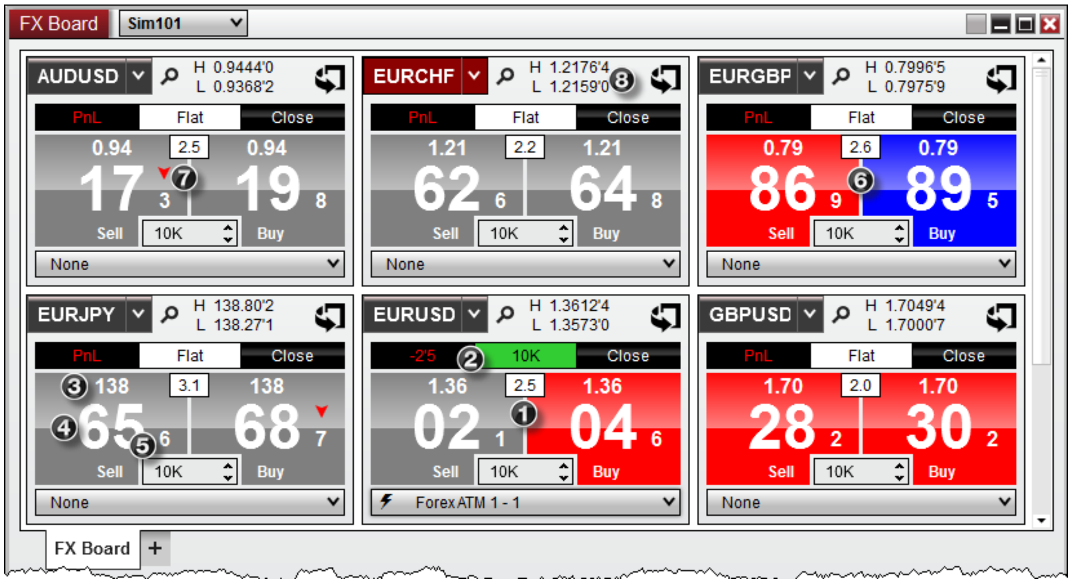
Market Display

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

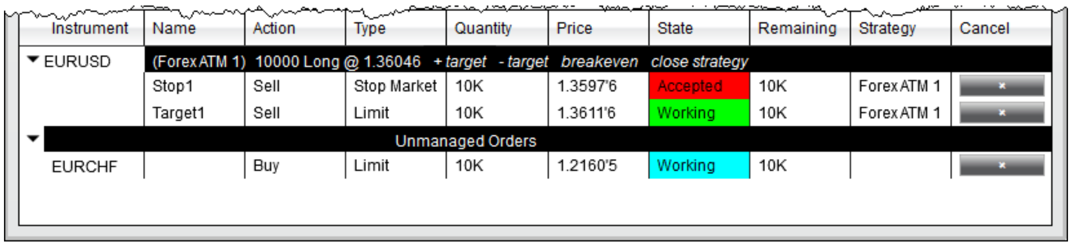

FX Board
Sim101
AUDUSD H 0.9444'0 L 0.9368'2
EURCHF H 1.2176'4 L 1.2159'0
EURGBP H 0.7996'5 L 0.7975'9
EURJPY H 138.80'2 L 138.27'1
EURUSD H 1.3612'4 L 1.3573'0
GBPUSD H 1.7049'4 L 1.7000'7
PnL
Flat
Close
-2'5
10K
Sell
Buy
None
Forex ATM 1 - 1
FX Board
+

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.



Instrument	Name	Action	Type	Quantity	Price	State	Remaining	Strategy	Cancel
EURUSD	(Forex ATM 1) 10000 Long @ 1.36046 +target -target breakeven close strategy								
	Stop1	Sell	Stop Market	10K	1.3597'6	Accepted	10K	Forex ATM 1	×
	Target1	Sell	Limit	10K	1.3611'6	Working	10K	Forex ATM 1	×
Unmanaged Orders									
EURCHF		Buy	Limit	10K	1.2160'5	Working	10K		×

Column Definitions

Instrume	Name of the instrument

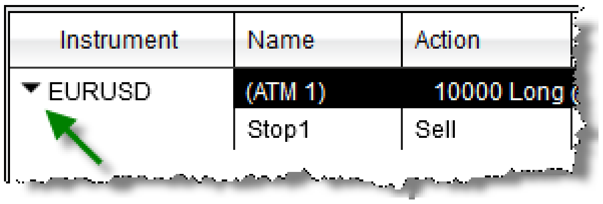
nt	
Name	Order name such as Stop1 or Target1
Action	Buy or Sell
Type	Order type
Quantity	Number of contracts submitted
Price	Order price
State	State of the order
Remainin g	Quantity remaining to be filled
Strategy	Name of ATM Strategy associated with the order
Cancel	Cancels the order(s)

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.



Instrument	Name	Action
EURUSD	(ATM 1)	10000 Long
	Stop1	Sell

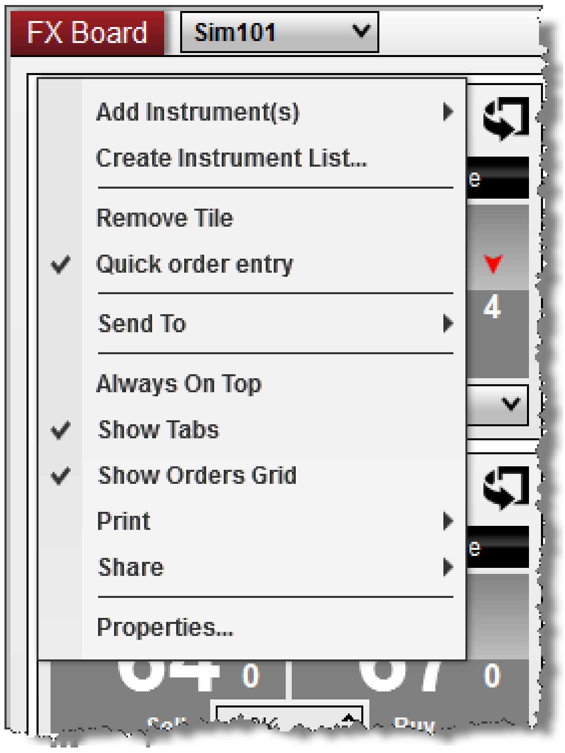
Understanding the right click menu

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**

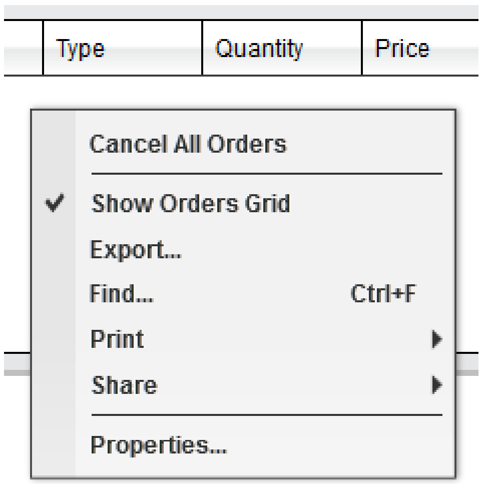


Add	Adds an individual instrument or list of

Instrument(s)	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

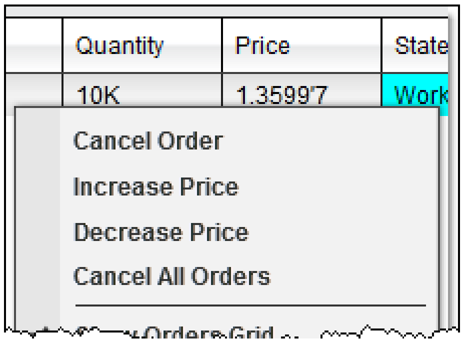
Order Grid Control Right Click Menu

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
Propert ies	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

11.19.9.2 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

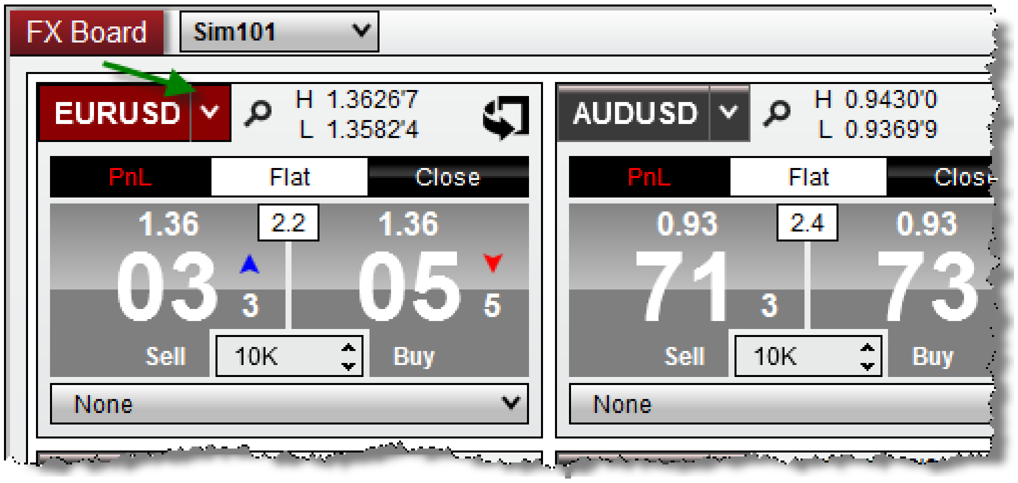
You can also rapidly add an entire list of predefined instruments to the **FX Board** window.

- 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**.

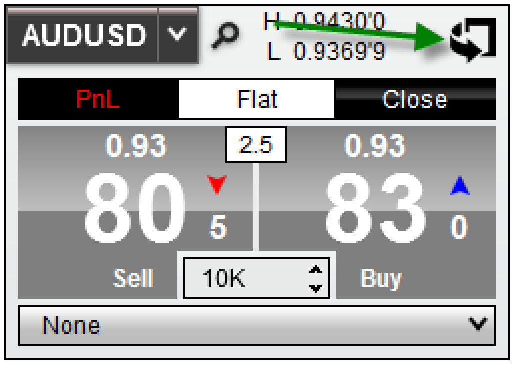
You will now be able to access this list from other features of NinjaTrader using the Instrument Selector. You can further edit this list by using the Instrument Lists window

Flipping tiles for custom order entry

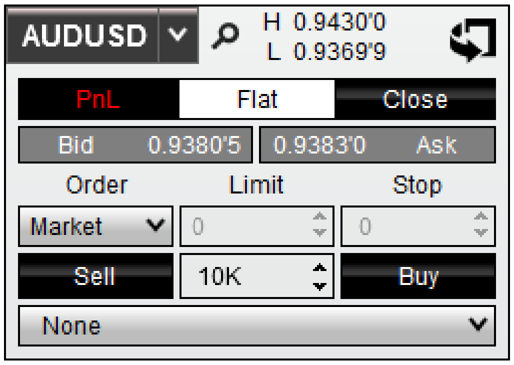
Flipping Tiles

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.**

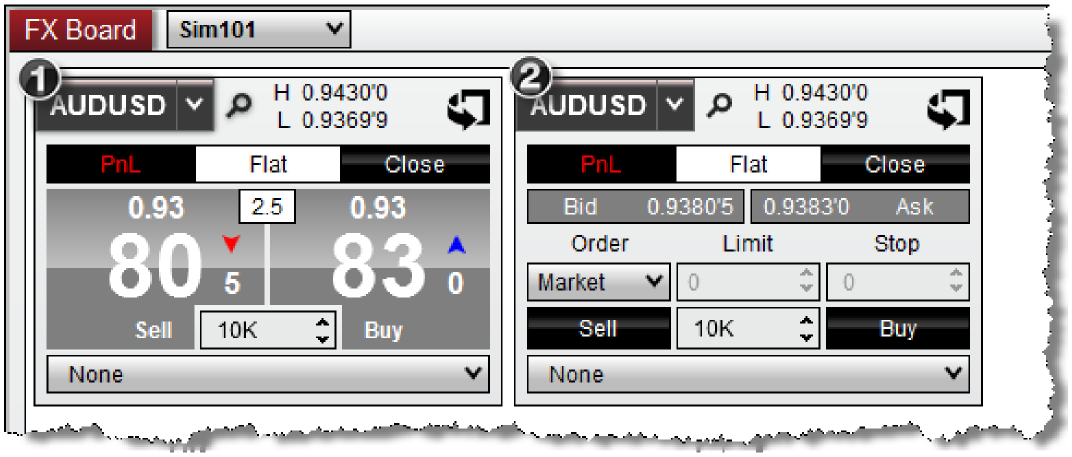


Once the tile has been flipped, you will see a number of additional controls which can be used to define and submit custom orders.



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.

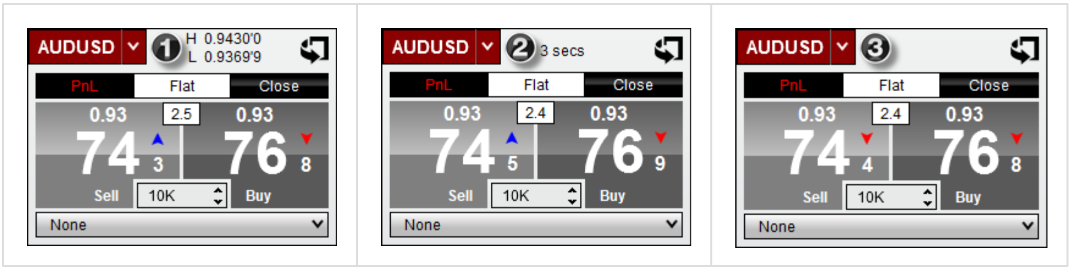
Market Data Display Text

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.

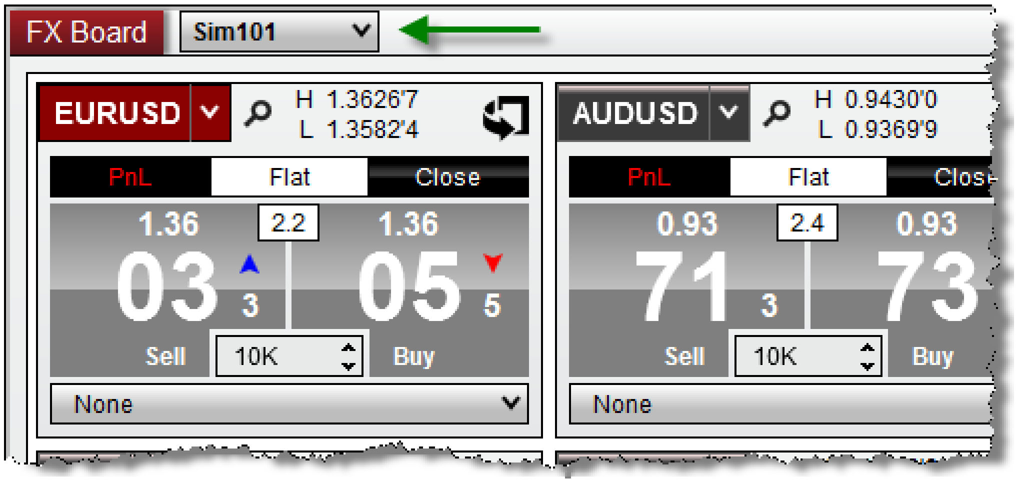
11.19.9.3 Submitting Orders

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 toNinjaTrader **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.

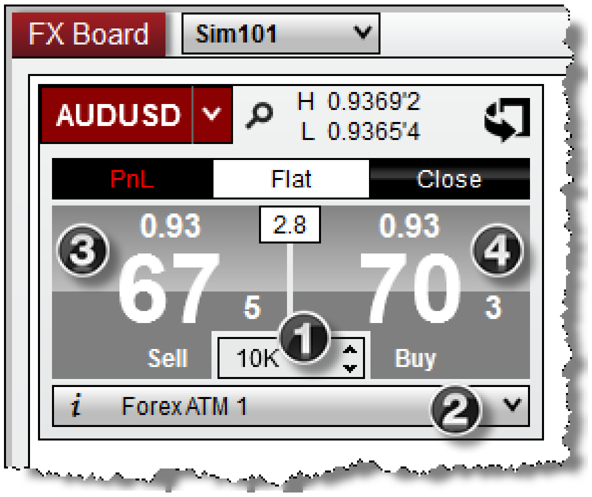


To change the account, select the account name or number you wish to trade through via the **account selector**. The account selected will be the account used for all instruments in the **FX Board** window.

How to submit orders with quick buttons

Quick Market Orders

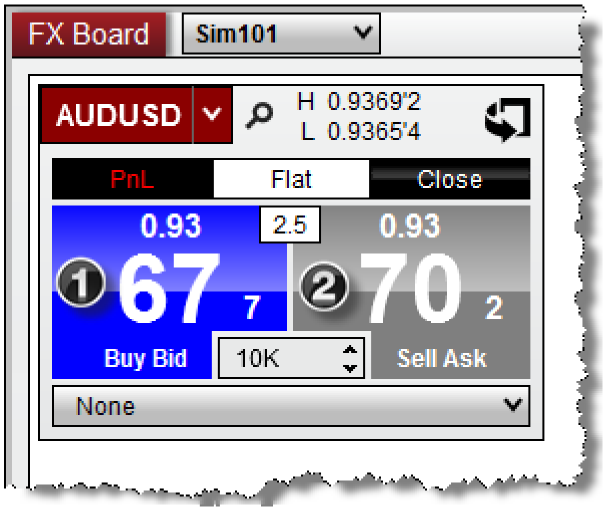
You can rapidly execute market orders directly from an **instrument tile**.



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

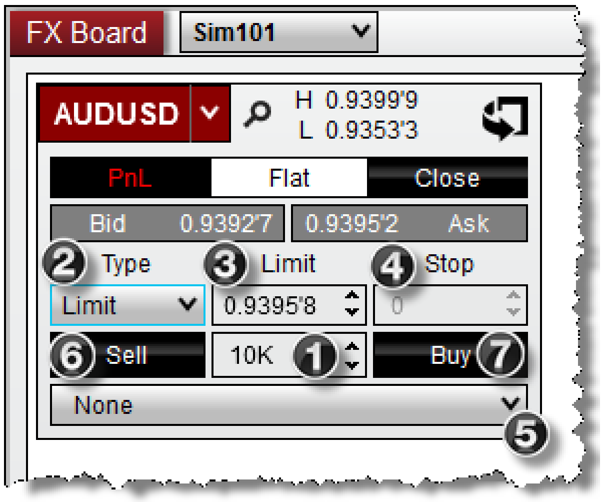
Custom Orders

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.

11.19.9.4 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

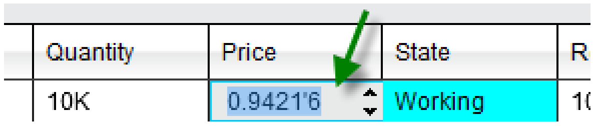
Changing the Price of an Order

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.



Quantity	Price	State	R
10K	0.9421'6	Working	1(

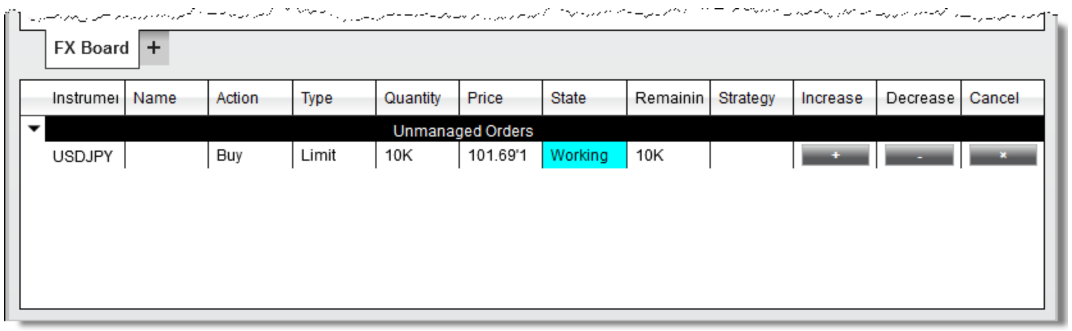
Tip: Hold down the CTRL key when scrolling in the **price editor** to change the price in steps of half-pip increments.

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**



Instrumei	Name	Action	Type	Quantity	Price	State	Remainin	Strategy	Increase	Decrease	Cancel
USDJPY		Buy	Limit	10K	101.69'1	Working	10K		+	-	×

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

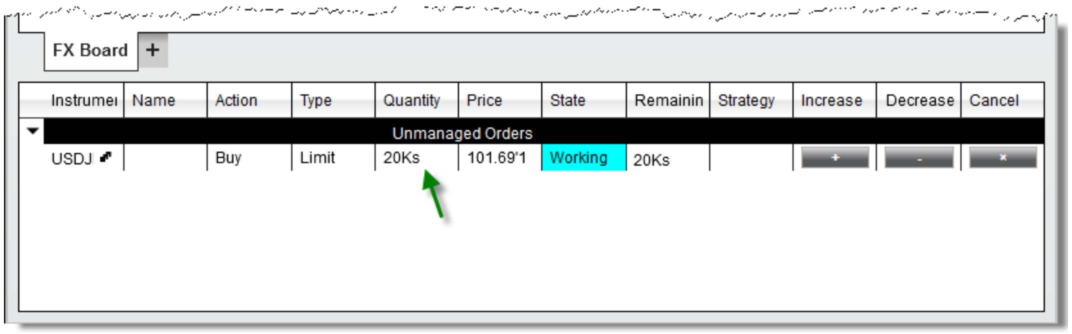
Changing the Quantity of an Order

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).

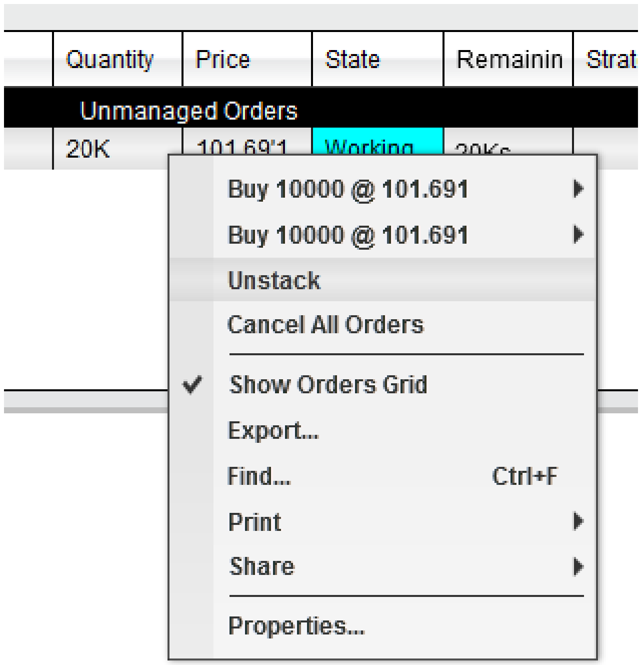
Tip: Holding down the CTRL key and scrolling will change the FX order size by 100K (100,000)

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.



Instrumei	Name	Action	Type	Quantity	Price	State	Remainin	Strategy	Increase	Decrease	Cancel
Unmanaged Orders											
USDJ		Buy	Limit	20Ks	101.69'1	Working	20Ks		+	-	×

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

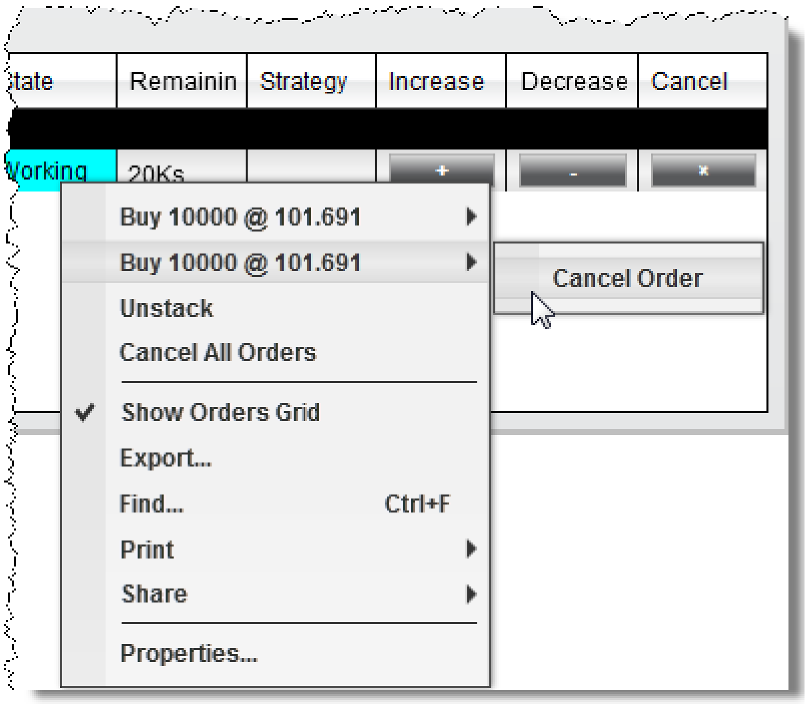
Cancelling an Individual Order

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:

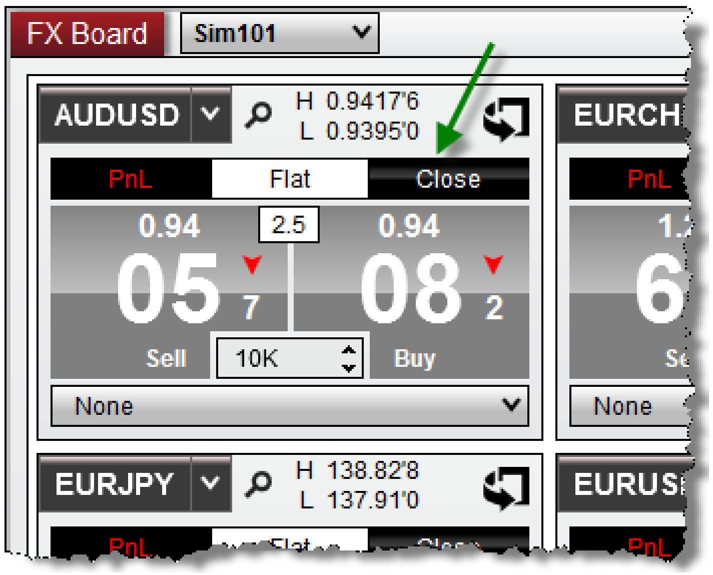
1. Right click on the stacked order row
2. Move your mouse over the order individual order
3. Select "**Cancel Order**"



11.19.9.5 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.



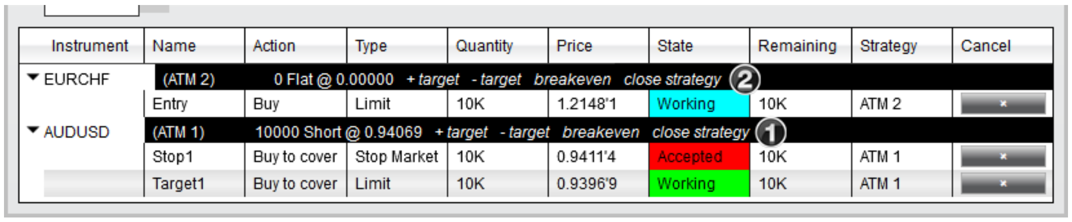
Clicking on this button with your middle mouse button (scroll wheel) will close the selected active **ATM Strategy** only. This means that the position size of the **ATM Strategy** will be closed and any working orders associated to that **ATM Strategy** will be cancelled.

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.**



Instrument	Name	Action	Type	Quantity	Price	State	Remaining	Strategy	Cancel
EURCHF	(ATM 2) 0 Flat @ 0.00000 *+ target* *- target* *breakeven* *close strategy* 2								
	Entry	Buy	Limit	10K	1.2148'1	Working	10K	ATM 2	×
AUDUSD	(ATM 1) 10000 Short @ 0.94069 *+ target* *- target* *breakeven* *close strategy* 1								
	Stop1	Buy to cover	Stop Market	10K	0.9411'4	Accepted	10K	ATM 1	×
	Target1	Buy to cover	Limit	10K	0.9396'9	Working	10K	ATM 1	×

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.

11.19.9.6 Properties

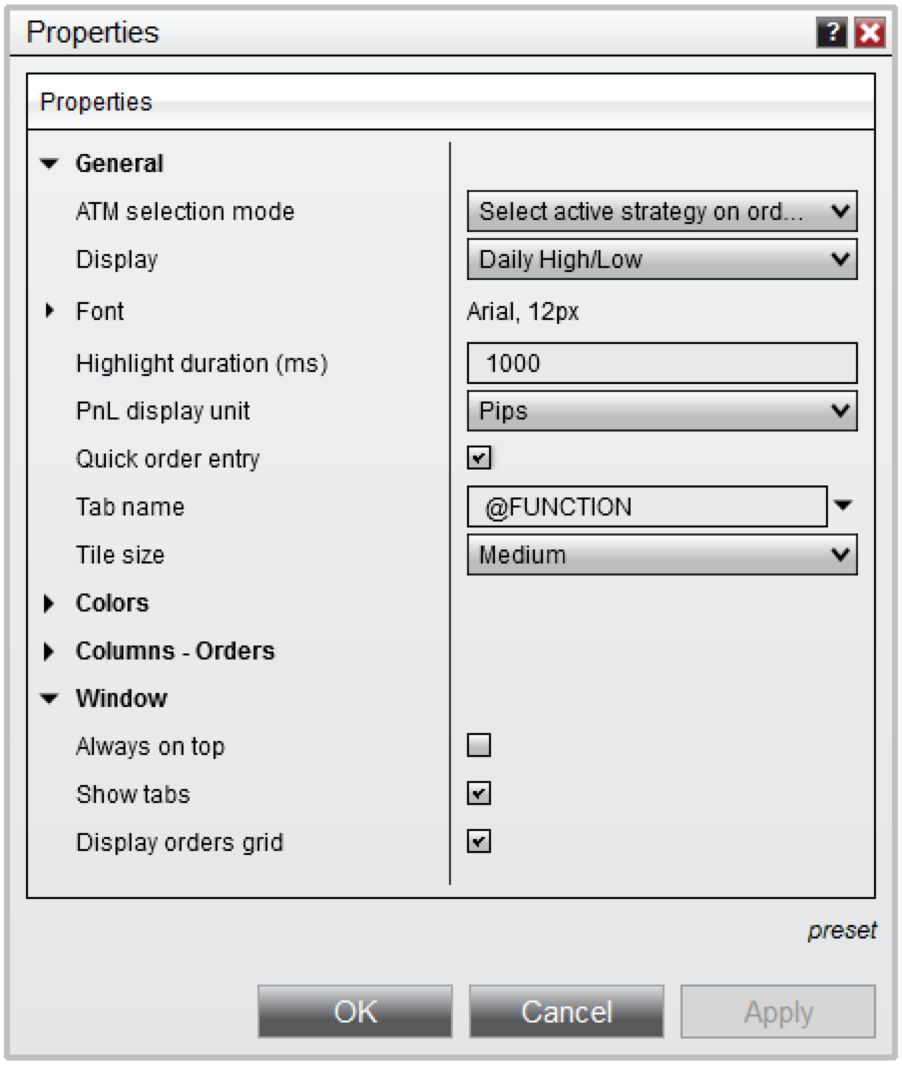
The FX Board order entry window is highly efficient by design but can also be customized to your preferences through the FX Board **Properties** menu.

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:


Properties

Properties

General
- ATM selection mode: Select active strategy on ord...
- Display: Daily High/Low
- Font: Arial, 12px
- Highlight duration (ms): 1000
- PnL display unit: Pips
- Quick order entry: ✓
- Tab name: @FUNCTION
- Tile size: Medium

Colors

Columns - Orders

Window
- Always on top: ☐
- Show tabs: ✓
- Display orders grid: ✓

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
Font	Sets the font options
Highlight duration	Sets the amount of time a tile will

(ms)	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.

11.19.10 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 **FX Board** menu item.

Order Ticket Overview

The Order Ticket order entry window allows you to view market data and submit orders.

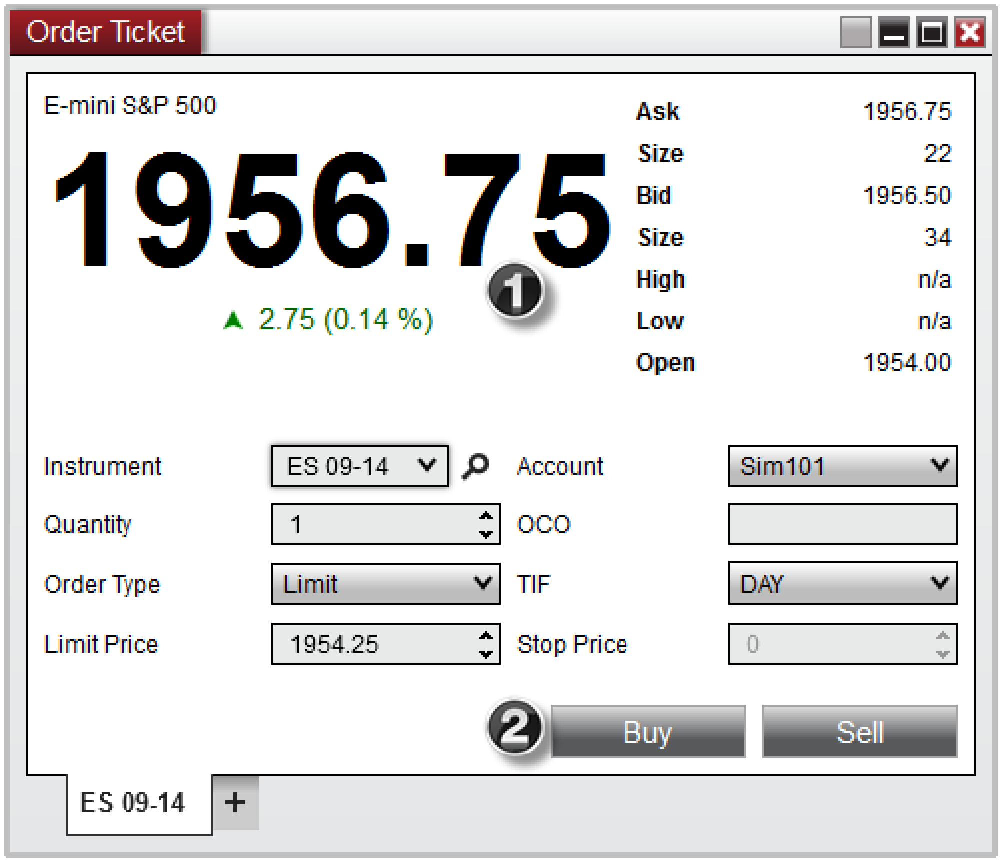
Display	Order Management
› Display Overview	› Submitting Orders
Misc	
› Properties	

11.19.10.1 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**



Instrument	ES 09-14	Account	Sim101
Quantity	1	OCO	
Order Type	Limit	TIF	DAY
Limit Price	1954.25	Stop Price	0

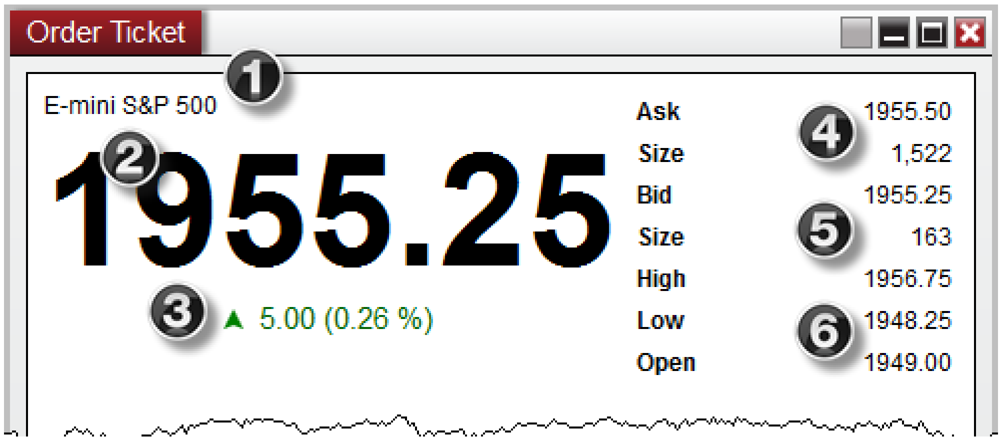
Understanding the market display section

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

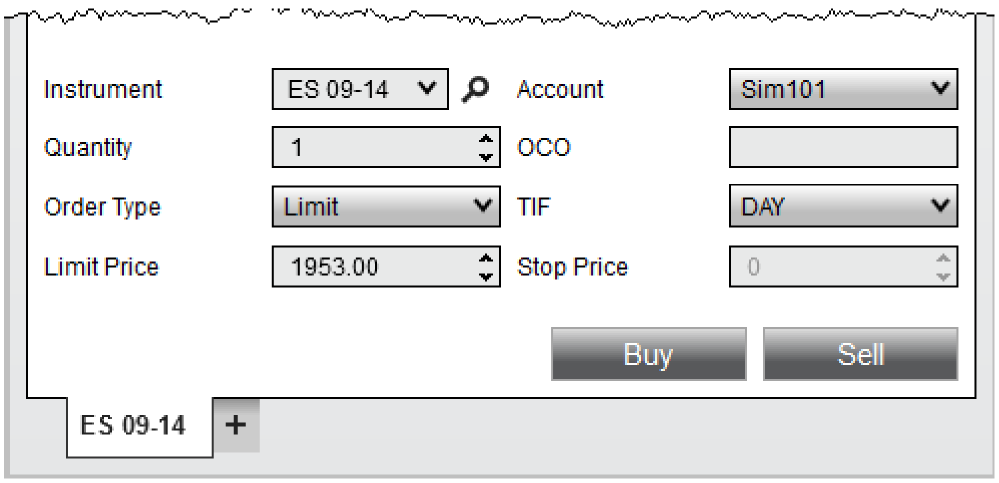


Ask	1955.50
Size	1,522
Bid	1955.25
Size	163
High	1956.75
Low	1948.25
Open	1949.00

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	ES 09-14	Account	Sim101
Quantity	1	OCO	
Order Type	Limit	TIF	DAY
Limit Price	1953.00	Stop Price	0

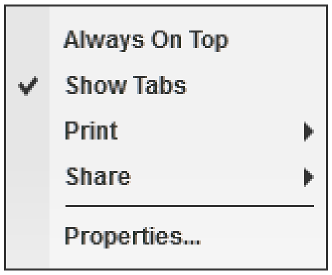
Instrument	Sets the Instrument
Account	Sets the Account
Quantity	Sets the order Quantity

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

11.19.10.2 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.

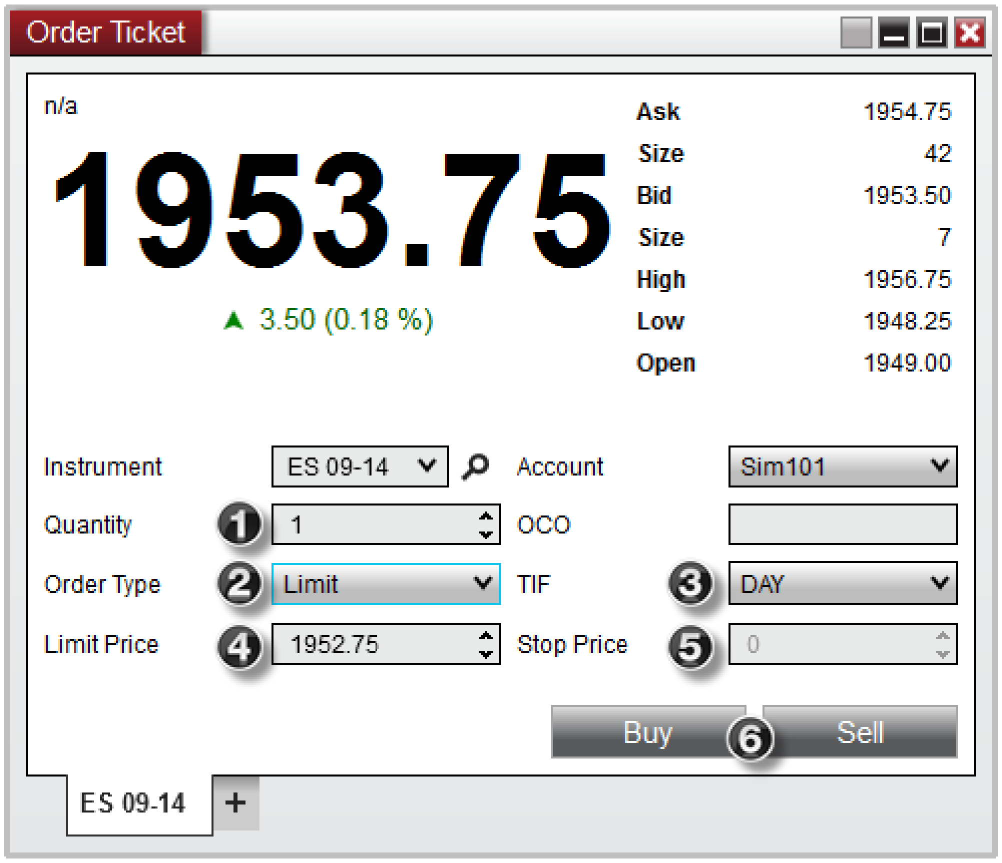
How to Select an Account

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


Order Ticket

n/a

1953.75

▲ 3.50 (0.18 %)

Ask	1954.75
Size	42
Bid	1953.50
Size	7
High	1956.75
Low	1948.25
Open	1949.00

Instrument	ES 09-14	Account	Sim101
Quantity	1	OCO	
Order Type	Limit	TIF	DAY
Limit Price	1952.75	Stop Price	0

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**

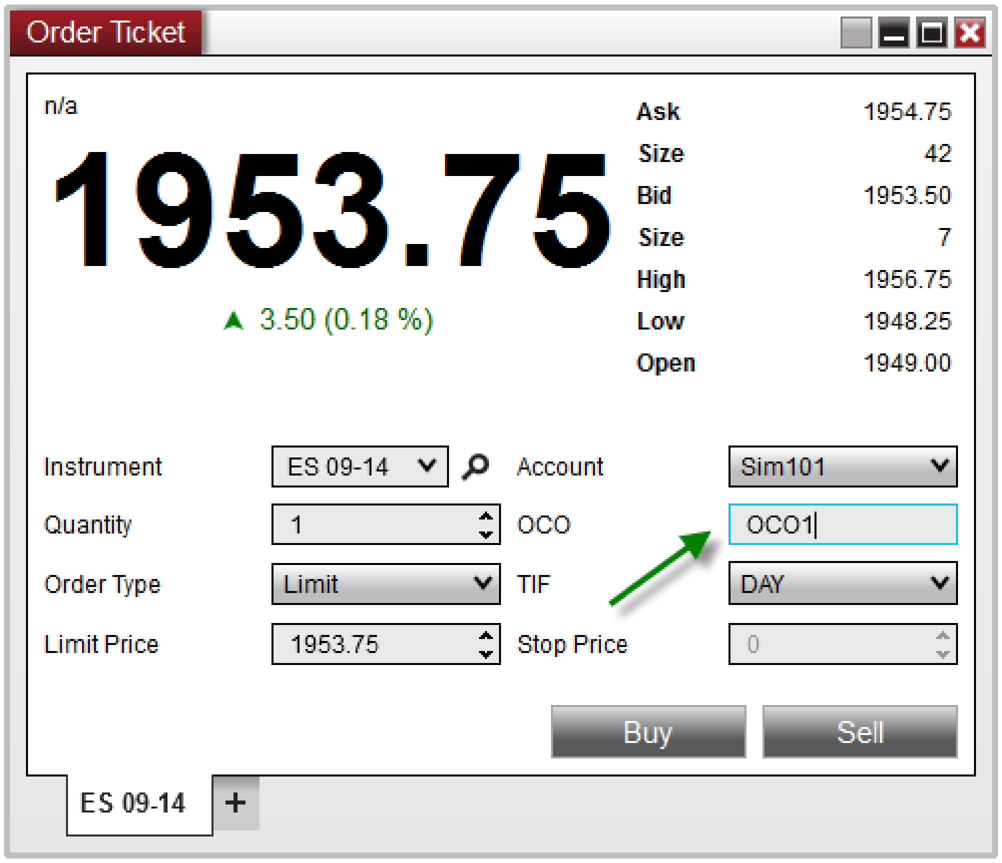
After the order has been submitted, it can be referenced from the Orders Tab of the **Account Data** or **Control Center** window.

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*".


Order Ticket

n/a

1953.75

▲ 3.50 (0.18 %)

Ask	1954.75
Size	42
Bid	1953.50
Size	7
High	1956.75
Low	1948.25
Open	1949.00

Instrument	ES 09-14	Account	Sim101
Quantity	1	OCO	OCO1
Order Type	Limit	TIF	DAY
Limit Price	1953.75	Stop Price	0

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.

11.19.10.3 Properties

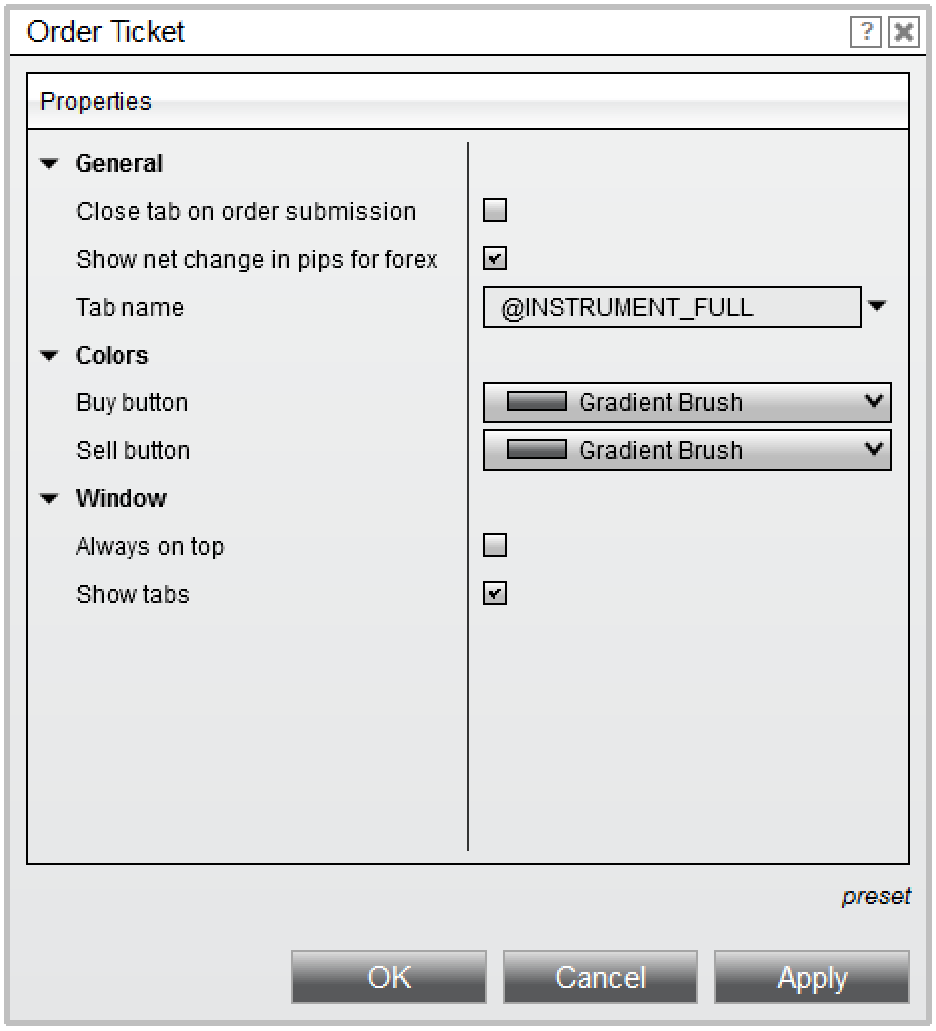
The Order Ticket order entry window is highly efficient by design but can also be customized to your preferences through the Order Ticket Properties menu.

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:



General
Close tab on order submission ☐
Show net change in pips for forex ✓
Tab name @INSTRUMENT_FULL
Colors
Buy button Gradient Brush
Sell button Gradient Brush
Window
Always on top ☐
Show tabs ✓

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.

11.19.1 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.

11.19.11.1 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

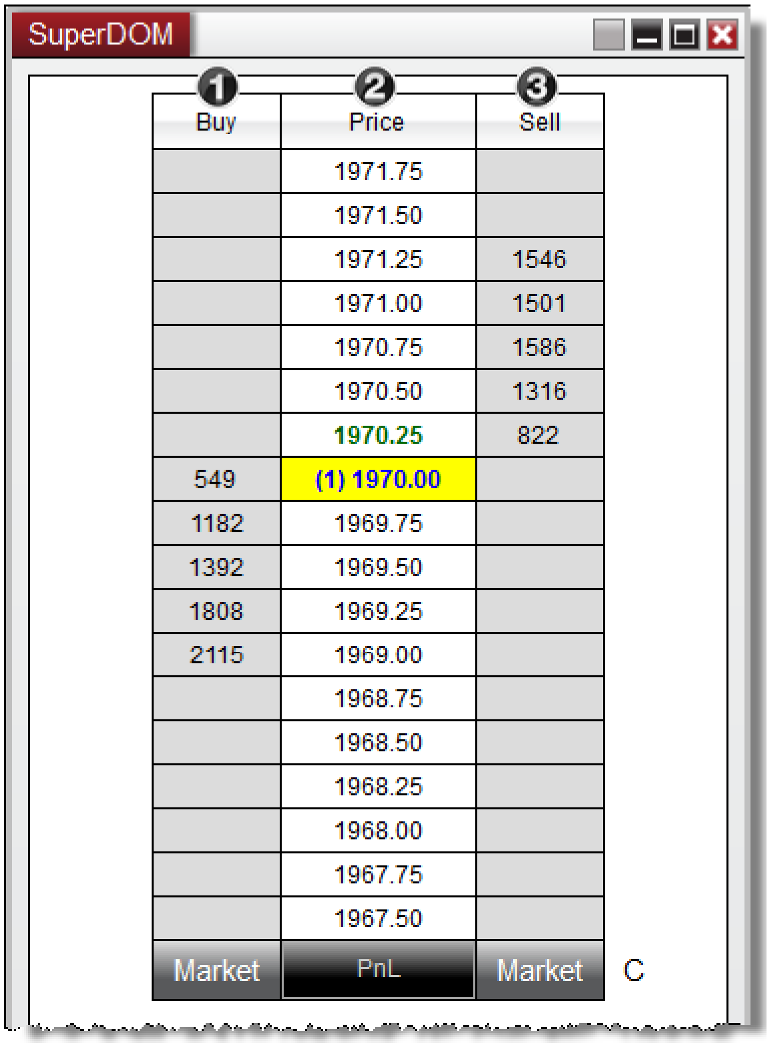
The center column, known as the **Price** ❷ column, is used to:

- 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** ❸ column which is used to:

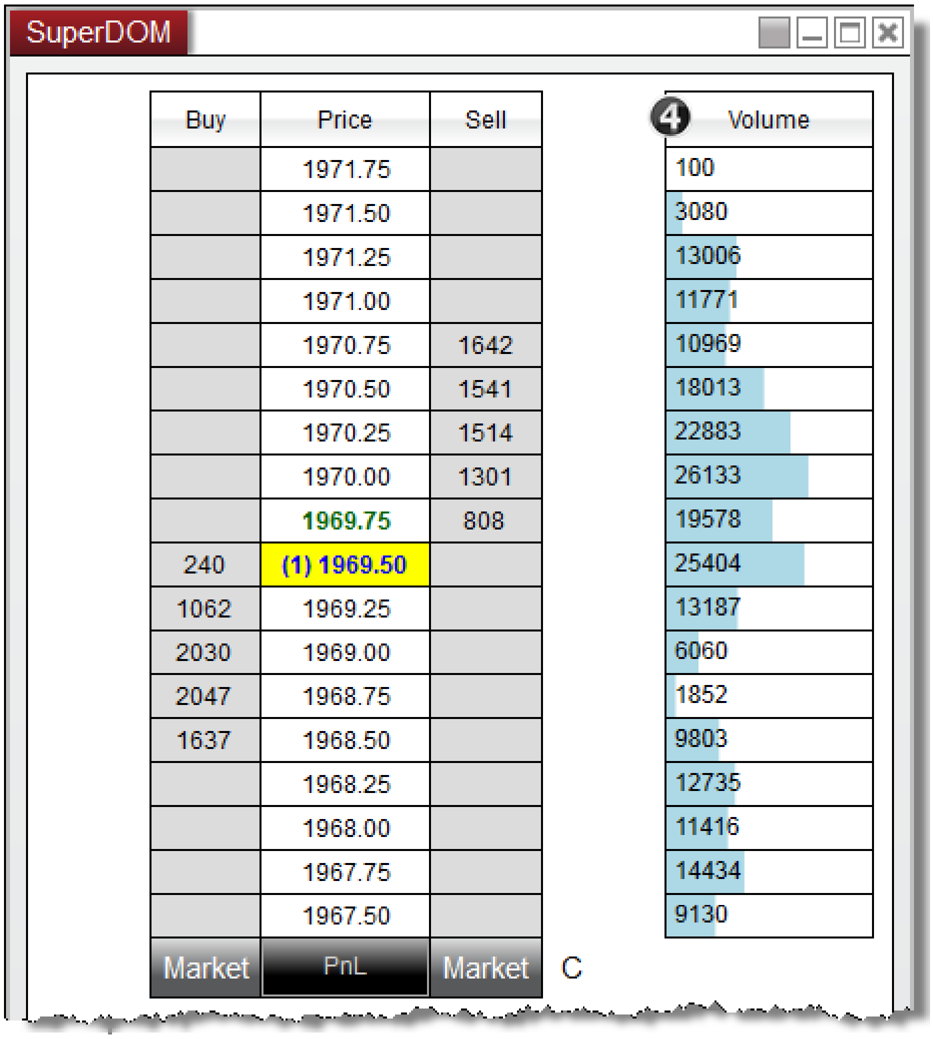
- Submit sell orders
- Modify sell orders
- Display the total contracts on the ask at their respective prices (also known as market depth)



Buy	Price	Sell
	1971.75	
	1971.50	
	1971.25	1546
	1971.00	1501
	1970.75	1586
	1970.50	1316
	1970.25	822
549	(1) 1970.00	
1182	1969.75	
1392	1969.50	
1808	1969.25	
2115	1969.00	
	1968.75	
	1968.50	
	1968.25	
	1968.00	
	1967.75	
	1967.50	

Columns

You can optionally configure additional **Columns** 4 to display other points of interest relative to the ladder display.



Buy	Price	Sell	Volume
	1971.75		100
	1971.50		3080
	1971.25		13006
	1971.00		11771
	1970.75	1642	10969
	1970.50	1541	18013
	1970.25	1514	22883
	1970.00	1301	26133
	1969.75	808	19578
240	(1) 1969.50		25404
1062	1969.25		13187
2030	1969.00		6060
2047	1968.75		1852
1637	1968.50		9803
	1968.25		12735
	1968.00		11416
	1967.75		14434
	1967.50		9130

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.

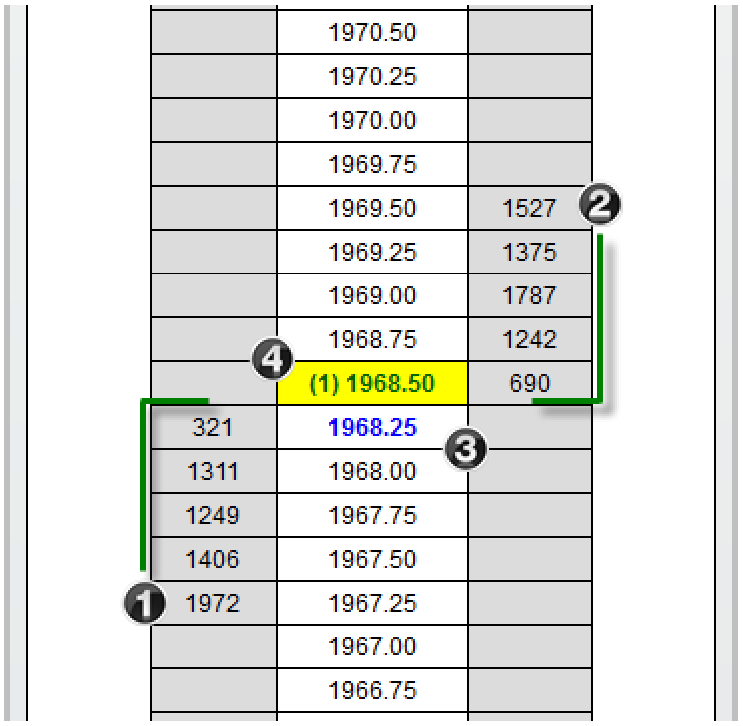
Understanding how market data is displayed

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)



	1970.50	
	1970.25	
	1970.00	
	1969.75	
	1969.50	1527
	1969.25	1375
	1969.00	1787
	1968.75	1242
	(1) 1968.50	690
321	1968.25	
1311	1968.00	
1249	1967.75	
1406	1967.50	
1972	1967.25	
	1967.00	
	1966.75	

How to use the quick buttons at the bottom of the Price Ladder display

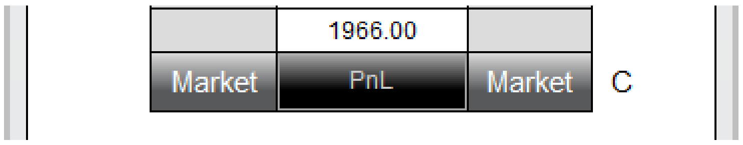
The bottom row of the price ladder contains three functions: Buy Market, PnL, and Sell **Market**.

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



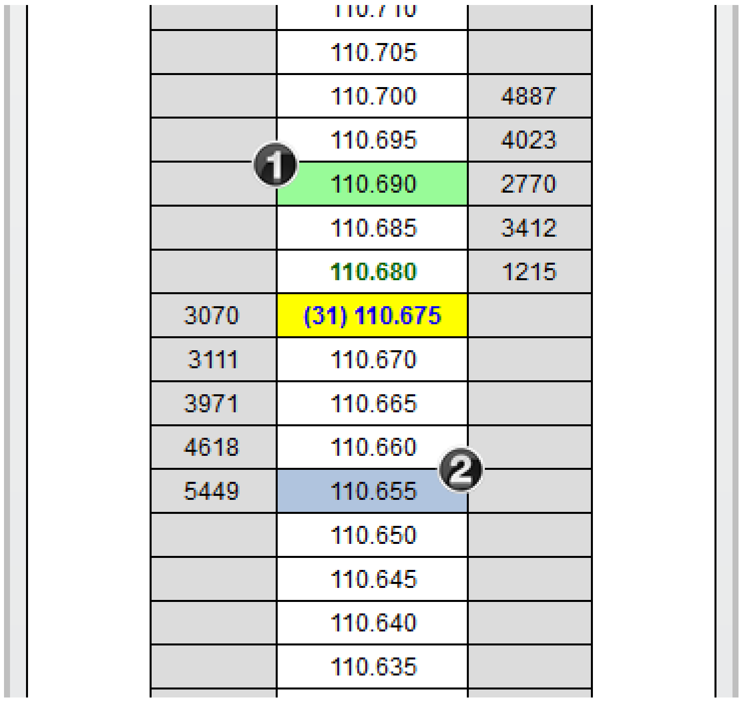
You can change the type of order the MARKET cells submit by holding down the SHIFT key to place limit orders at the ask, or by holding down the CTRL key to place limit orders at the bid. Clicking with your left mouse button on the PnL cell will change the display between points, ticks, currency, percent 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.



	110.710	
	110.705	
	110.700	4887
	110.695	4023
	110.690	2770
	110.685	3412
	110.680	1215
3070	(31) 110.675	
3111	110.670	
3971	110.665	
4618	110.660	
5449	110.655	
	110.650	
	110.645	
	110.640	
	110.635	

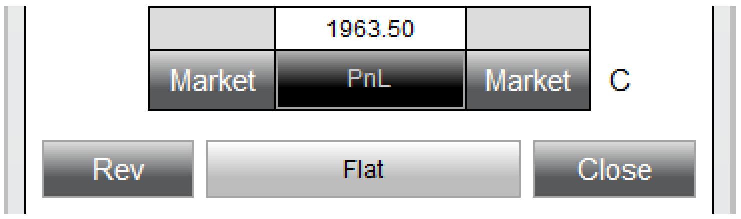
Note: Daily High and Low values are not calculated by NinjaTrader and are sent from your data provider. Not all data providers provide this information for all instrument types.

▽ 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

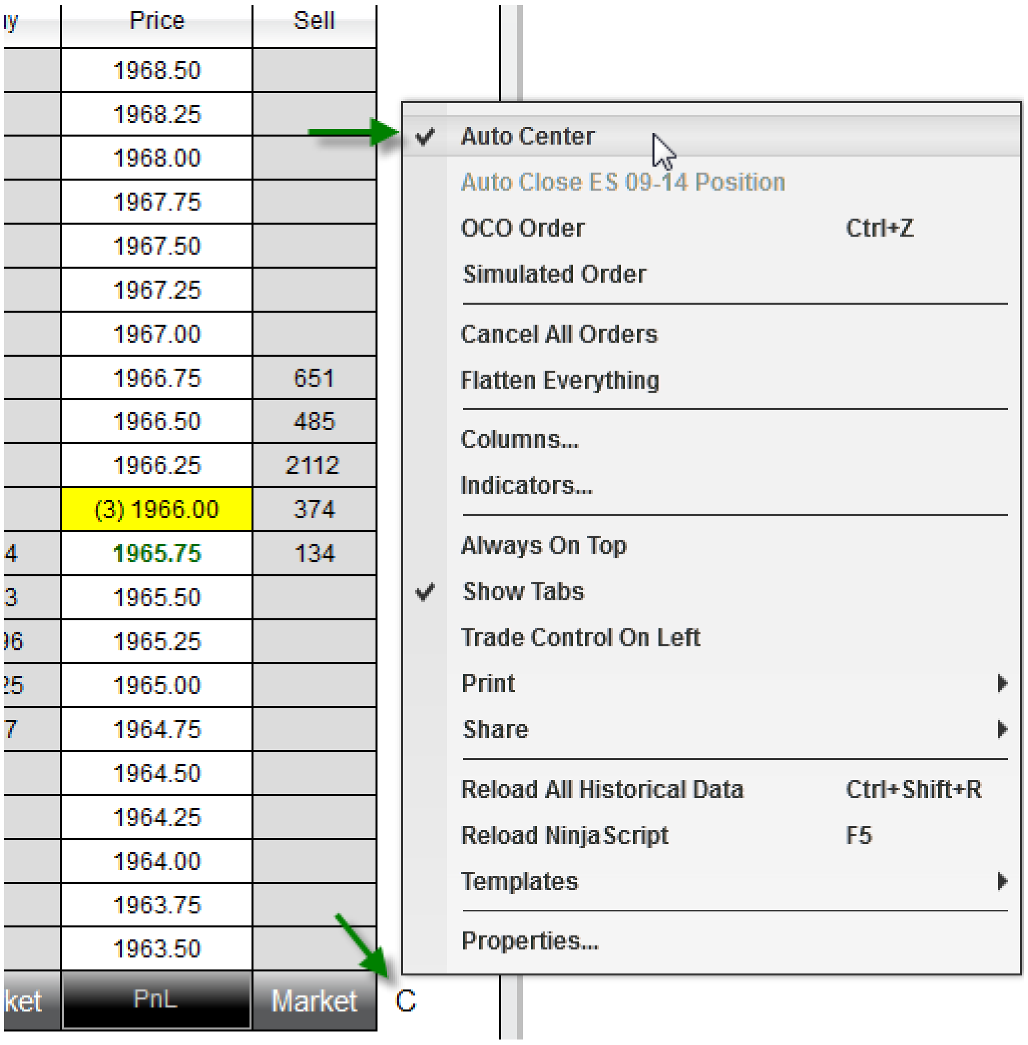
Play Video

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

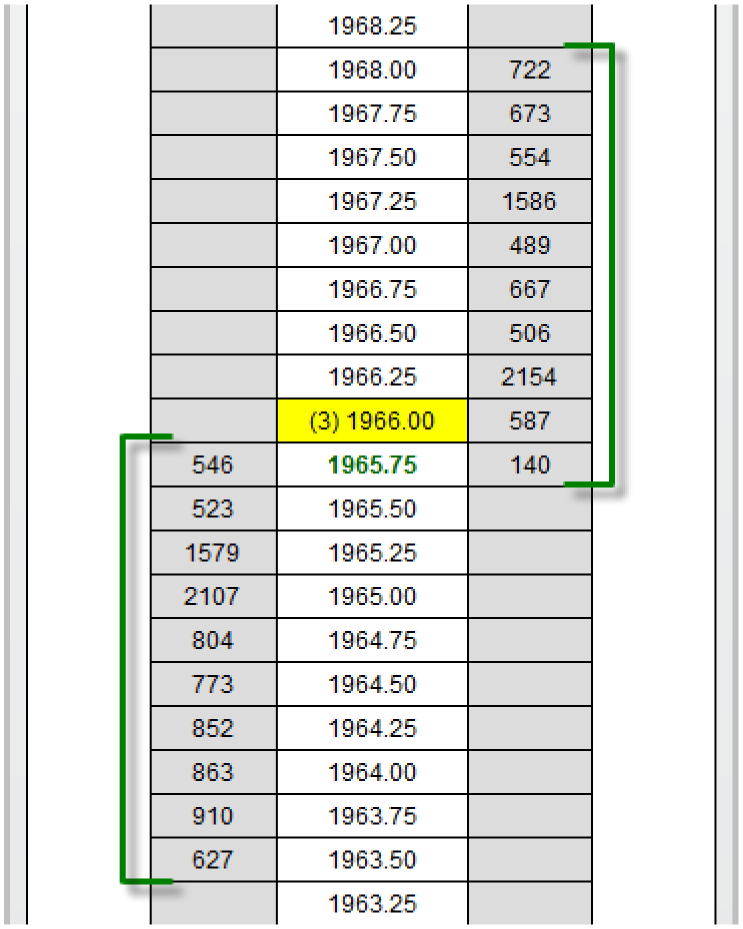
The number of price rows will be dynamically adjusted by vertically re-sizing the **SuperDOM** window. The larger the size of the **SuperDOM** window, the more price levels will be added in automatically.

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 5 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 10 levels of market depth for an instrument you are trading. If you would like to view all 10 of these levels on the **SuperDOM**, simply right click on the **SuperDOM**, select **Properties**, and set the **# of market depth levels property** value to 10 and the press **OK**.



	1968.25	
	1968.00	722
	1967.75	673
	1967.50	554
	1967.25	1586
	1967.00	489
	1966.75	667
	1966.50	506
	1966.25	2154
	(3) 1966.00	587
546	1965.75	140
523	1965.50	
1579	1965.25	
2107	1965.00	
804	1964.75	
773	1964.50	
852	1964.25	
863	1964.00	
910	1963.75	
627	1963.50	
	1963.25	

11.19.11.2 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.

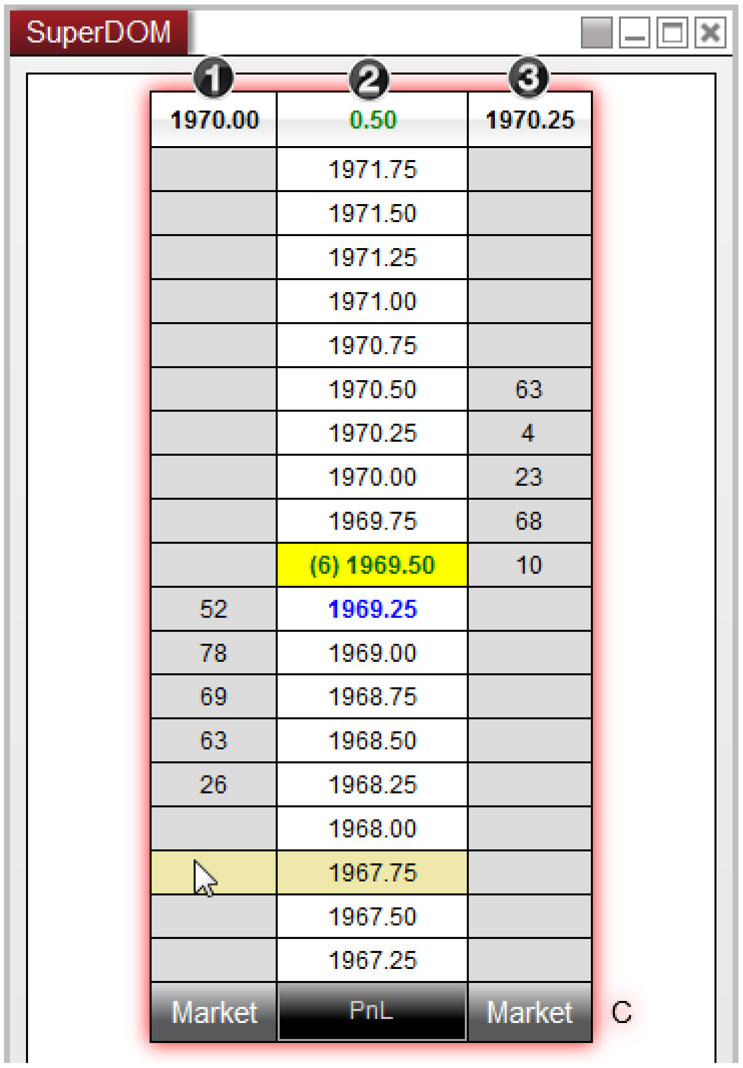
Suspending the Dynamic Price Ladder

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



1970.00	0.50	1970.25
	1971.75	
	1971.50	
	1971.25	
	1971.00	
	1970.75	
	1970.50	63
	1970.25	4
	1970.00	23
	1969.75	68
	(6) 1969.50	10
52	1969.25	
78	1969.00	
69	1968.75	
63	1968.50	
26	1968.25	
	1968.00	
	1967.75	
	1967.50	
	1967.25	

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.

11.19.11.: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.

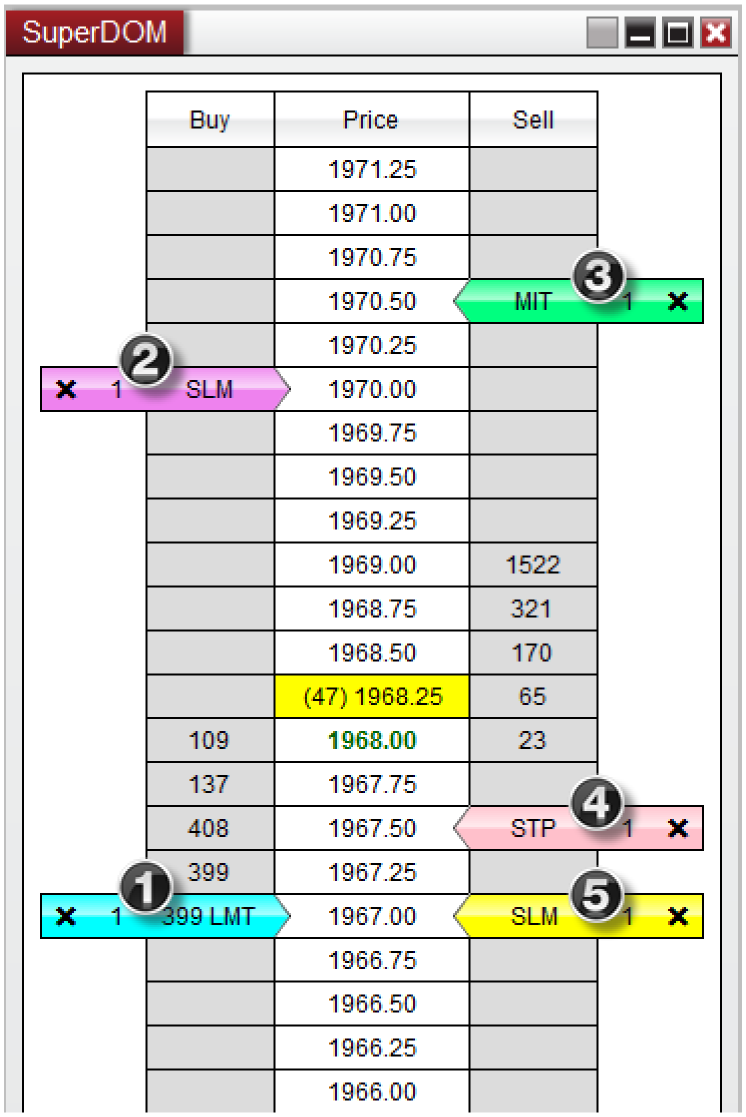
Understanding how orders are displayed

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.



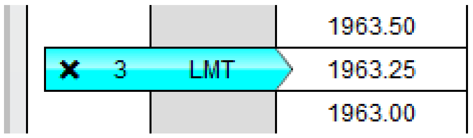
SuperDOM

Buy	Price	Sell
	1971.25	
	1971.00	
	1970.75	
	1970.50	MIT 1 ×
	1970.25	
× 1 SLM	1970.00	
	1969.75	
	1969.50	
	1969.25	
	1969.00	1522
	1968.75	321
	1968.50	170
	(47) 1968.25	65
109	1968.00	23
137	1967.75	
408	1967.50	STP 1 ×
399	1967.25	
× 1 399 LMT	1967.00	SLM 1 ×
	1966.75	
	1966.50	
	1966.25	
	1966.00	

Understanding how the quantity of an order are displayed

Size Marker

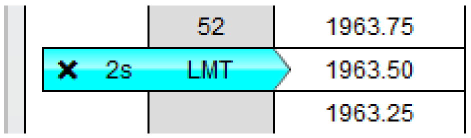
There is also an associated Size Marker which displays the remaining contracts to be filled for the order(s) at the corresponding price. In the image blow, three contracts are remaining to be filled and are working at a price of 1963.25



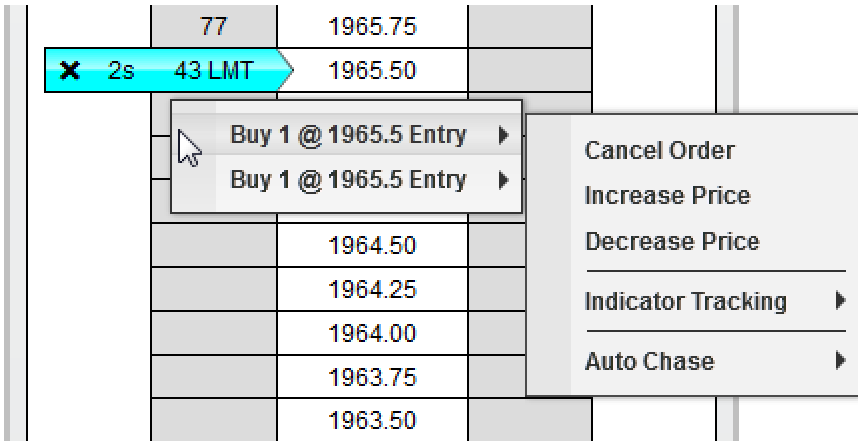
Understanding how multiple orders at the same price are displayed

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.

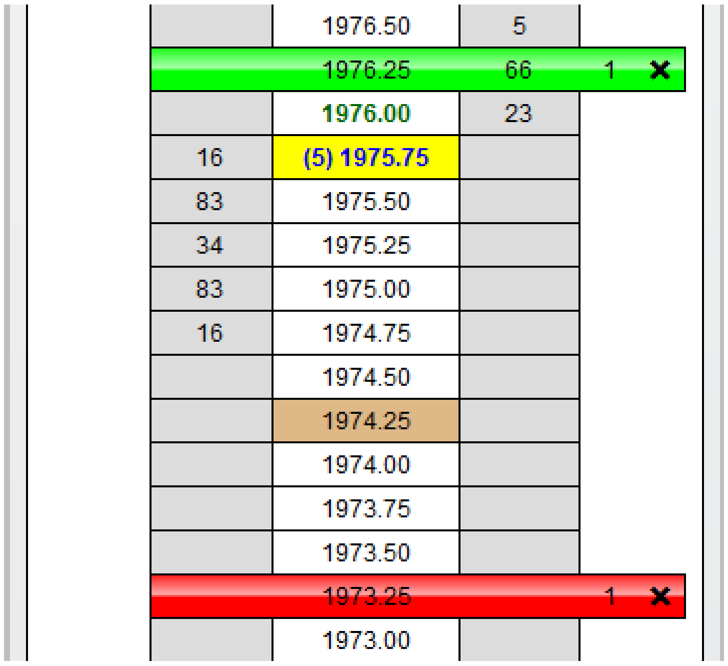


Understanding how Stop Loss and Profit Target orders are displayed

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.



	1976.50	5	
	1976.25	66	1 ✕
	1976.00	23	
16	(5) 1975.75		
83	1975.50		
34	1975.25		
83	1975.00		
16	1974.75		
	1974.50		
	1974.25		
	1974.00		
	1973.75		
	1973.50		
	1973.25		1 ✕
	1973.00		

▽ How to view out of range Stop Loss and Profit Target orders

Displaying Stop Loss and Profit Target orders outside the visible range

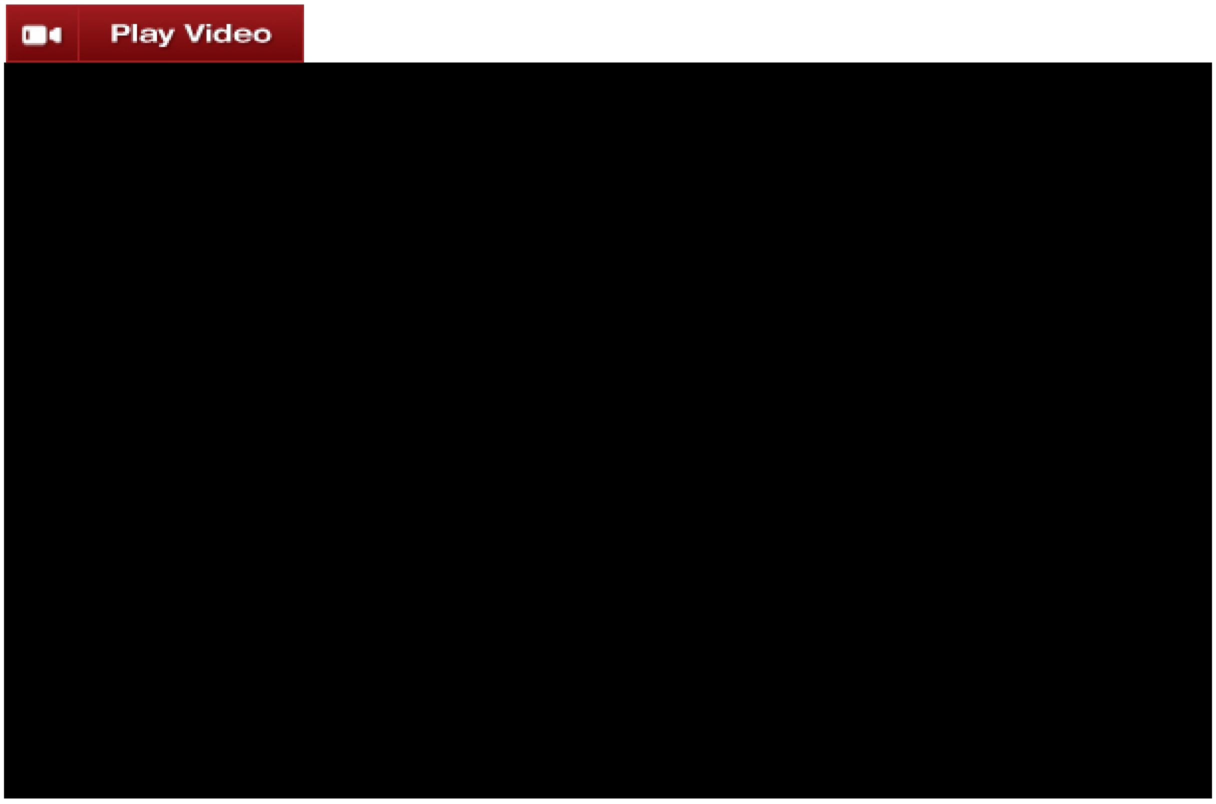
There may be times when your Stop Loss or Profit Target orders are outside of the visible price range of the SuperDOM price ladder. You can easily bring these orders in range by first disabling **Auto Center** from the **SuperDOM** right click menu, and then clicking with your middle mouse button in 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

11.19.11.4 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

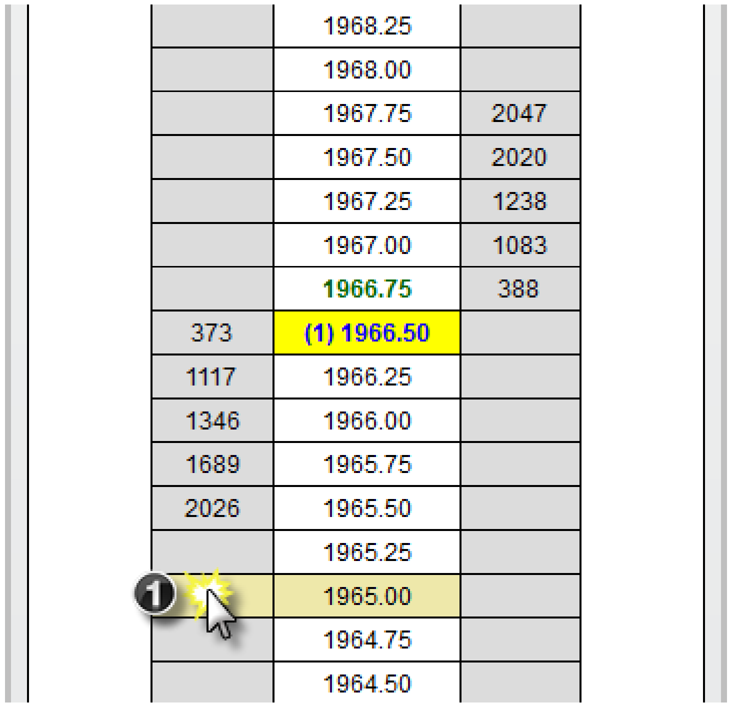
To submit an Order

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 ❶ left mouse button in the cell that corresponds to the price you wish the **limit** order to be submitted at.



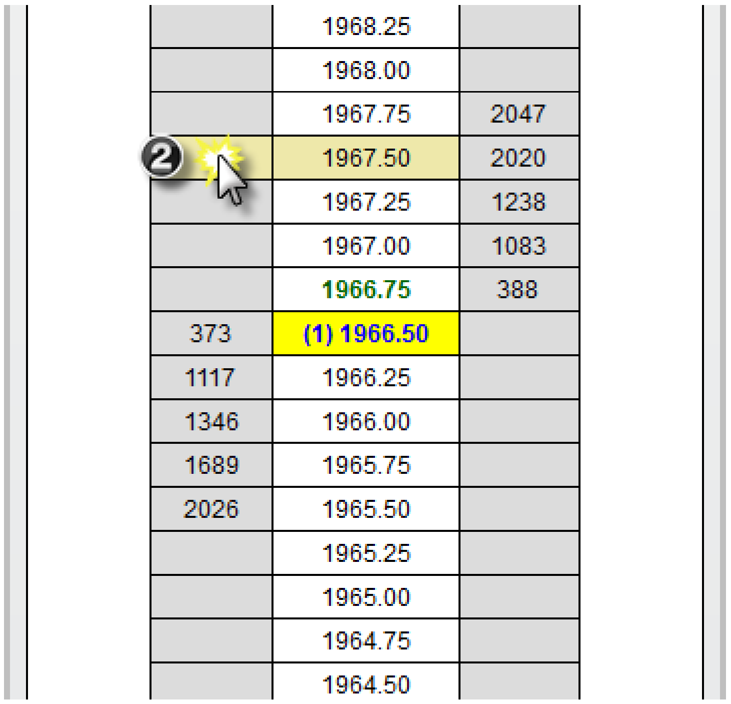
	1968.25	
	1968.00	
	1967.75	2047
	1967.50	2020
	1967.25	1238
	1967.00	1083
	1966.75	388
373	(1) 1966.50	
1117	1966.25	
1346	1966.00	
1689	1965.75	
2026	1965.50	
	1965.25	
❶	1965.00	
	1964.75	
	1964.50	

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.



	1968.25	
	1968.00	
	1967.75	2047
	1967.50	2020
	1967.25	1238
	1967.00	1083
	1966.75	388
373	(1) 1966.50	
1117	1966.25	
1346	1966.00	
1689	1965.75	
2026	1965.50	
	1965.25	
	1965.00	
	1964.75	
	1964.50	

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

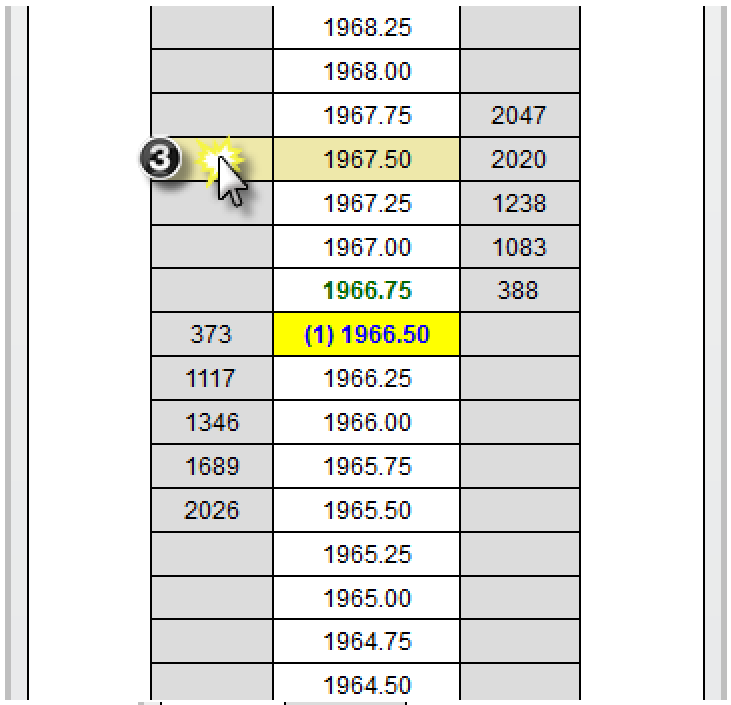
Stop-Limit Order

To submit a **stop-limit** order, select either the **Buy** column for buy orders or the

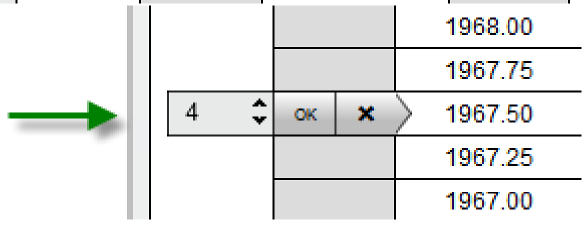
Sell column for sell orders and press down on your 3 middle mouse button (scroll wheel) in the cell that corresponds to the price you wish the **stop limit** order to be submitted at.

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 1966.50.



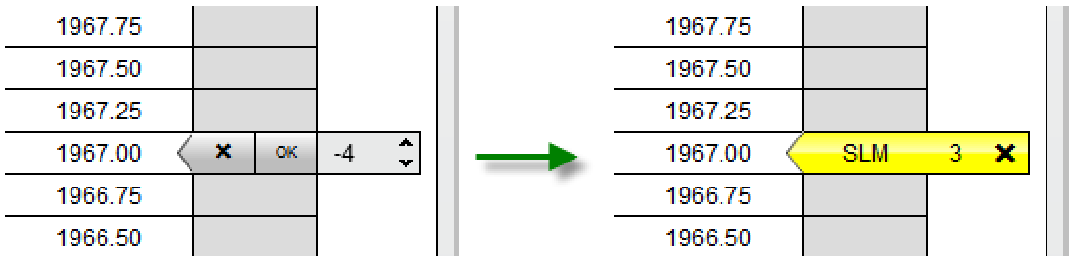
	1968.25	
	1968.00	
	1967.75	2047
	1967.50	2020
	1967.25	1238
	1967.00	1083
	1966.75	388
373	(1) 1966.50	
1117	1966.25	
1346	1966.00	
1689	1965.75	
2026	1965.50	
	1965.25	
	1965.00	
	1964.75	
	1964.50	



		1968.00
		1967.75
4	OK ×	1967.50
		1967.25
		1967.00

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.



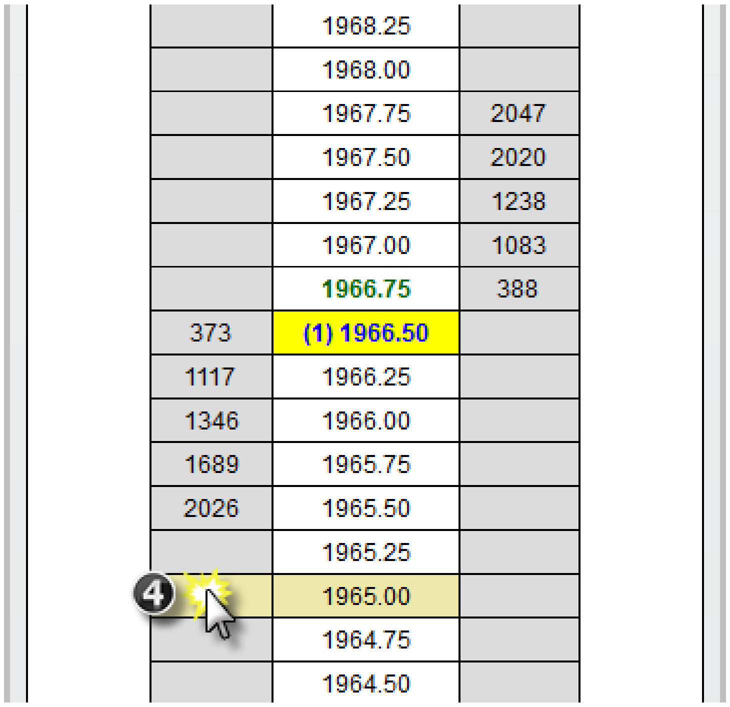
Single Click Stop-Limit Orders

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 ❹ 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.



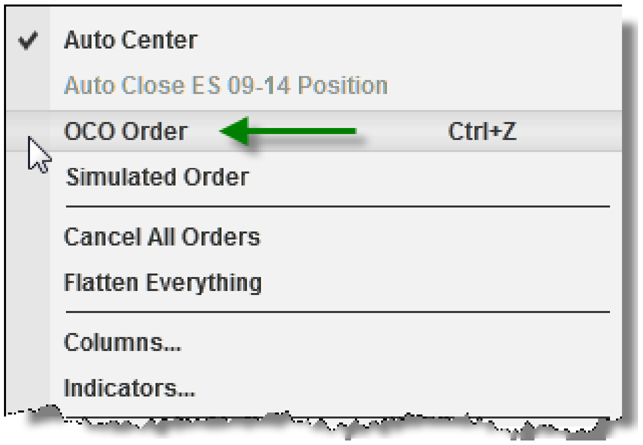
	1968.25	
	1968.00	
	1967.75	2047
	1967.50	2020
	1967.25	1238
	1967.00	1083
	1966.75	388
373	(1) 1966.50	
1117	1966.25	
1346	1966.00	
1689	1965.75	
2026	1965.50	
	1965.25	
	1965.00	
	1964.75	
	1964.50	

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

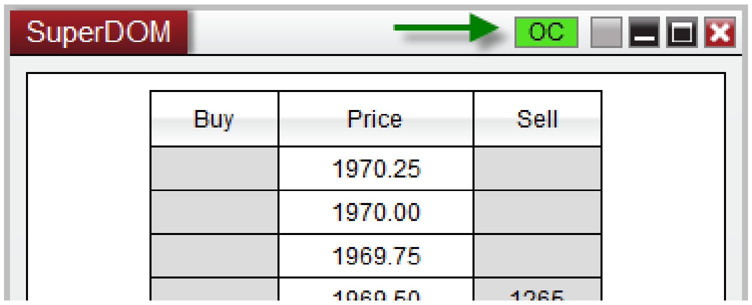
Understanding the OCO order (one cancels other) function

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.

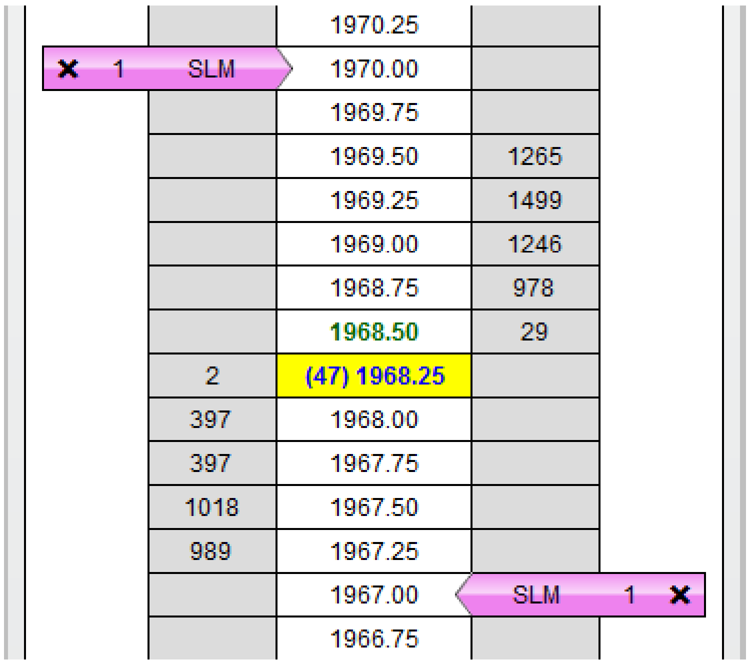
To reset an **OCO** ID, simply disable the **OCO** function, and re-enable. This will generate a new **OCO ID** and allow you to place new orders.

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.



		1970.25		
✕ 1 SLM		1970.00		
		1969.75		
		1969.50	1265	
		1969.25	1499	
		1969.00	1246	
		1968.75	978	
		1968.50	29	
	2	(47) 1968.25		
	397	1968.00		
	397	1967.75		
	1018	1967.50		
	989	1967.25		
		1967.00		SLM 1 ✕
		1966.75		

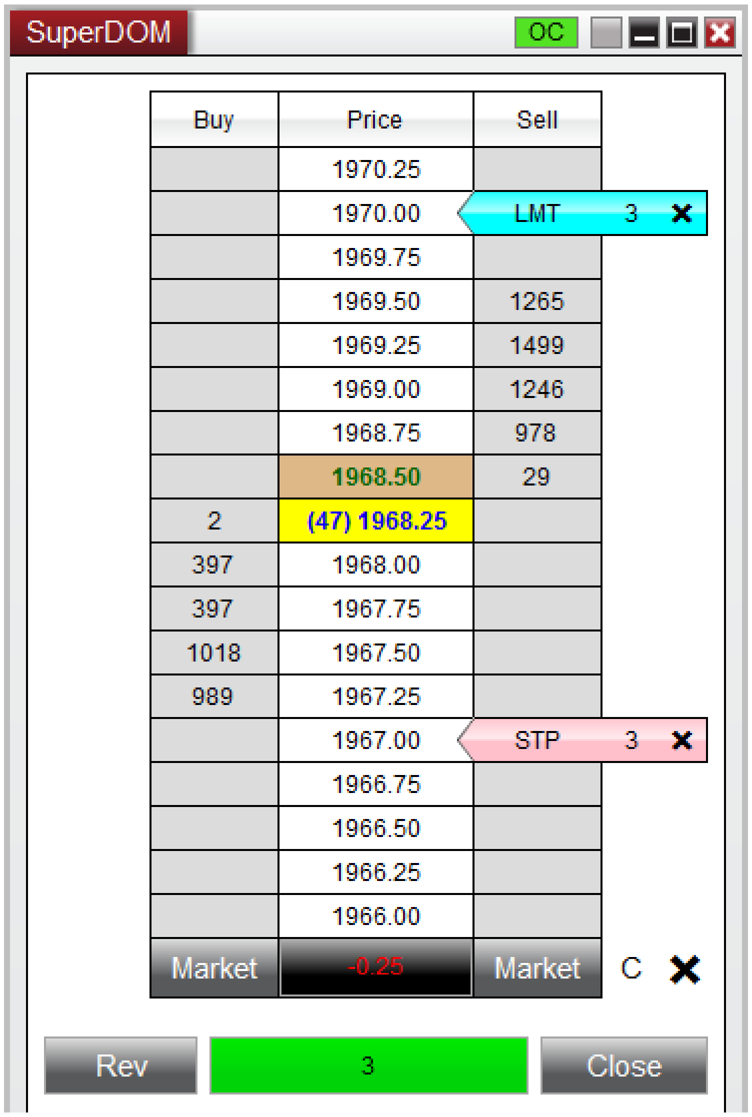
For a market fade approach just substitute **limit** orders for stop orders.

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



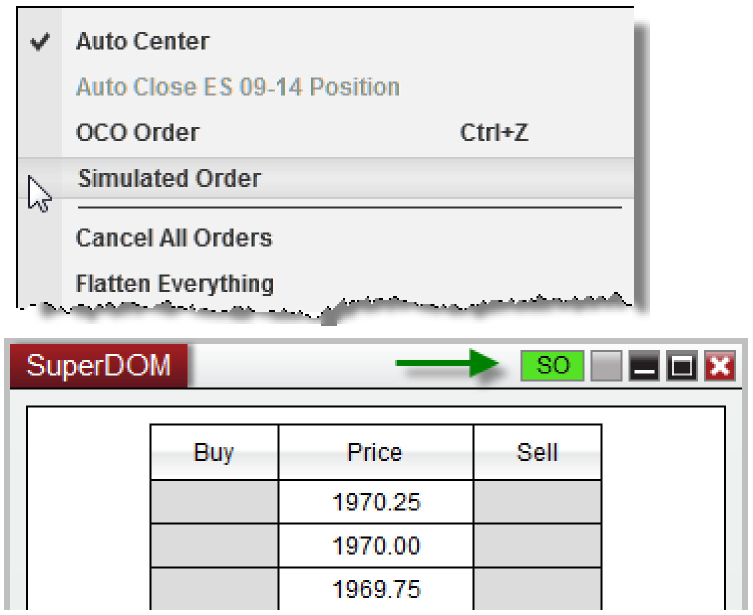
Buy	Price	Sell
	1970.25	
	1970.00	LMT 3 ✕
	1969.75	
	1969.50	1265
	1969.25	1499
	1969.00	1246
	1968.75	978
	1968.50	29
2	(47) 1968.25	
397	1968.00	
397	1967.75	
1018	1967.50	
989	1967.25	
	1967.00	STP 3 ✕
	1966.75	
	1966.50	
	1966.25	
	1966.00	
Market	-0.25	Market

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.



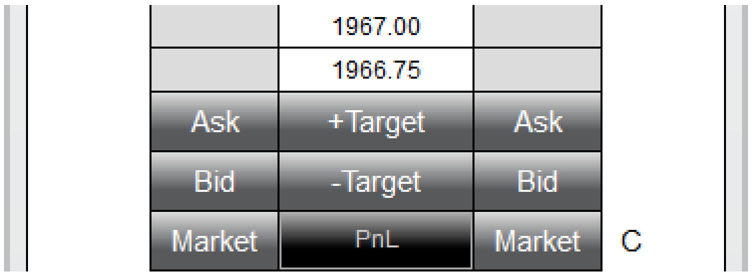
One of the powerful features of Simulated Stop Orders is that you can submit a "negative limit stop-limit" order. This means that you can place an order where the limit price is better than the stop price. As an example, you may want to buy on strength indicated by a move up to a particular price. Once 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.

11.19.11.5 Modifying and Cancelling Orders

Play Video

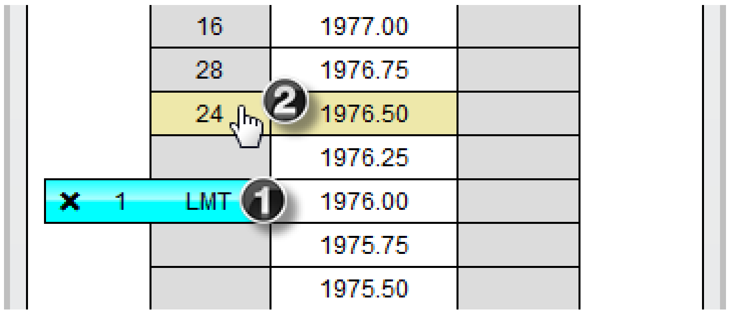
Orders are modified within the SuperDOM by selecting the order and clicking on the new price cell. Optionally you can also enable Single Click Order Modification of your Profit Target and Stop Loss orders within the SuperDOM Properties.

▽ How to modify the price of entry and exit orders

Modifying entry and exit orders

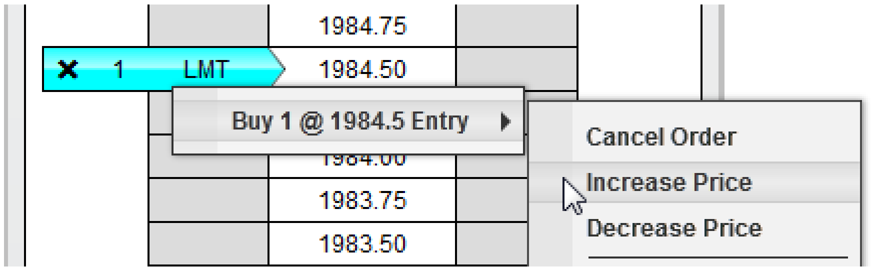
Pending orders in NinjaTrader may be modified by clicking to select the order and clicking once more at the new price point. This approach is more effective than drag and drop because it eliminates the potential errors made by accidentally letting go of your mouse button and dropping an order on the 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



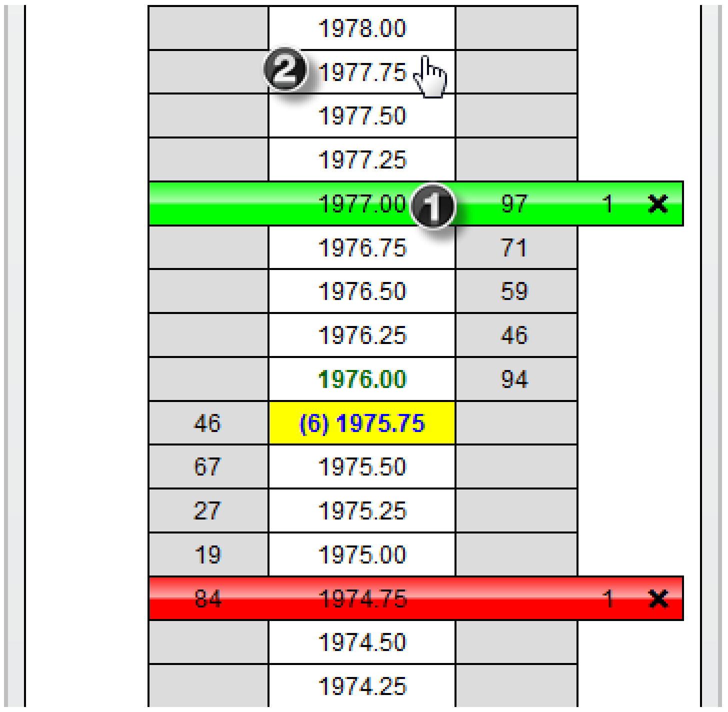
How to modify the price of Stop Loss and Profit Target orders

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.

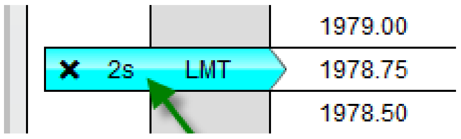


Note: If there are multiple orders consolidated at a price level, modifying the price will modify all orders at that price level.

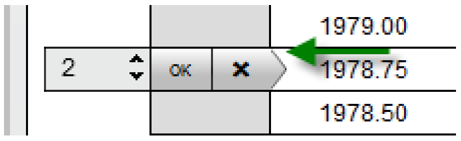
How to modify the size of an order

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

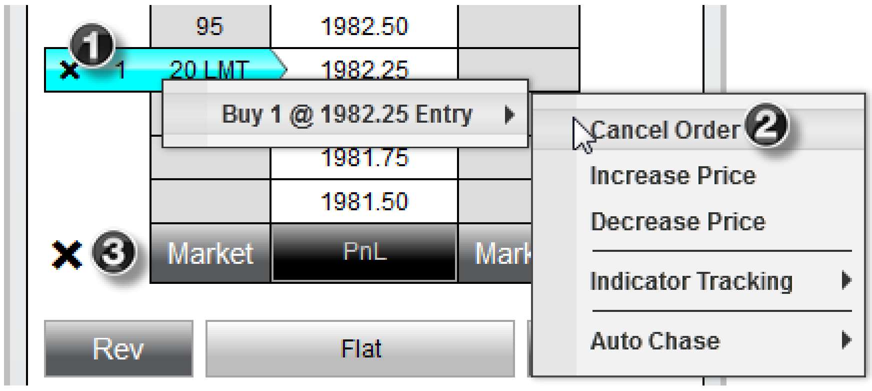
Cancelling Orders

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.

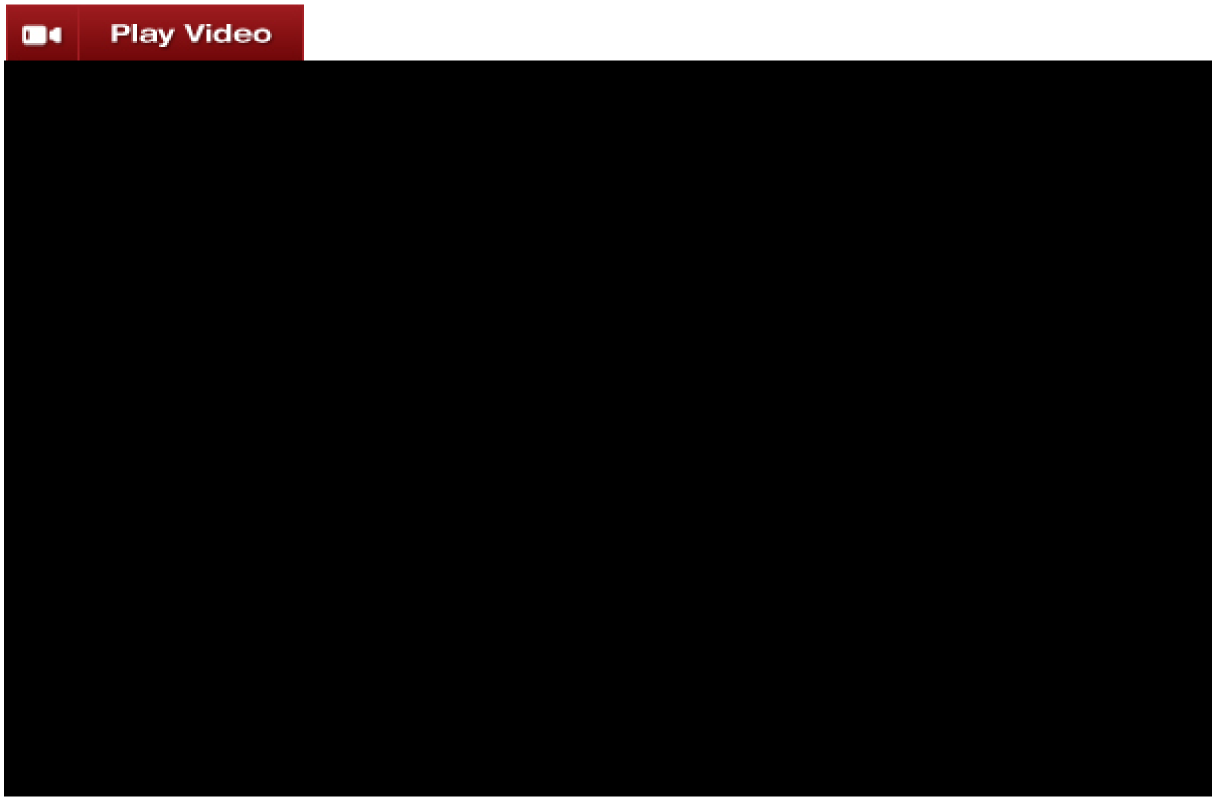
2. Pressing down on the right mouse button with the mouse cursor hovering over the order will display all orders consolidated at that price. You can then select any individual order for cancellation.

3. Pressing on the large "**X**" will cancel all orders on either the "BUY" side (in this example) or the sell side.

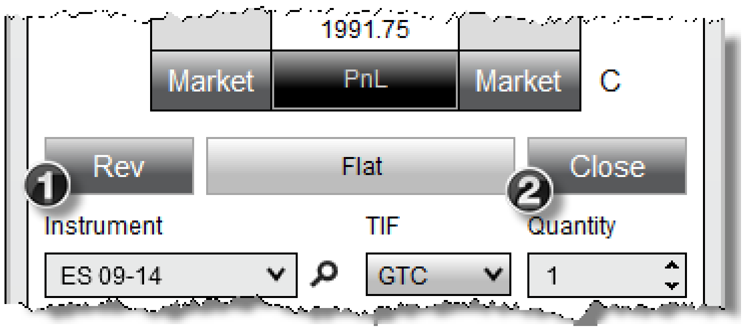


You can also cancel "ALL" orders by right mouse clicking inside the **SuperDOM** and selecting the menu item **Cancel All Orders**.

11.19.11.0 Managing Positions



The SuperDOM has action buttons that allow you to quickly: close open positions, reverse positions, or even add/remove targets to your ATM Strategy

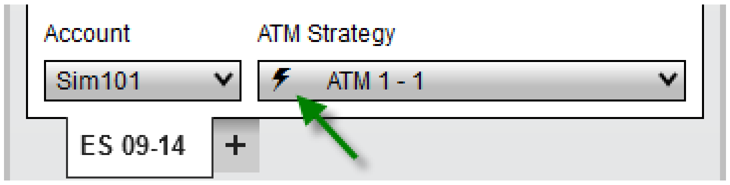


1. Left mouse clicking on the "Rev" will close the current open position and open a reverse position.

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.

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

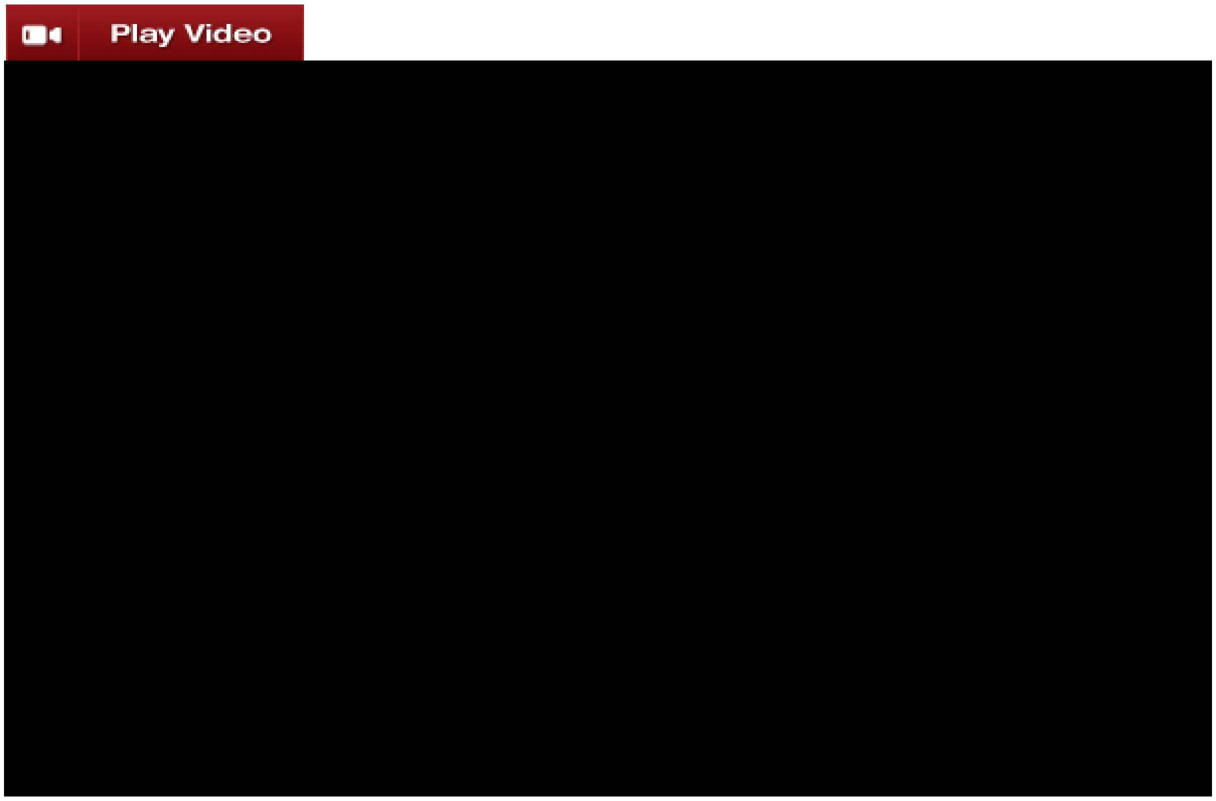
There are two locations within the SuperDOM where you can add or remove a target.

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

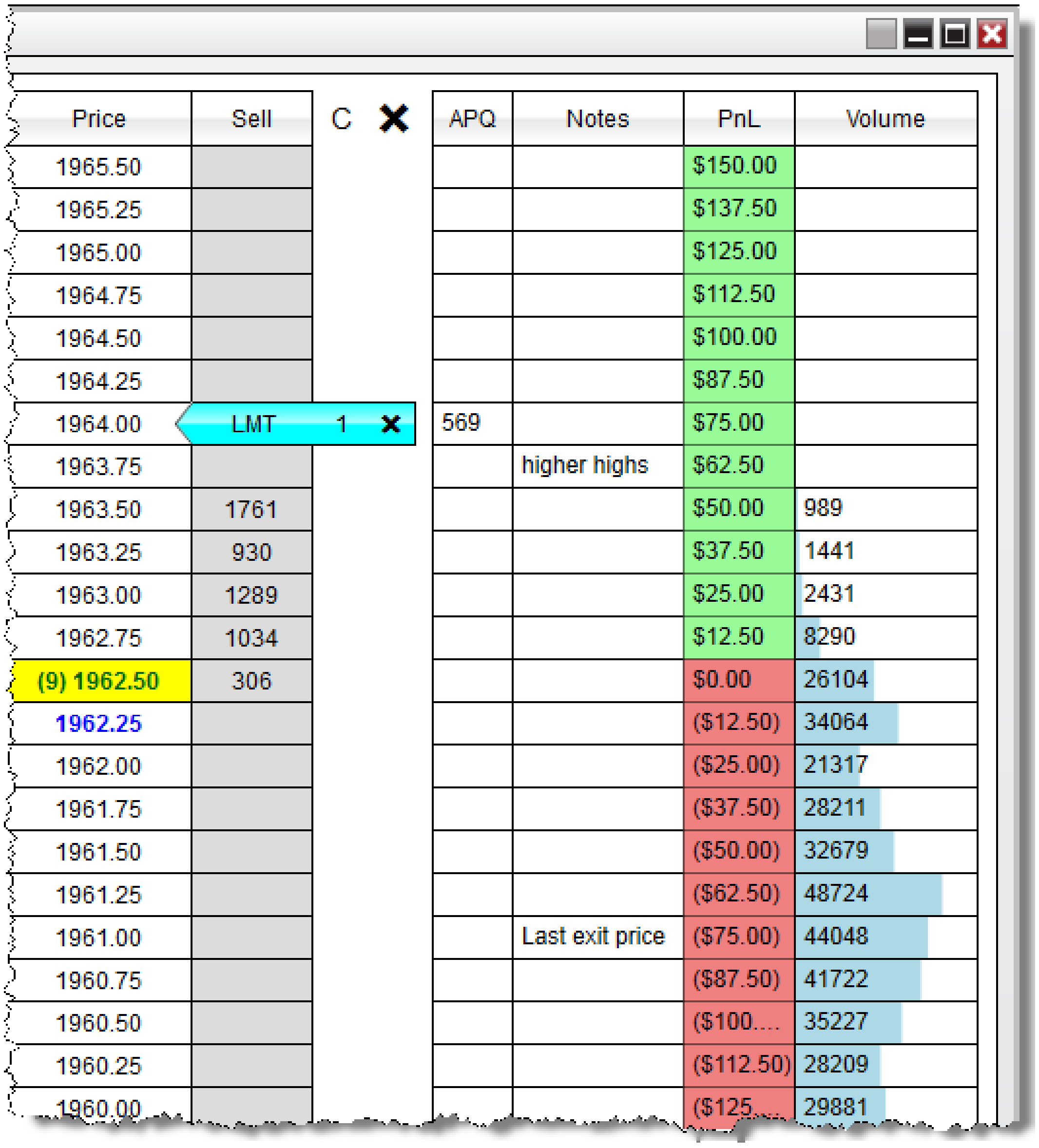
SuperDOM properties dialog window

2. Right mouse click context menu and select **Add Target** or **Remove Target**

11.19.11.7 Using SuperDOM Columns



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.



Price	Sell	C	✖	APQ	Notes	PnL	Volume
1965.50						$150.00	
1965.25						$137.50	
1965.00						$125.00	
1964.75						$112.50	
1964.50						$100.00	
1964.25						$87.50	
1964.00	LMT	1	×	569		$75.00	
1963.75					higher highs	$62.50	
1963.50	1761					$50.00	989
1963.25	930					$37.50	1441
1963.00	1289					$25.00	2431
1962.75	1034					$12.50	8290
(9) 1962.50	306					$0.00	26104
1962.25						($12.50)	34064
1962.00						($25.00)	21317
1961.75						($37.50)	28211
1961.50						($50.00)	32679
1961.25						($62.50)	48724
1961.00					Last exit price	($75.00)	44048
1960.75						($87.50)	41722
1960.50						($100....	35227
1960.25						($112.50)	28209
1960.00						($125....	29881

Understanding the Columns window

The Columns window is used to add, remove and edit all columns within a **SuperDOM**

Accessing the Columns Window

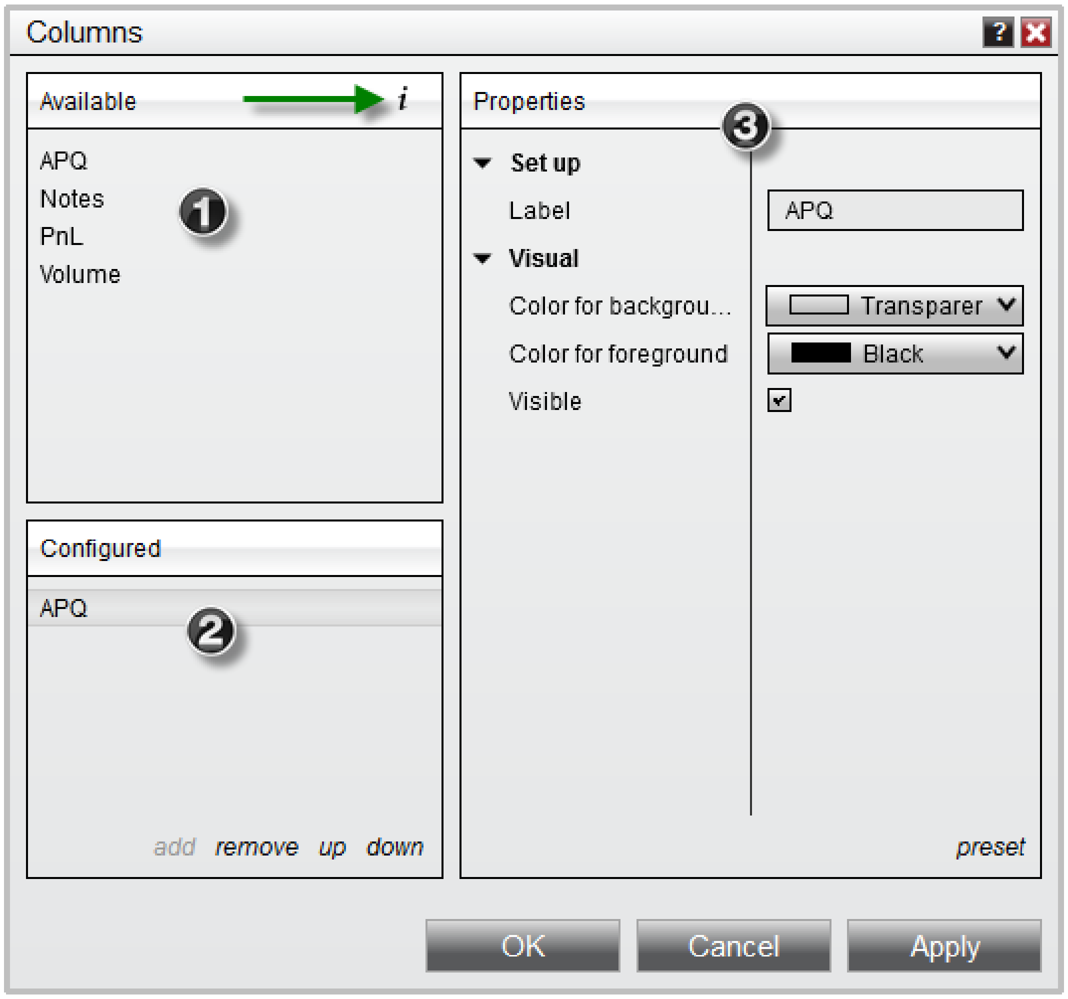
- Right mouse click on the **SuperDOM** window and select the menu **Columns**

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**



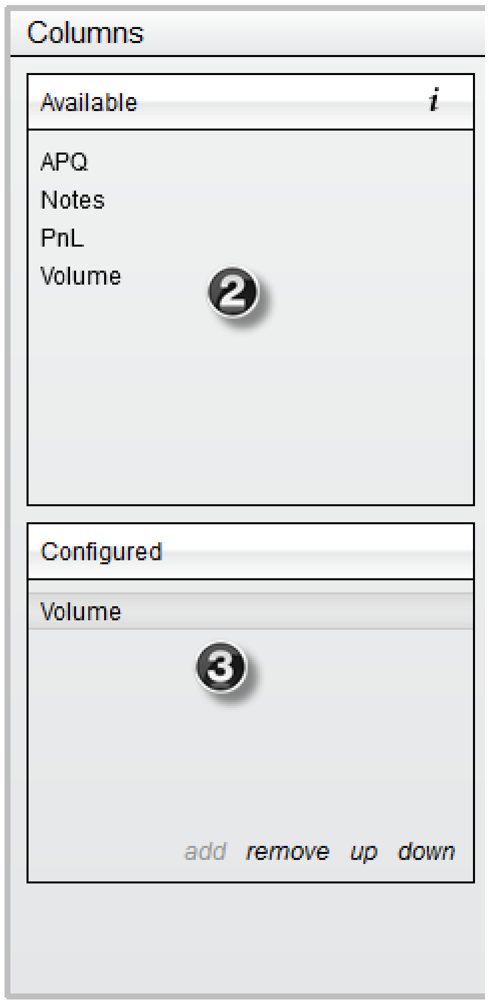
How to add columns

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)

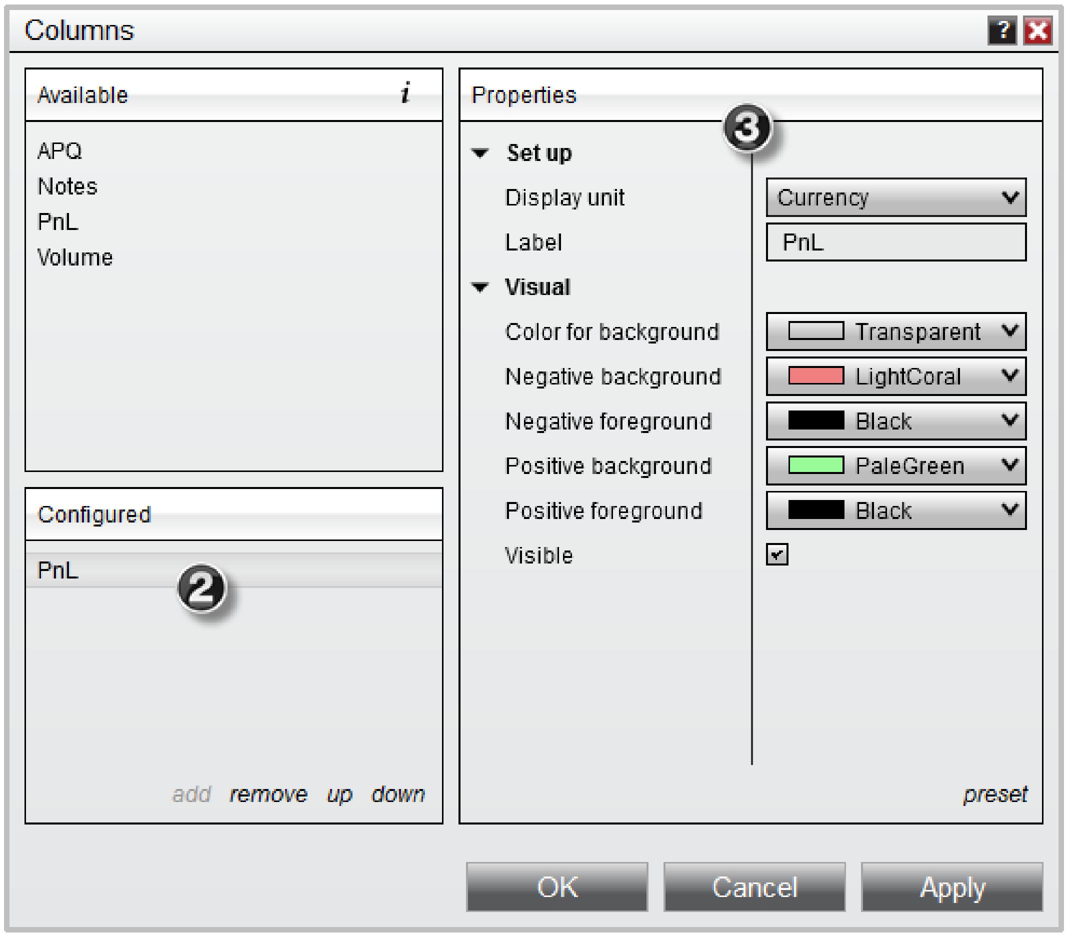


▽ How to edit a column's parameters

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.



Set up
Display unit: Currency
Label: PnL
Visual
Color for background: Transparent
Negative background: LightCoral
Negative foreground: Black
Positive background: PaleGreen
Positive foreground: Black
Visible: ✓

Column Parameters

The following parameters are common to most columns:

Setup	
Label	Sets the text used for the column header
Visual	
Color for backgroun d	Sets the color used for the column cells
Color for foreground	Sets the color used for the column text

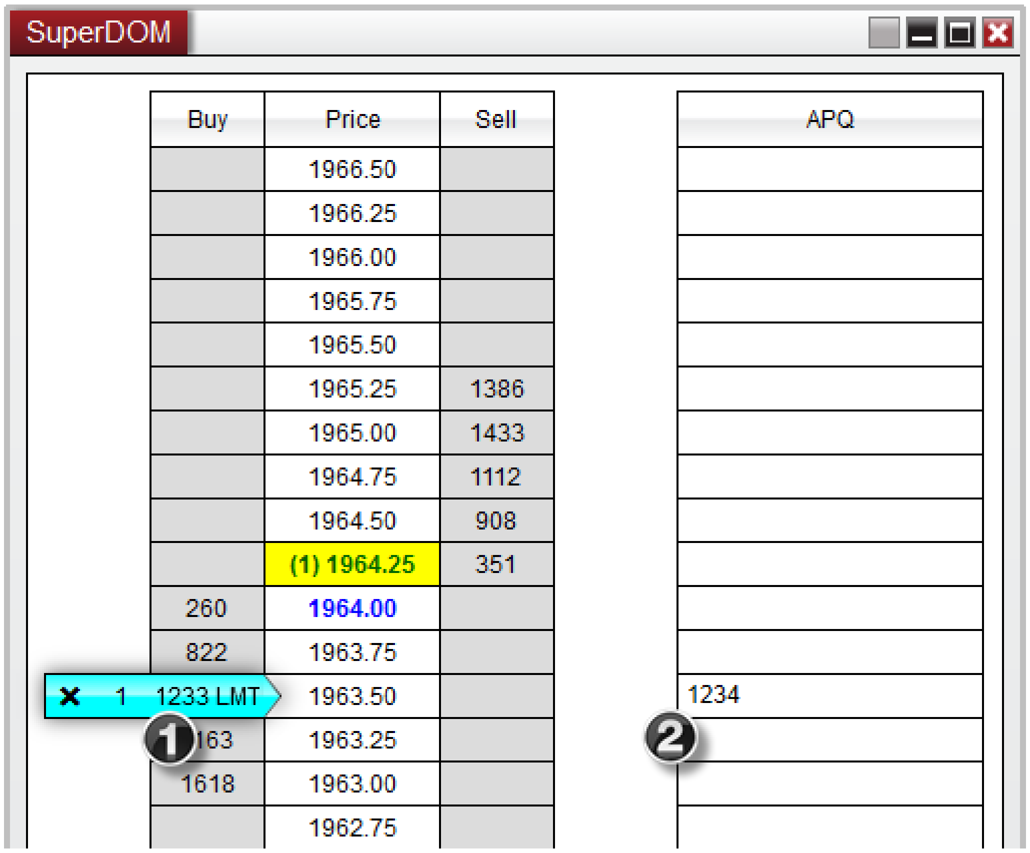
Visible	Enables / Disables if the column should be displayed on the SuperDOM
Time Frame	
Trading Hours	Sets the hours used for historical bar calculations

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 default system columns

APQ (Approximate Position in Queue) Column

The **APQ** column will calculate the number of contract resting ahead of your **Limit** or **MIT** 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.



Buy	Price	Sell	APQ
	1966.50		
	1966.25		
	1966.00		
	1965.75		
	1965.50		
	1965.25	1386	
	1965.00	1433	
	1964.75	1112	
	1964.50	908	
	(1) 1964.25	351	
260	1964.00		
822	1963.75		
× 1 1233 LMT	1963.50		1234
163	1963.25		
1618	1963.00		
	1962.75		

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.

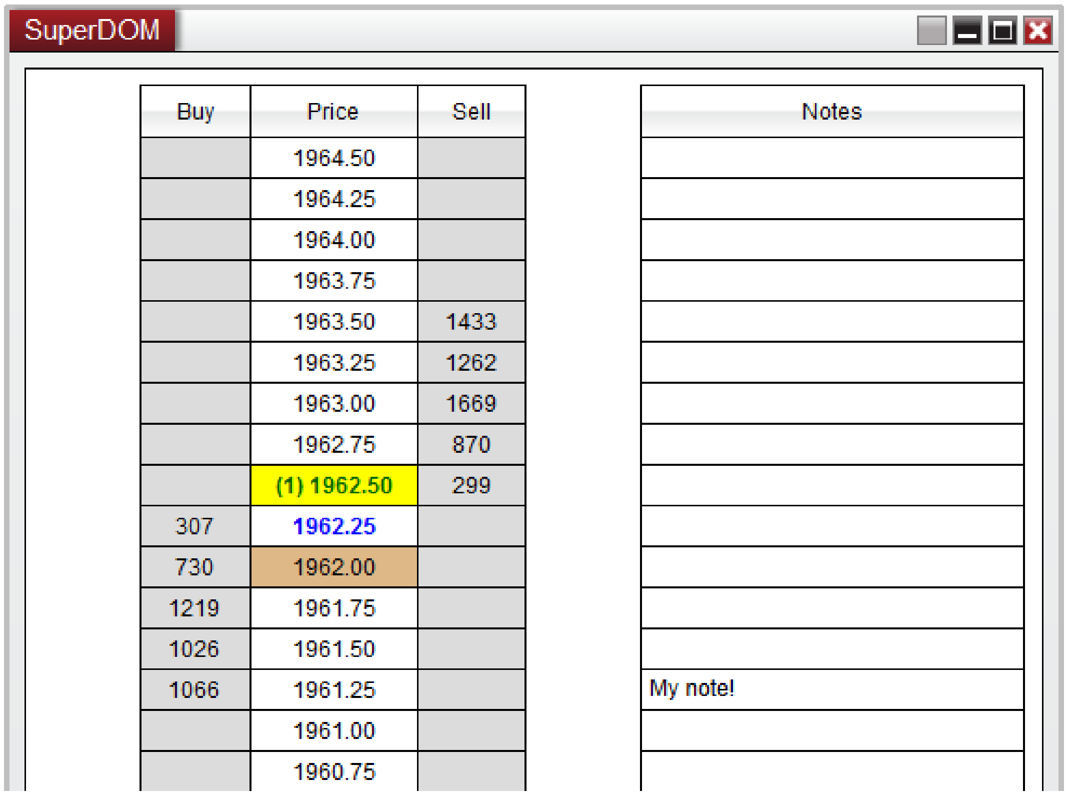
Note: The value displayed in the **APQ** is a calculation based on the volume from your data provider. For simulated orders, there is no way to accurately track your order against the live orders that are being sent from the data provider and filled at a live exchange, and as a result, the estimate will 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.

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.



Buy	Price	Sell	Notes
	1964.50		
	1964.25		
	1964.00		
	1963.75		
	1963.50	1433	
	1963.25	1262	
	1963.00	1669	
	1962.75	870	
	(1) 1962.50	299	
307	1962.25		
730	1962.00		
1219	1961.75		
1026	1961.50		
1066	1961.25		My note!
	1961.00		
	1960.75		

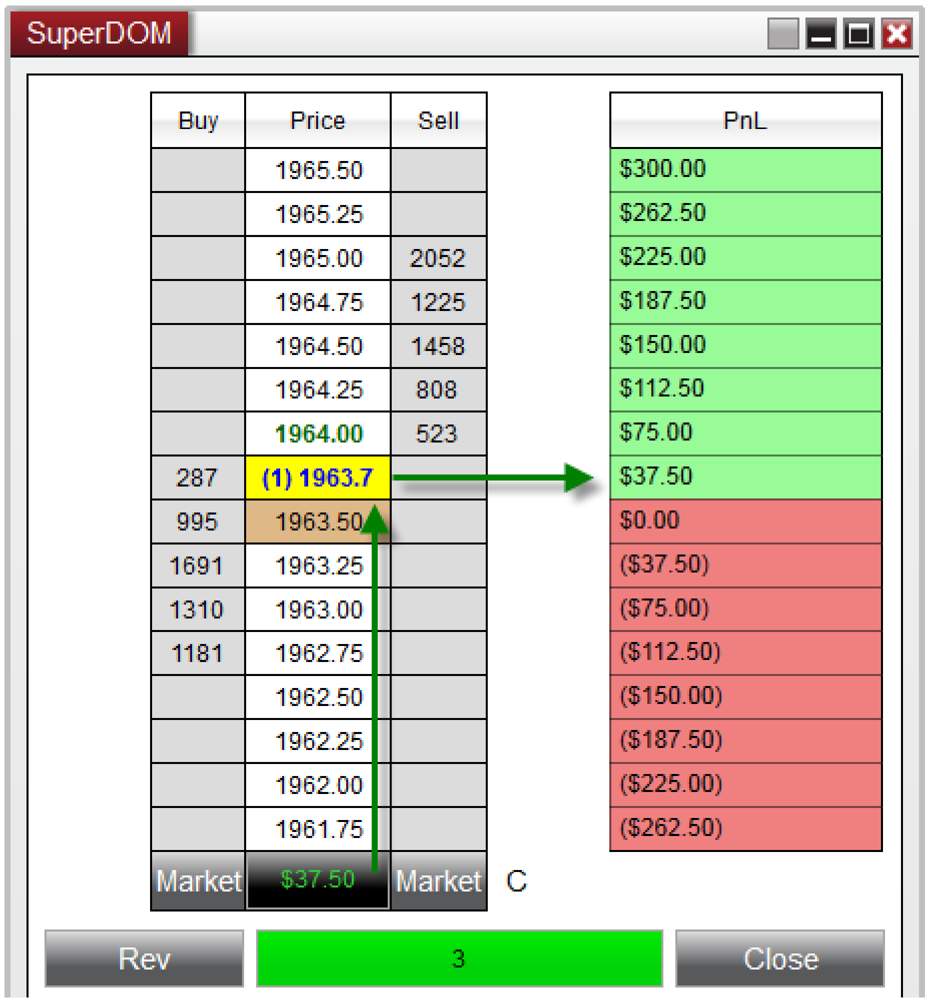
Your custom note will now be synchronized with the price corresponding price row and will remain at that price level as your scroll up or down on the **SuperDOM**.

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.

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.



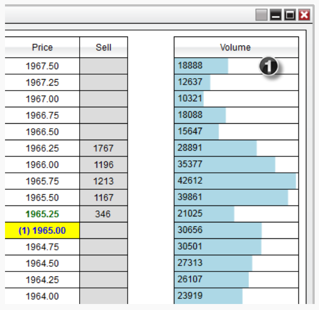
Buy	Price	Sell	PnL
	1965.50		$300.00
	1965.25		$262.50
	1965.00	2052	$225.00
	1964.75	1225	$187.50
	1964.50	1458	$150.00
	1964.25	808	$112.50
	1964.00	523	$75.00
287	(1) 1963.7		$37.50
995	1963.50		$0.00
1691	1963.25		($37.50)
1310	1963.00		($75.00)
1181	1962.75		($112.50)
	1962.50		($150.00)
	1962.25		($187.50)
	1962.00		($225.00)
	1961.75		($262.50)
Market	$37.50	Market	

Volume Column

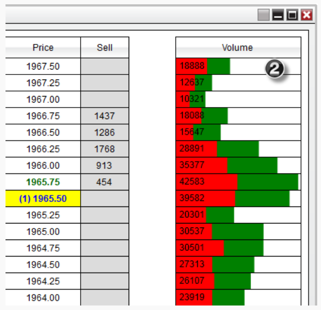
The **Volume Column** will display the number of contracts that have traded in the current session. This column has two Setup Properties to determine how the volume information is displayed. You data feed provider must support historical tick data and is using the **Volume Column** in Buy/Sell mode must 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



Price	Sell	Volume
1967.50		18888
1967.25		12637
1967.00		10321
1966.75		18088
1966.50		15647
1966.25	1767	28891
1966.00	1196	35377
1965.75	1213	42612
1965.50	1167	39861
1965.25	346	21025
(1) 1965.00		30656
1964.75		30501
1964.50		27313
1964.25		26107
1964.00		23919



Price	Sell	Volume
1967.50		18888
1967.25		12637
1967.00		10321
1966.75	1437	18088
1966.50	1286	15647
1966.25	1768	28891
1966.00	913	35377
1965.75	454	42583
(1) 1965.50		39582
1965.25		20301
1965.00		30537
1964.75		30501
1964.50		27313
1964.25		26107
1964.00		23919

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.

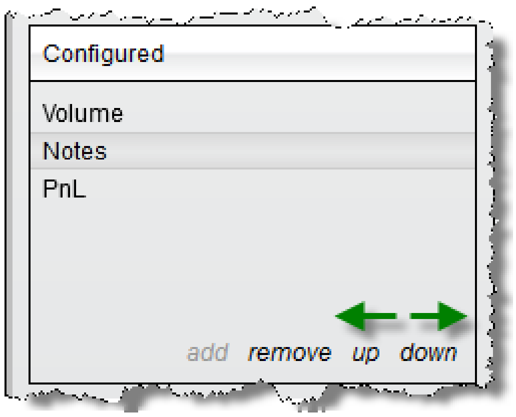
Customize the display of columns

Moving/Resizing Columns

Each column added to the **SuperDOM** can be individually resized or moved.

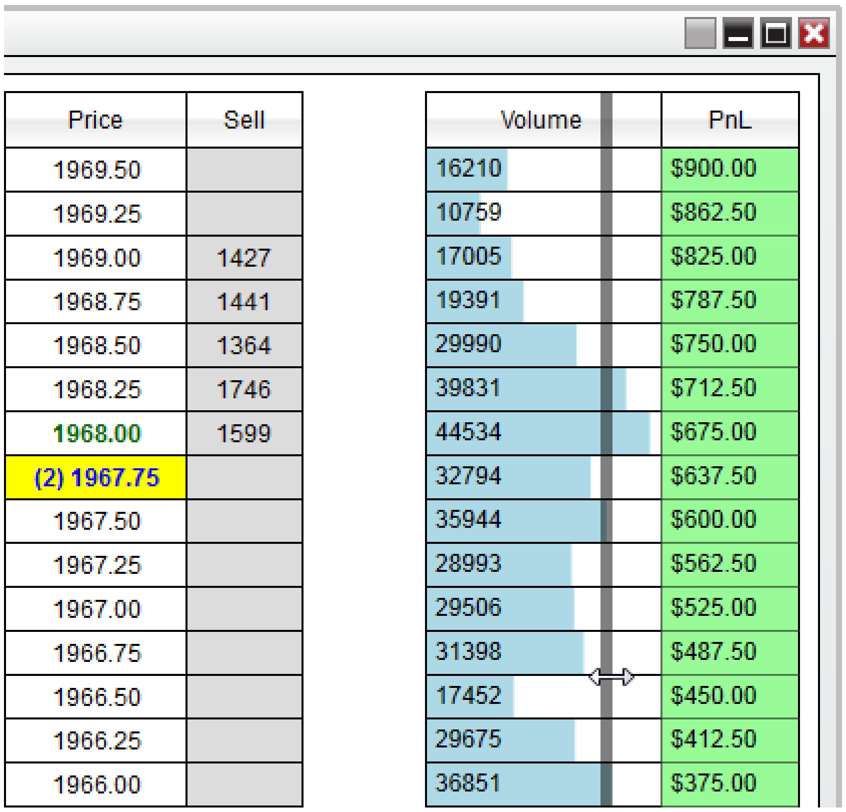
To move the order of columns in the **SuperDOM** window

- 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:

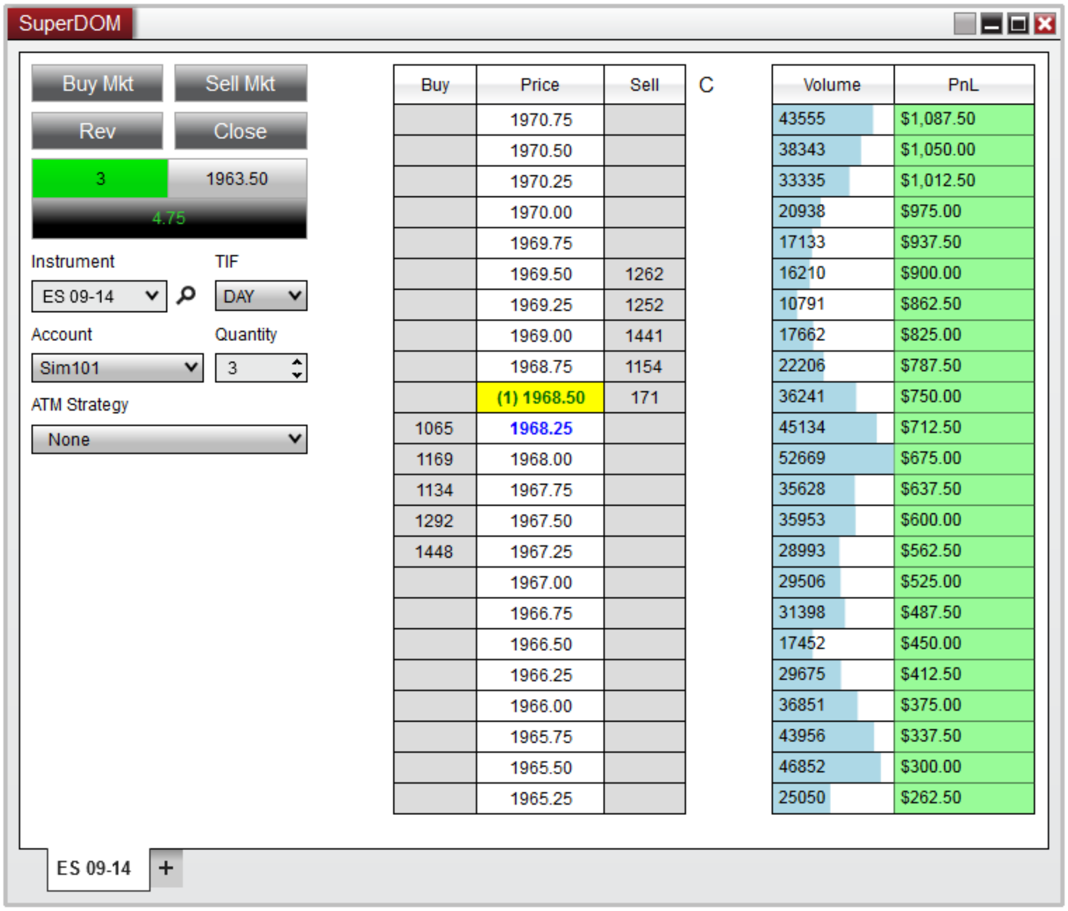
- Move your cursor to the edge of the column you wish to resize, where your cursor will turn into a left and right facing arrow
- Left mouse click and drag to meet the width you desire



Price	Sell	Volume		PnL
1969.50		16210		$900.00
1969.25		10759		$862.50
1969.00	1427	17005		$825.00
1968.75	1441	19391		$787.50
1968.50	1364	29990		$750.00
1968.25	1746	39831		$712.50
1968.00	1599	44534		$675.00
(2) 1967.75		32794		$637.50
1967.50		35944		$600.00
1967.25		28993		$562.50
1967.00		29506		$525.00
1966.75		31398		$487.50
1966.50		17452		$450.00
1966.25		29675		$412.50
1966.00		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.



Instrument: ES 09-14
TIF: DAY
Account: Sim101
Quantity: 3
ATM Strategy: None

Buy	Price	Sell	C	Volume	PnL
	1970.75			43555	$1,087.50
	1970.50			38343	$1,050.00
	1970.25			33335	$1,012.50
	1970.00			20938	$975.00
	1969.75			17133	$937.50
	1969.50	1262		16210	$900.00
	1969.25	1252		10791	$862.50
	1969.00	1441		17662	$825.00
	1968.75	1154		22206	$787.50
	(1) 1968.50	171		36241	$750.00
1065	1968.25			45134	$712.50
1169	1968.00			52669	$675.00
1134	1967.75			35628	$637.50
1292	1967.50			35953	$600.00
1448	1967.25			28993	$562.50
	1967.00			29506	$525.00
	1966.75			31398	$487.50
	1966.50			17452	$450.00
	1966.25			29675	$412.50
	1966.00			36851	$375.00
	1965.75			43956	$337.50
	1965.50			46852	$300.00
	1965.25			25050	$262.50

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 http://www.ninjatrader.com/partners.php#NinjaScript-Consultants to build your custom indicators is also available.

11.19.11.8 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.

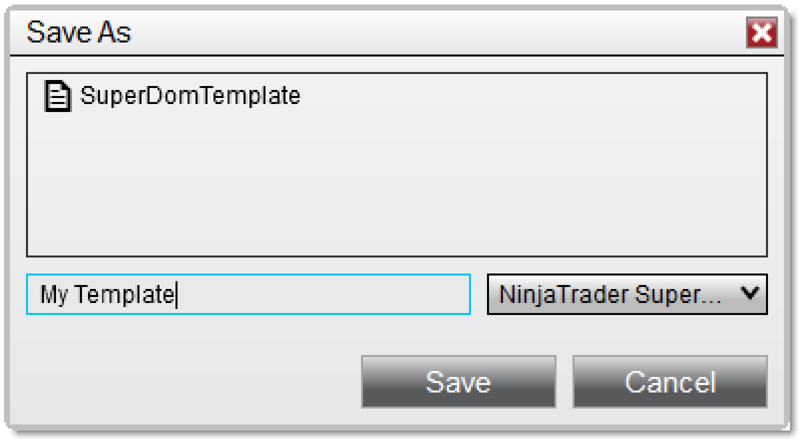
Changing the Default SuperDOM Template

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:

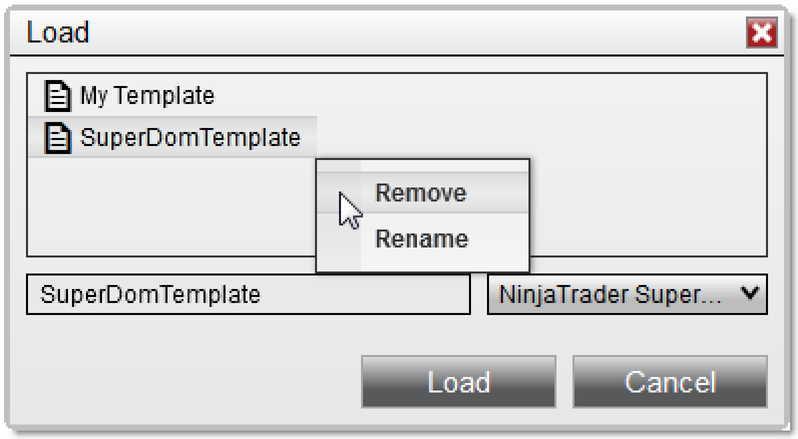
1. Right mouse click within a chart and select the menu item **Templates** followed by either the **Save As** or **Load** menu items
2. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the template for removal from the list of templates, then select the **Remove** menu item.

Renaming a SuperDOM Template

To rename an existing **SuperDOM Template** from the list of saved templates:

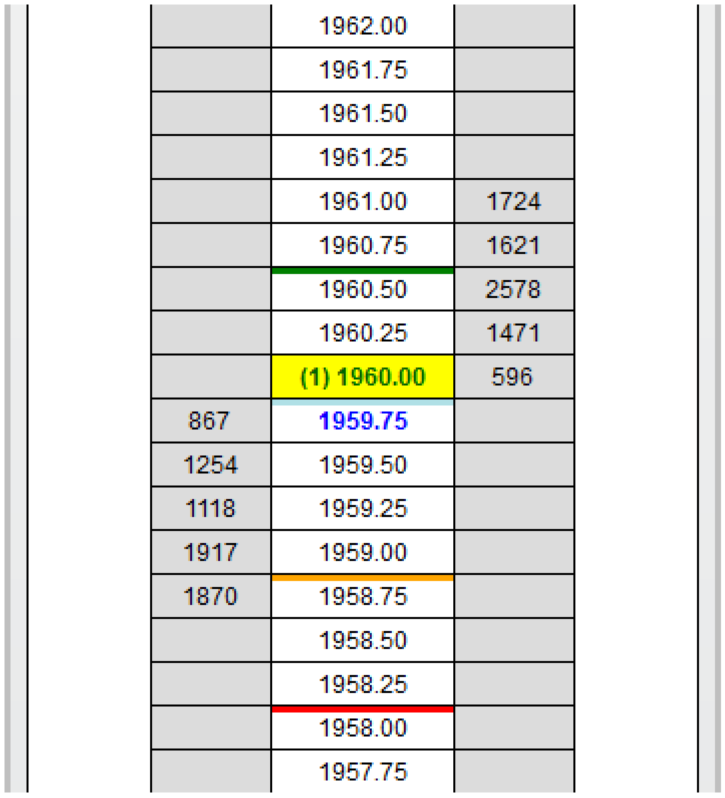
3. Right mouse click within a chart and select the menu item **Templates** followed by either the **Save As** or **Load** menu items
4. The **Save** or **Load** window will appear, depending on which menu item you selected. Right mouse click the template from the list of templates, then select the **Rename** menu item.

In the image below, we can either remove or rename the selected **SuperDOM Template**.



11.19.11.9 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.



	1962.00	
	1961.75	
	1961.50	
	1961.25	
	1961.00	1724
	1960.75	1621
	1960.50	2578
	1960.25	1471
	(1) 1960.00	596
867	1959.75	
1254	1959.50	
1118	1959.25	
1917	1959.00	
1870	1958.75	
	1958.50	
	1958.25	
	1958.00	
	1957.75	

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**.

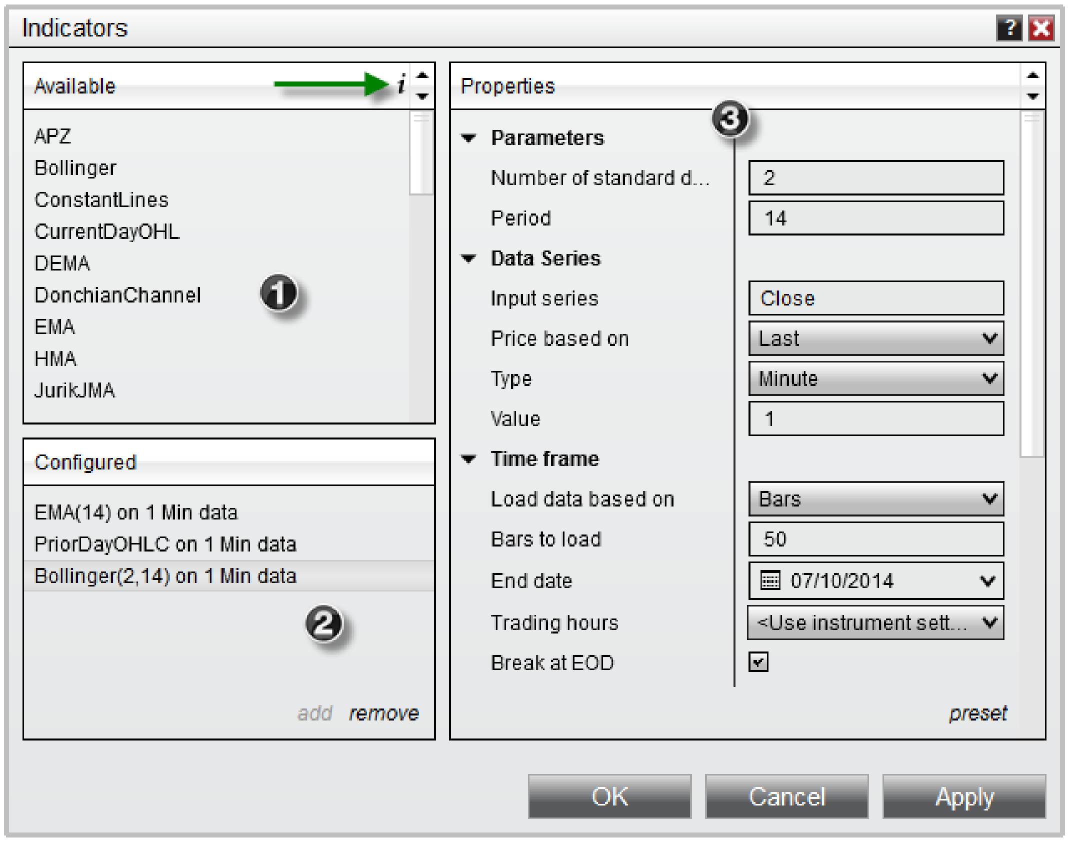
Accessing the Indicators Window

- 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**



Available: APZ, Bollinger, ConstantLines, CurrentDayOHL, DEMA, DonchianChannel, EMA, HMA, JurikJMA

Configured: EMA(14) on 1 Min data, PriorDayOHLC on 1 Min data, Bollinger(2,14) on 1 Min data

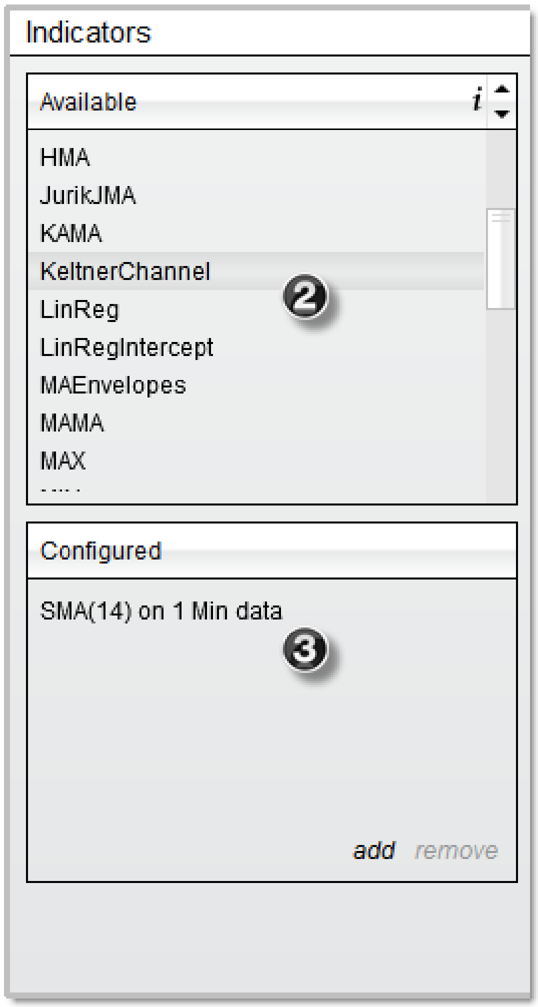
Parameters	
Number of standard d...	2
Period	14
Data Series	
Input series	Close
Price based on	Last
Type	Minute
Value	1
Time frame	
Load data based on	Bars
Bars to load	50
End date	07/10/2014
Trading hours	<Use instrument sett...
Break at EOD	✓

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)*

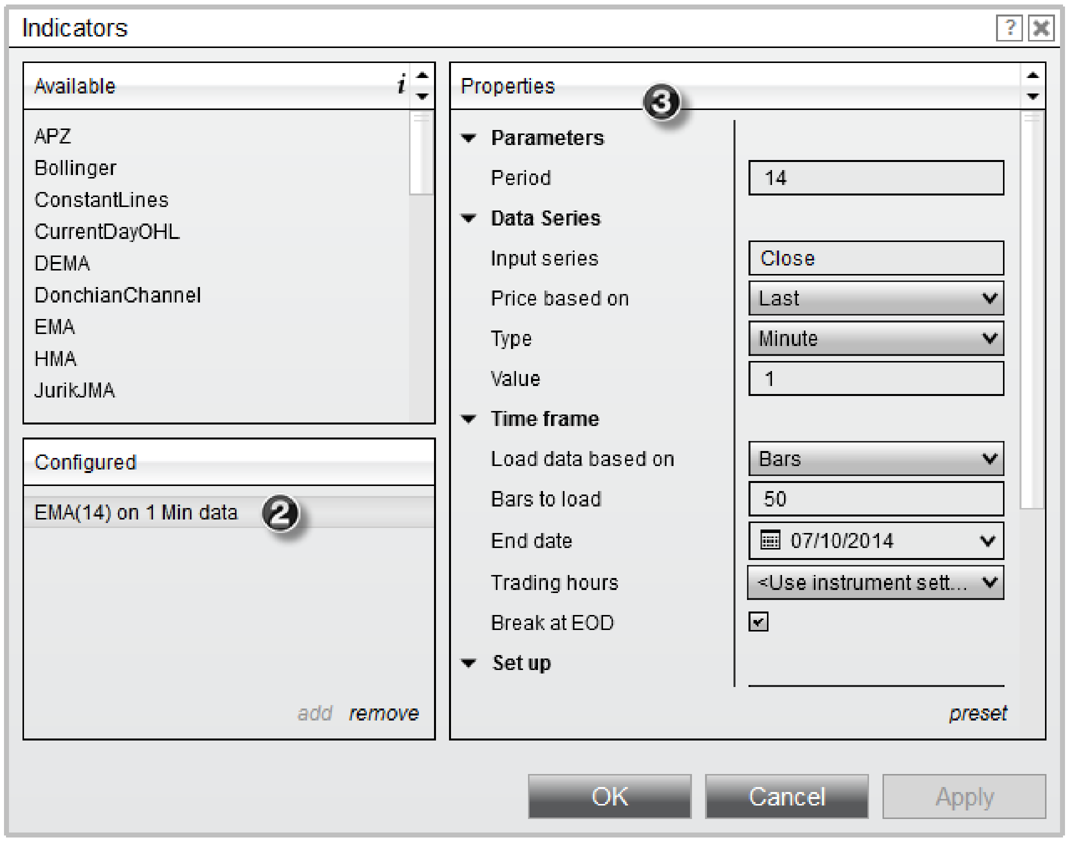


How to edit an indicator's parameters

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.



Parameters
Period 14
Data Series
Input series Close
Price based on Last
Type Minute
Value 1
Time frame
Load data based on Bars
Bars to load 50
End date 07/10/2014
Trading hours <Use instrument sett...
Break at EOD ✓
Set up

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.

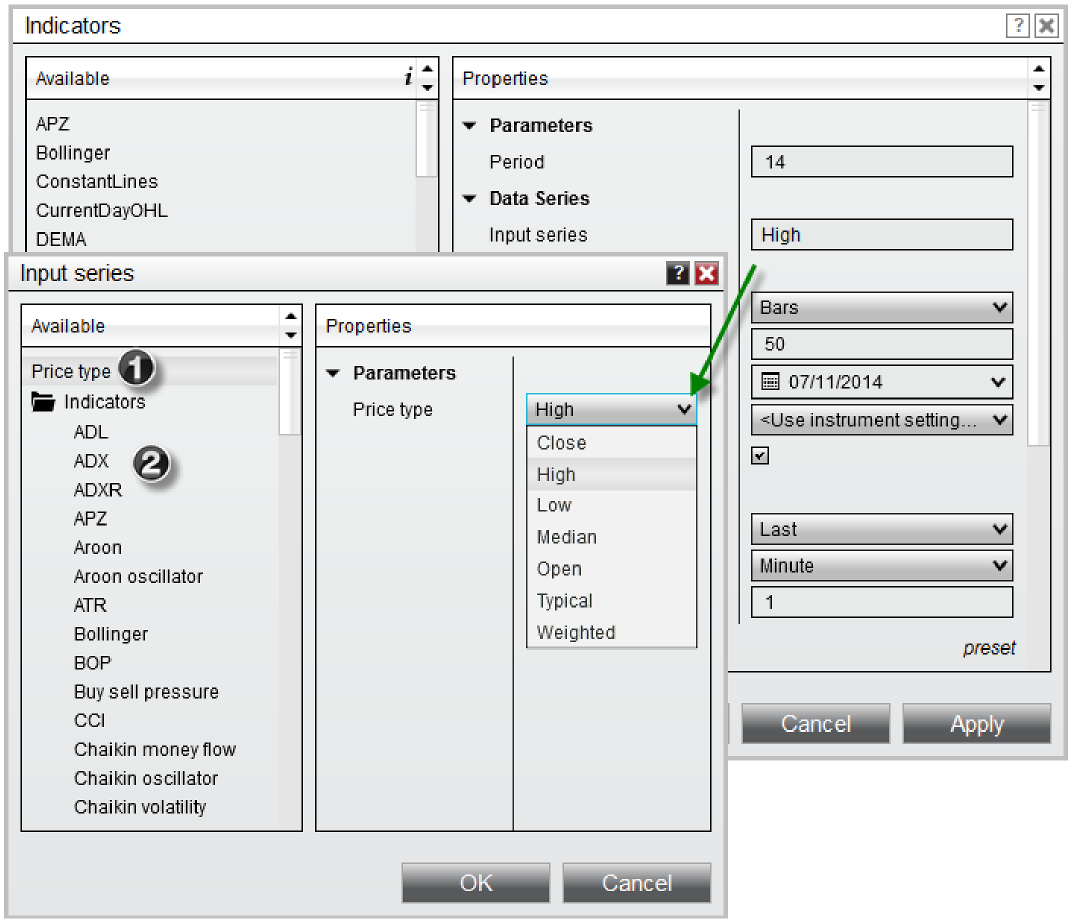
Indicator Input Series

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**.

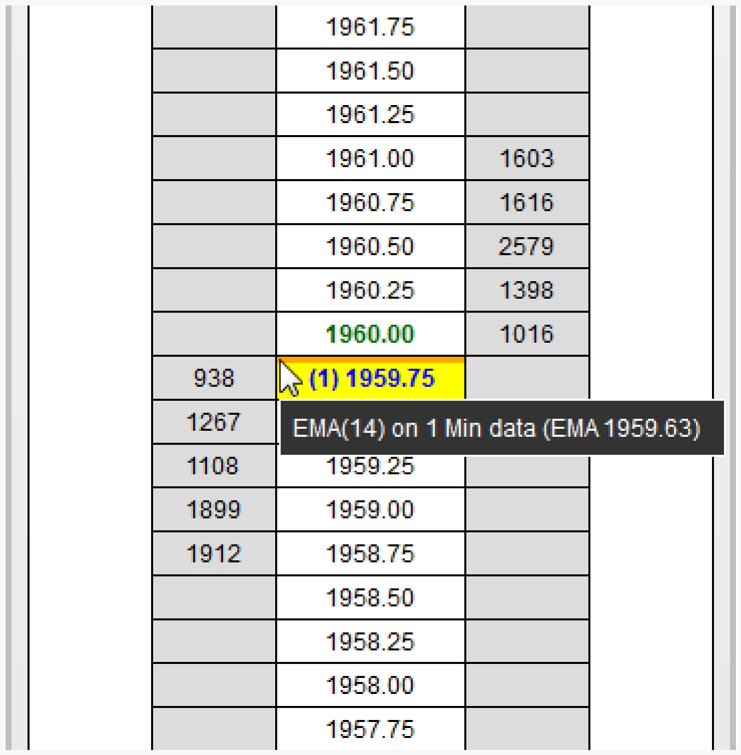
2. Additionally, you can also choose another indicator as the input series. When you select another indicator as the input series, you can define the properties used in the input series for the second indicator. Once you have selected the input series of your choice left mouse click the OK button to 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.



	1961.75	
	1961.50	
	1961.25	
	1961.00	1603
	1960.75	1616
	1960.50	2579
	1960.25	1398
	1960.00	1016
938	(1) 1959.75	
1267		
1108	1959.25	
1899	1959.00	
1912	1958.75	
	1958.50	
	1958.25	
	1958.00	
	1957.75	

EMA(14) on 1 Min data (EMA 1959.63)

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.

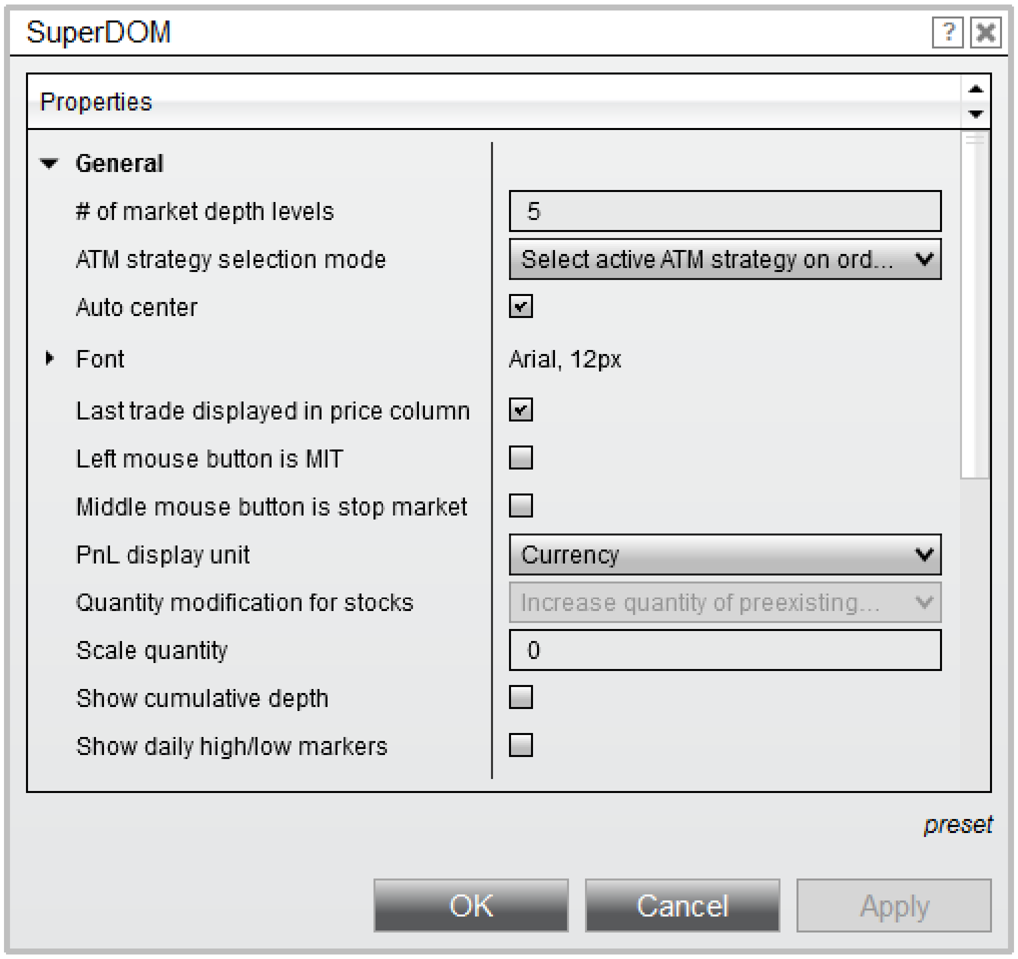
11.19.11.´Properites

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


SuperDOM

Properties

General

# of market depth levels	5
ATM strategy selection mode	Select active ATM strategy on ord...
Auto center	✓
Font	Arial, 12px
Last trade displayed in price column	✓
Left mouse button is MIT	☐
Middle mouse button is stop market	☐
PnL display unit	Currency
Quantity modification for stocks	Increase quantity of preexisting...
Scale quantity	0
Show cumulative depth	☐
Show daily high/low markers	☐

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
Font	Sets the font options

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 MIT (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 modificatio n 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
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.

11.20 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.

Playback is the ability to 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

11.20.1 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.

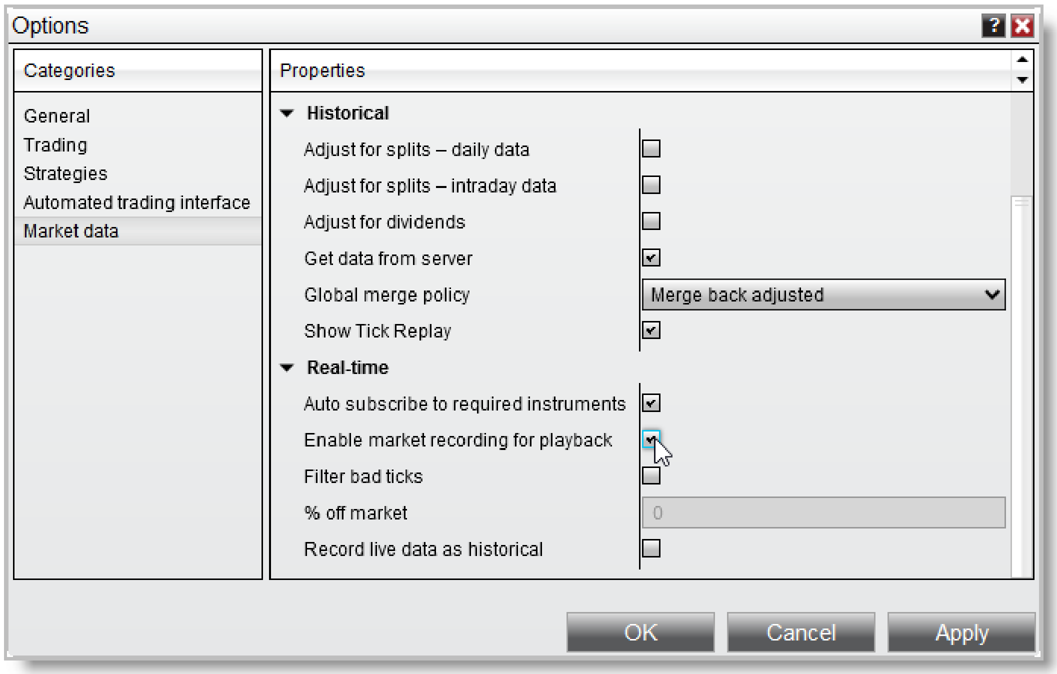
Play Video

▽ 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**".


Options

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

Properties

Historical

Adjust for splits – daily data	☐
Adjust for splits – intraday data	☐
Adjust for dividends	☐
Get data from server	✓
Global merge policy	Merge back adjusted
Show Tick Replay	✓

Real-time

Auto subscribe to required instruments	✓
Enable market recording for playback	✓
Filter bad ticks	☐
% off market	0
Record live data as historical	☐

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.

How to download playback data from the NinjaTrader server

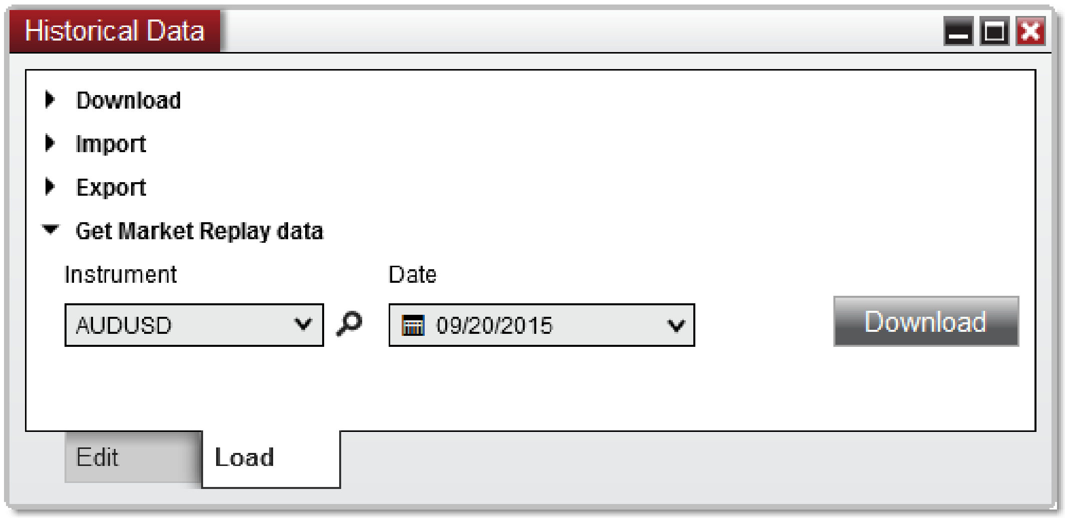
Downloading Market Replay data for the Playback connection

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. Select the **Tools** menu in the Control Center, select the menu item **Historical Data** and select "**Load**" tab. 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.

11.20.2 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.

▽ 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.

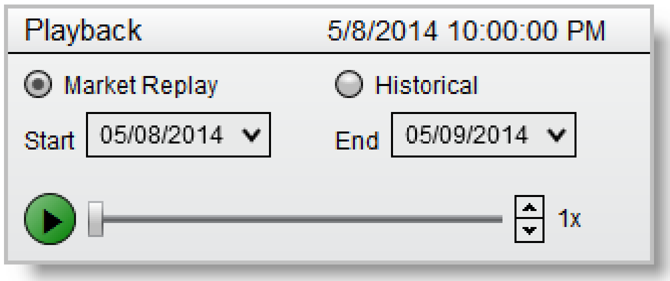
In the caption bar of the **Playback** control you will see the current date and time of where the play head is located.

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

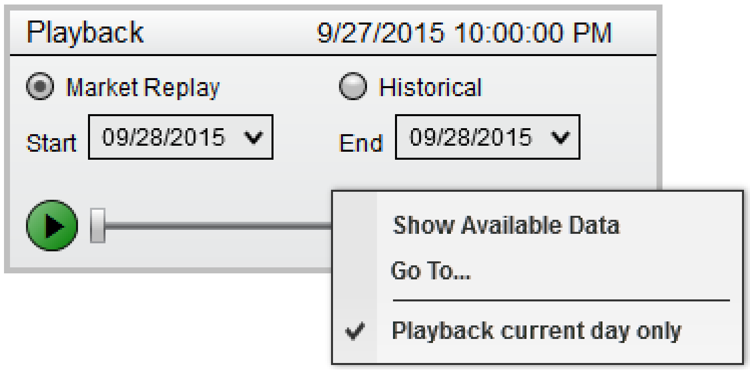
	slider
Play	Starts the market replay.
Slide control	Selects a point in time to start replay (sliding during playback will reset the Replay101 account trade history)
Speed control	Each successive click increases the speed of the playback. Playback of "**Max**" will process data at fastest possible speed.



Note: When changing dates in the Playback control window, open windows will not update to the new dates right away, but will wait until focus is taken away from the field being edited.

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	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. Uncheck this if you want to ensure 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 most process more data but is usefull 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 and indicators will calculate just as they were running in real-time.

To provide the highest possible playback performance the visual update interval is 1 second, please note though that order fills and indicator / strategy calculations are processed with the highest time stamp granularity possible still.

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.

11.20.3 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\data 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.

11.21 Risk

Risk Overview

The **Risk** window allows you to define margin control and limits to be used on simulation accounts.

› Using the Risk window

11.21.1 Using the Risk window

Within the **Risk** window, **Risk** Templates hold the risk definitions for simulation accounts. A risk definition holds the amount of margin needed per contract, it also limits the amount of contracts allowed to trade.

▿ 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.

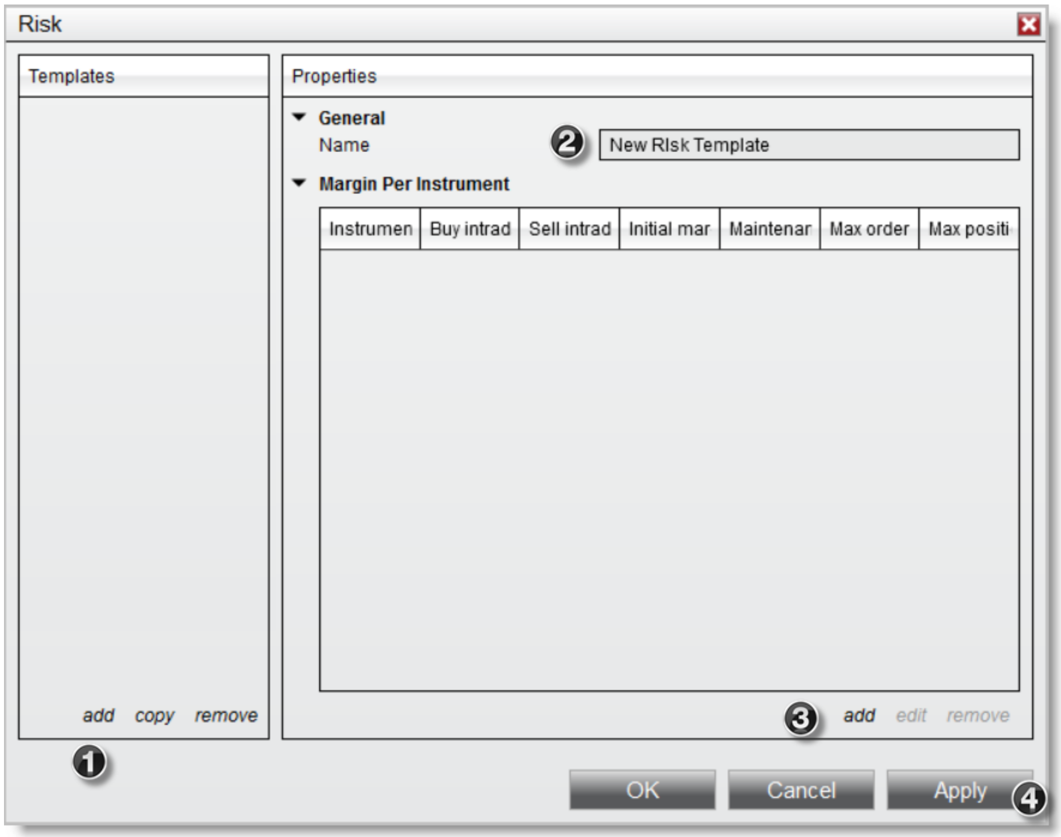
How to create and edit a Risk Template

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

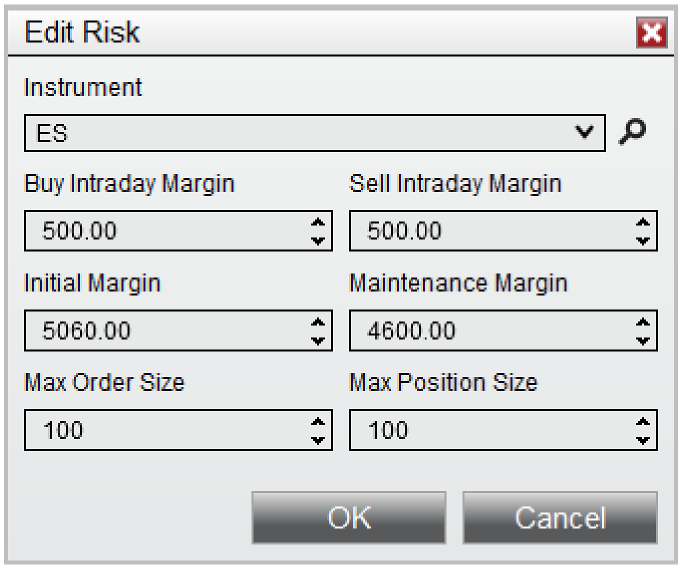
Risk Templates can be edited in the following ways:

- 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.


Edit Risk

Instrument: ES

Buy Intraday Margin: 500.00

Sell Intraday Margin: 500.00

Initial Margin: 5060.00

Maintenance Margin: 4600.00

Max Order Size: 100

Max Position Size: 100

Buy Intrada y Margin	Sets the intraday margin required for buy orders.
Sell Intrada	Sets the intraday margin required for sell orders.

y Margin	
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.

11.22 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

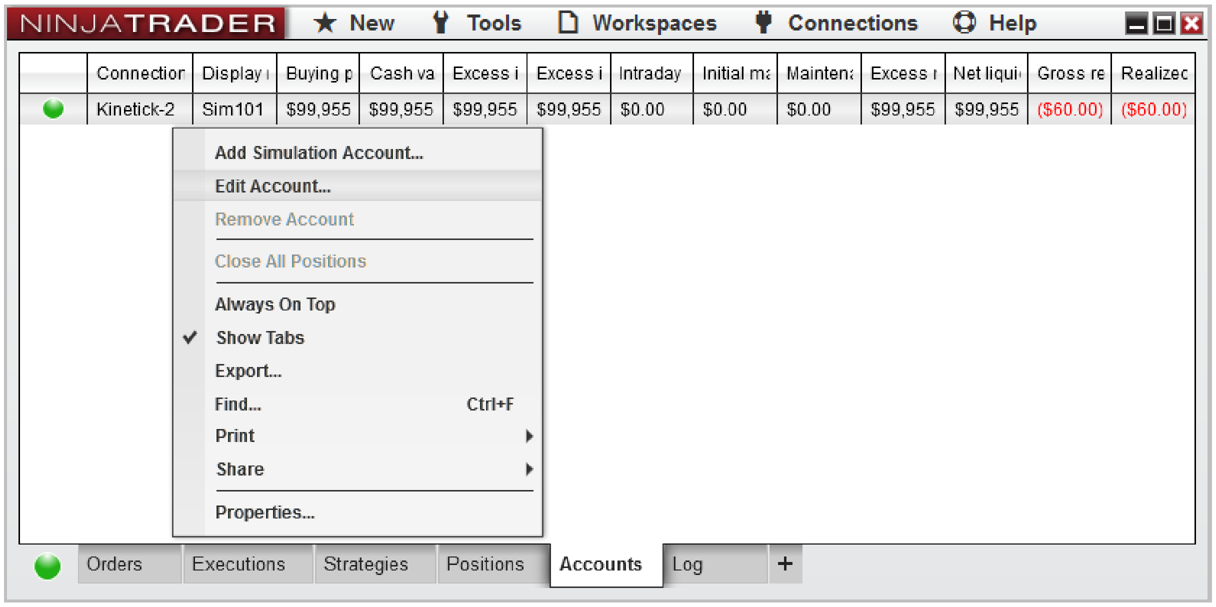
11.22.1 The Sim101 Account

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

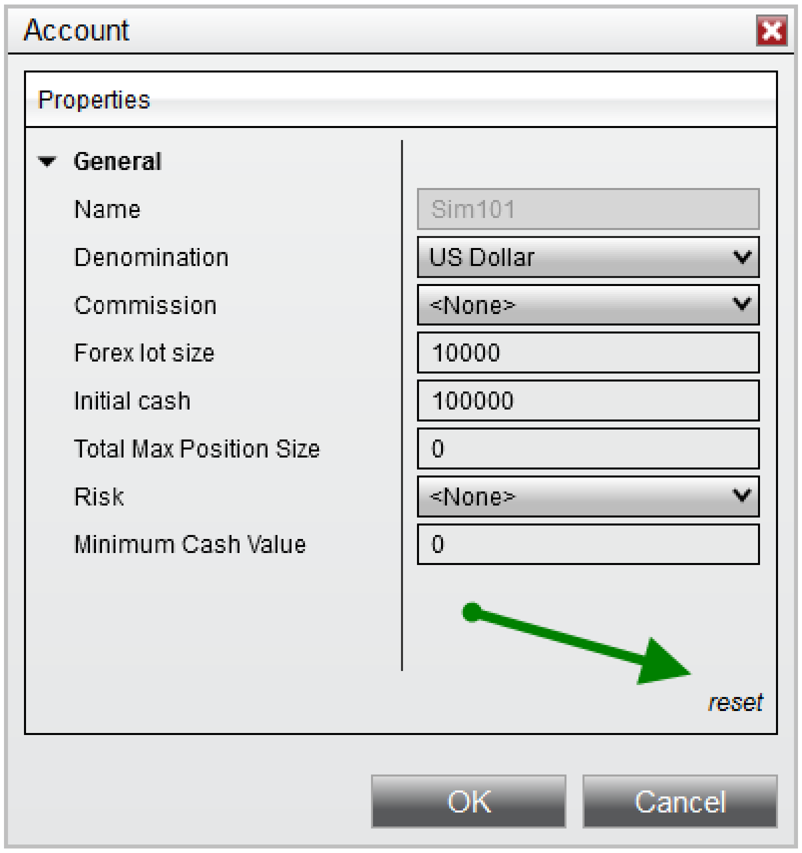
You can set initial Sim101 account values, reset simulator values, and clear order history. To access these settings open the Control Center window select the "**Accounts**" tab. If the account tab is not visible select the "+" tab button and select 'New accounts'



	Connectior	Display	Buying p	Cash va	Excess i	Excess i	Intraday	Initial ma	Maintena	Excess r	Net liqui	Gross re	Realizec
●	Kinetick-2	Sim101	$99,955	$99,955	$99,955	$99,955	$0.00	$0.00	$0.00	$99,955	$99,955	($60.00)	($60.00)

Resetting Initial Cash value on the Sim101 account

To reset the initial cash value on your account please edit the account as shown above.


Account

Properties

General

Name	Sim101
Denomination	US Dollar
Commission	<None>
Forex lot size	10000
Initial cash	100000
Total Max Position Size	0
Risk	<None>
Minimum Cash Value	0

In the Simulation Accounts window:

1. Set the "**Initial cash**" property to what you want to reset the account to.
2. Click "reset"

11.22.2 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.

11.22.3 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.

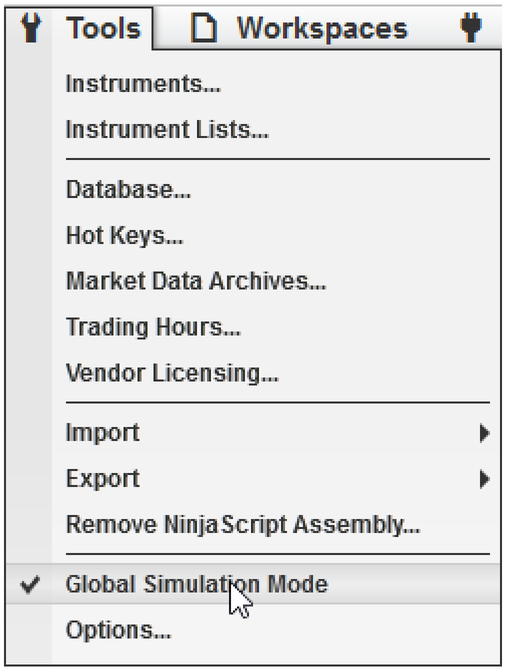
11.22.4 Global Simulation Mode

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.



11.22.5 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.

11.23 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

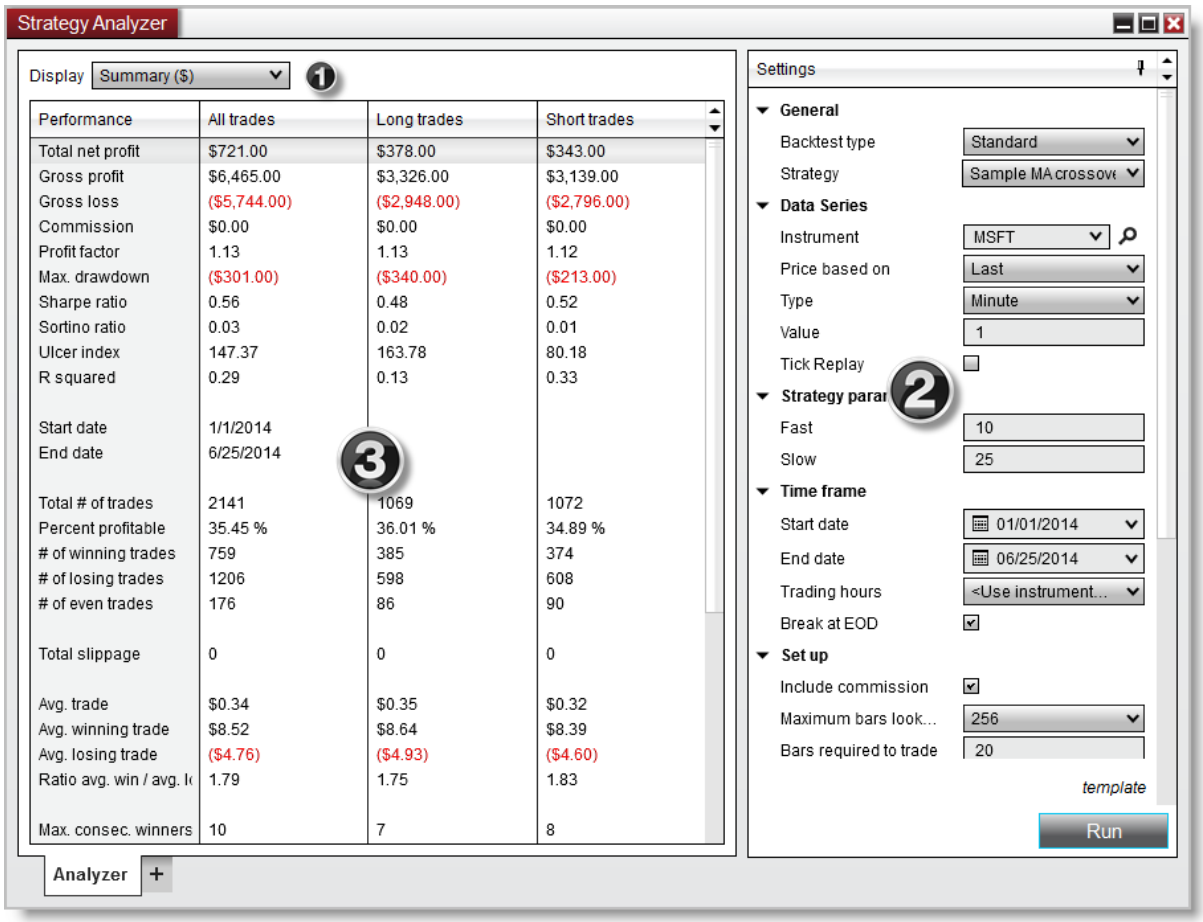
11.23.1 Understanding the Layout

Play Video

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.



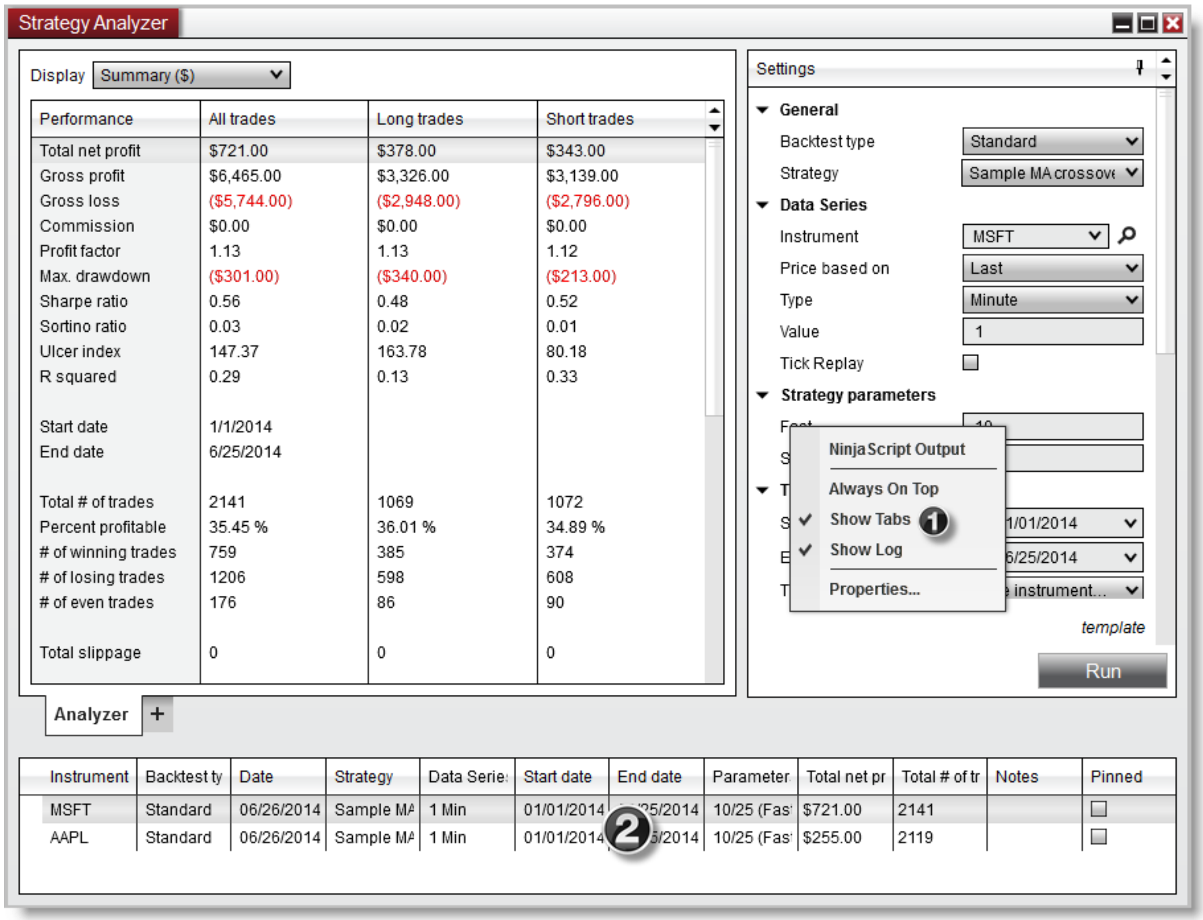
Performance	All trades	Long trades	Short trades
Total net profit	$721.00	$378.00	$343.00
Gross profit	$6,465.00	$3,326.00	$3,139.00
Gross loss	($5,744.00)	($2,948.00)	($2,796.00)
Commission	$0.00	$0.00	$0.00
Profit factor	1.13	1.13	1.12
Max. drawdown	($301.00)	($340.00)	($213.00)
Sharpe ratio	0.56	0.48	0.52
Sortino ratio	0.03	0.02	0.01
Ulcer index	147.37	163.78	80.18
R squared	0.29	0.13	0.33
Start date	1/1/2014		
End date	6/25/2014		
Total # of trades	2141	1069	1072
Percent profitable	35.45 %	36.01 %	34.89 %
# of winning trades	759	385	374
# of losing trades	1206	598	608
# of even trades	176	86	90
Total slippage	0	0	0
Avg. trade	$0.34	$0.35	$0.32
Avg. winning trade	$8.52	$8.64	$8.39
Avg. losing trade	($4.76)	($4.93)	($4.60)
Ratio avg. win / avg. lo	1.79	1.75	1.83
Max. consec. winners	10	7	8

General	
Backtest type	Standard
Strategy	Sample MA crossove
Data Series	
Instrument	MSFT
Price based on	Last
Type	Minute
Value	1
Tick Replay	☐
Strategy parar	
Fast	10
Slow	25
Time frame	
Start date	01/01/2014
End date	06/25/2014
Trading hours	<Use instrument...
Break at EOD	✓
Set up	
Include commission	✓
Maximum bars look...	256
Bars required to trade	20

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.


Strategy Analyzer

Display: Summary ($)

Performance	All trades	Long trades	Short trades
Total net profit	$721.00	$378.00	$343.00
Gross profit	$6,465.00	$3,326.00	$3,139.00
Gross loss	($5,744.00)	($2,948.00)	($2,796.00)
Commission	$0.00	$0.00	$0.00
Profit factor	1.13	1.13	1.12
Max. drawdown	($301.00)	($340.00)	($213.00)
Sharpe ratio	0.56	0.48	0.52
Sortino ratio	0.03	0.02	0.01
Ulcer index	147.37	163.78	80.18
R squared	0.29	0.13	0.33
Start date	1/1/2014		
End date	6/25/2014		
Total # of trades	2141	1069	1072
Percent profitable	35.45 %	36.01 %	34.89 %
# of winning trades	759	385	374
# of losing trades	1206	598	608
# of even trades	176	86	90
Total slippage	0	0	0

Settings
- General
  - Backtest type: Standard
  - Strategy: Sample MA crossove
- Data Series
  - Instrument: MSFT
  - Price based on: Last
  - Type: Minute
  - Value: 1
  - Tick Replay: ☐
- Strategy parameters
  - Fast: 10

Instrument	Backtest ty	Date	Strategy	Data Serie	Start date	End date	Parameter	Total net pr	Total # of tr	Notes	Pinned
MSFT	Standard	06/26/2014	Sample MA	1 Min	01/01/2014	06/25/2014	10/25 (Fas	$721.00	2141		☐
AAPL	Standard	06/26/2014	Sample MA	1 Min	01/01/2014	06/25/2014	10/25 (Fas	$255.00	2119		☐

11.23.2 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

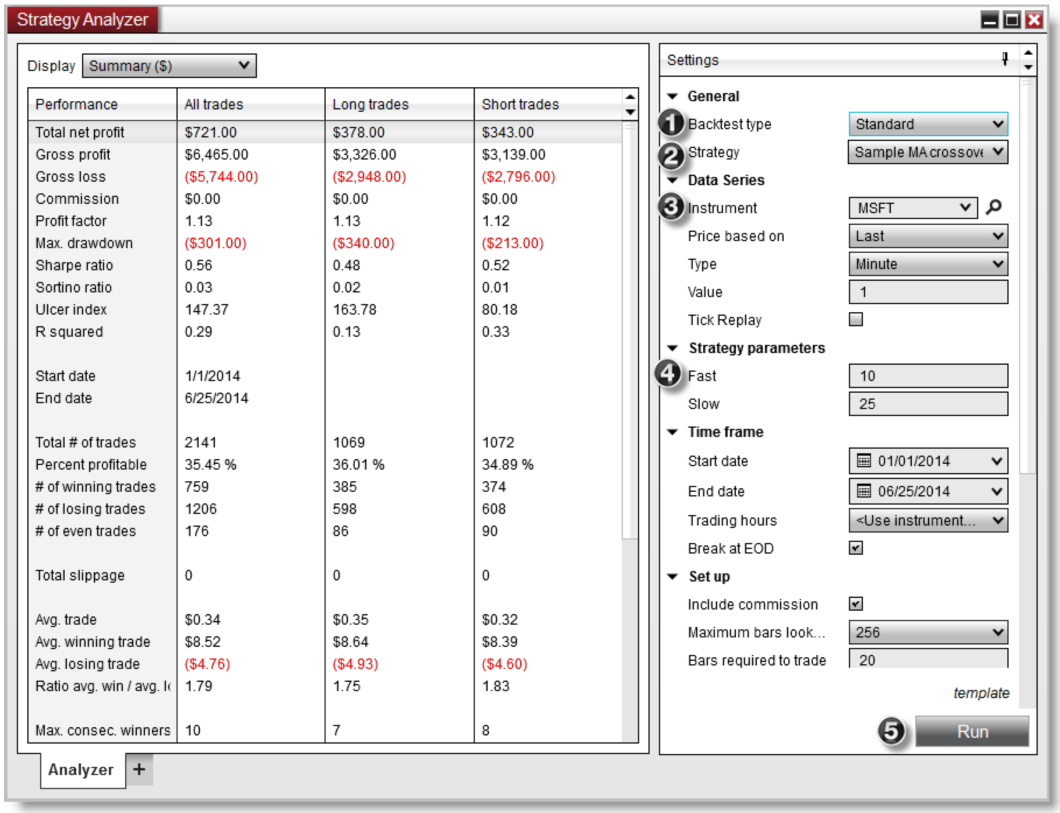
Notes:

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.

How to run a backtest

Start a Backtest

To run a **Backtest** of a strategy:



Display: Summary ($)

Performance	All trades	Long trades	Short trades
Total net profit	$721.00	$378.00	$343.00
Gross profit	$6,465.00	$3,326.00	$3,139.00
Gross loss	($5,744.00)	($2,948.00)	($2,796.00)
Commission	$0.00	$0.00	$0.00
Profit factor	1.13	1.13	1.12
Max. drawdown	($301.00)	($340.00)	($213.00)
Sharpe ratio	0.56	0.48	0.52
Sortino ratio	0.03	0.02	0.01
Ulcer index	147.37	163.78	80.18
R squared	0.29	0.13	0.33
Start date	1/1/2014		
End date	6/25/2014		
Total # of trades	2141	1069	1072
Percent profitable	35.45 %	36.01 %	34.89 %
# of winning trades	759	385	374
# of losing trades	1206	598	608
# of even trades	176	86	90
Total slippage	0	0	0
Avg. trade	$0.34	$0.35	$0.32
Avg. winning trade	$8.52	$8.64	$8.39
Avg. losing trade	($4.76)	($4.93)	($4.60)
Ratio avg. win / avg. l	1.79	1.75	1.83
Max. consec. winners	10	7	8

Settings

General	
Backtest type	Standard
Strategy	Sample MA crossove
Data Series	
Instrument	MSFT
Price based on	Last
Type	Minute
Value	1
Tick Replay	☐
Strategy parameters	
Fast	10
Slow	25
Time frame	
Start date	01/01/2014
End date	06/25/2014
Trading hours	<Use instrument...
Break at EOD	☑
Set up	
Include commission	☑
Maximum bars look...	256
Bars required to trade	20

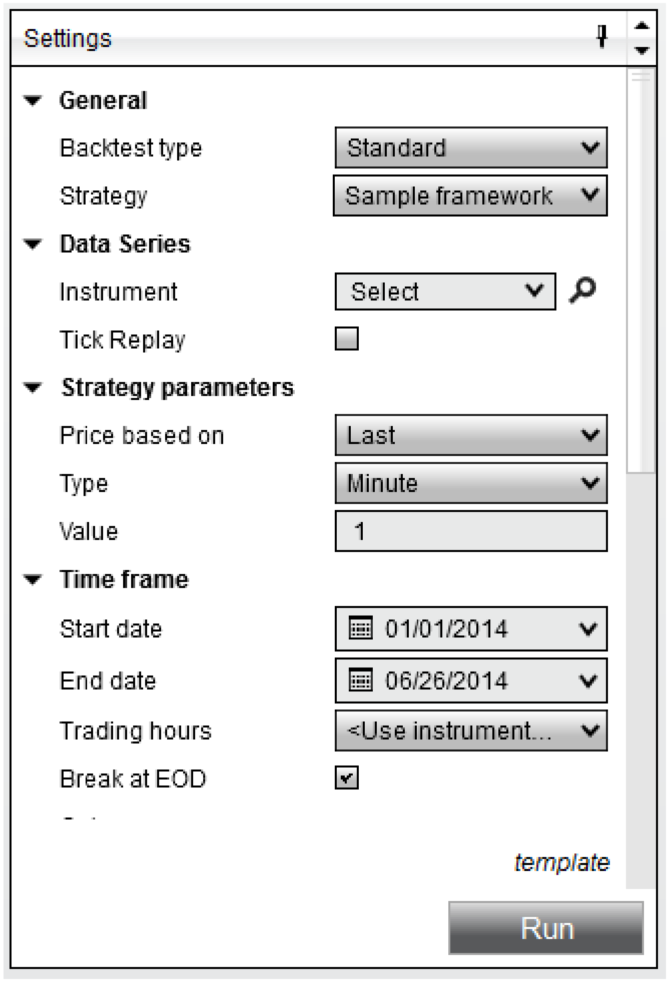
1. Select the Backtest type of "**Standard**"
2. Select the strategy you would like to backtest
3. Select the instrument and Data Series you would like to backtest
4. Set the strategy and backtest parameters (See the "*Understanding backtest properties*" section below for property definitions)
5. Select 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:


Settings

General

Backtest type: Standard

Strategy: Sample framework

Data Series

Instrument: Select

Tick Replay: ☐

Strategy parameters

Price based on: Last

Type: Minute

Value: 1

Time frame

Start date: 01/01/2014

End date: 06/26/2014

Trading hours: <Use instrument...

Break at EOD: ✓

General	
Backtest type	Sets the backtest type. 1. Standard 2. Optimization 3. Walk Forward Optimization 4. Multi-Objective Optimization
Strategy	Sets the strategy you would like to test.
Data Series	

Instrume nt	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.
Strateg y paramet ers	
Paramet ers (...)	Each strategy parameter is listed dynamically depending on the strategy selection
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 commis	Enables or disables commissions in the backtest performance results (See the "Commission Tab"

sion	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 to apply to market / stop market order executions (default is 0)
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

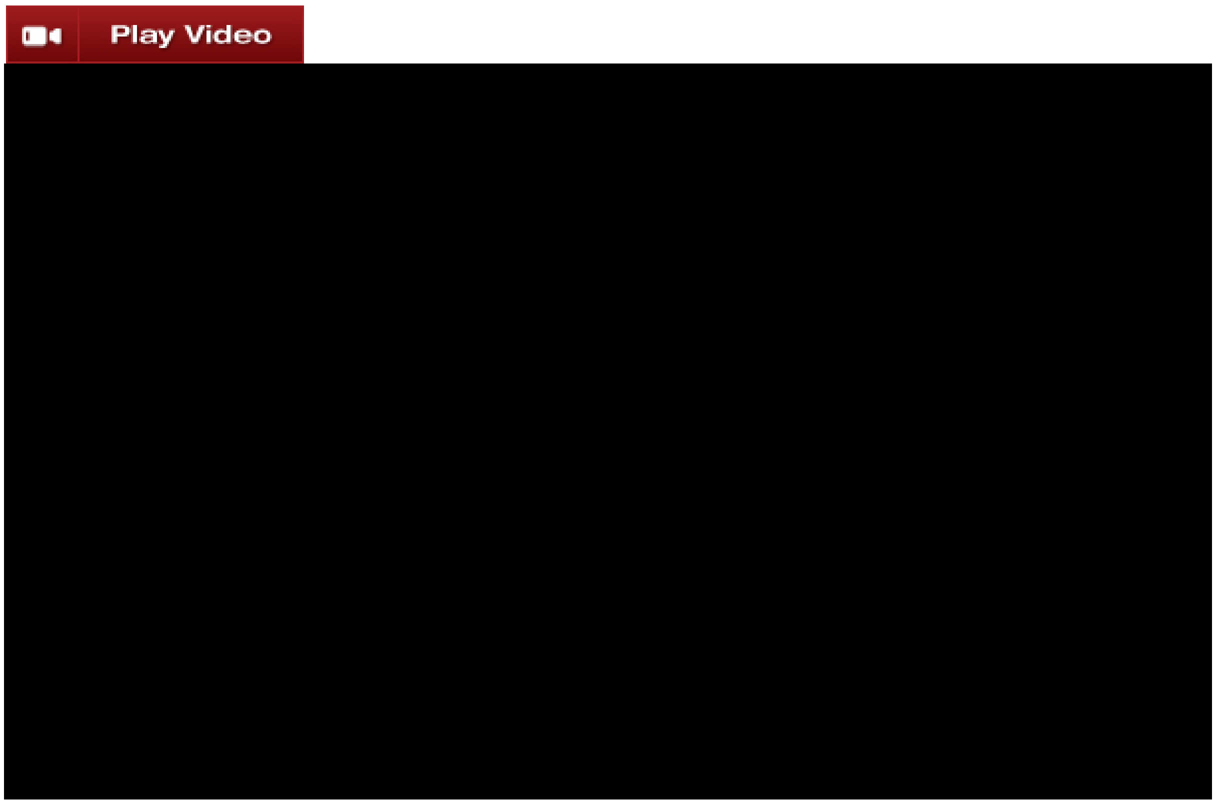
	been reached while in an open position. If set to "UniqueEntries", strategy will process entry orders until the maximum allowable entries set by the "Entries per direction" property per each uniquely named entry.
Exit on session close	When enabled, open positions are closed on the last bar of a session
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 "by account" - Allows you to set a virtual account value that is used to determine maximum order size based on margin settings per instrument set in the Instruments window
Time in force	Sets the order's time in force

11.23.3 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

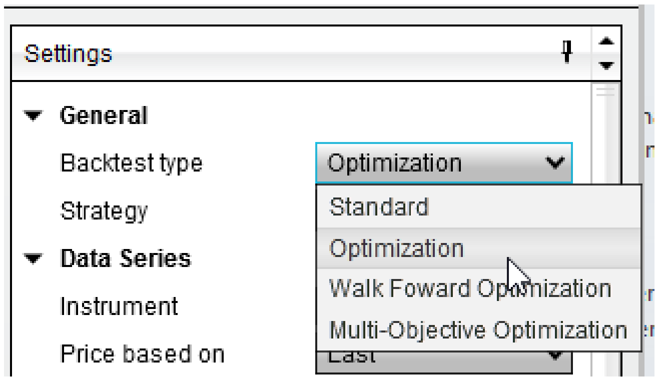
Tip: There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore.



▽ 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**.


Settings

General

Backtest type: Optimization

Strategy

Data Series

Instrument

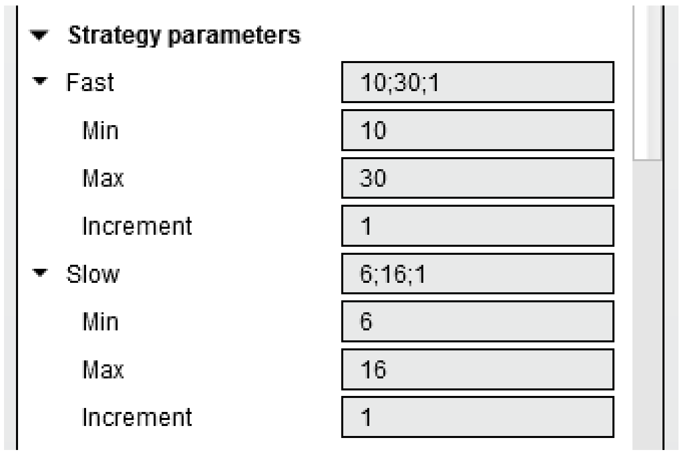
Price based on: Last

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**".



Strategy parameters	
Fast	10;30;1
Min	10
Max	30
Increment	1
Slow	6;16;1
Min	6
Max	16
Increment	1

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.

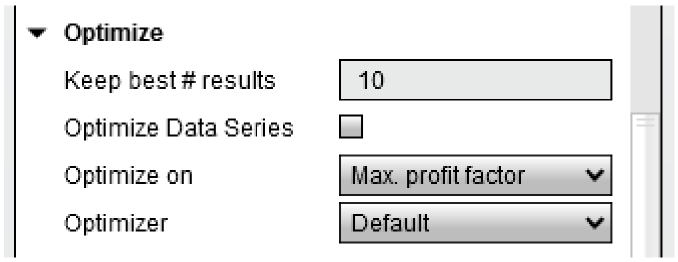
Please see the "*Understanding Optimization properties*" section below for more information.

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:



Keep best # results	10
Optimize Data Series	☐
Optimize on	Max. profit factor
Optimizer	Default

Tip: You can optionally "**Optimize on**" multiple objectives by using a Multi-Objective optimization

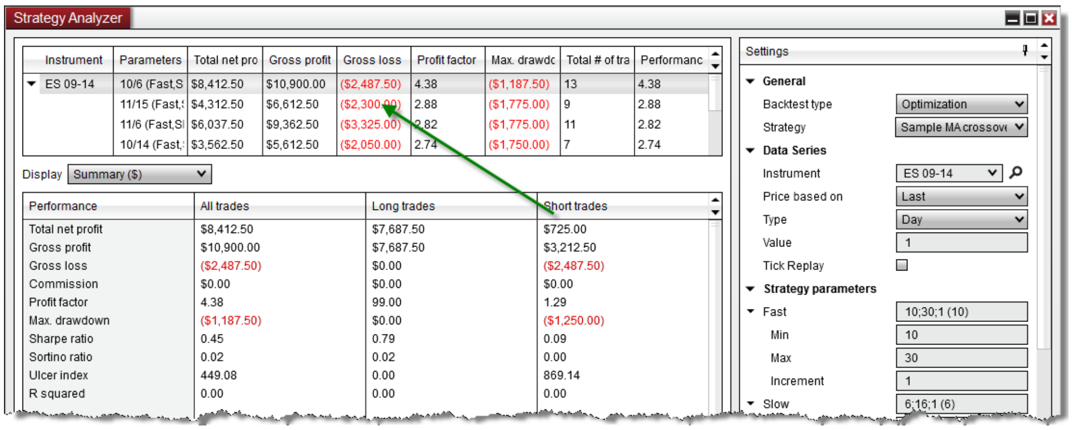
Keep best # results	Sets the number of best results to display
Optimize data series	If set to true, the Data Series Value property will be available for optimization (Not supported for Kagi, PointAndFigure, and Line Break period Types)
Optimize on...	Sets the optimization fitness to base the optimization 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.

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.


Strategy Analyzer

Instrument	Parameters	Total net pro	Gross profit	Gross loss	Profit factor	Max. drawdc	Total # of tra	Performanc
ES 09-14	10/6 (Fast,S	$8,412.50	$10,900.00	($2,487.50)	4.38	($1,187.50)	13	4.38
	11/15 (Fast,:	$4,312.50	$6,612.50	($2,300.00)	2.88	($1,775.00)	9	2.88
	11/6 (Fast,Sl	$6,037.50	$9,362.50	($3,325.00)	2.82	($1,775.00)	11	2.82
	10/14 (Fast,:	$3,562.50	$5,612.50	($2,050.00)	2.74	($1,750.00)	7	2.74

Display Summary ($)

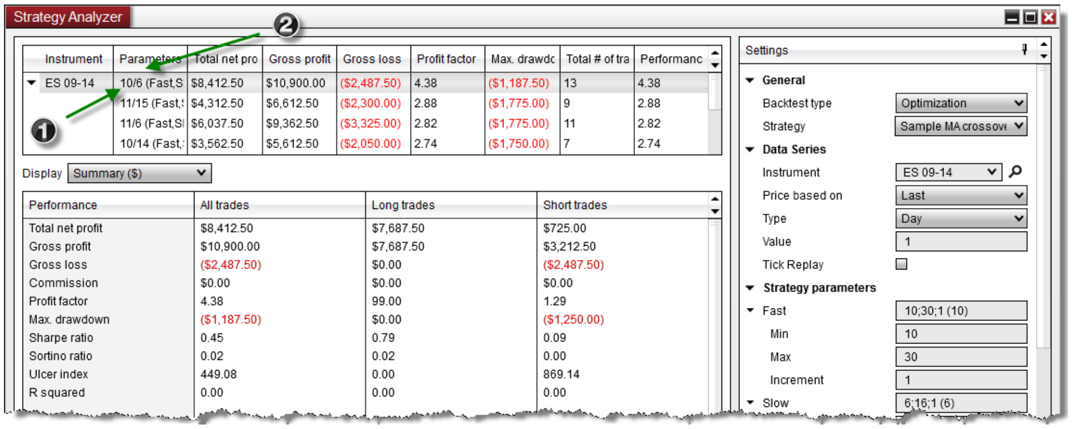
Performance	All trades	Long trades	Short trades
Total net profit	$8,412.50	$7,687.50	$725.00
Gross profit	$10,900.00	$7,687.50	$3,212.50
Gross loss	($2,487.50)	$0.00	($2,487.50)
Commission	$0.00	$0.00	$0.00
Profit factor	4.38	99.00	1.29
Max. drawdown	($1,187.50)	$0.00	($1,250.00)
Sharpe ratio	0.45	0.79	0.09
Sortino ratio	0.02	0.02	0.00
Ulcer index	449.08	0.00	869.14
R squared	0.00	0.00	0.00

Settings

General	
Backtest type	Optimization
Strategy	Sample MA crossove
Data Series	
Instrument	ES 09-14
Price based on	Last
Type	Day
Value	1
Tick Replay	☐
Strategy parameters	
Fast	10;30;1 (10)
Min	10
Max	30
Increment	1
Slow	6;16;1 (6)

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.



Instrument	Parameters	Total net pro	Gross profit	Gross loss	Profit factor	Max. drawdc	Total # of tra	Performanc
ES 09-14	10/6 (Fast,S	$8,412.50	$10,900.00	($2,487.50)	4.38	($1,187.50)	13	4.38
	11/15 (Fast,:	$4,312.50	$6,612.50	($2,300.00)	2.88	($1,775.00)	9	2.88
	11/6 (Fast,Sl	$6,037.50	$9,362.50	($3,325.00)	2.82	($1,775.00)	11	2.82
	10/14 (Fast,:	$3,562.50	$5,612.50	($2,050.00)	2.74	($1,750.00)	7	2.74

Performance	All trades	Long trades	Short trades
Total net profit	$8,412.50	$7,687.50	$725.00
Gross profit	$10,900.00	$7,687.50	$3,212.50
Gross loss	($2,487.50)	$0.00	($2,487.50)
Commission	$0.00	$0.00	$0.00
Profit factor	4.38	99.00	1.29
Max. drawdown	($1,187.50)	$0.00	($1,250.00)
Sharpe ratio	0.45	0.79	0.09
Sortino ratio	0.02	0.02	0.00
Ulcer index	449.08	0.00	869.14
R squared	0.00	0.00	0.00

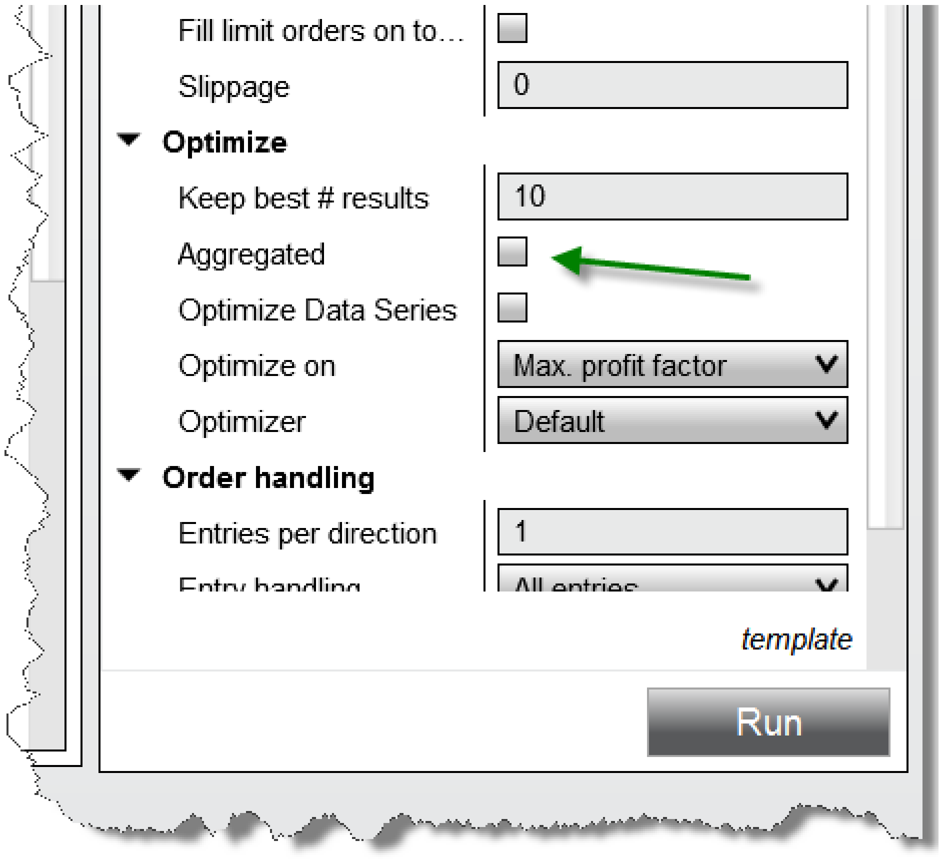
General	
Backtest type	Optimization
Strategy	Sample MA crossove
Data Series	
Instrument	ES 09-14
Price based on	Last
Type	Day
Value	1
Tick Replay	☐
Strategy parameters	
Fast	10;30;1 (10)
Min	10
Max	30
Increment	1
Slow	6;16;1 (6)

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.


Fill limit orders on to...
Slippage: 0
Optimize
Keep best # results: 10
Aggregated
Optimize Data Series
Optimize on: Max. profit factor
Optimizer: Default
Order handling
Entries per direction: 1

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.

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.

11.23.3.1 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.

Play Video

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

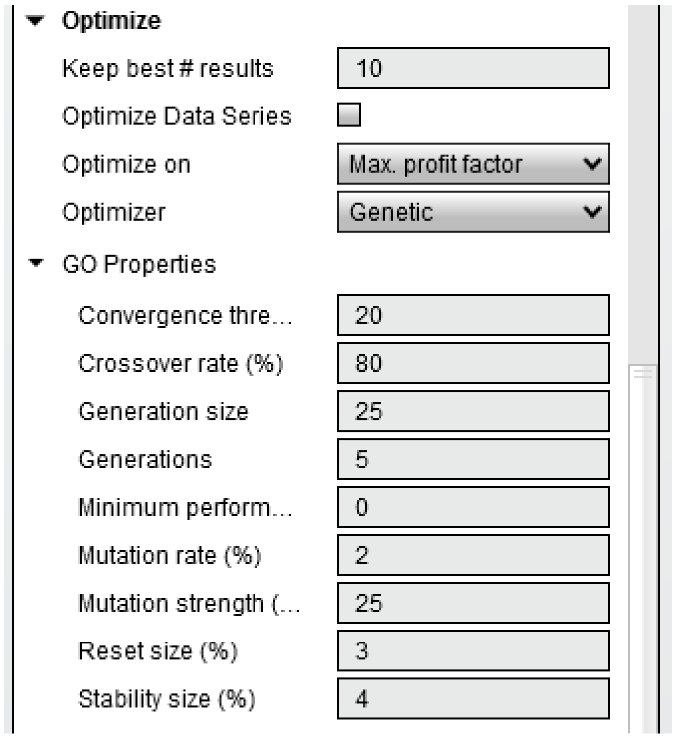
Crossover and Mutation

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.



Optimize	
Keep best # results	10
Optimize Data Series	☐
Optimize on	Max. profit factor
Optimizer	Genetic
GO Properties	
Convergence thre...	20
Crossover rate (%)	80
Generation size	25
Generations	5
Minimum perform...	0
Mutation rate (%)	2
Mutation strength (...	25
Reset size (%)	3
Stability size (%)	4

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.
Generatio	Sets the number of combinations to test in each

n size	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.
Generatio ns	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 performa nce	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
Mutation strength (%)	Sets the maximum offset from crossover values that an offspring marked for mutation can have its parameters changed
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 %"

11.23.3.2 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 (drawdown) in account size experienced from the highest high seen in each trade, and is used to find the iteration with the lowest drawdown.

```
See the Maximum Drawdown formula on the Statistics Definitions page
```

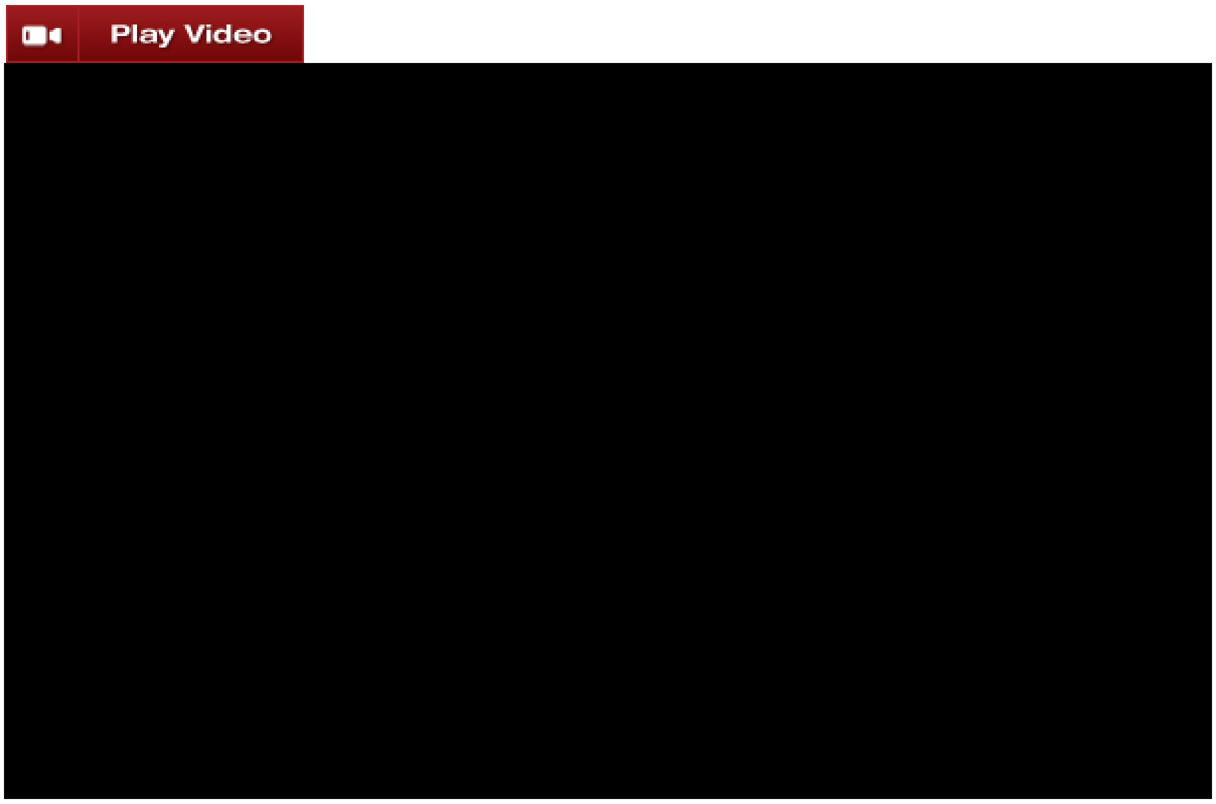
```
Min Drawdown = the smallest single drawdown
```

11.23.4 Walk Forward Optimizion

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.

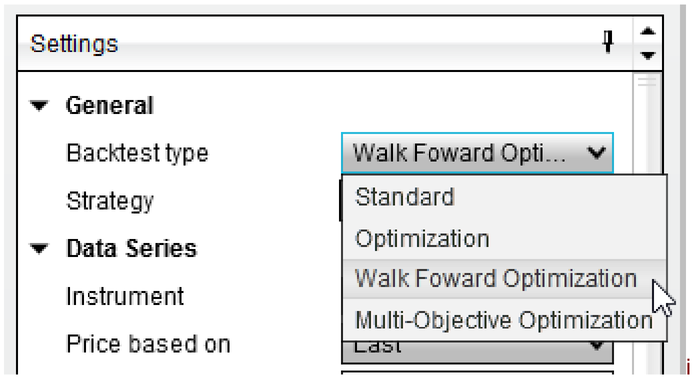


Note: The IncludeTradeHistoryInBacktest property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for optimization. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as 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**.

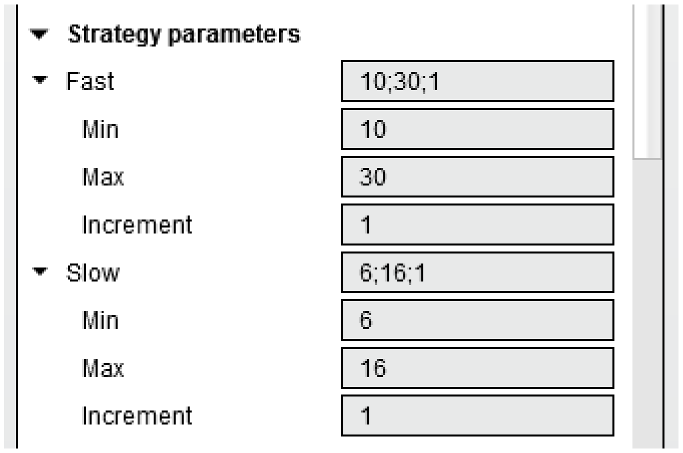


Note: When making the selection additional parameters to configure your optimization will be made visible.

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**".



Strategy parameters	
Fast	10;30;1
Min	10
Max	30
Increment	1
Slow	6;16;1
Min	6
Max	16
Increment	1

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.

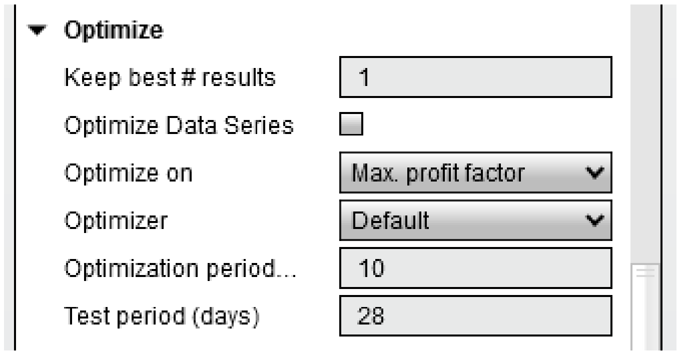
Setting the Optimization Fitness

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.


Optimize

Keep best # results	1
Optimize Data Series	☐
Optimize on	Max. profit factor
Optimizer	Default
Optimization period...	10
Test period (days)	28

Tip: You can optionally "**Optimize on**" multiple objectives by using a Multi-Objective optimization

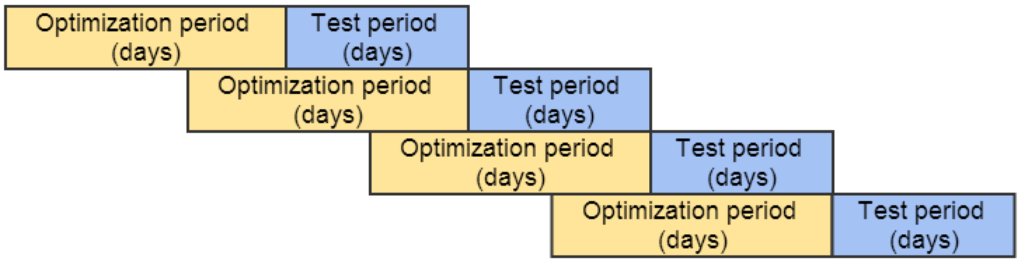
Keep best # results	Sets the number of best results to display
Optimize data series	If set to true, the Data Series Value property will be available for optimization (Not supported for Kagi, PointAndFigure, and Line Break period Types)
Optimize on...	Sets the optimization fitness to base the optimization 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

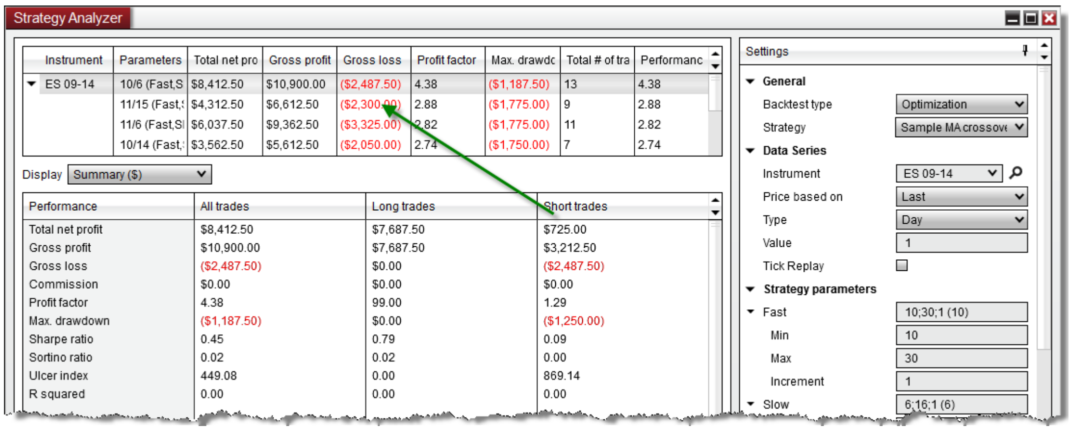
Understanding Walk Forward results

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.


Optimization period (days)
Test period (days)
Optimization period (days)
Test period (days)
Optimization period (days)
Test period (days)
Optimization period (days)
Test period (days)

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.


Strategy Analyzer

Instrument	Parameters	Total net pro	Gross profit	Gross loss	Profit factor	Max. drawdc	Total # of tra	Performanc
ES 09-14	10/6 (Fast,S	$8,412.50	$10,900.00	($2,487.50)	4.38	($1,187.50)	13	4.38
	11/15 (Fast,:	$4,312.50	$6,612.50	($2,300.00)	2.88	($1,775.00)	9	2.88
	11/6 (Fast,Sl	$6,037.50	$9,362.50	($3,325.00)	2.82	($1,775.00)	11	2.82
	10/14 (Fast,:	$3,562.50	$5,612.50	($2,050.00)	2.74	($1,750.00)	7	2.74

Display Summary ($)

Performance	All trades	Long trades	Short trades
Total net profit	$8,412.50	$7,687.50	$725.00
Gross profit	$10,900.00	$7,687.50	$3,212.50
Gross loss	($2,487.50)	$0.00	($2,487.50)
Commission	$0.00	$0.00	$0.00
Profit factor	4.38	99.00	1.29
Max. drawdown	($1,187.50)	$0.00	($1,250.00)
Sharpe ratio	0.45	0.79	0.09
Sortino ratio	0.02	0.02	0.00
Ulcer index	449.08	0.00	869.14
R squared	0.00	0.00	0.00

Settings

General	
Backtest type	Optimization
Strategy	Sample MA crossove
Data Series	
Instrument	ES 09-14
Price based on	Last
Type	Day
Value	1
Tick Replay	☐
Strategy parameters	
Fast	10;30;1 (10)
Min	10
Max	30
Increment	1
Slow	6;16;1 (6)

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".

11.23.5 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.

- 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.

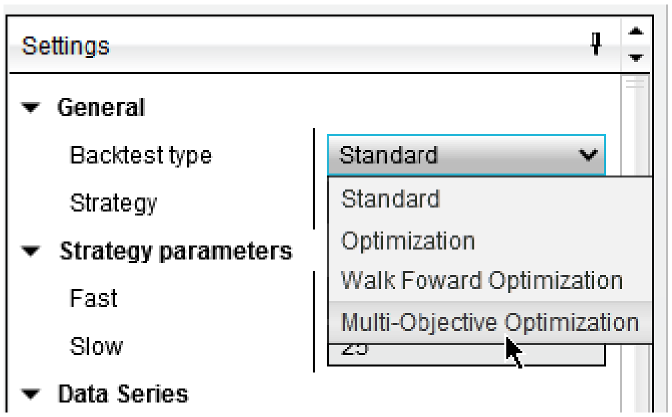
Play Video

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**.

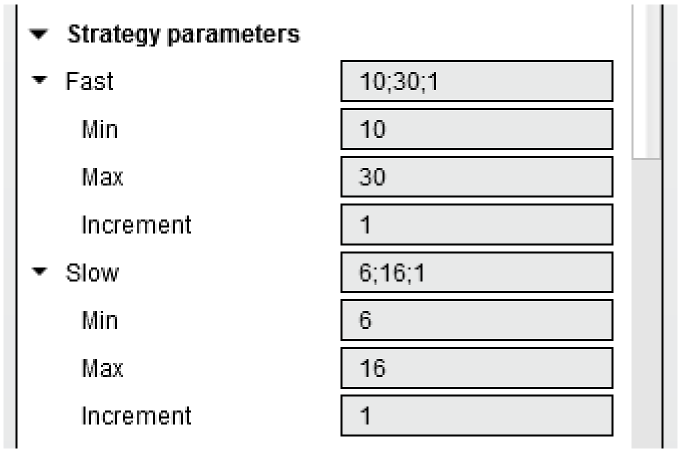


Note: When making the selection additional parameters to configure your optimization will be made visible.

Setting the Test Range

You can 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**".



Strategy parameters	
Fast	10;30;1
Min	10
Max	30
Increment	1
Slow	6;16;1
Min	6
Max	16
Increment	1

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

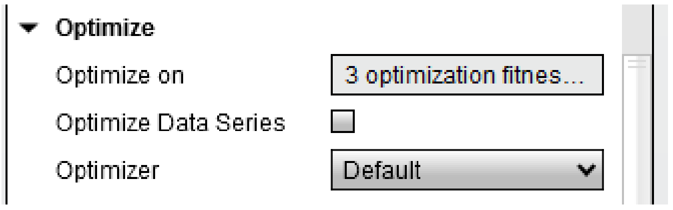
In the image above, the input "Fast" has a starting (initial) value of 10 and an ending value of 30 with an increment of 1. This means that the first value tested will be 10, then 11, then 12 all the way through 30. The input "Slow" has a starting value of 6, ending value of 16 with an increment of 1. 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

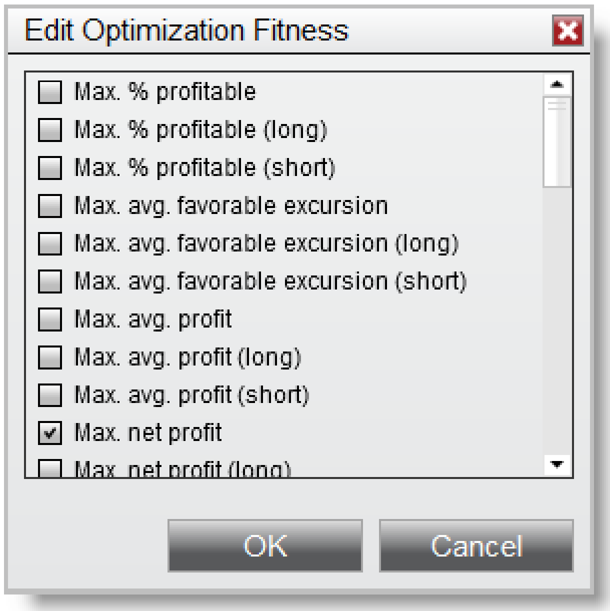
Setting Multiple Optimization Fitness

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
Optimize on 3 optimization fitnes...
Optimize Data Series ☐
Optimizer Default

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.

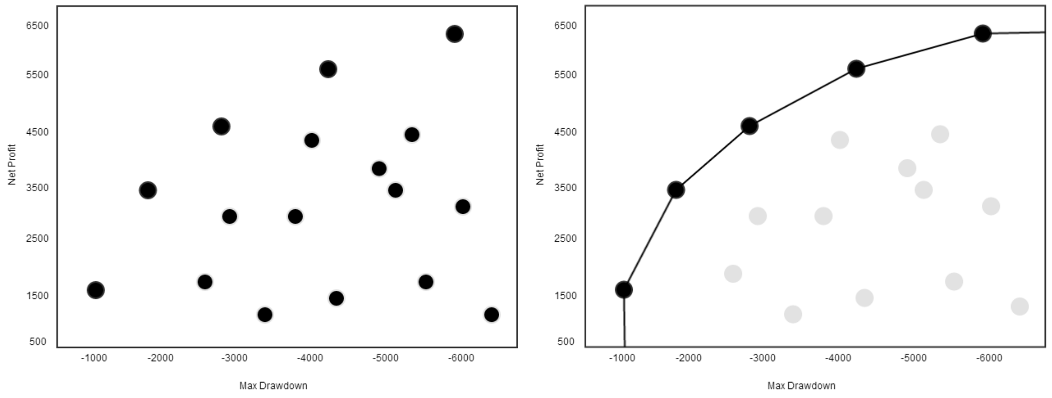

Edit Optimization Fitness

- [ ] Max. % profitable
- [ ] Max. % profitable (long)
- [ ] Max. % profitable (short)
- [ ] Max. avg. favorable excursion
- [ ] Max. avg. favorable excursion (long)
- [ ] Max. avg. favorable excursion (short)
- [ ] Max. avg. profit
- [ ] Max. avg. profit (long)
- [ ] Max. avg. profit (short)
- [✓] Max. net profit
- [ ] Max. net profit (long)

Understanding Multi-Objective results

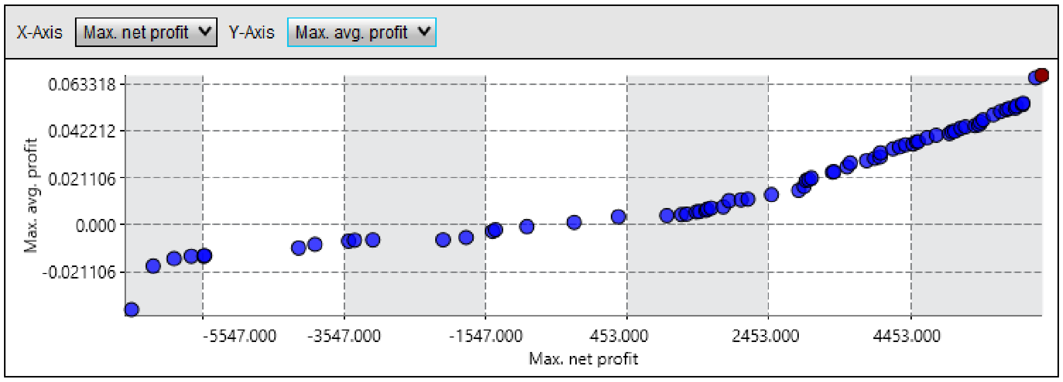
Understanding Multi-Objective Results

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.

11.23.6 Understanding Historical Fill Processing

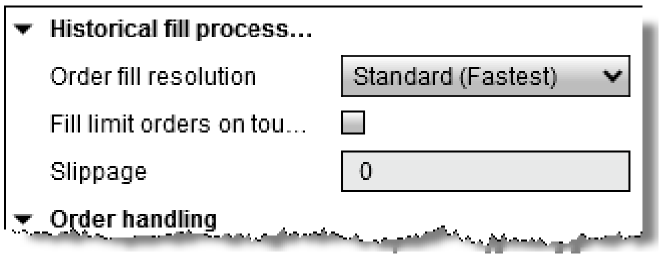
NinjaTrader uses advanced historical fill processing methods and techniques to get the most realistic results possible on historical backtests.

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.



Historical fill process...

Order fill resolution: Standard (Fastest)

Fill limit orders on tou...: ☐

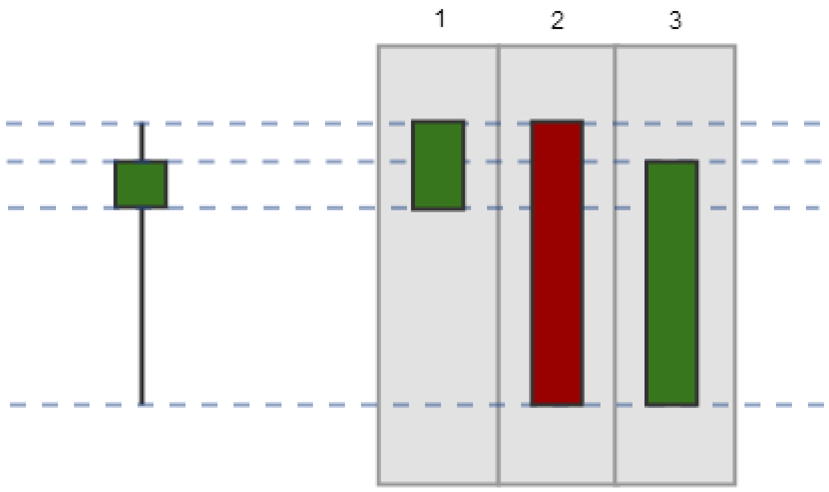
Slippage: 0

Order handling

The Standard setting creates virtual bars according to the following logic:

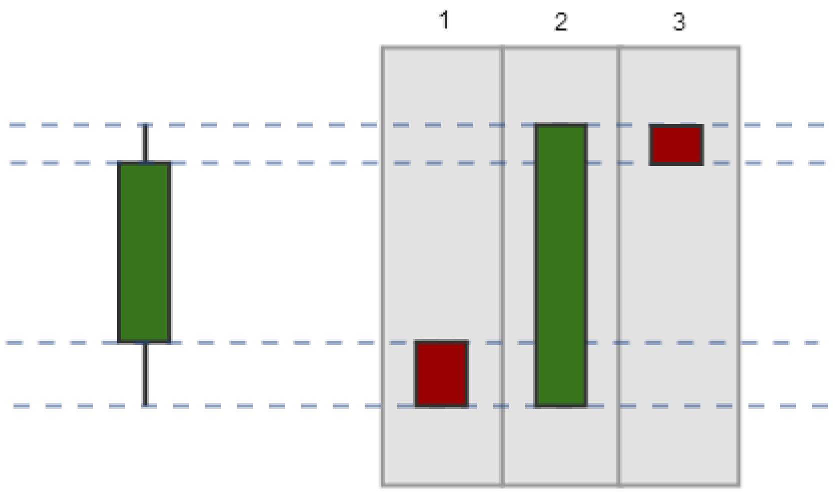
When the **Open** price of the bar is closer to the **High** price then the **Low** price:

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 then the **High** price:

1. **Open** price to the **Low** price
2. **Low** price to the **High** price
3. **High** price to the **Close** price



How the Fill Algorithm Works

During a backtest order quantity is an absolute value, which is in most cases different than in a real-time brokerage account. As an example, 1 traded FX lot at a live brokerage account might be the equivalent to 100,000 of notional value (check with your broker) however, in backtest a value of 1 is a literal value of 1 and not 100,000. Thus if you want to trade 100,000 in a backtest, you need to put in a value of 100,000. Just remember that if you convert your strategy from backtest to live you will need to amend the order quantities appropriately. (Please see the Running FX Strategies section for more information).

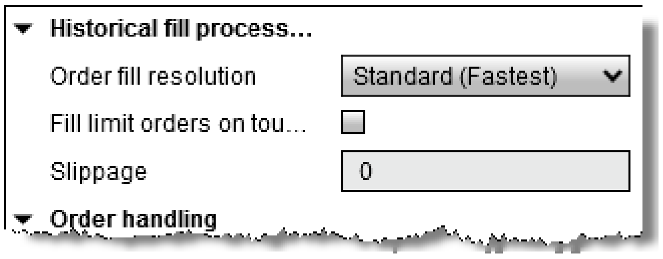
Slippage

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 and stop-market 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 then 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: Standard (Fastest)
Fill limit orders on tou...
Slippage: 0

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.

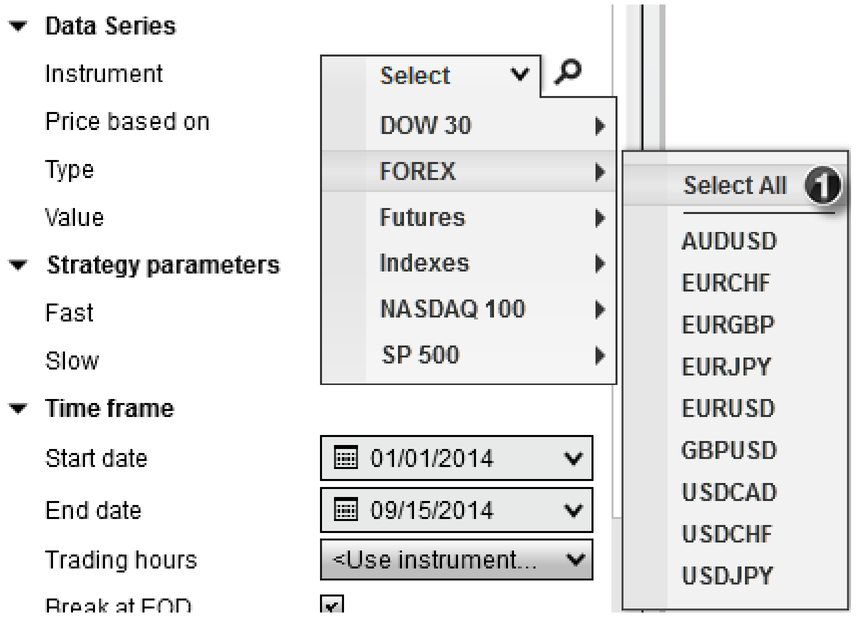
Selecting **order fill resolution** of "**High**" will allow you to set a secondary bar series to be used as the price data to fill your orders, this allows you to bring in more granular data then you are currently running the strategy on. For example you may have a strategy that you run on "**Daily**" 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.

Note: 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.

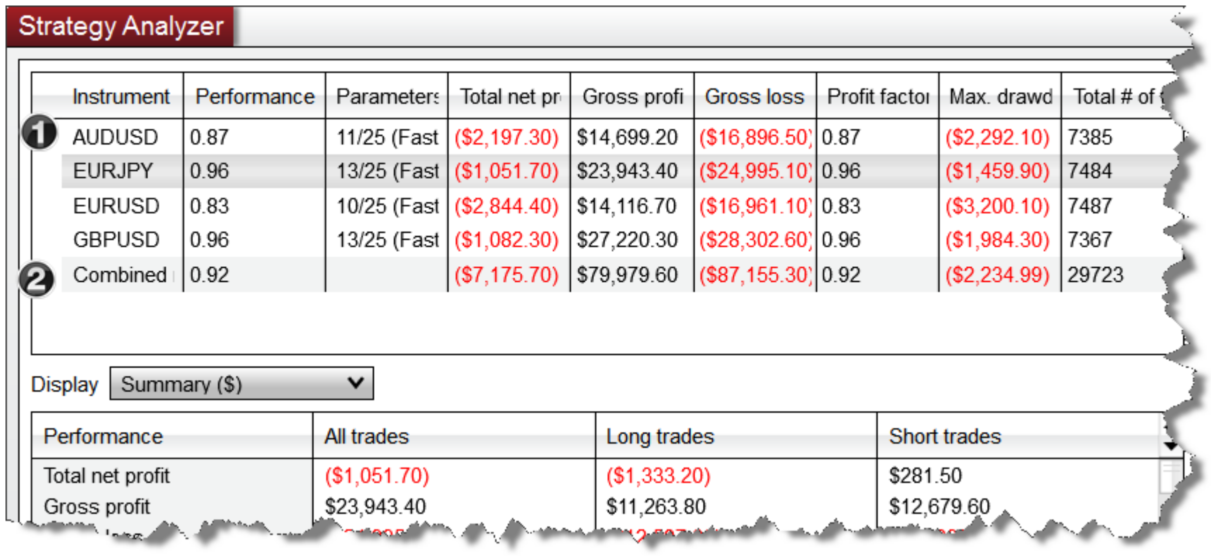
11.23.7 Basket testing multiple instruments

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.



Start date 01/01/2014
End date 09/15/2014
Trading hours <Use instrument...
Break at EOD ✓

Once the test is complete, a listing of all the results will be displayed.



Instrument	Performance	Parameter	Total net pr	Gross profi	Gross loss	Profit facto	Max. drawd	Total # of
AUDUSD	0.87	11/25 (Fast	($2,197.30)	$14,699.20	($16,896.50	0.87	($2,292.10)	7385
EURJPY	0.96	13/25 (Fast	($1,051.70)	$23,943.40	($24,995.10	0.96	($1,459.90)	7484
EURUSD	0.83	10/25 (Fast	($2,844.40)	$14,116.70	($16,961.10	0.83	($3,200.10)	7487
GBPUSD	0.96	13/25 (Fast	($1,082.30)	$27,220.30	($28,302.60	0.96	($1,984.30)	7367
Combined	0.92		($7,175.70)	$79,979.60	($87,155.30	0.92	($2,234.99)	29723

Display Summary ($)

Performance	All trades	Long trades	Short trades
Total net profit	($1,051.70)	($1,333.20)	$281.50
Gross profit	$23,943.40	$11,263.80	$12,679.60

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:

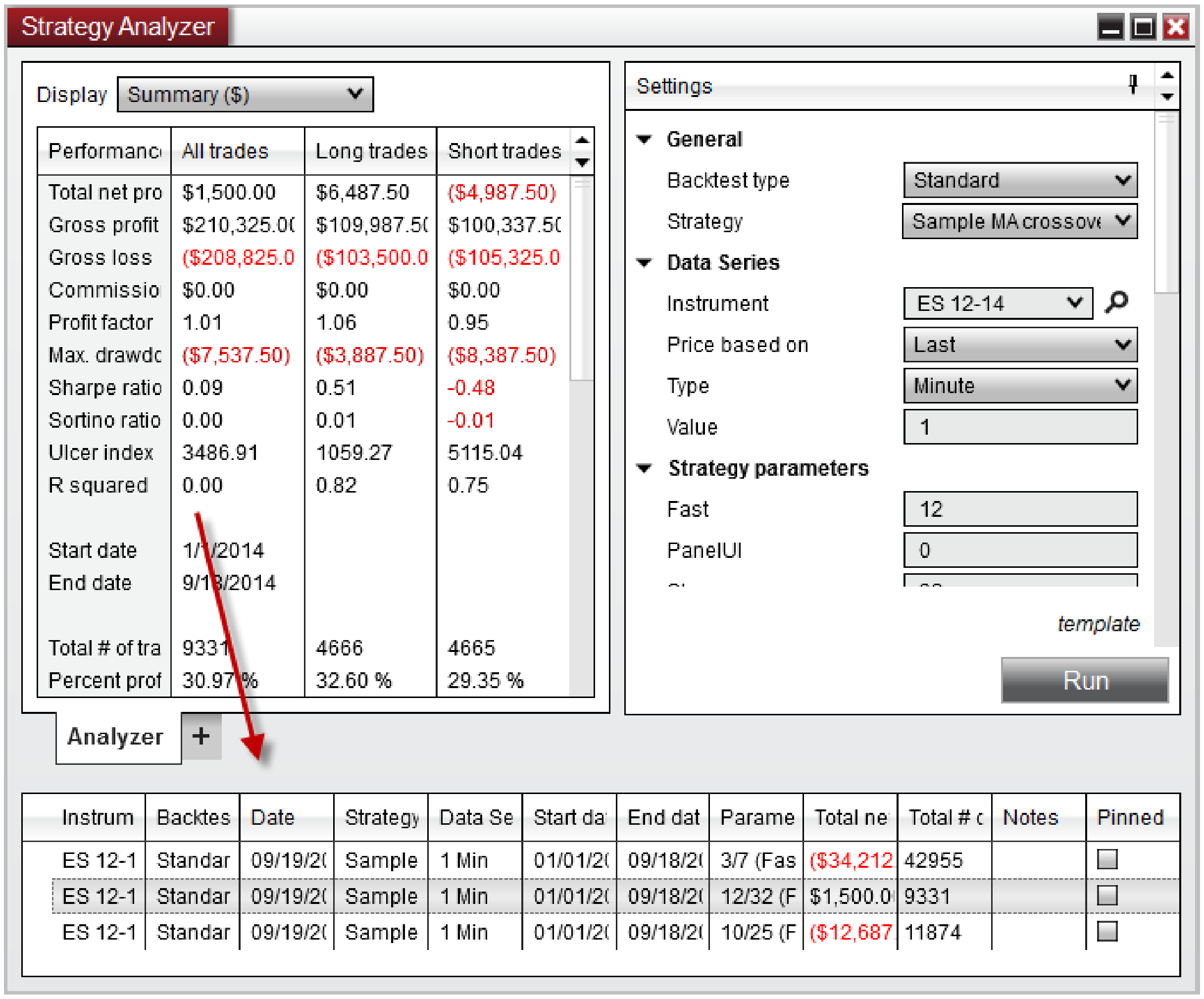
```
Combined Max.Drawdown = SUM((AUDUSD Drawdown * AUDUSD Trade Count) + (EURJPY Drawdown
* EURJPY Trade Count) + (EURUSD Drawdown * EURUSD Trade Count) + (GBPUSD Drawdown *
GBPUSD Trade Count)) / Total Trade Count of All Instruments
```

11.23.8 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.



Display: Summary ($)

Performanc	All trades	Long trades	Short trades
Total net pro	$1,500.00	$6,487.50	($4,987.50)
Gross profit	$210,325.0(	$109,987.5(	$100,337.5(
Gross loss	($208,825.0	($103,500.0	($105,325.0
Commissio	$0.00	$0.00	$0.00
Profit factor	1.01	1.06	0.95
Max. drawdc	($7,537.50)	($3,887.50)	($8,387.50)
Sharpe ratio	0.09	0.51	-0.48
Sortino ratio	0.00	0.01	-0.01
Ulcer index	3486.91	1059.27	5115.04
R squared	0.00	0.82	0.75
Start date	1/1/2014		
End date	9/18/2014		
Total # of tra	9331	4666	4665
Percent prof	30.97 %	32.60 %	29.35 %

Settings

General
- Backtest type: Standard
- Strategy: Sample MA crossove

Data Series
- Instrument: ES 12-14
- Price based on: Last
- Type: Minute
- Value: 1

Strategy parameters
- Fast: 12
- PanelUI: 0

Instrum	Backtes	Date	Strategy	Data Se	Start da	End dat	Parame	Total ne	Total # c	Notes	Pinned
ES 12-1	Standar	09/19/2(	Sample	1 Min	01/01/2(	09/18/2(	3/7 (Fas	($34,212	42955		☐
ES 12-1	Standar	09/19/2(	Sample	1 Min	01/01/2(	09/18/2(	12/32 (F	$1,500.0	9331		☐
ES 12-1	Standar	09/19/2(	Sample	1 Min	01/01/2(	09/18/2(	10/25 (F	($12,687	11874		☐

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:

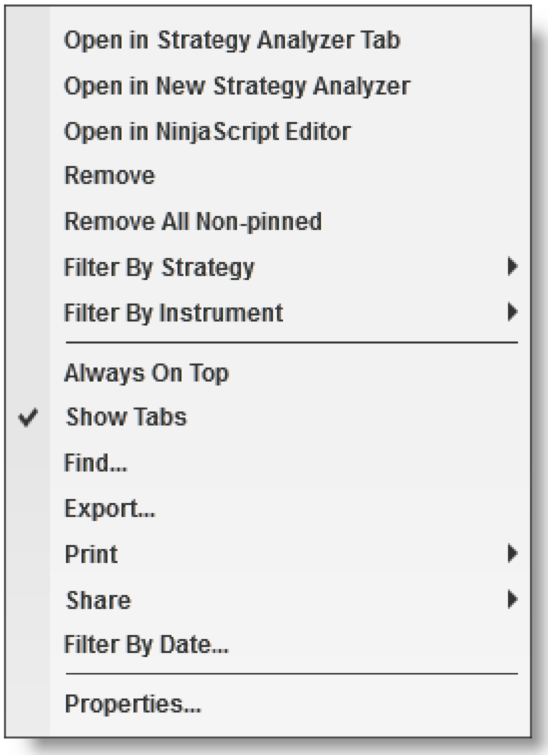
Instrument	The instrument the test was performed on.
Backtest	The type of backtest that was performed
Date	The date the backtest was performed
Strategy	The strategy used for the backtest
Data Series	The data series used for the backtest
Start date	The start date used for the backtest
End date	The end 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:



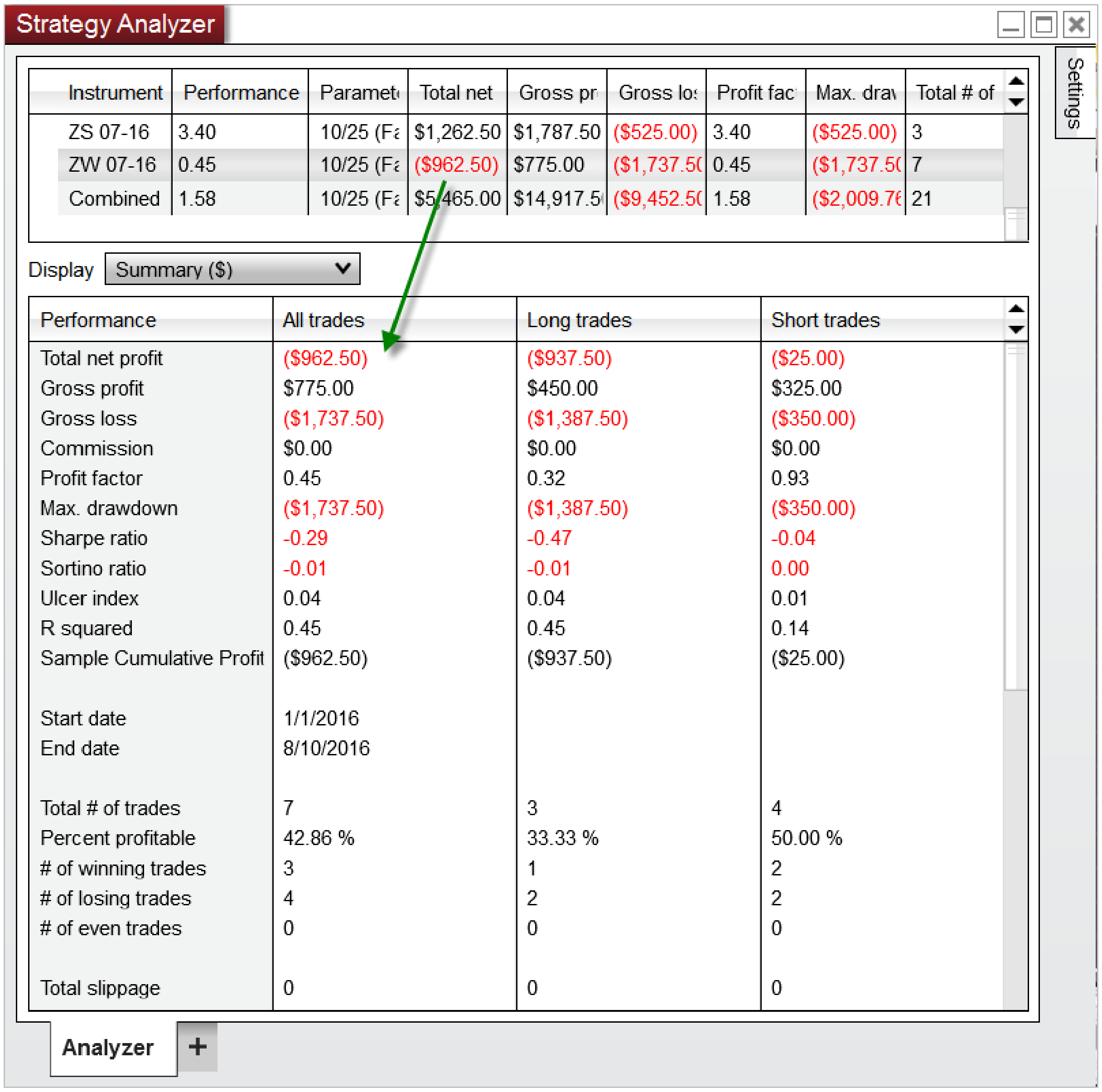
Open In Strategy Analyzer Tab	Opens the backtest in a new tab in the current Strategy Analyzer window
Open in New Strategy Analyzer	Opens the backtest in a new Strategy Analyzer
Open in NinjaScript Editor	Opens the saves revision of the code as used when the backtest was run. You can restore any set 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

11.23.9 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



Instrument	Performance	Paramet	Total net	Gross pr	Gross lo:	Profit fac	Max. drav	Total # of
ZS 07-16	3.40	10/25 (Fa	$1,262.50	$1,787.50	($525.00)	3.40	($525.00)	3
ZW 07-16	0.45	10/25 (Fa	($962.50)	$775.00	($1,737.5(	0.45	($1,737.5(	7
Combined	1.58	10/25 (Fa	$5,465.00	$14,917.5	($9,452.5(	1.58	($2,009.76	21

Performance	All trades	Long trades	Short trades
Total net profit	($962.50)	($937.50)	($25.00)
Gross profit	$775.00	$450.00	$325.00
Gross loss	($1,737.50)	($1,387.50)	($350.00)
Commission	$0.00	$0.00	$0.00
Profit factor	0.45	0.32	0.93
Max. drawdown	($1,737.50)	($1,387.50)	($350.00)
Sharpe ratio	-0.29	-0.47	-0.04
Sortino ratio	-0.01	-0.01	0.00
Ulcer index	0.04	0.04	0.01
R squared	0.45	0.45	0.14
Sample Cumulative Profit	($962.50)	($937.50)	($25.00)
Start date	1/1/2016		
End date	8/10/2016		
Total # of trades	7	3	4
Percent profitable	42.86 %	33.33 %	50.00 %
# of winning trades	3	1	2
# of losing trades	4	2	2
# of even trades	0	0	0
Total slippage	0	0	0

11.23.10 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

11.23.10.1 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 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.

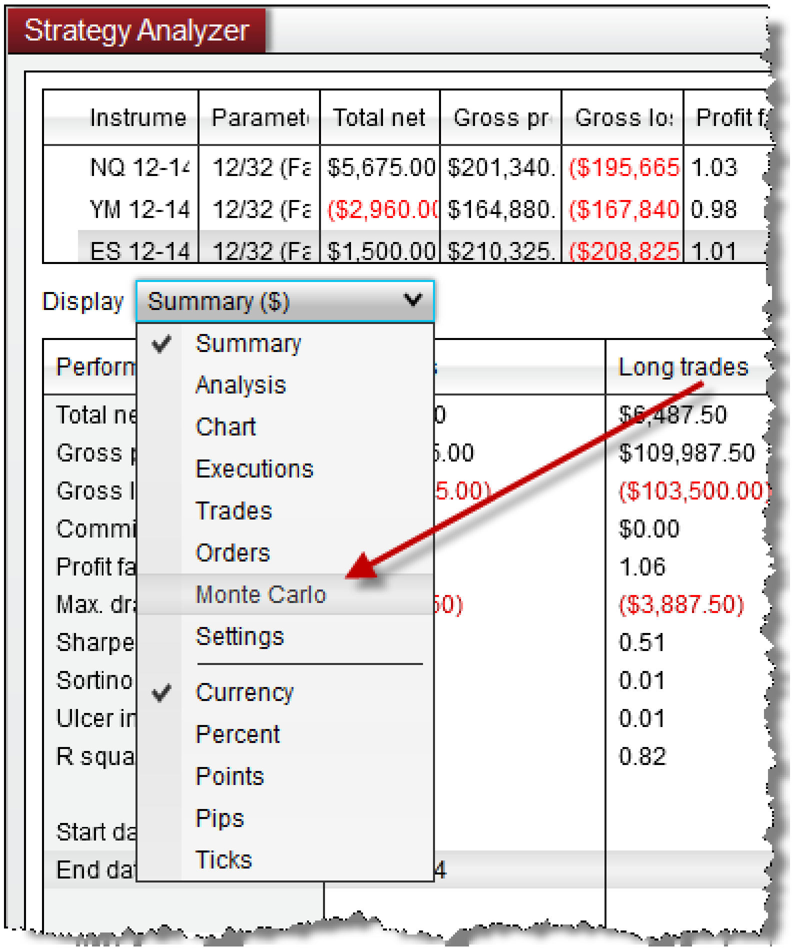
Why use Monte Carlo Simulation?
Although a backtest of a NinjaScript strategy may produce profitable results, those results may have just been due to good luck. In real life, you may have a string of bad trades that can wipe out the account before the good trades appear, therefore it would be helpful 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, Walk-Forward Optimization, or run an Account Performance report.
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...**



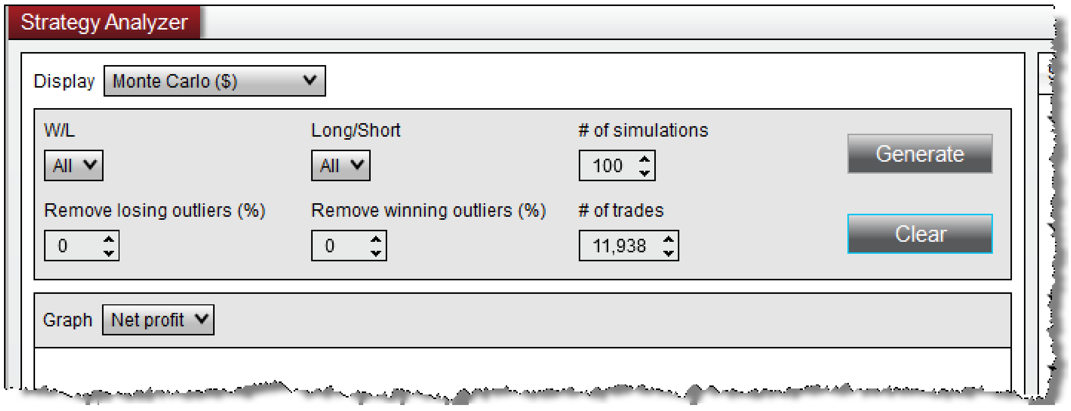
Instrume	Paramet	Total net	Gross pr	Gross lo:	Profit fa
NQ 12-14	12/32 (Fa	$5,675.00	$201,340.	($195,665	1.03
YM 12-14	12/32 (Fa	($2,960.0(	$164,880.	($167,840	0.98
ES 12-14	12/32 (Fa	$1,500.00	$210,325.	($208,825	1.01

Perform		Long trades
Total ne	0	$6,487.50
Gross p	.00	$109,987.50
Gross l	5.00)	($103,500.00)
Commi		$0.00
Profit fa		1.06
Max. dr	0)	($3,887.50)
Sharpe		0.51
Sortino		0.01
Ulcer ir		0.01
R squa		0.82
Start da		
End da	4	

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.



Display: Monte Carlo ($)

W/L	Long/Short	# of simulations
All	All	100

Remove losing outliers (%)	Remove winning outliers (%)	# of trades
0	0	11,938

Graph: Net profit

Monte Carlo Simulation Parameters

The following parameters are adjustable when running a Monte Carlo Simulation:

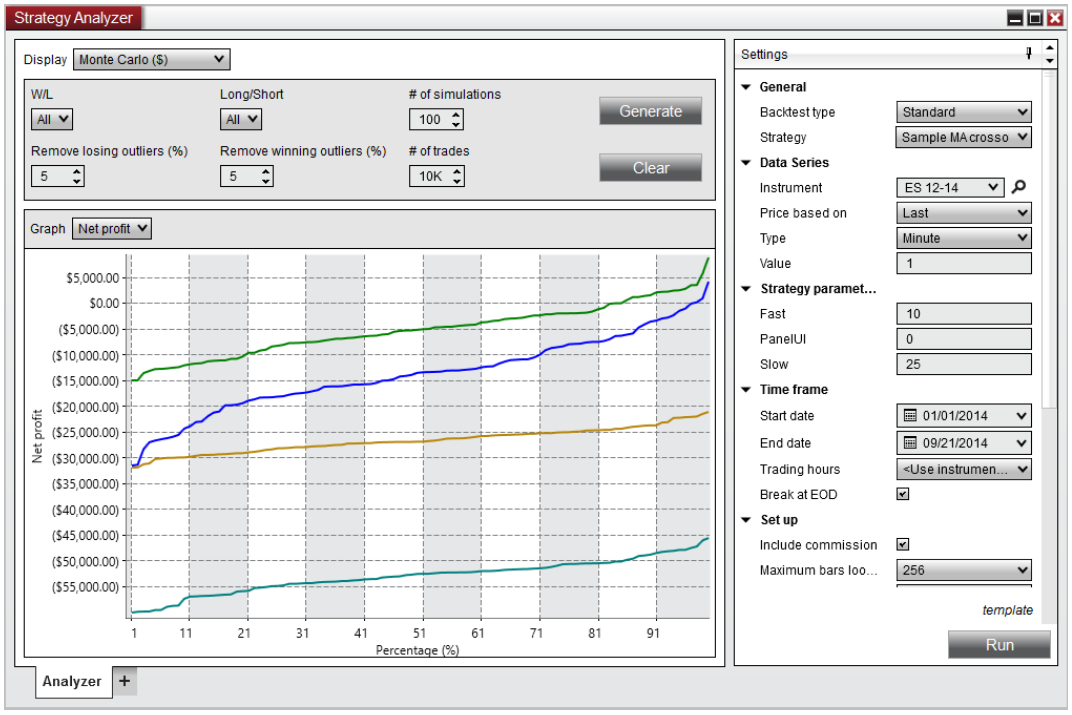
Graph	Sets the statistic to generate the report on
W/L	Sets the results to show only winners, only loser, or both
Long/ Short	Sets the results to show only long trades, only short trades, or both
Remove winning outliers (%)	Removes the top % outliers from the results
Remove losing outliers (%)	Removes the bottom % outliers from the results
# of simulatio ns	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)

simulatio n	

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.



Display: Monte Carlo ($)

W/L	Long/Short	# of simulations
All	All	100

Remove losing outliers (%)	Remove winning outliers (%)	# of trades
5	5	10K

Graph: Net profit

Settings

General	
Backtest type	Standard
Strategy	Sample MA crosso
Data Series	
Instrument	ES 12-14
Price based on	Last
Type	Minute
Value	1
Strategy paramet...	
Fast	10
PanelUI	0
Slow	25
Time frame	
Start date	01/01/2014
End date	09/21/2014
Trading hours	<Use instrumen...
Break at EOD	✓
Set up	
Include commission	✓
Maximum bars loo...	256

X-Axis

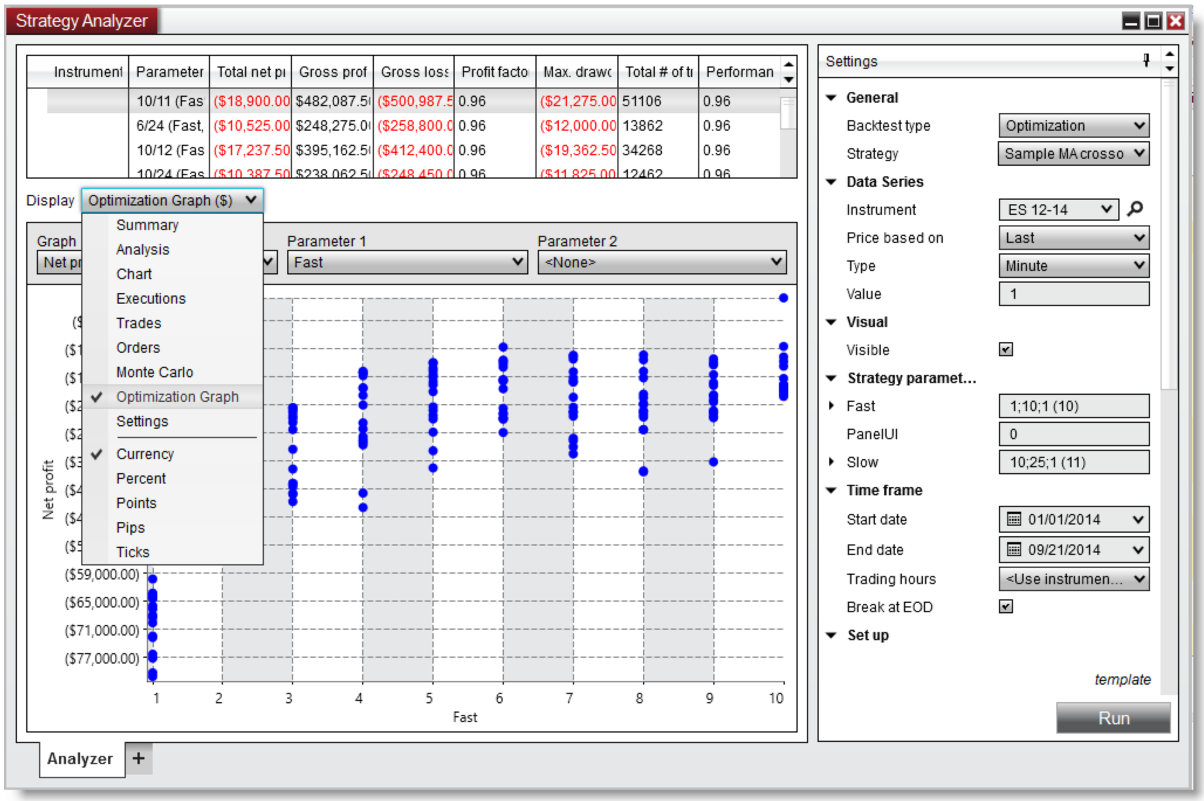
The horizontal axis of the Monte Carlo Simulation graph shows the percentage of simulations that have fallen below the Y - axis value. For example, if you run a Monte Carlo Simulation setting the # of Simulations to "100" and using the Cumulative Profit graph, the intersection of the 50% X - value and 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.

11.23.1 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.



Instrument	Parameter	Total net pı	Gross prof	Gross loss	Profit facto	Max. drawd	Total # of tı	Performan
	10/11 (Fas	($18,900.00	$482,087.5	($500,987.5	0.96	($21,275.00	51106	0.96
	6/24 (Fast,	($10,525.00	$248,275.0	($258,800.0	0.96	($12,000.00	13862	0.96
	10/12 (Fas	($17,237.50	$395,162.5	($412,400.0	0.96	($19,362.50	34268	0.96
	10/24 (Fas	($10,387.50	$238,062.5	($248,450.0	0.96	($11,825.00	12462	0.96

Display Optimization Graph ($)

Graph Net pr Parameter 1 Fast Parameter 2 <None>

Settings

General	
Backtest type	Optimization
Strategy	Sample MA crosso
Data Series	
Instrument	ES 12-14
Price based on	Last
Type	Minute
Value	1
Visual	
Visible	✓
Strategy paramet...	
Fast	1;10;1 (10)
PanelUI	0
Slow	10;25;1 (11)
Time frame	
Start date	01/01/2014
End date	09/21/2014
Trading hours	<Use instrumen...
Break at EOD	✓
Set up	

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.

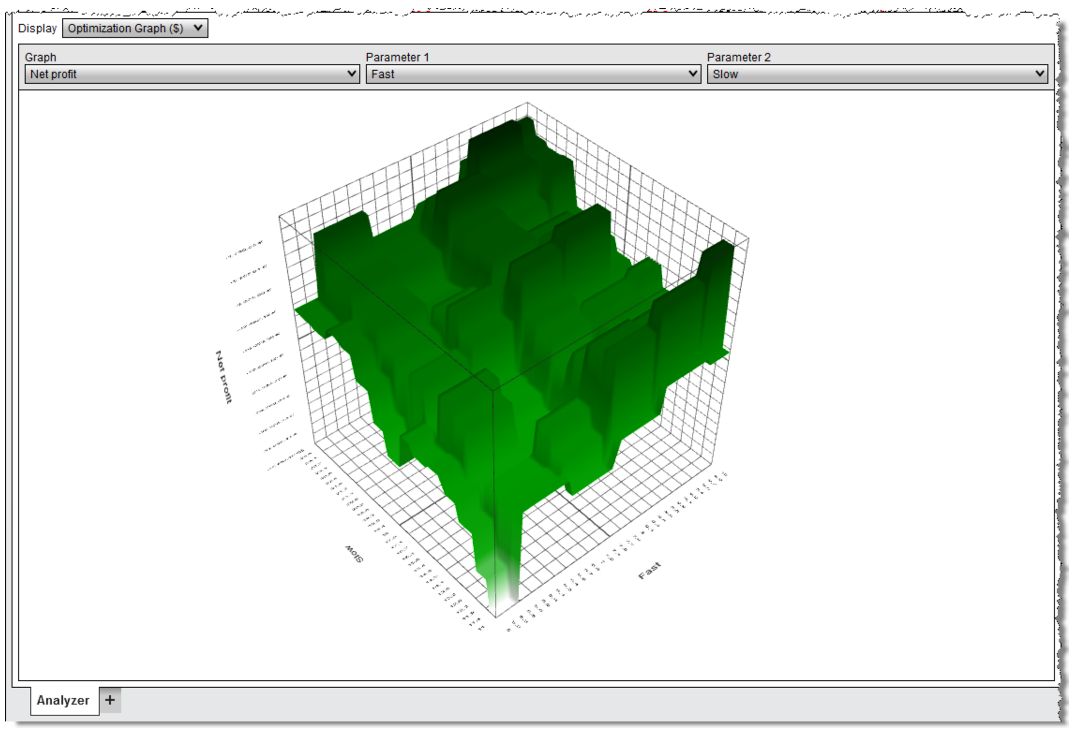
The 2D Optimization graph will be displayed when you have only selected a single parameter and is the default graph view.

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


Display: Optimization Graph ($)
Graph: Net profit
Parameter 1: Fast
Parameter 2: Slow

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

11.23.12 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) 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.

11.23.13 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.

11.23.14 Strategy Analyzer Properties

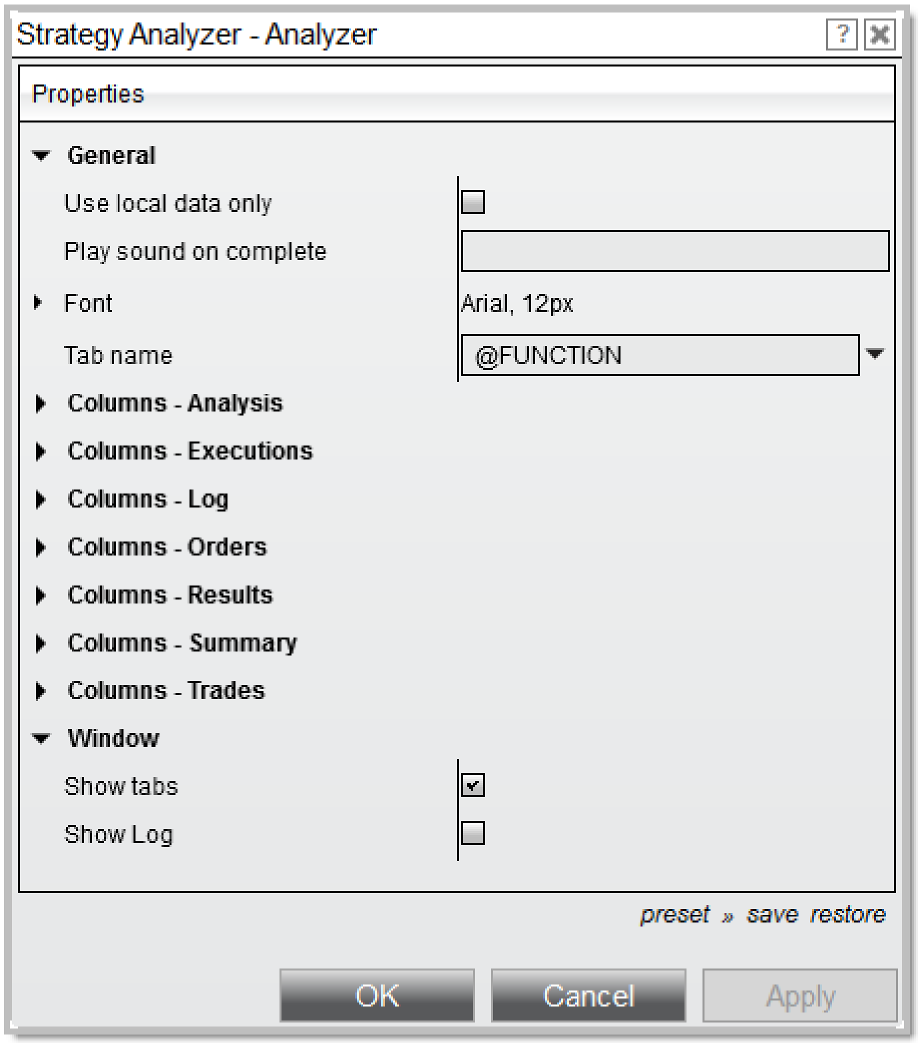
Many of the **Strategy Analyzer** visual display settings can be customized using the **Strategy Analyzer Properties** window.

▽ 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:



General
Use local data only ☐
Play sound on complete
Font Arial, 12px
Tab name @FUNCTION
Columns - Analysis
Columns - Executions
Columns - Log
Columns - Orders
Columns - Results
Columns - Summary
Columns - Trades
Window
Show tabs ☑
Show Log ☐

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.
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 "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.

11.23.15 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.

Note: This property is set to **true** by default **ONLY** when applying a strategy to a chart. If you wish to disable it when applying a strategy to a chart, you will need to explicitly set it to **false** in your script.

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.

Note: This property is set to **false** by default when applying a strategy directly in the

Strategies tab of the **Control Center**, or when using the **Strategy Analyzer** for backtesting or strategy optimization. If you wish to enable it when applying a strategy in either of these ways, you will need to explicitly set it to **true** in your script.

11.24 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:

› Builder Screens
› Condition Builder
› Strategy Actions

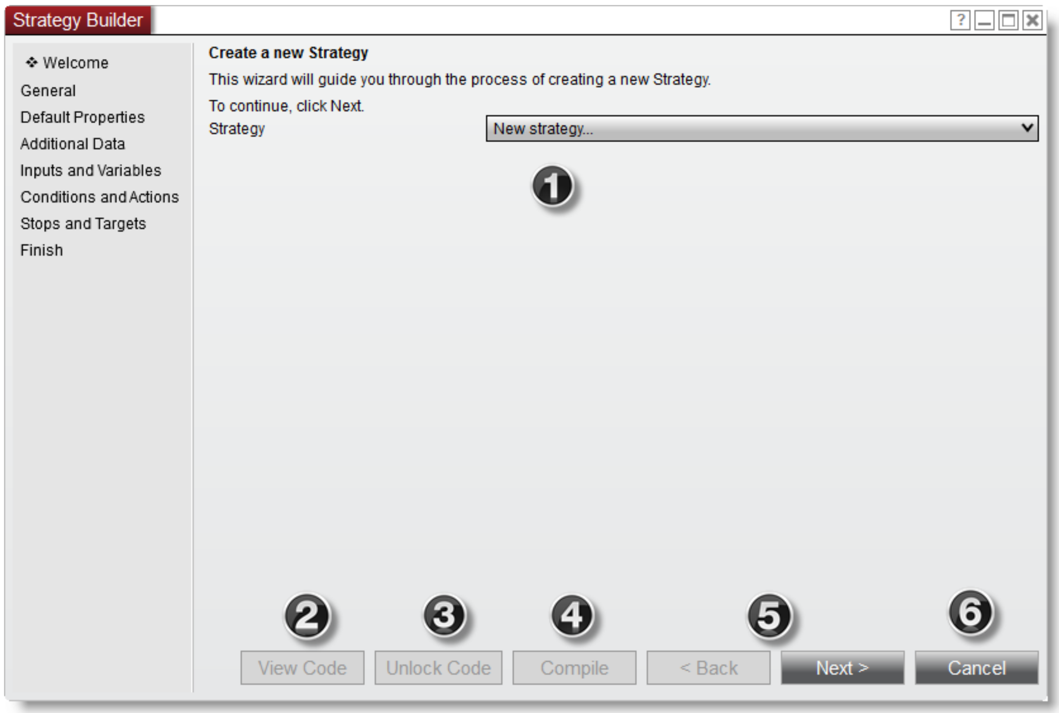
11.24.1 Builder Screens

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**.



Create a new Strategy

This wizard will guide you through the process of creating a new Strategy.

To continue, click Next.

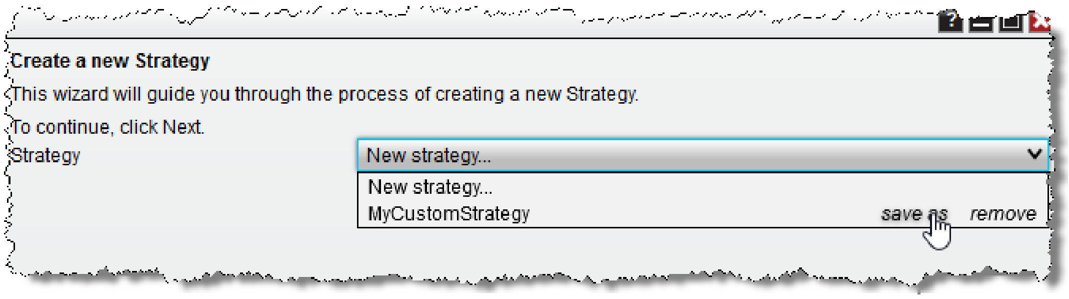
Strategy: New strategy...

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.


Create a new Strategy

This wizard will guide you through the process of creating a new Strategy.

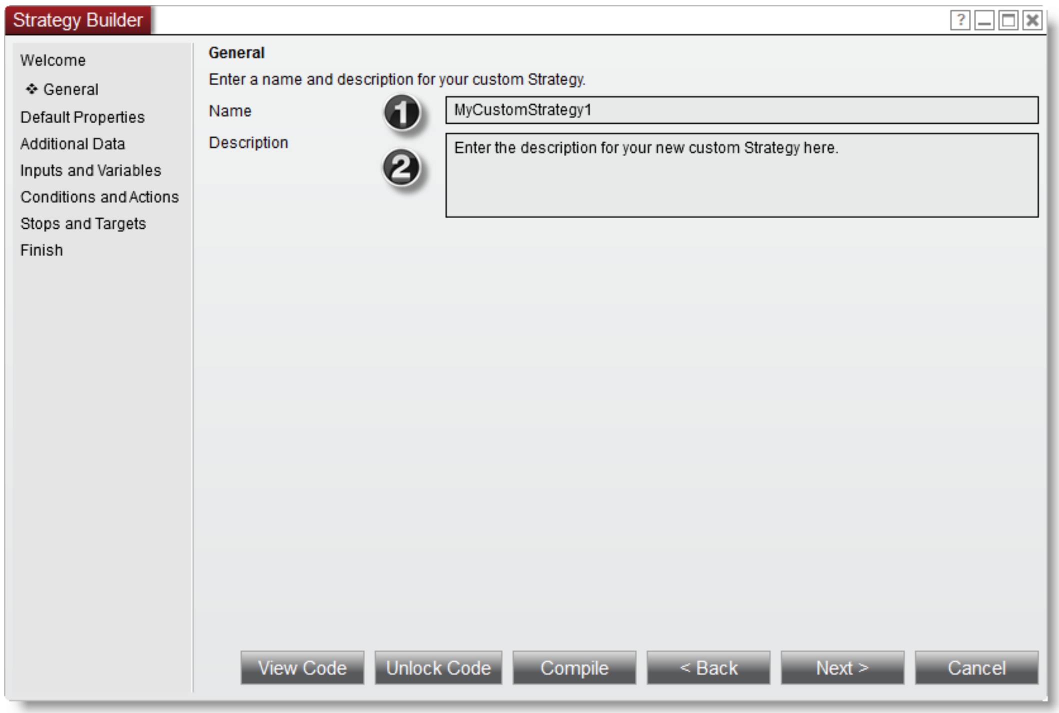
To continue, click Next.

Strategy: New strategy...

Understanding the General screen

General Screen Layout

The **General** screen is where you enter the name and description of your strategy.



General

Enter a name and description for your custom Strategy.

Name: MyCustomStrategy1

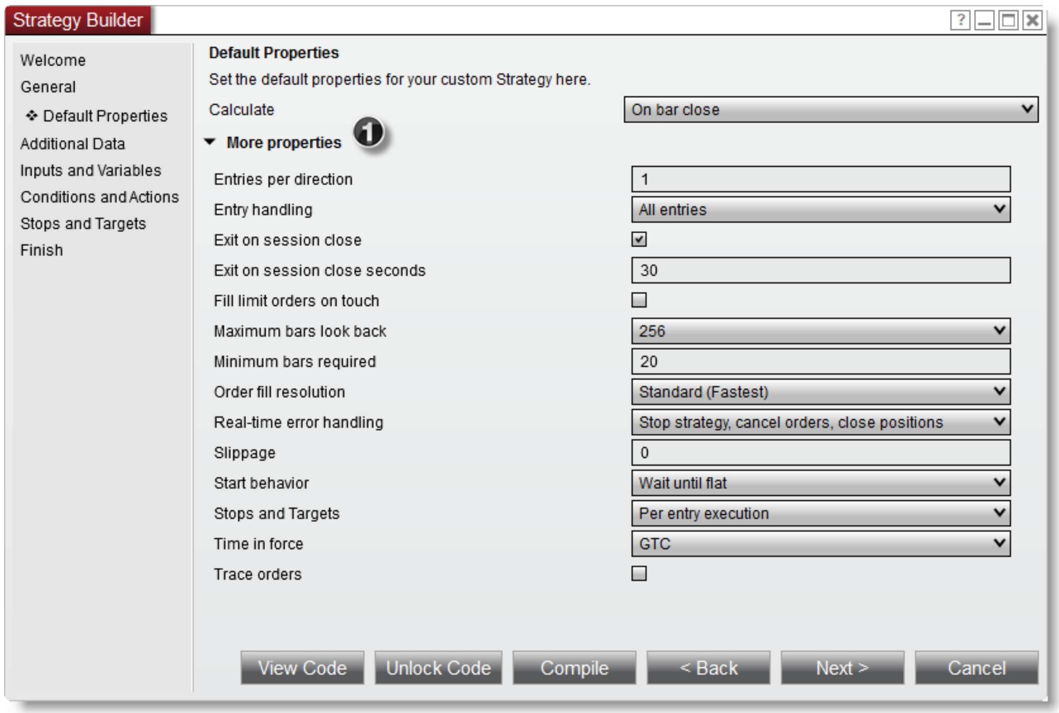
Description: Enter the description for your new custom Strategy here.

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.



Default Properties
Set the default properties for your custom Strategy here.

Calculate	On bar close
More properties	
Entries per direction	1
Entry handling	All entries
Exit on session close	✓
Exit on session close seconds	30
Fill limit orders on touch	☐
Maximum bars look back	256
Minimum bars required	20
Order fill resolution	Standard (Fastest)
Real-time error handling	Stop strategy, cancel orders, close positions
Slippage	0
Start behavior	Wait until flat
Stops and Targets	Per entry execution
Time in force	GTC
Trace orders	☐

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

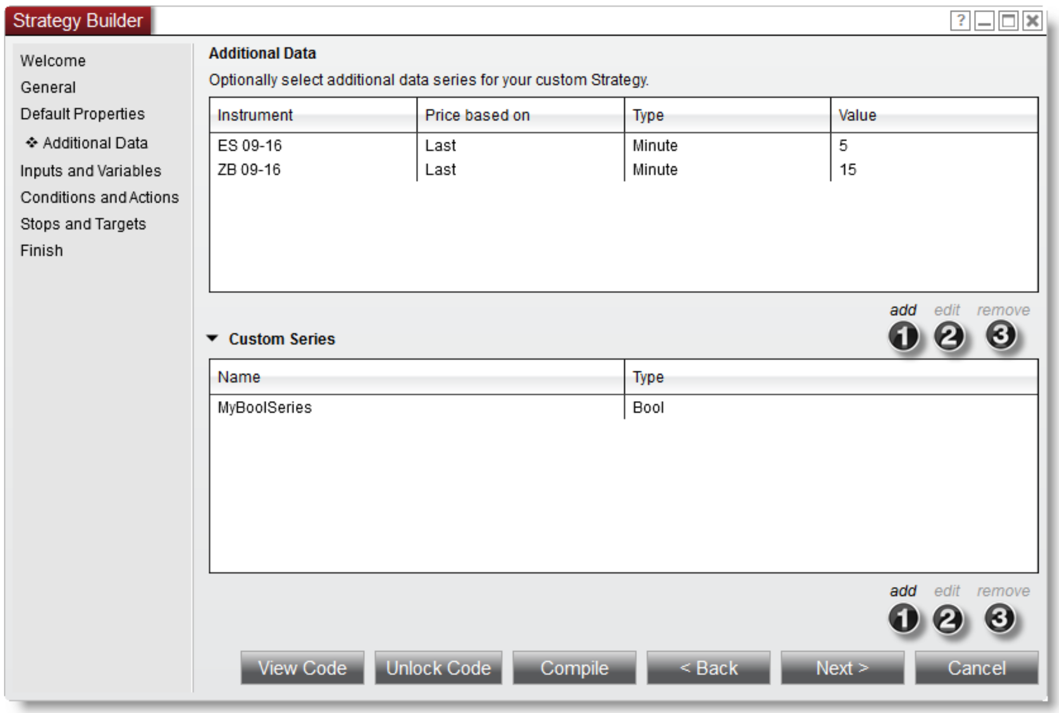
	strategy will process entry orders until the maximum allowable entries set by the "Entries per direction" property per each uniquely named entry have been reached.
Exit on close	When enabled, open positions will be closed on the last bar of a session
Exit on close seconds	Sets the number of seconds prior 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.



Additional Data

Optionally select additional data series for your custom Strategy.

Instrument	Price based on	Type	Value
ES 09-16	Last	Minute	5
ZB 09-16	Last	Minute	15

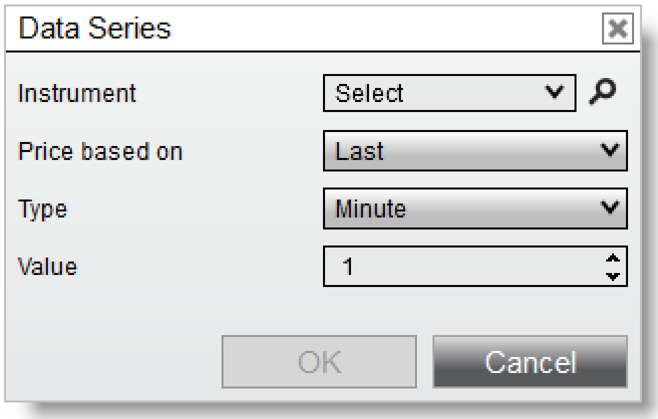
Custom Series

Name	Type
MyBoolSeries	Bool

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


Data Series

Instrument: Select

Price based on: Last

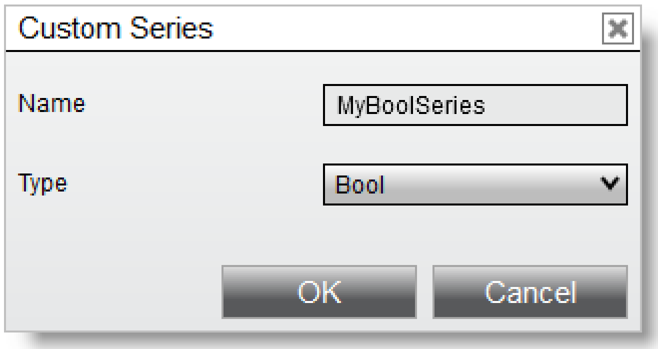
Type: Minute

Value: 1

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



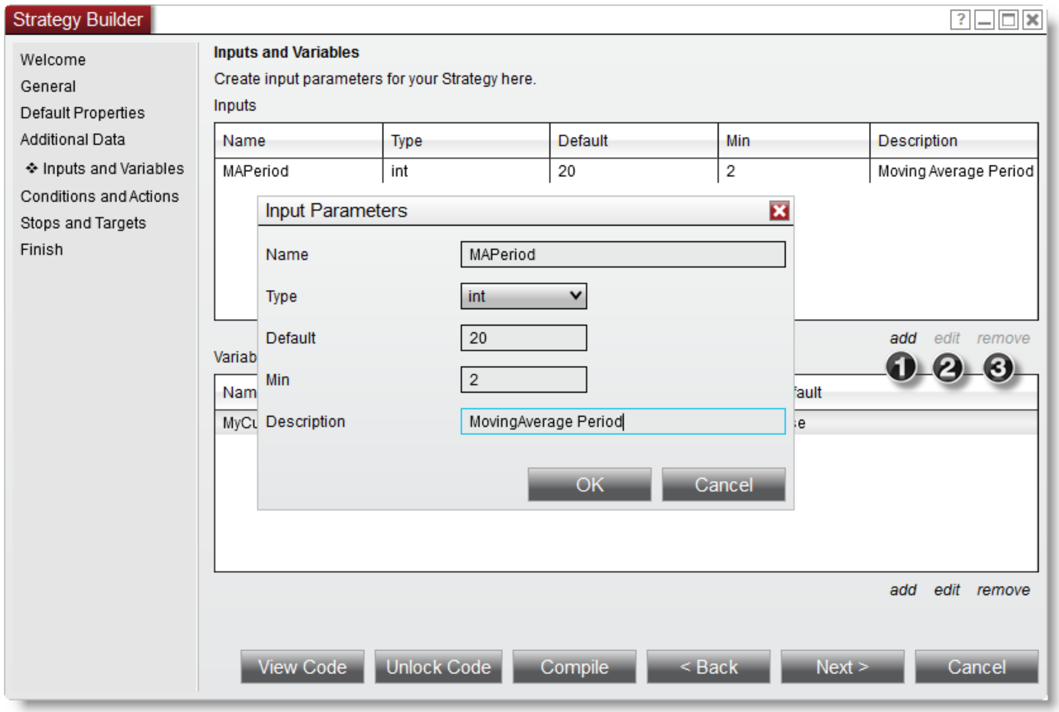
Name: MyBoolSeries
Type: Bool

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.



Inputs and Variables

Create input parameters for your Strategy here.

Inputs

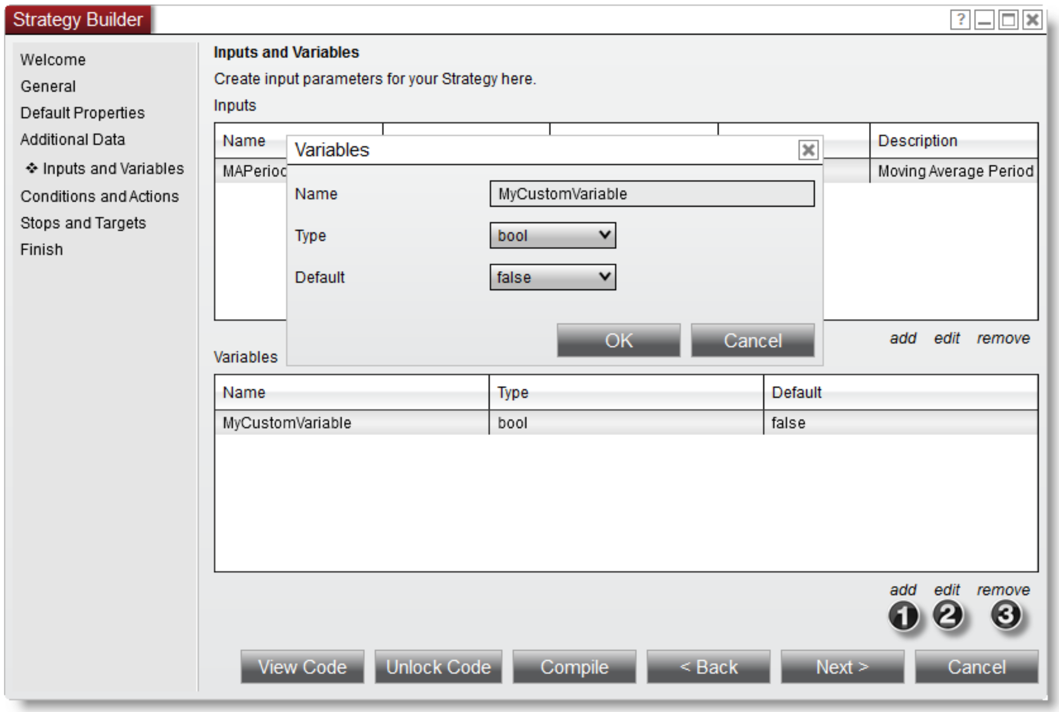
Name	Type	Default	Min	Description
MAPeriod	int	20	2	Moving Average Period

Input Parameters

Name	MAPeriod
Type	int
Default	20
Min	2
Description	MovingAverage Period

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



Name	Type	Default
MyCustomVariable	bool	false

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

Understanding the Conditions and Actions screen

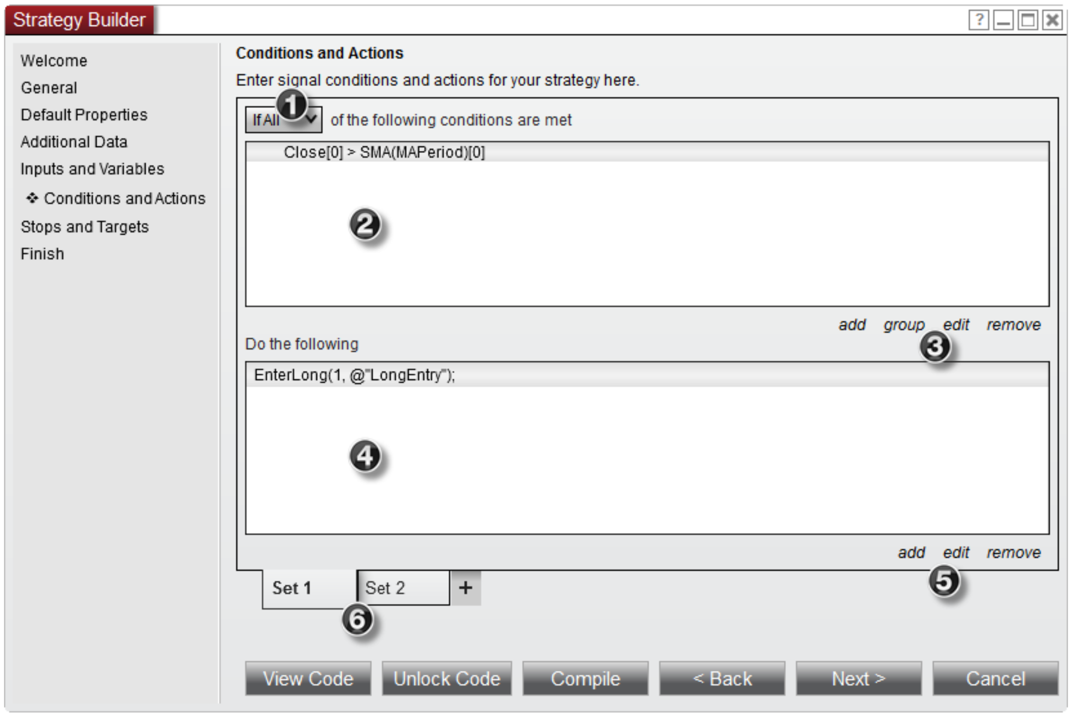
Conditions and Actions screen Layout

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.



```
Close[0] > SMA(MAPeriod)[0]
```

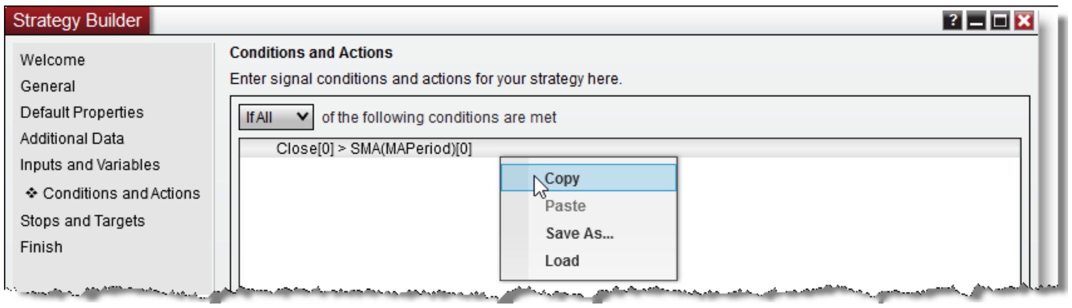
```
EnterLong(1, @"LongEntry");
```

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

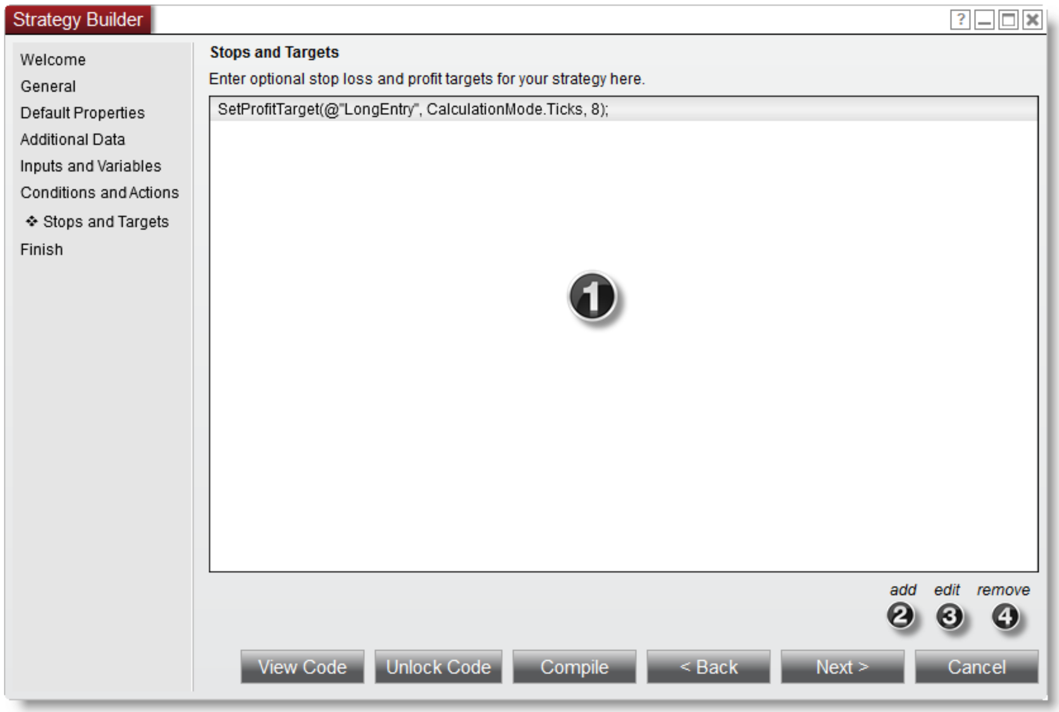
You can copy and paste conditions from one set to another and you can even save a condition set as a template and load for future use via the right mouse button click context menu as show in the image below. To save a condition set as a template, select the **Save As...** menu item and then to re-use 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 and profit target orders that are automatically submitted and managed once your strategy opens a position.



Enter optional stop loss and profit targets for your strategy here.

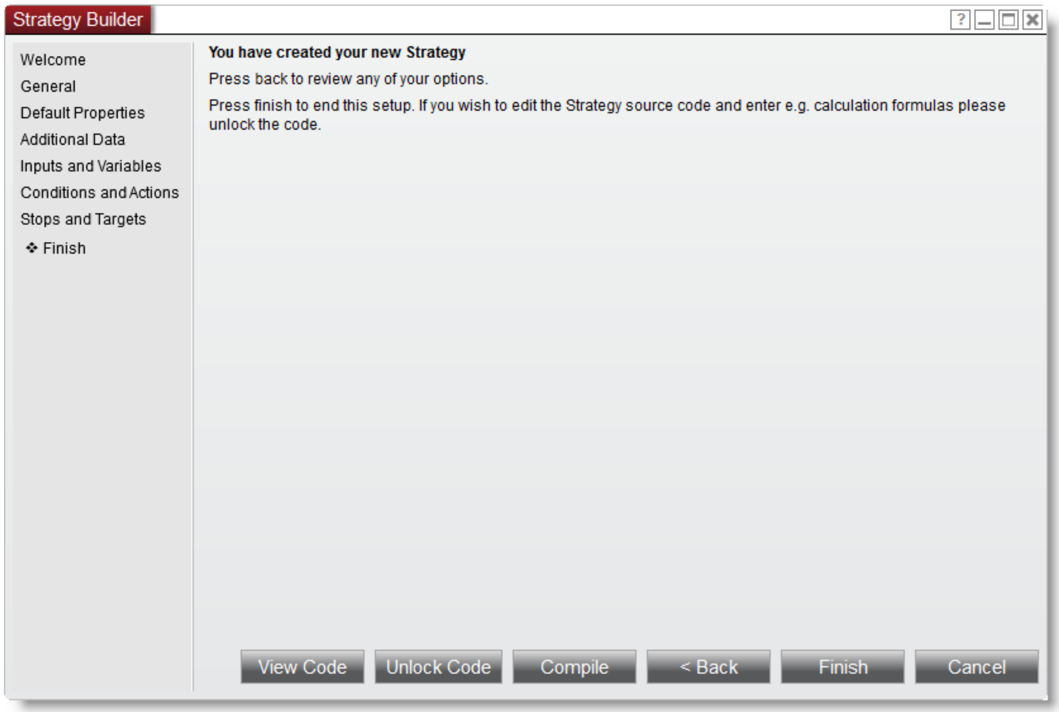
```
SetProfitTarget(@"LongEntry", CalculationMode.Ticks, 8);
```

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.



You have created your new Strategy

Press back to review any of your options.

Press finish to end this setup. If you wish to edit the Strategy source code and enter e.g. calculation formulas please unlock the code.

11.24.2 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.

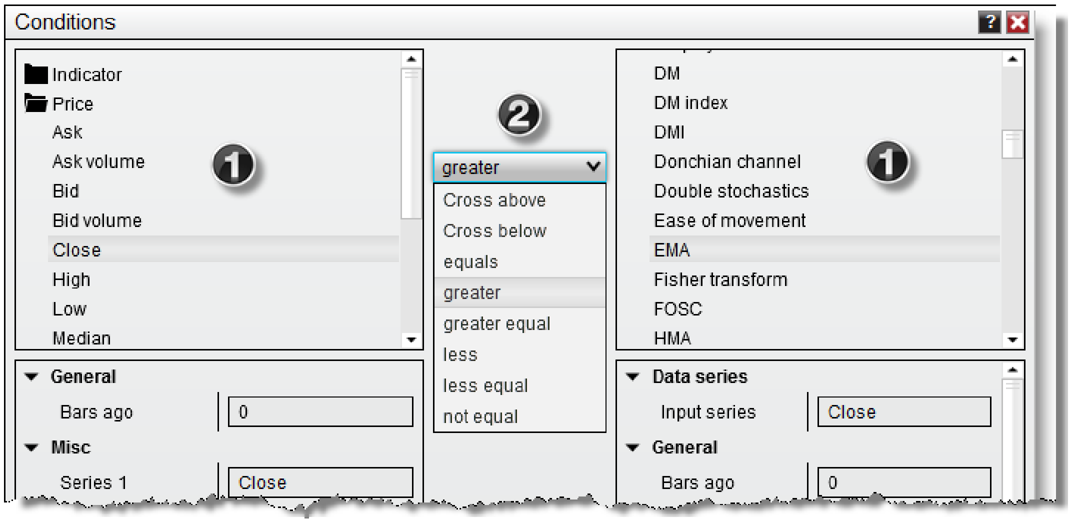
Basic Operation

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).

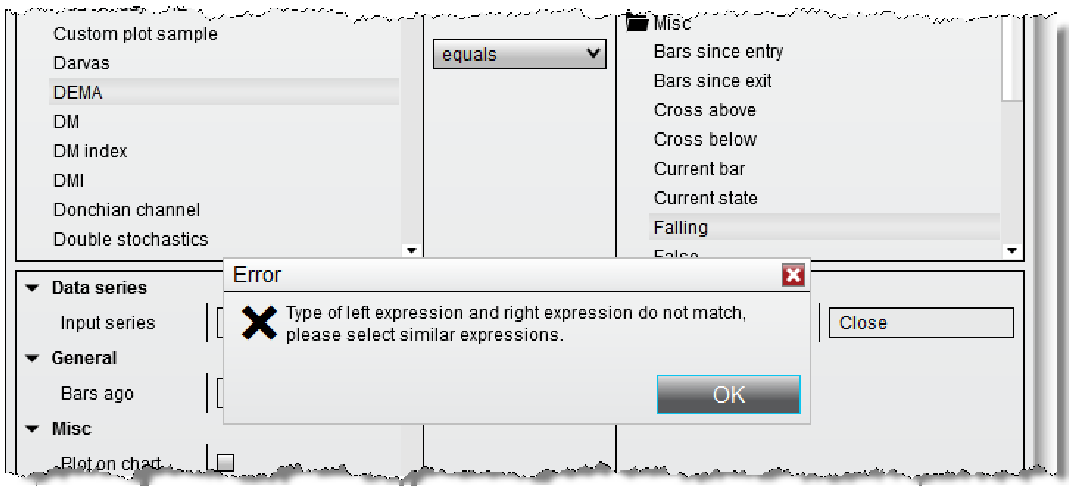


1. Available items such as indicators, price data, etc. to use for the comparison
2. List of relational operators

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 -

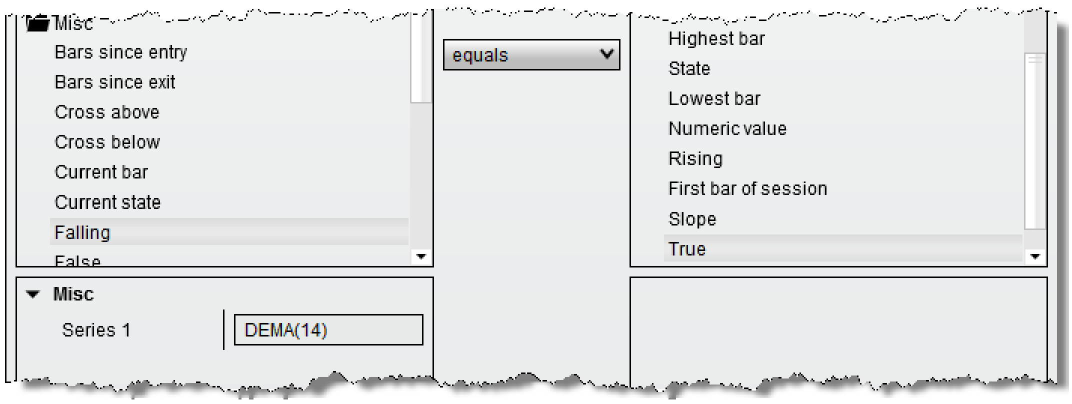
"Type of left expression and right expression do not match, please select similar expressions"



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.

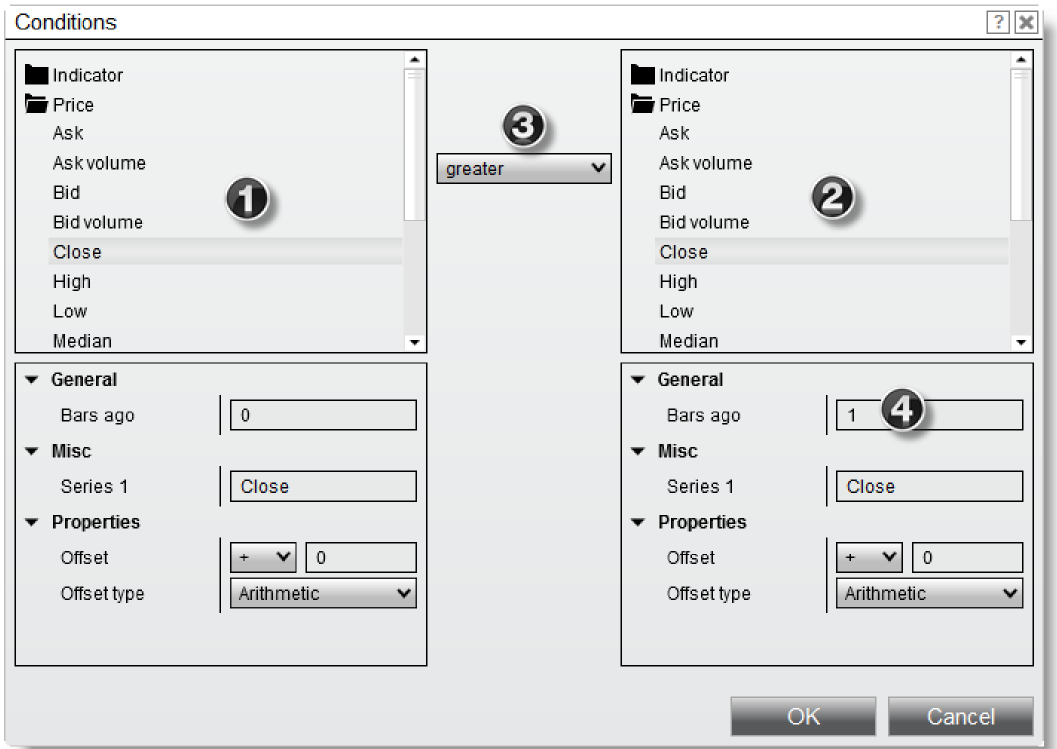


▽ How to make price data comparisons

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"



Bars ago	0
Series 1	Close
Offset	+ 0
Offset type	Arithmetic

Bars ago	1
Series 1	Close
Offset	+ 0
Offset type	Arithmetic

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.

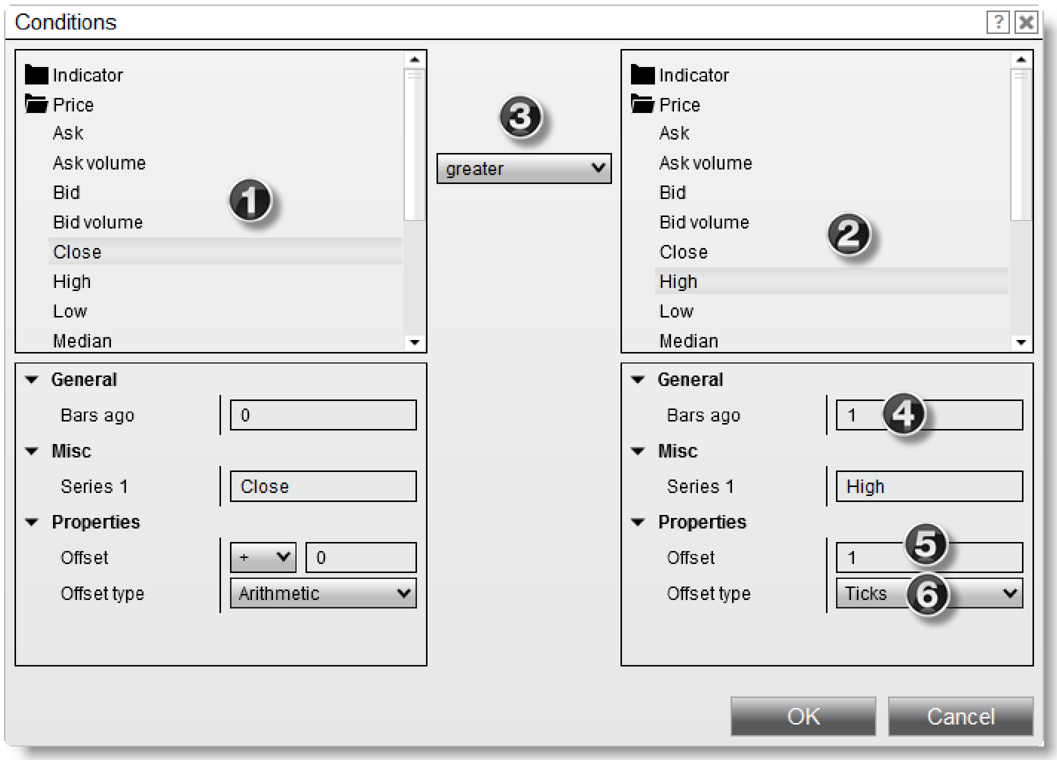
Offset type can be set to:

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"



Left: Bars ago 0; Series 1 Close; Offset + 0; Offset type Arithmetic

Right: Bars ago 1; Series 1 High; Offset 1; Offset type Ticks

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

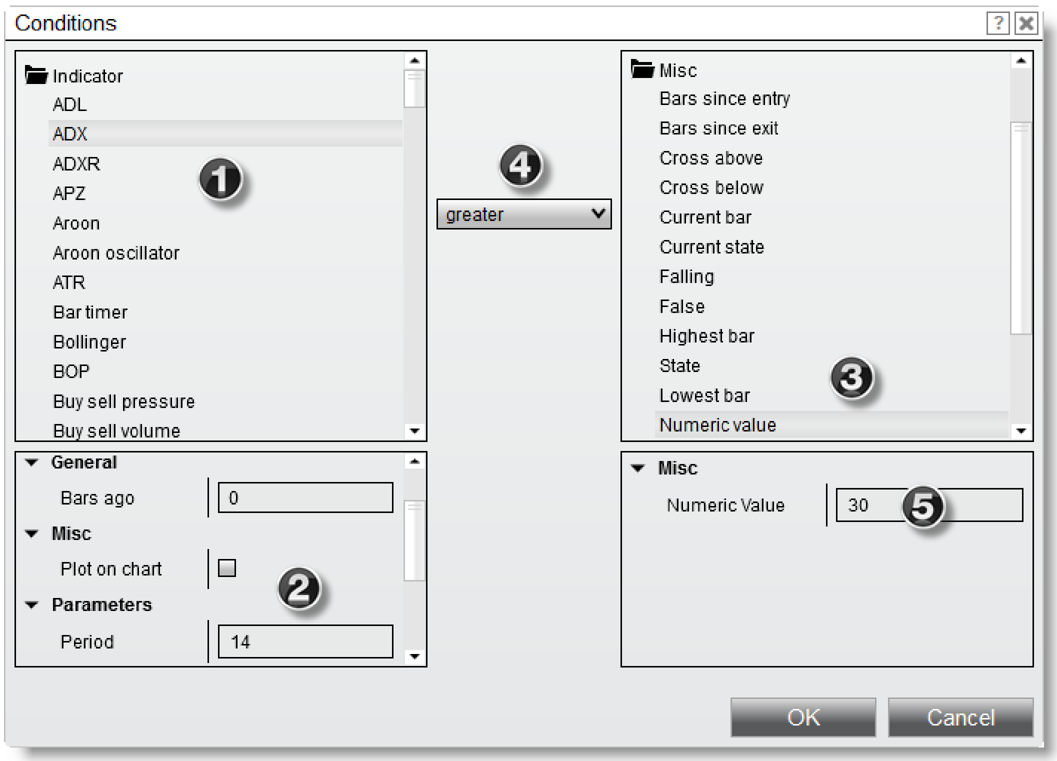
Indicator to Value comparisons

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"

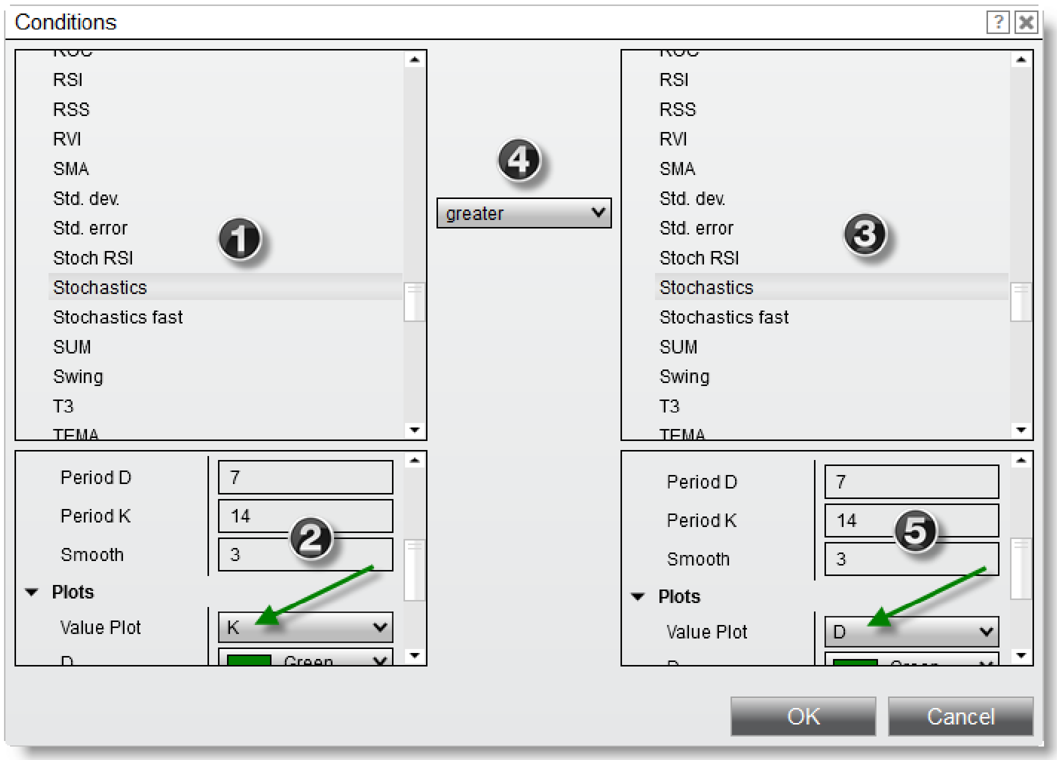
How to compare plot values of multi-plot indicators

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)



Period D: 7
Period K: 14
Smooth: 3
Value Plot: K (left) / D (right)

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.

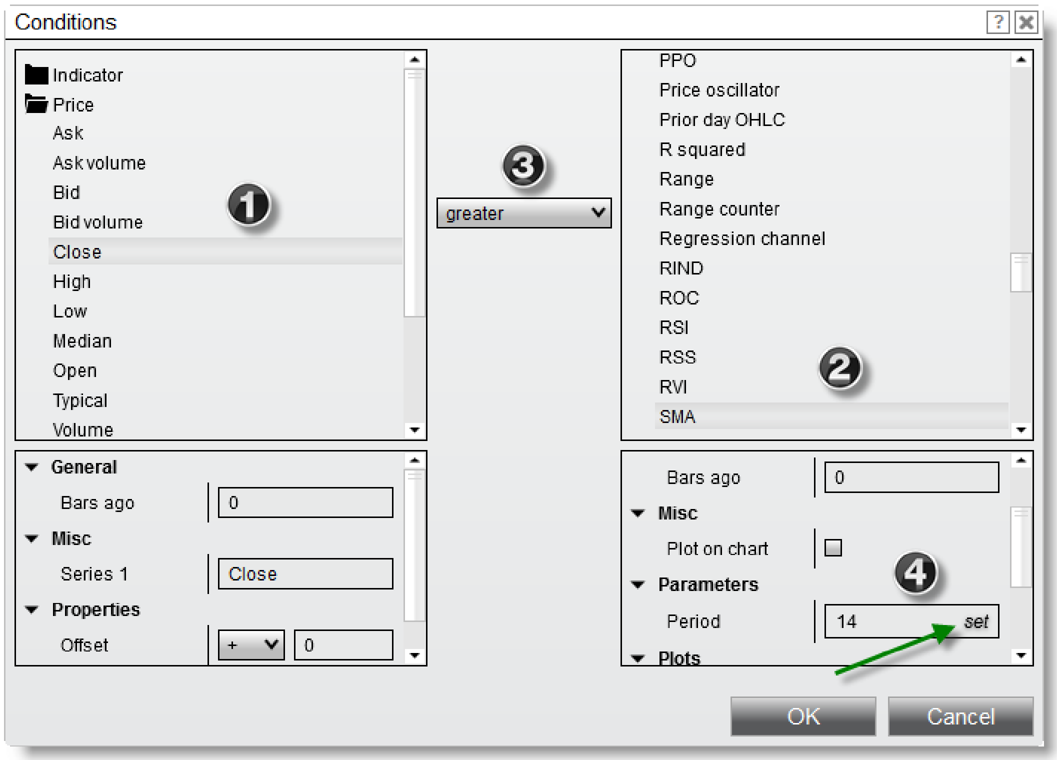
SMA(9) is how you express a 9 period simple moving average in NinjaScript. If you run a strategy, you would always be using a 9 period simple moving average. At run time, you might want to change this value to 10. User defined inputs accomplish this. If you created an input named "MyInput", you could 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.

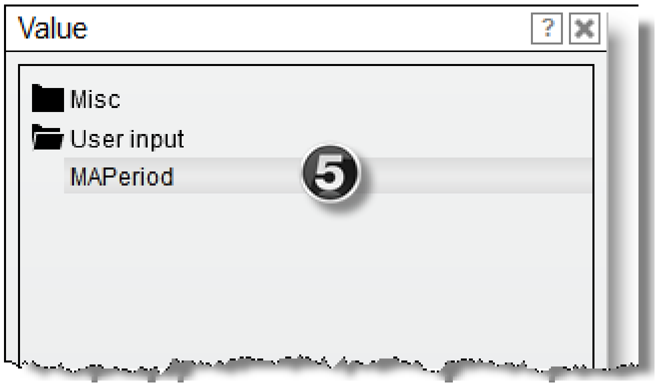
- User inputs are created from the Builder screen
- User variables can be set in the strategy logic through the **Condition Builder** (see the sections above)

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.

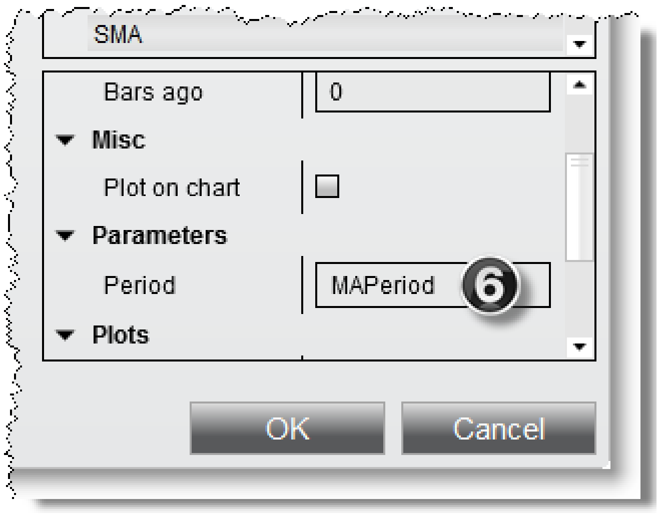
1. Expand the **Price** category and select the **Close**.
2. Expand the **Indicator** category and select the **SMA** indicator
3. Select the **greater** relational operator
4. Set the **Period** parameter to a user defined input by pressing the "**Set**" button (green arrow) to open the **Value** window



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.



Bars ago: 0
Plot on chart: ☐
Period: MAPeriod

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"

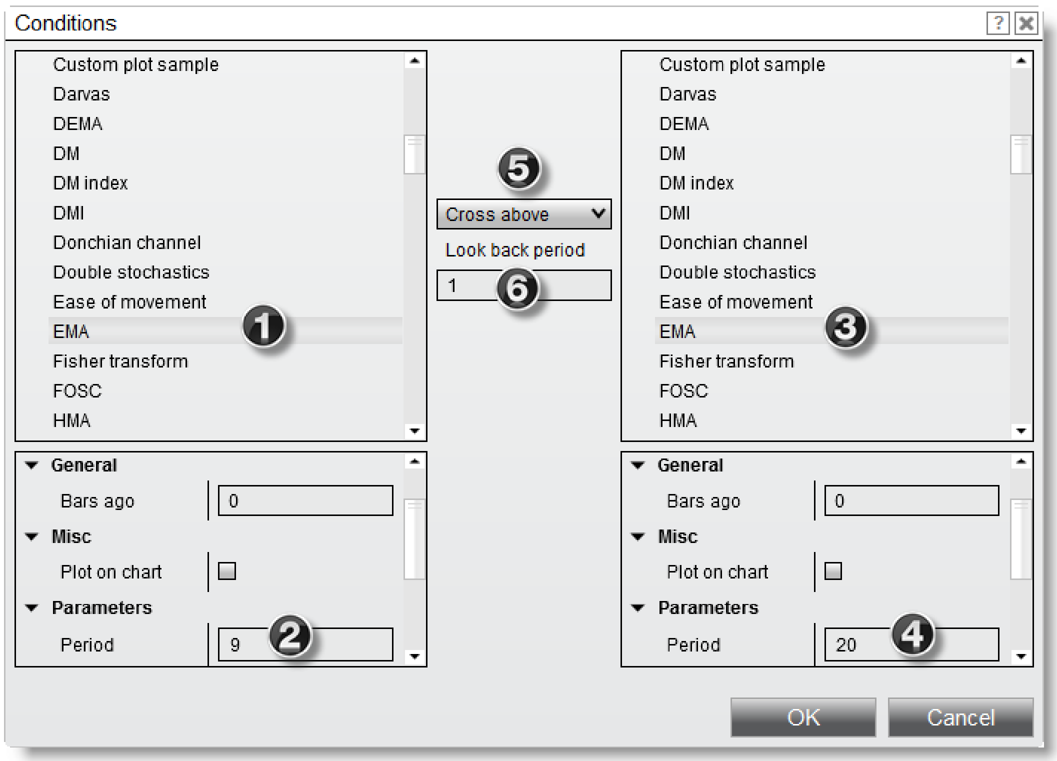
How to create a cross over condition

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**


Conditions

Custom plot sample
Darvas
DEMA
DM
DM index
DMI
Donchian channel
Double stochastics
Ease of movement
EMA 1
Fisher transform
FOSC
HMA

General
Bars ago 0
Misc
Plot on chart ☐
Parameters
Period 9 2

5
Cross above
Look back period
1 6

Custom plot sample
Darvas
DEMA
DM
DM index
DMI
Donchian channel
Double stochastics
Ease of movement
EMA 3
Fisher transform
FOSC
HMA

General
Bars ago 0
Misc
Plot on chart ☐
Parameters
Period 20 4

OK Cancel

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"

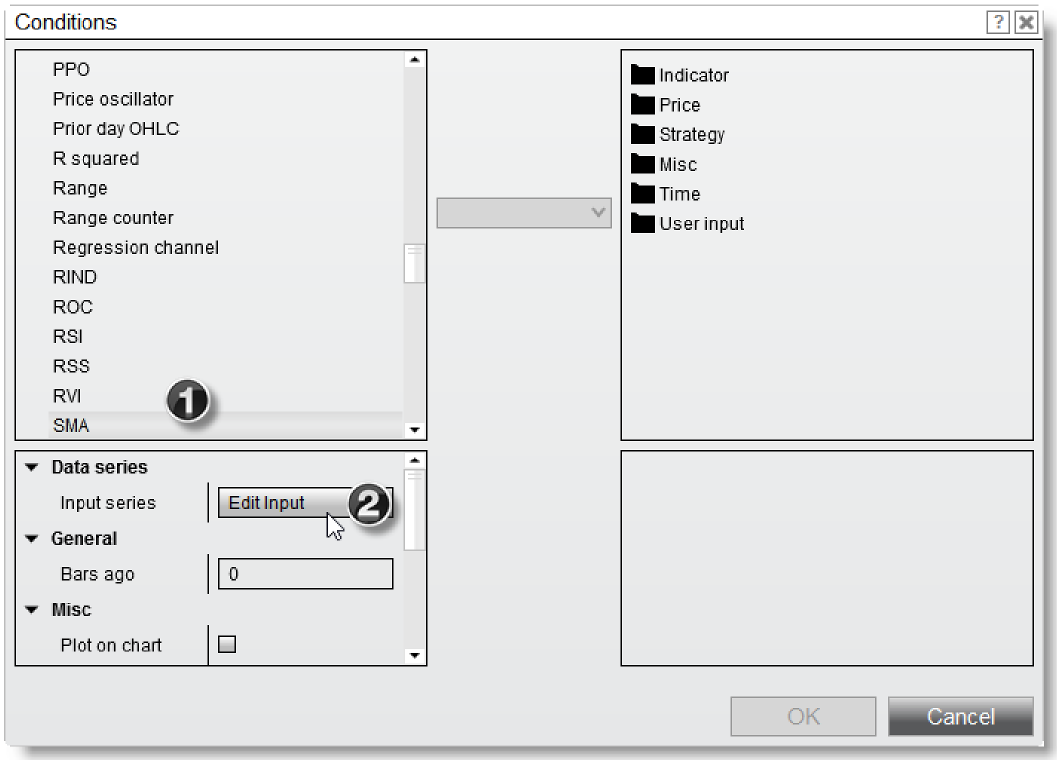
How to use indicator inputs in other indicators

Indicator on Indicator

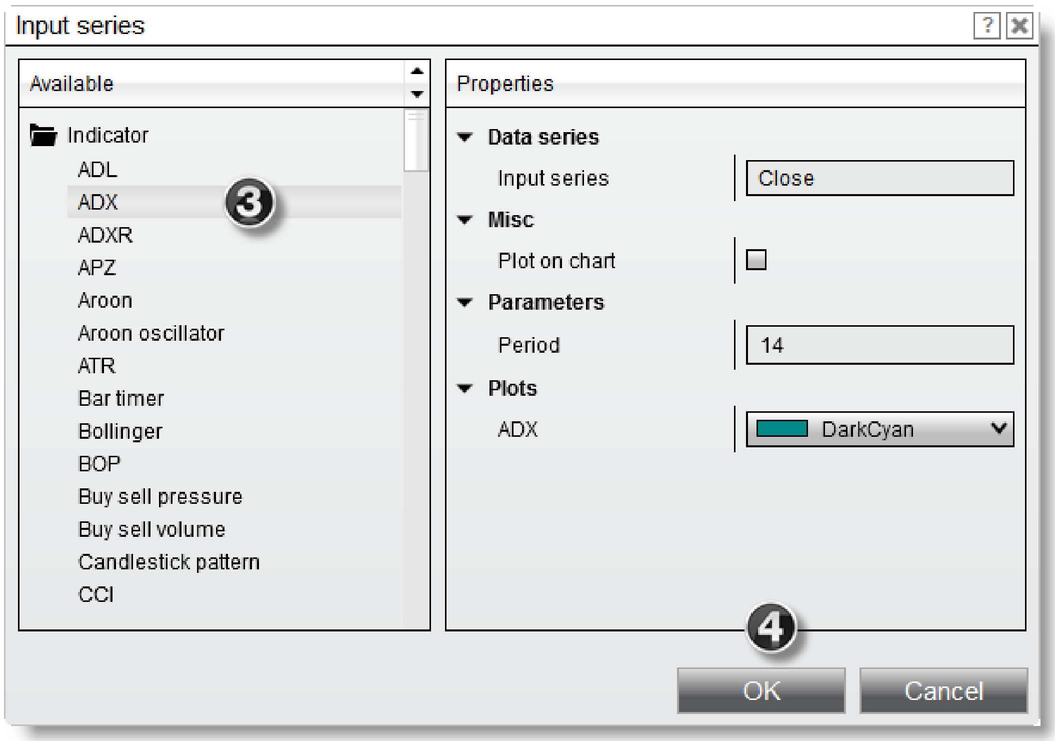
You can use indicators as input for other indicators ... actually, you can nest indicators within indicators infinitely if you really wanted to!

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

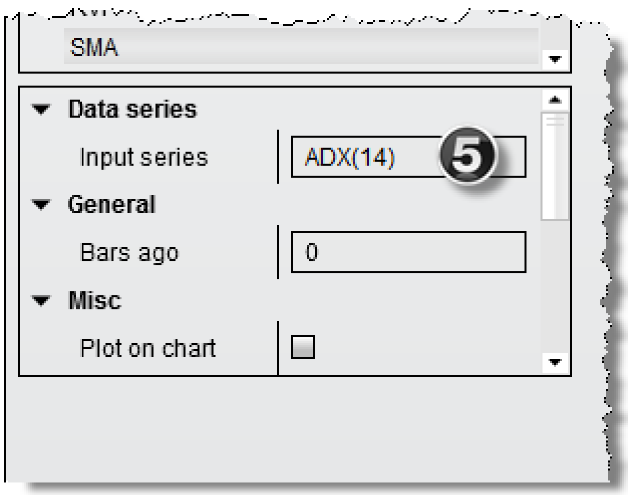


3. Select the **ADX** indicator and set any properties in the **Properties** window
4. Press the **OK** button



Data series
Input series: Close
Misc
Plot on chart: ☐
Parameters
Period: 14
Plots
ADX: DarkCyan

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.



Data series
Input series: ADX(14)
General
Bars ago: 0
Misc
Plot on chart: ☐

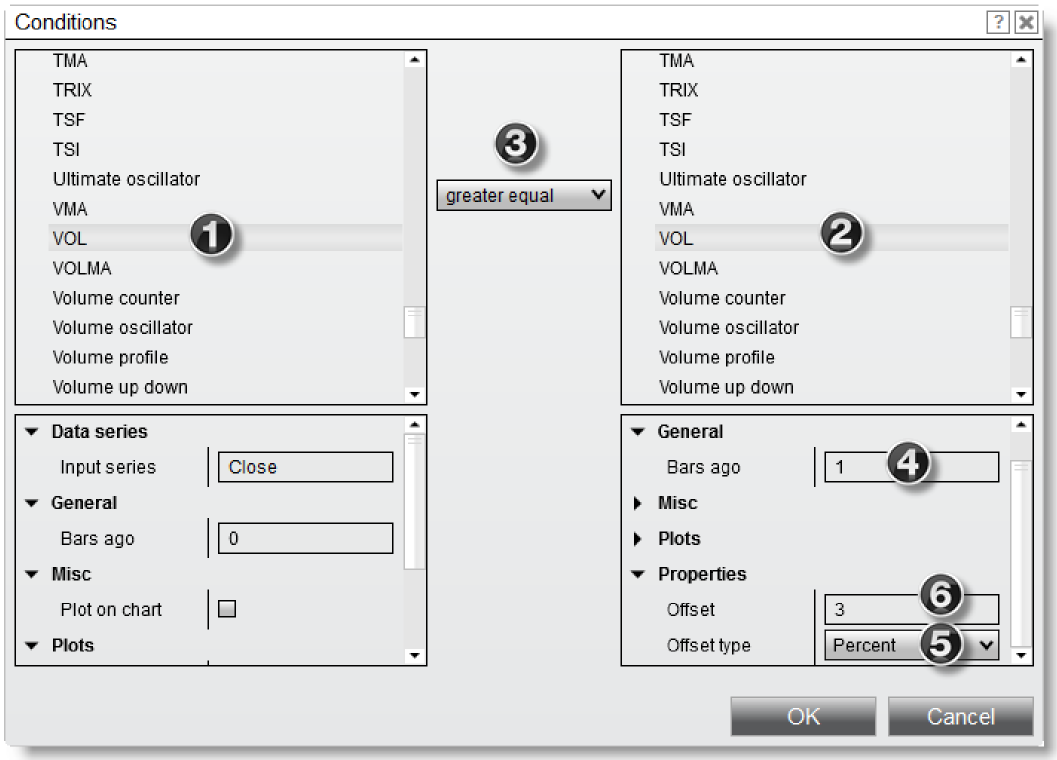
How to check for volume expansion

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*


Conditions

Left list: TMA, TRIX, TSF, TSI, Ultimate oscillator, VMA, VOL, VOLMA, Volume counter, Volume oscillator, Volume profile, Volume up down

Data series
Input series: Close
General
Bars ago: 0
Misc
Plot on chart: ☐
Plots

greater equal

Right list: TMA, TRIX, TSF, TSI, Ultimate oscillator, VMA, VOL, VOLMA, Volume counter, Volume oscillator, Volume profile, Volume up down

General
Bars ago: 1
Misc
Plots
Properties
Offset: 3
Offset type: Percent

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%"

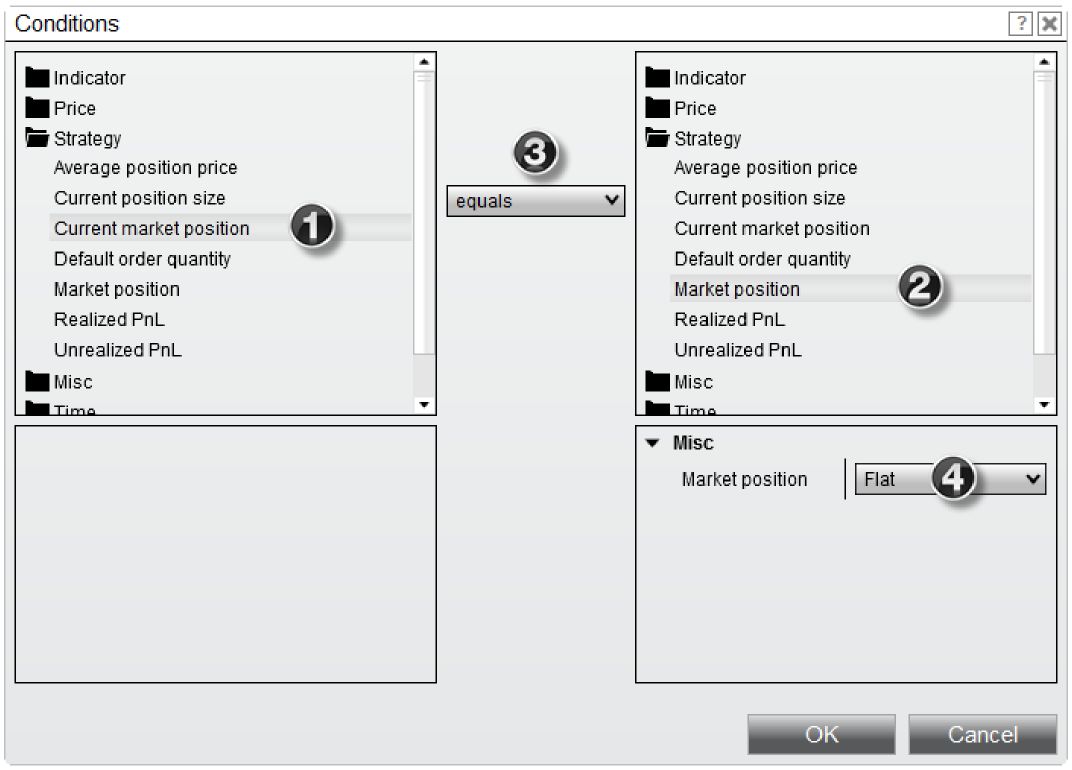
How to create market position comparisons

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 dropdown 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

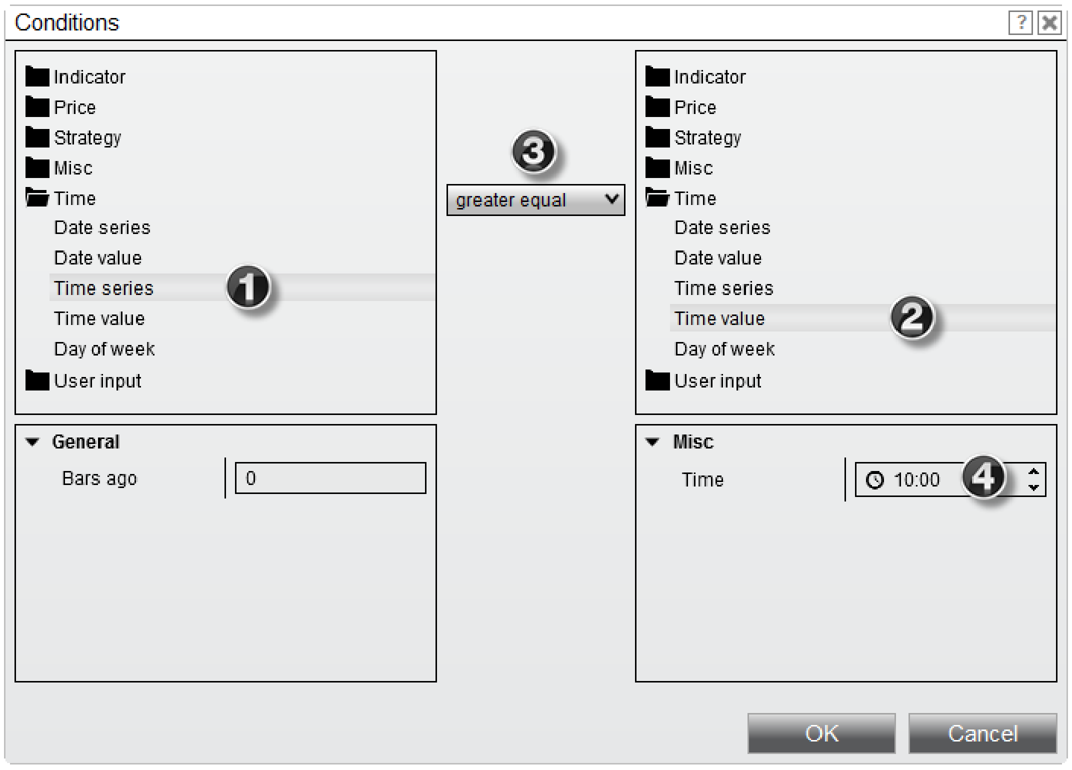
Creating Time Comparisons

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"



greater equal

General
Bars ago 0

Misc
Time 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"

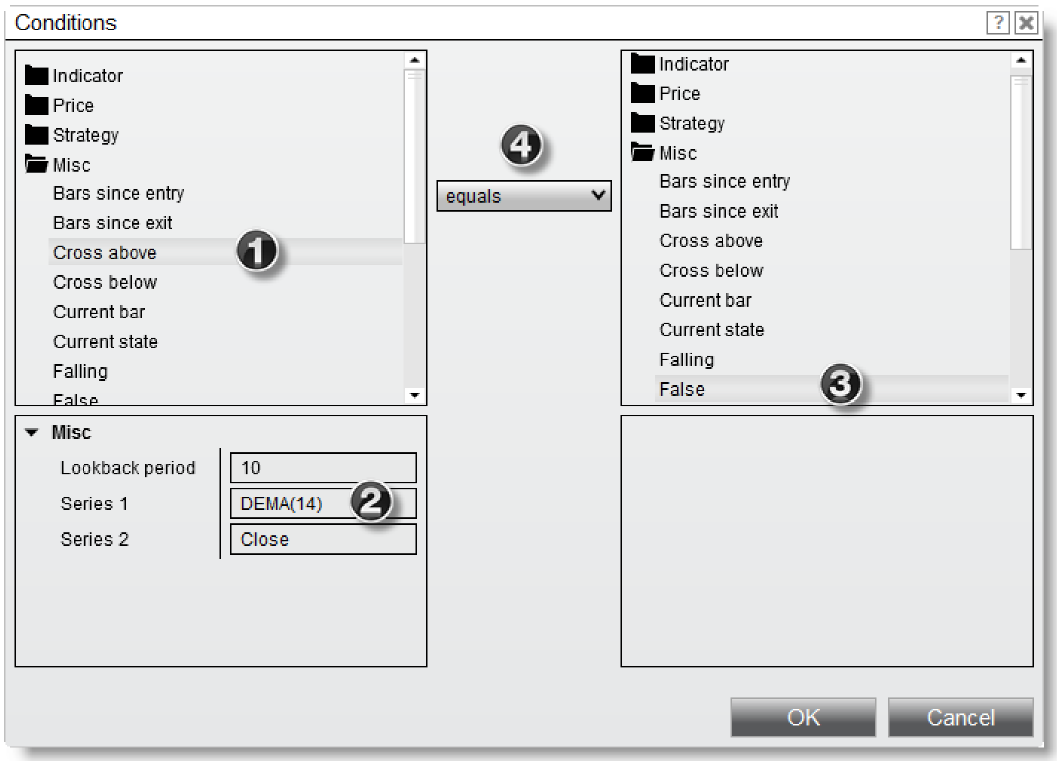
How to negate a condition

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.

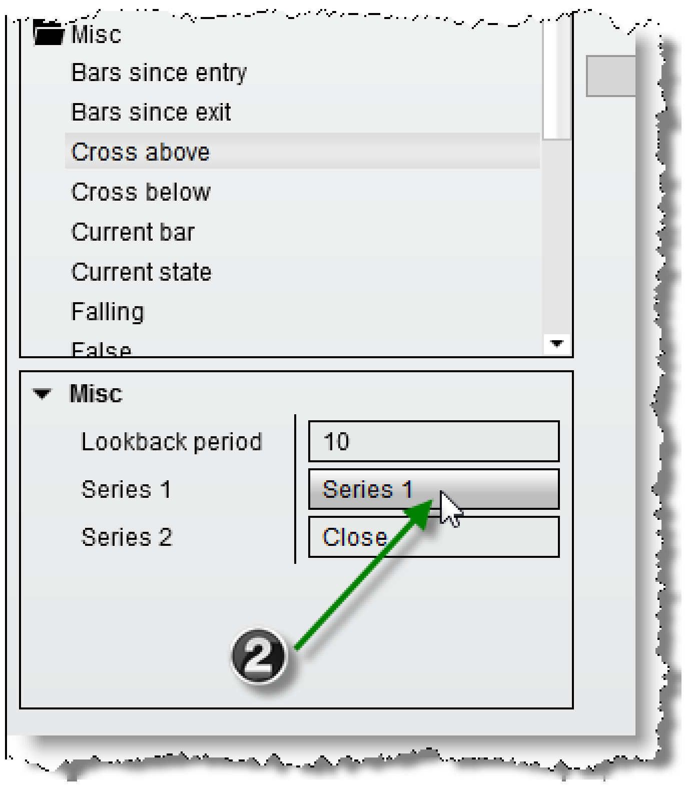
The following is an example and represents one of many possible combinations.

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



Misc

Lookback period	10
Series 1	DEMA(14)
Series 2	Close



Lookback period: 10
Series 1: Series 1
Series 2: Close

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"

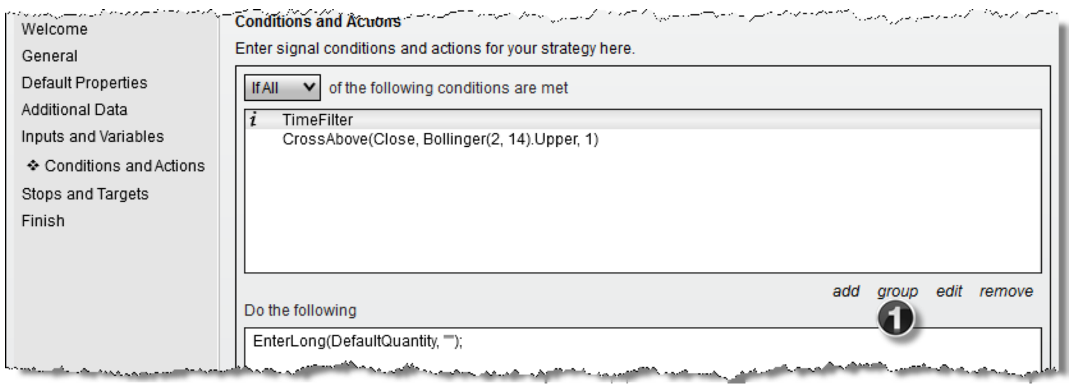
How to create time filters

Creating a Time Filter

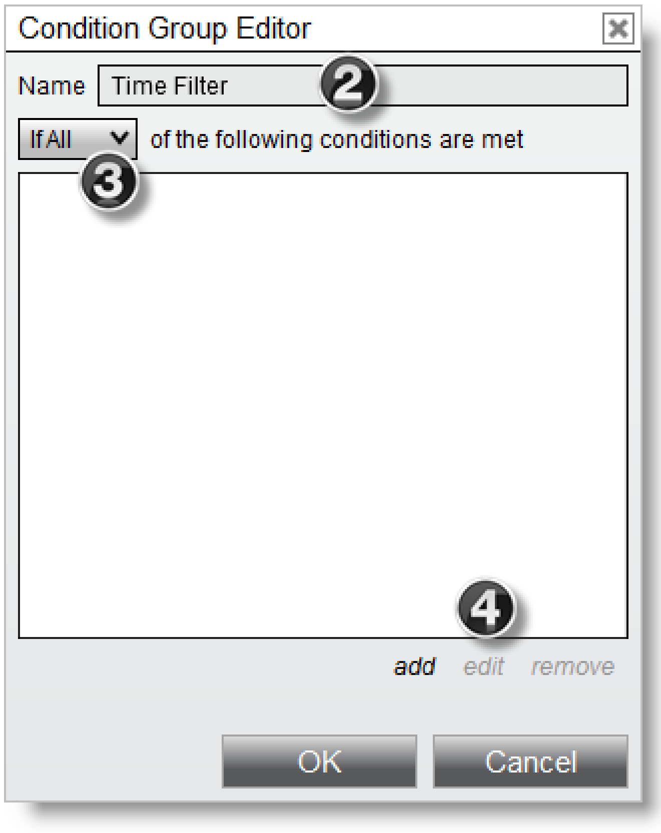
Time filters can be a useful tool of your custom strategy to help make its trades more efficient and devise a way to test for various parts of the trading session. The Condition Group Editor is ideally suited to set those up for your Strategy Builder scripts.

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

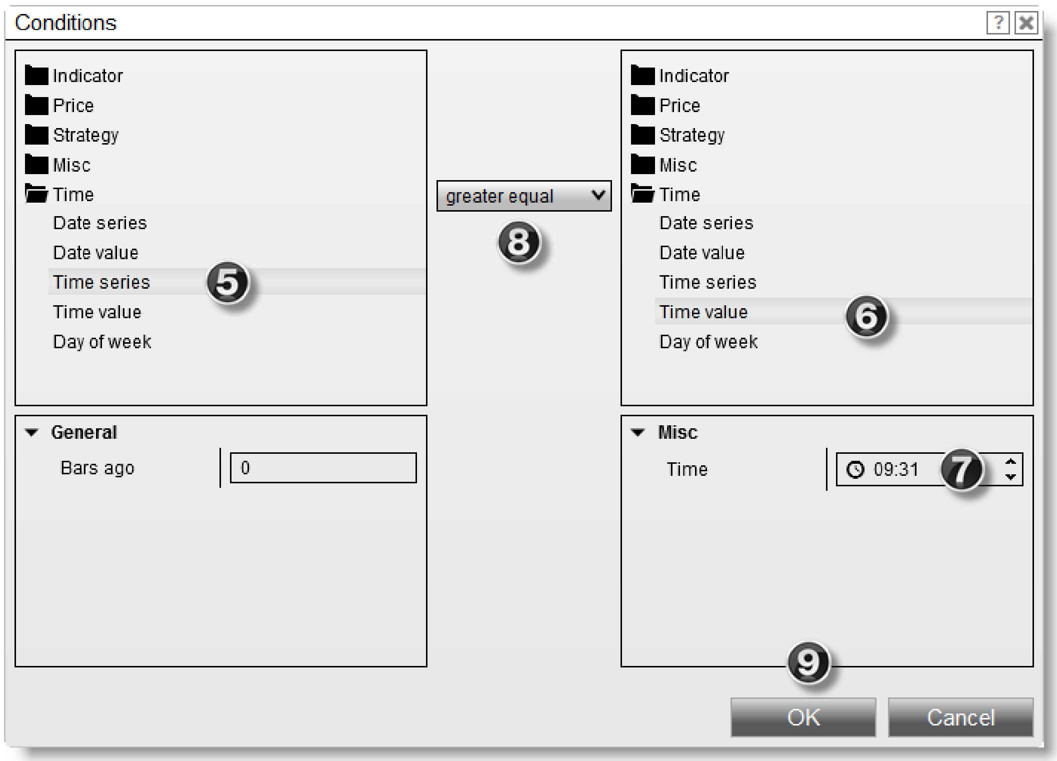


2. Optionally set a custom name for your Condition Group, i.e. Time Filter.
3. Selects **if all** of the individual conditions in the group have to be met in order to allow for a 'true' result evaluation, or **if any** will be sufficient.
4. Press **add**, **edit** or **remove** to add new condition 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**

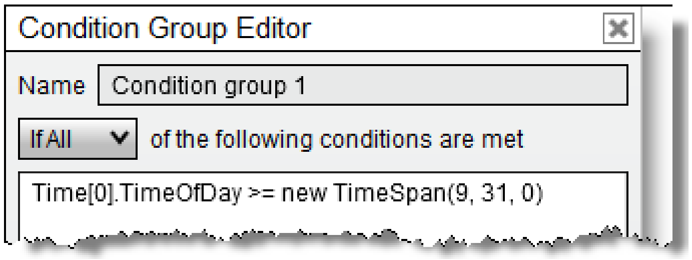
7. Enter your desired **Time** under Misc for the start of the time filter, i.e. 9:31
8. Select the **greater equal** relational operator



Bars ago: 0

Time: 09:31

9. Press the **OK** button then to return to the Condition Group Editor with your first filter condition created.

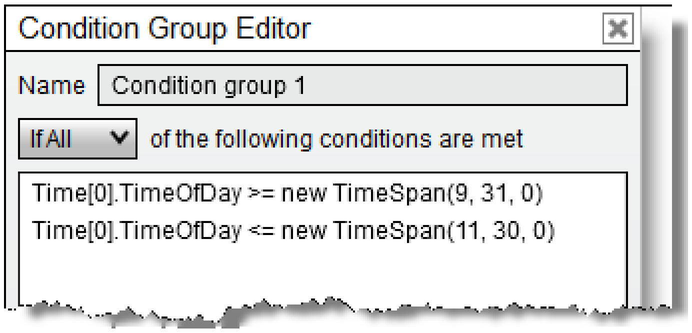


Name: Condition group 1

If All — of the following conditions are met

```
Time[0].TimeOfDay >= new TimeSpan(9, 31, 0)
```

Having setup the second, opposing condition as well the Condition Group for the time filter could look like :



Condition group 1

```
Time[0].TimeOfDay >= new TimeSpan(9, 31, 0)
Time[0].TimeOfDay <= new TimeSpan(11, 30, 0)
```

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"

11.24.3 Actions

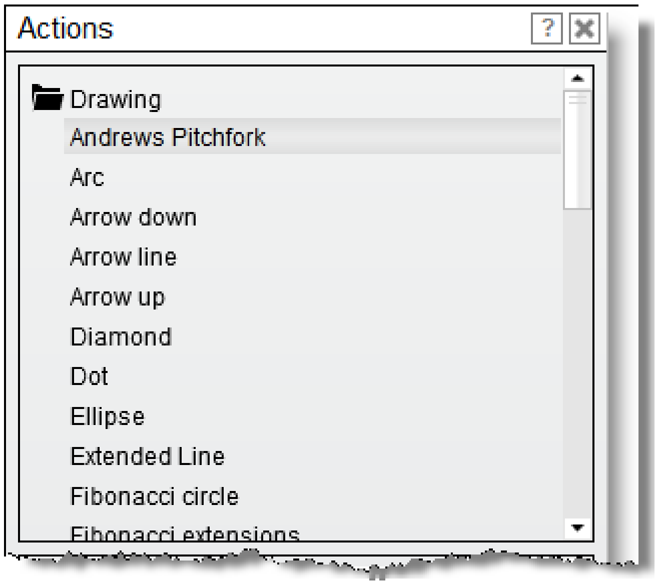
The **Actions** window allows you to select actions to execute for your script's conditions, for example executing an order or visualizing outcomes via draw objects.

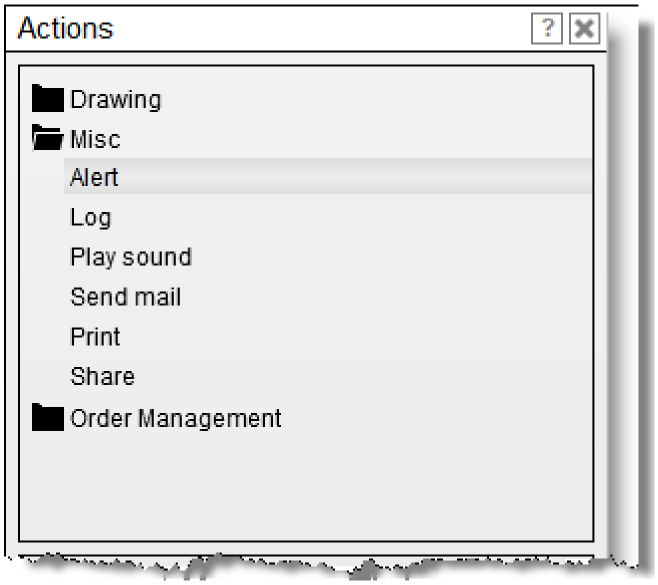
Understanding the Actions window

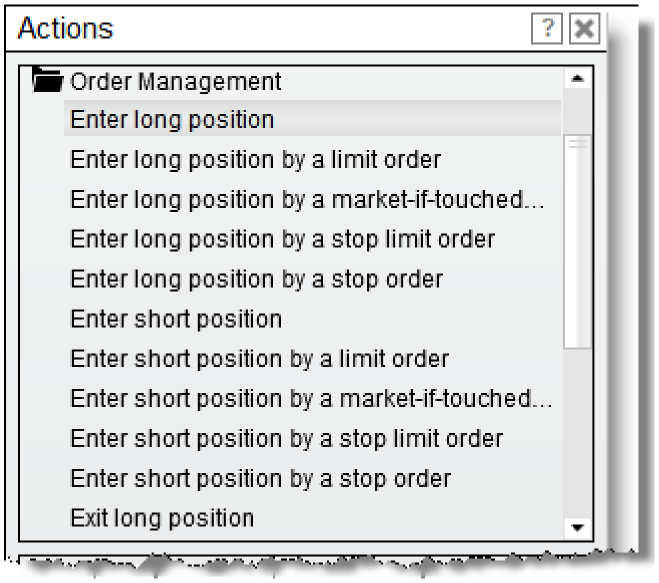
Strategy Action Window

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.


Actions
Drawing
Andrews Pitchfork
Arc
Arrow down
Arrow line
Arrow up
Diamond
Dot
Ellipse
Extended Line
Fibonacci circle
Fibonacci extensions


Actions
Drawing
Misc
Alert
Log
Play sound
Send mail
Print
Share
Order Management


Actions
Order Management
Enter long position
Enter long position by a limit order
Enter long position by a market-if-touched...
Enter long position by a stop limit order
Enter long position by a stop order
Enter short position
Enter short position by a limit order
Enter short position by a market-if-touched...
Enter short position by a stop limit order
Enter short position by a stop order
Exit long position

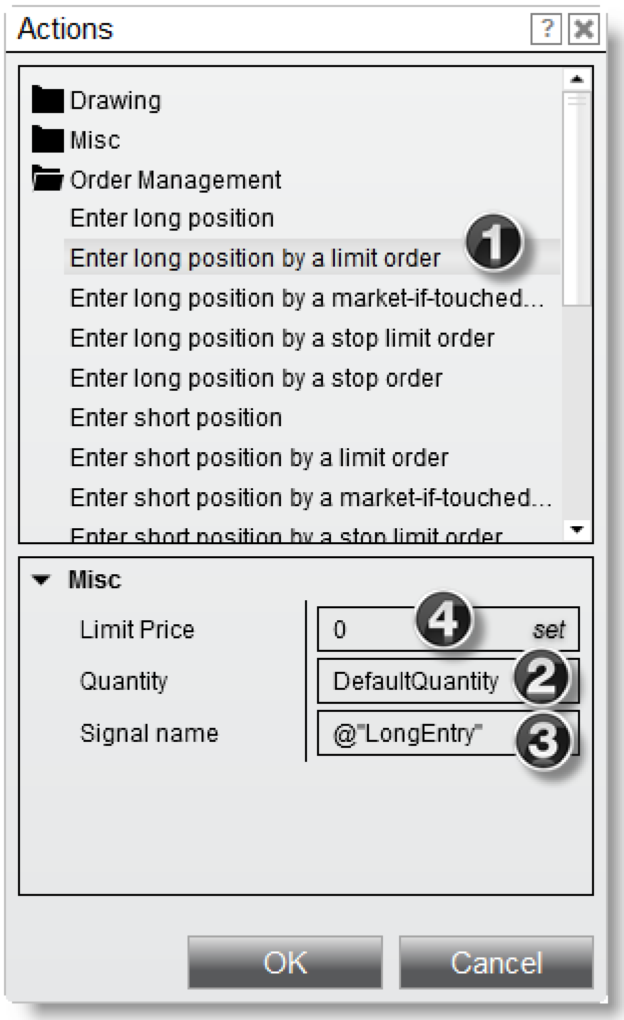
How to enter a market position

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.

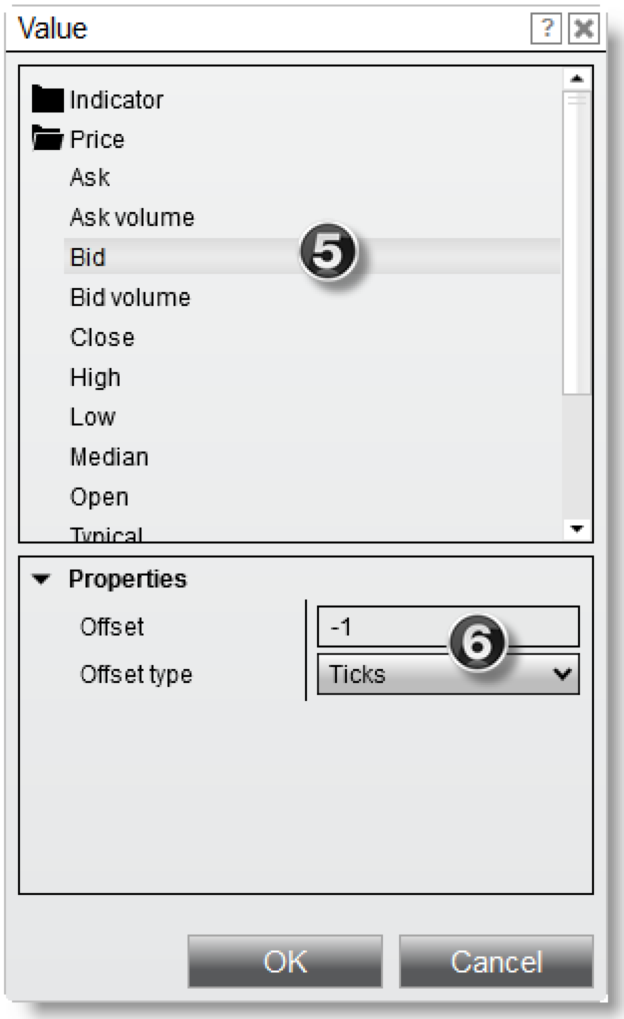
1. Expand the **Order management** category and select **Enter a long position by a limit order**
2. You can optionally set the number of contracts / shares for the order or leave the **DefaultQuantity** value which allows you set the quantity when starting a strategy
3. Set the ***Signal name** property 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


Actions
Drawing
Misc
Order Management
Enter long position
Enter long position by a limit order
1
Enter long position by a market-if-touched...
Enter long position by a stop limit order
Enter long position by a stop order
Enter short position
Enter short position by a limit order
Enter short position by a market-if-touched...
Enter short position by a stop limit order
Misc
4
2
3
OK
Cancel

Limit Price	0 set
Quantity	DefaultQuantity
Signal name	@"LongEntry"

***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)



Offset: -1
Offset type: Ticks

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"

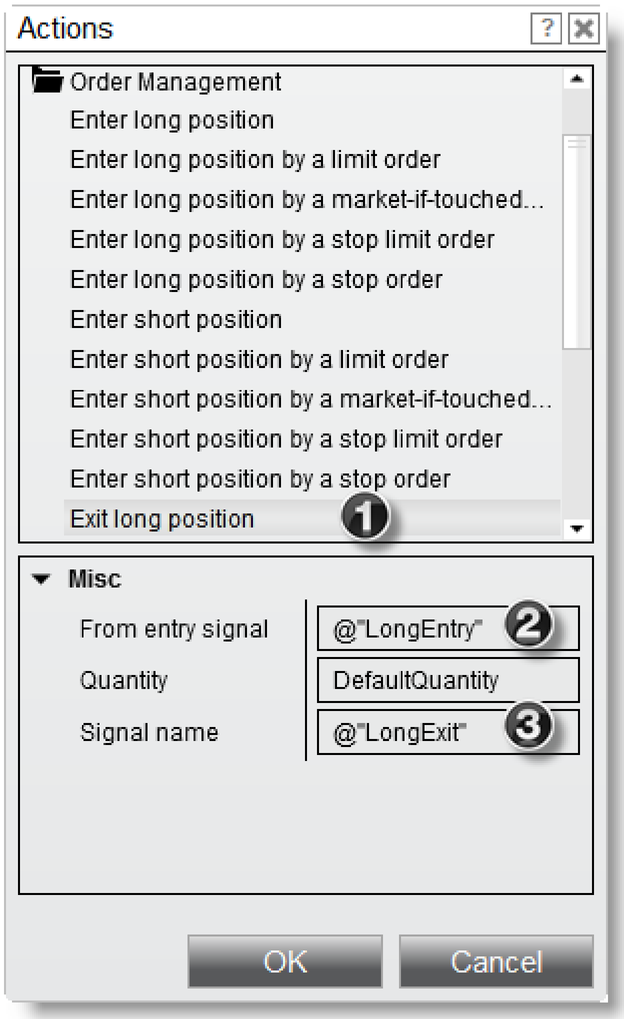
How to exit a market position

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)


Actions
Order Management
Enter long position
Enter long position by a limit order
Enter long position by a market-if-touched...
Enter long position by a stop limit order
Enter long position by a stop order
Enter short position
Enter short position by a limit order
Enter short position by a market-if-touched...
Enter short position by a stop limit order
Enter short position by a stop order
Exit long position
1
Misc
2
3
OK
Cancel

From entry signal: @"LongEntry"
Quantity: DefaultQuantity
Signal name: @"LongExit"

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'."

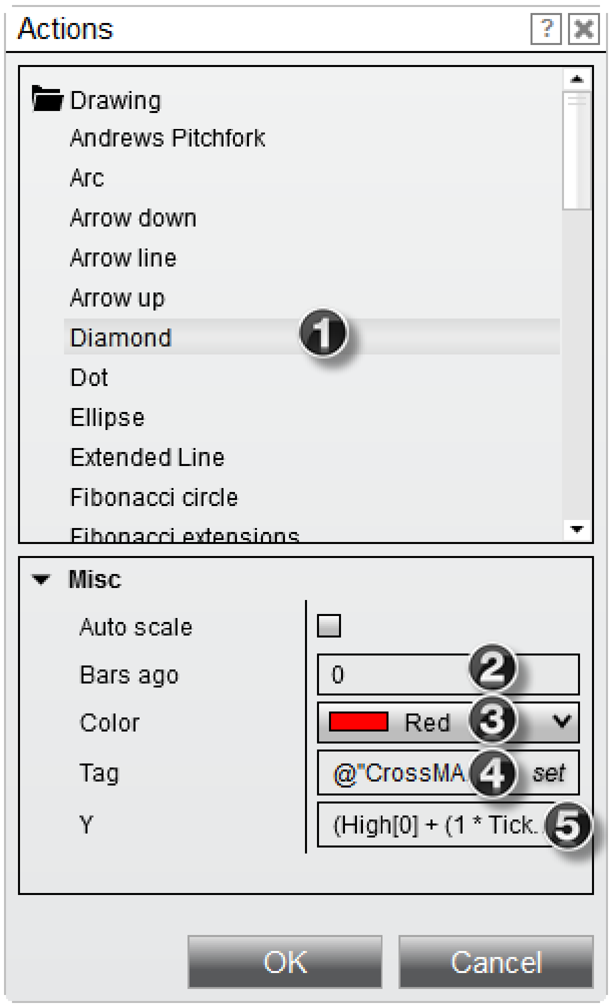
▽ How to draw on a chart

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

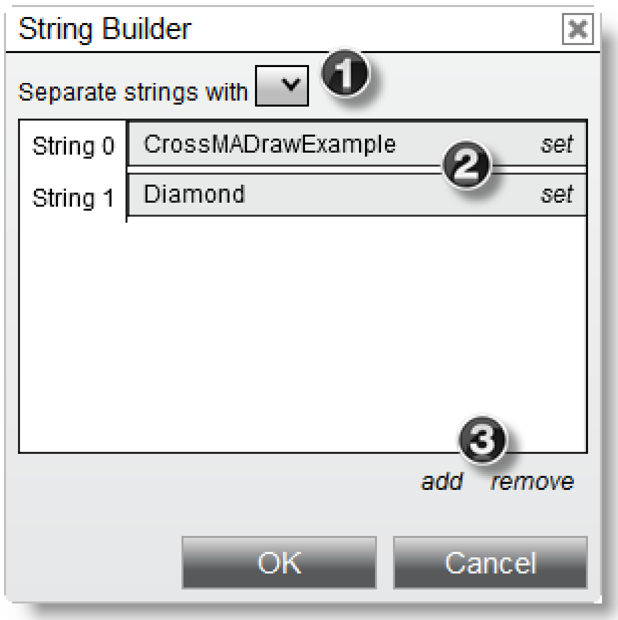


Auto scale	☐
Bars ago	0
Color	🟥 Red
Tag	@"CrossMA set
Y	(High[0] + (1 * Tick.

Once the **OK** button is pressed, an action is created that would translate to the following:

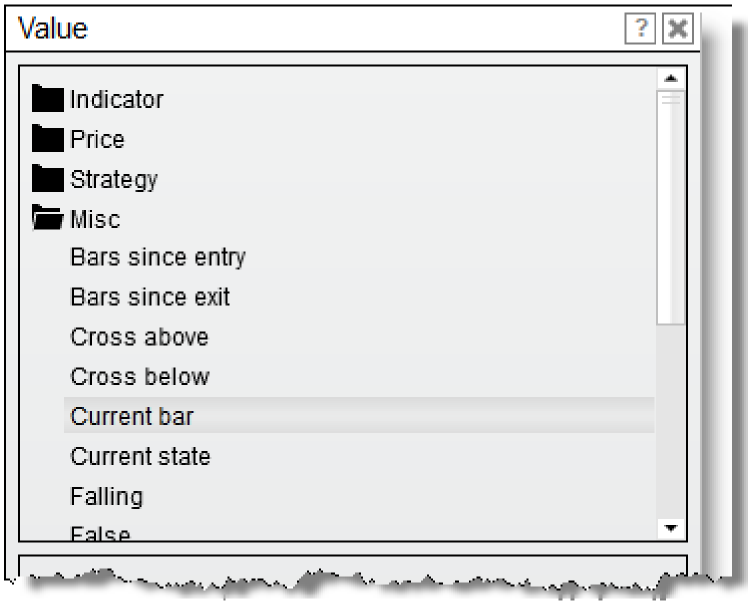
"Draw a red diamond above the high of the current bar plus one tick"

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.



11.25 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

11.25.1 Using the Time & Sales Window

Play Video

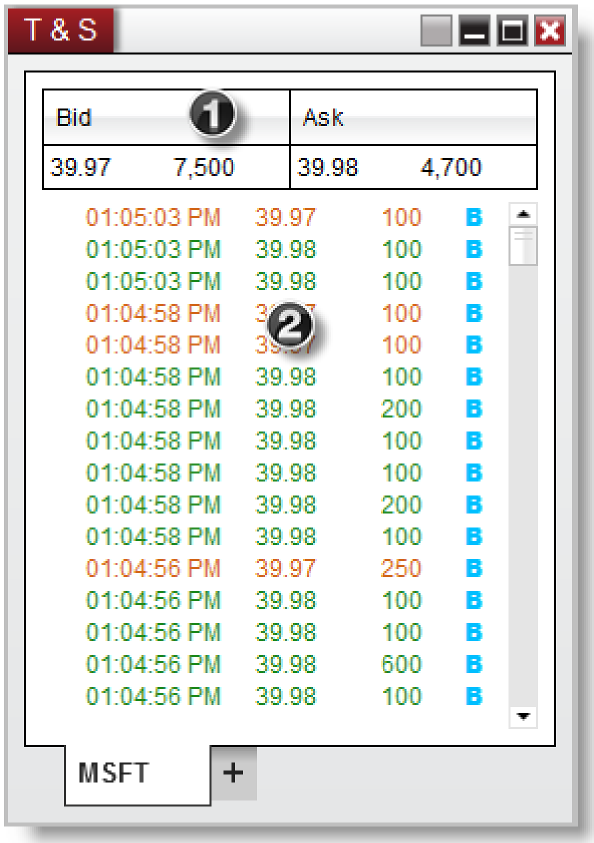
Selecting an Instrument

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



Bid		Ask	
39.97	7,500	39.98	4,700

```
01:05:03 PM   39.97   100   B
01:05:03 PM   39.98   100   B
01:05:03 PM   39.98   100   B
01:04:58 PM   39.97   100   B
01:04:58 PM   39.97   100   B
01:04:58 PM   39.98   100   B
01:04:58 PM   39.98   200   B
01:04:58 PM   39.98   100   B
01:04:58 PM   39.98   100   B
01:04:58 PM   39.98   200   B
01:04:58 PM   39.98   100   B
01:04:56 PM   39.97   250   B
01:04:56 PM   39.98   100   B
01:04:56 PM   39.98   100   B
01:04:56 PM   39.98   600   B
01:04:56 PM   39.98   100   B
```

1 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**.

❷ Time & Sales Grid

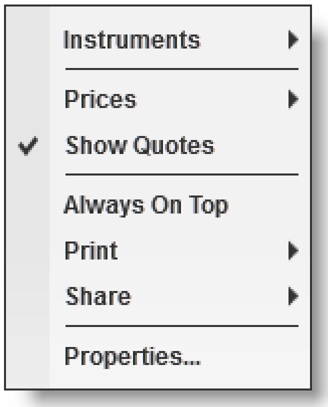
The **Time & Sales Grid** displays Bid, Ask, and Last data.

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 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...**

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.

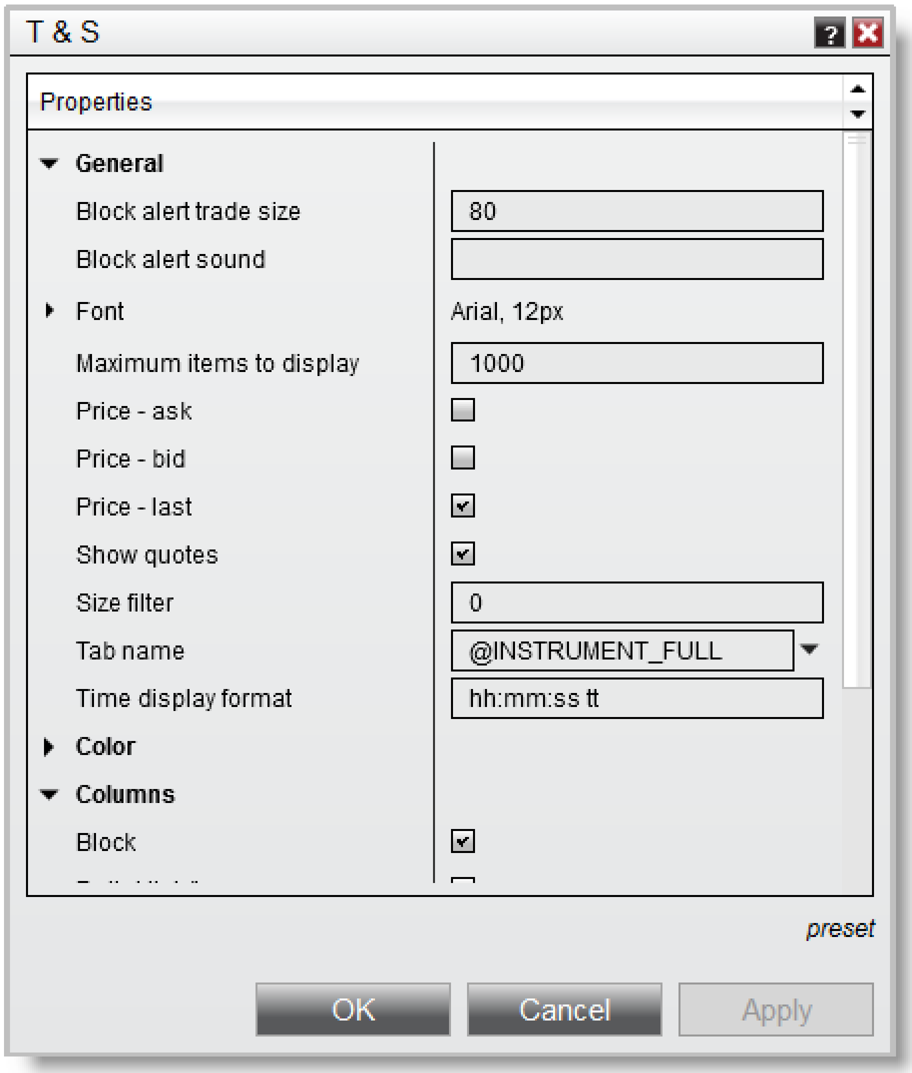
11.25.2 Time & Sales Properties

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



General
Block alert trade size: 80
Block alert sound:
Font: Arial, 12px
Maximum items to display: 1000
Price - ask: ☐
Price - bid: ☐
Price - last: ✓
Show quotes: ✓
Size filter: 0
Tab name: @INSTRUMENT_FULL
Time display format: hh:mm:ss tt
Color
Columns
Block: ✓

Property Definitions

General	
Block alert trade size	Sets a value indicating the minimum trade size required to register a block trader alert
Block alert sound	Select a sound file to play when the block alert is triggered

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.

	<table><tr><td>fff</td><td>The milliseconds in a date and time value.</td></tr><tr><td>hh</td><td>The hour, using a 12-hour clock from 01 to 12.</td></tr><tr><td>HH</td><td>The hour, using a 24-hour clock from 00 to 23.</td></tr><tr><td>mm</td><td>The minute, from 00 through 59.</td></tr><tr><td>s</td><td>The second, from 0 through 59.</td></tr><tr><td>ss</td><td>The second, from 00 through 59.</td></tr><tr><td>tt</td><td>The AM/PM designator.</td></tr></table>
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 backgrou	Sets the back color for trades at the ask price

nd	
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 foreground	Sets the text color for trades between the bid and ask price
Bid background	Sets the back color for bid data
Bid foreground	Sets the text color for bid data
Block	Sets the block icon color.

alert	
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.

11.25.3 Window Linking

Please see the Window Linking section of the Help Guide for more information on linking the **Time & Sales** window.

11.26 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

11.26.1 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**.

Play Video

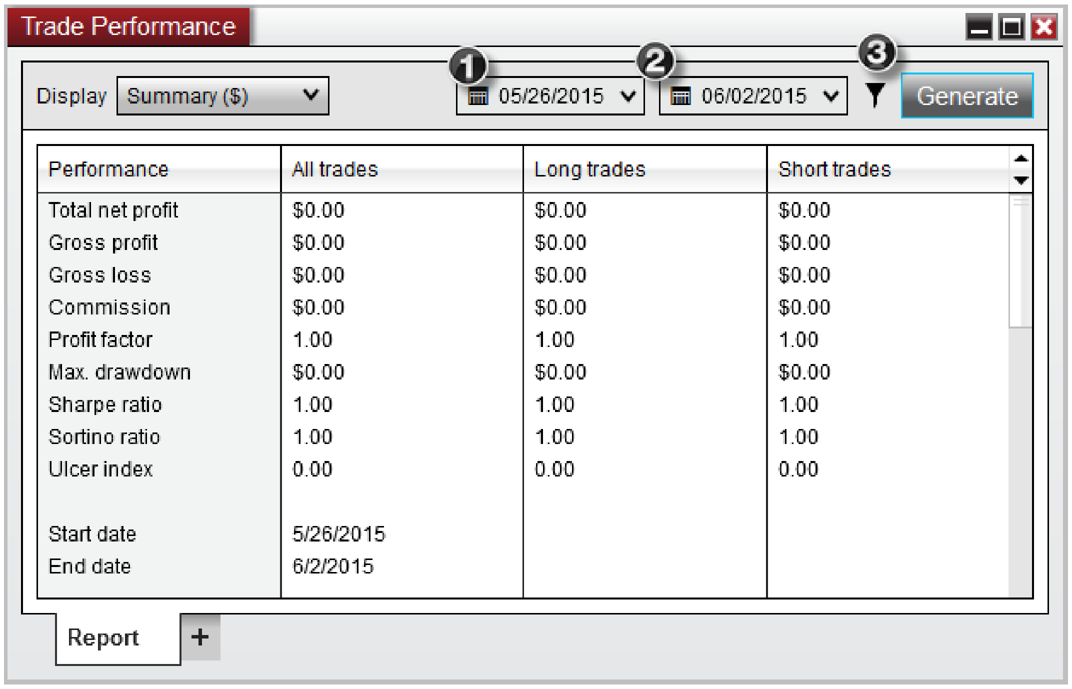
Understanding the performance report

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.



Performance	All trades	Long trades	Short trades
Total net profit	$0.00	$0.00	$0.00
Gross profit	$0.00	$0.00	$0.00
Gross loss	$0.00	$0.00	$0.00
Commission	$0.00	$0.00	$0.00
Profit factor	1.00	1.00	1.00
Max. drawdown	$0.00	$0.00	$0.00
Sharpe ratio	1.00	1.00	1.00
Sortino ratio	1.00	1.00	1.00
Ulcer index	0.00	0.00	0.00
Start date	5/26/2015		
End date	6/2/2015		

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

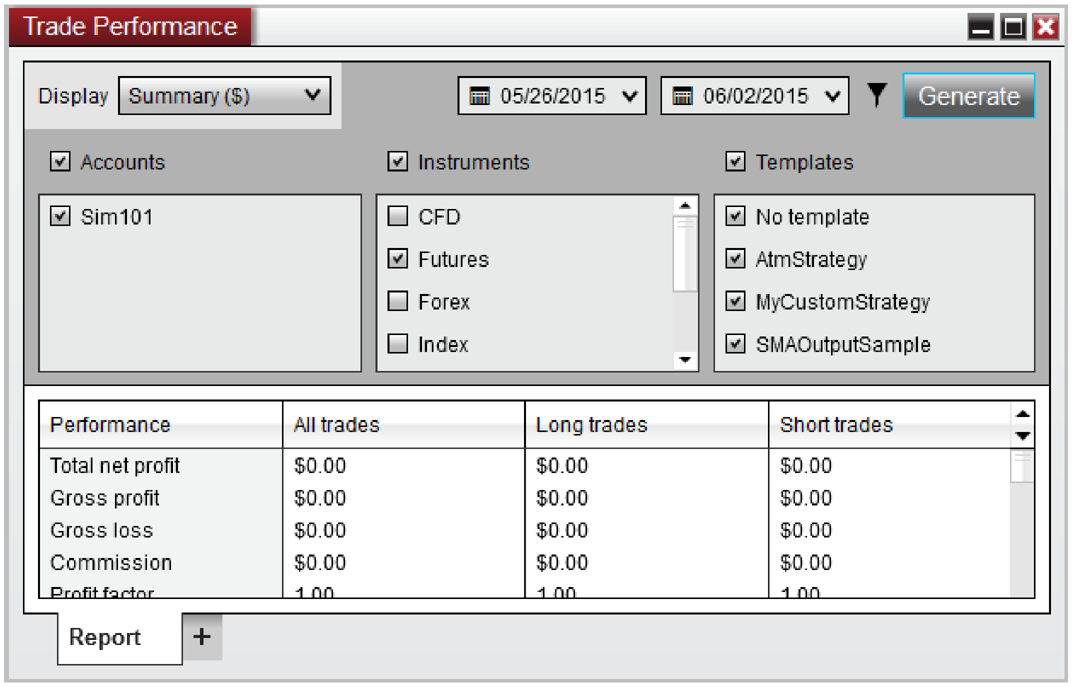
Note: As **Forex** trade quantities are typically in multiples of 1,000, 10K, and 100K a NinjaTrader feature is to divide (normalize) your **Forex** trades by your account lot size. Account lot size is recorded per execution and is set by the connection. A micro **Forex** account is 1,000, a mini **Forex** 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.



Performance	All trades	Long trades	Short trades
Total net profit	$0.00	$0.00	$0.00
Gross profit	$0.00	$0.00	$0.00
Gross loss	$0.00	$0.00	$0.00
Commission	$0.00	$0.00	$0.00
Profit factor	1.00	1.00	1.00

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.

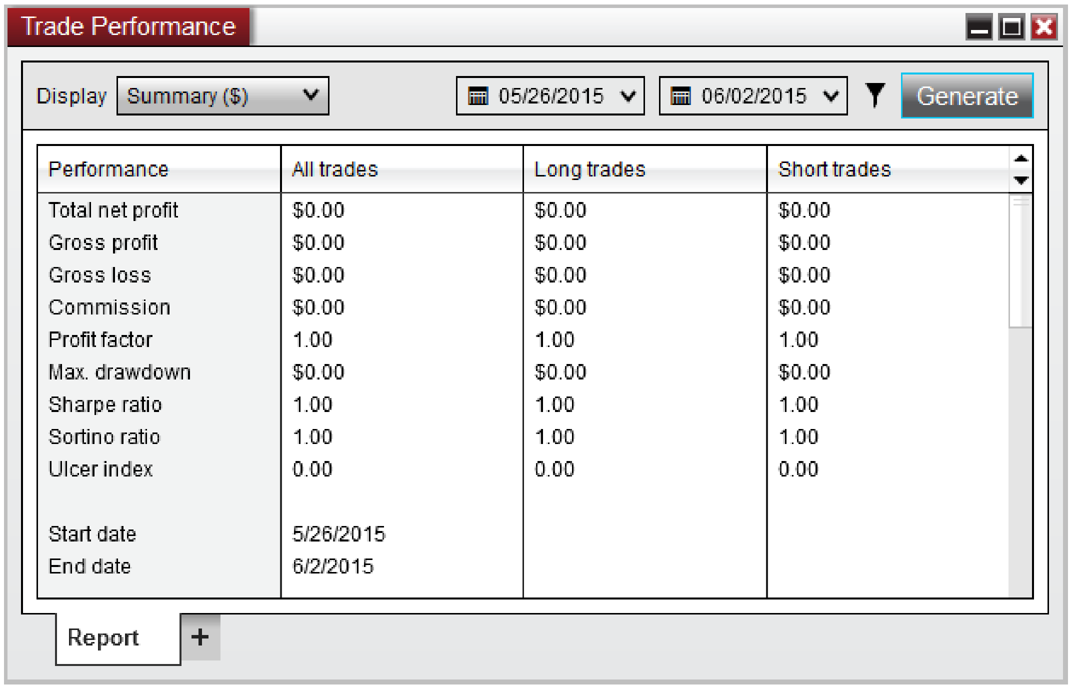
11.26.2 Performance Displays

The **Account Performance** window displays performance data in a variety of ways.

Understanding the Summary display

Summary Display

Displays all performance statistics and metrics.



Performance	All trades	Long trades	Short trades
Total net profit	$0.00	$0.00	$0.00
Gross profit	$0.00	$0.00	$0.00
Gross loss	$0.00	$0.00	$0.00
Commission	$0.00	$0.00	$0.00
Profit factor	1.00	1.00	1.00
Max. drawdown	$0.00	$0.00	$0.00
Sharpe ratio	1.00	1.00	1.00
Sortino ratio	1.00	1.00	1.00
Ulcer index	0.00	0.00	0.00
Start date	5/26/2015		
End date	6/2/2015		

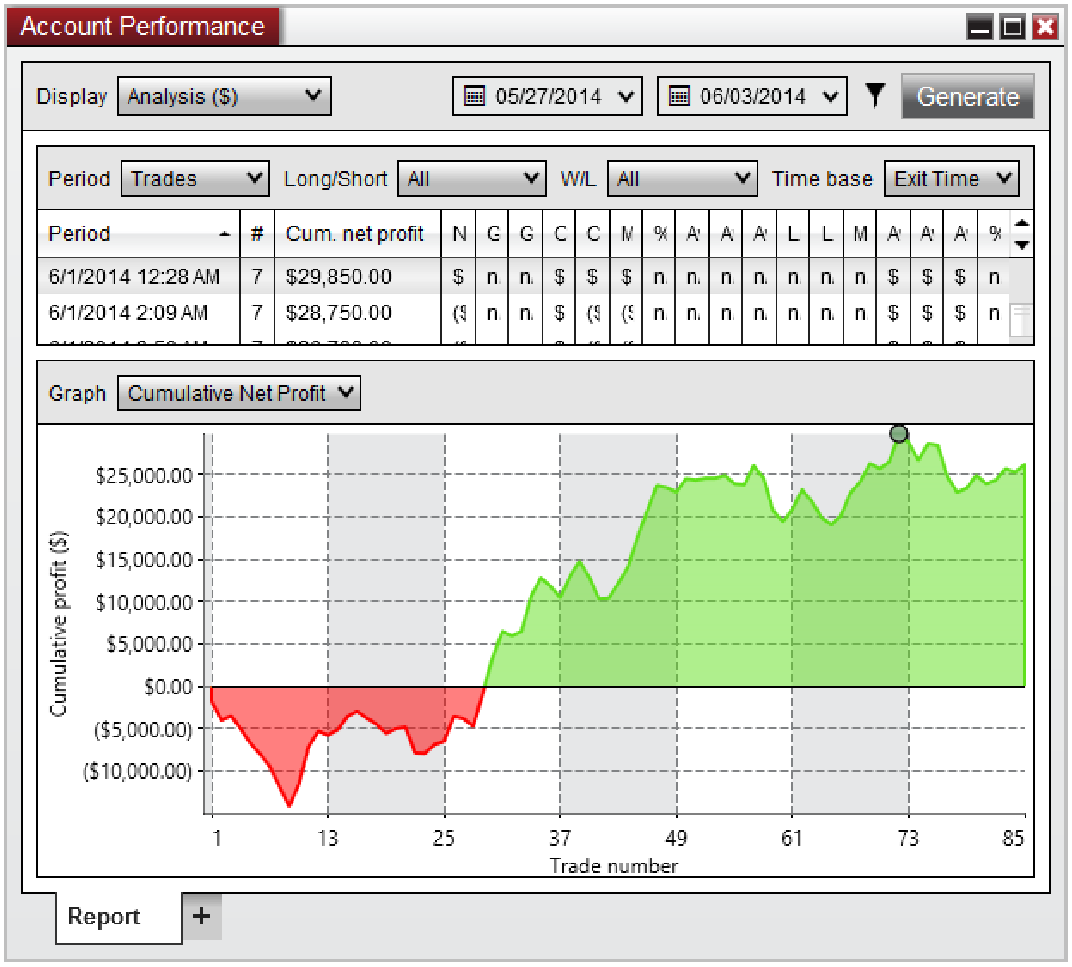
Please see the Statistic Definition section of the help guide for details on how each statistic is calculated.

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.



Display: Analysis ($) | 05/27/2014 | 06/03/2014

Period: Trades | Long/Short: All | W/L: All | Time base: Exit Time

Period	#	Cum. net profit	N	G	G	C	C	M	%	A	A	A	L	L	M	A	A	A	%
6/1/2014 12:28 AM	7	$29,850.00	$	n	n	$	$	$	n	n	n	n	n	n	n	$	$	$	n
6/1/2014 2:09 AM	7	$28,750.00	($	n	n	$	($	($	n	n	n	n	n	n	n	$	$	$	n

Graph: Cumulative Net Profit

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.

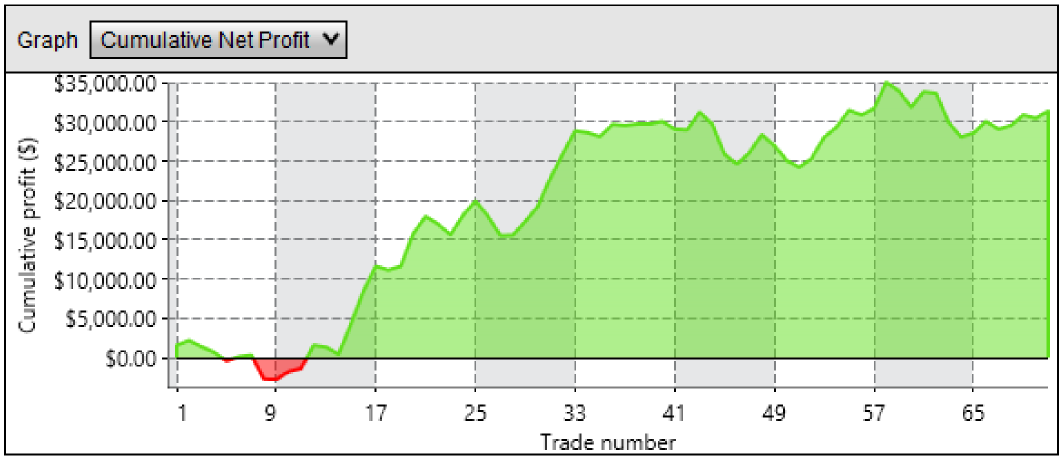
Period	Sets the periodicity you want the trade data displayed in, this also drives the graph below. **Available Period Selections** • Daily • Weekly • Monthly

	• 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

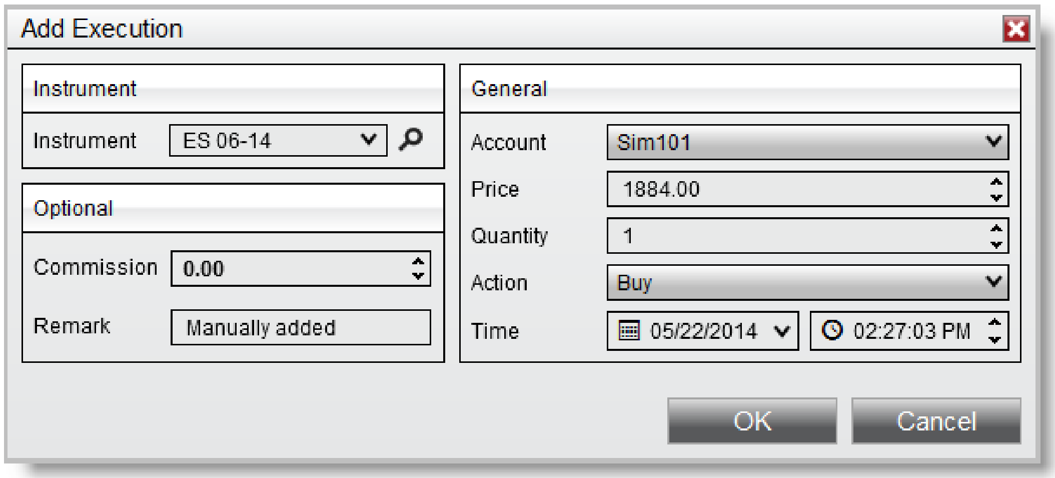
Adding Executions

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.



Instrument	
Instrument	ES 06-14
Optional	
Commission	0.00
Remark	Manually added
General	
Account	Sim101
Price	1884.00
Quantity	1
Action	Buy
Time	05/22/2014 02:27:03 PM

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 **Note**: ATM Strategies which have not been saved as a template will not be reflected (i.e., "Custom")
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
Cumula tive net profit	Summation of all the profit earned by all your trades
Commi ssion	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**".

11.26.3 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 loss in percent) - 1) of all wining 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 wining 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.

Cur	`SQRT(Summation((cumulative currency profit -`

ren cy	`maximum realized currency profit) ^2 ) / Total # of trades)`
Per cen t	`SQRT(Summation((1 + cumulative percent profit / (1 + maximum realized percent profit) - 1) ^2 ) / Total # of trades)`
Poi nts	`SQRT(Summation((cumulative point profit - maximum realized point profit) ^2 ) / Total # of trades)`
Pip s	`SQRT((Summation((cumulative point profit - maximum realized point profit) ^2) / PipSize ) / Total # of trades)`
Tic ks	`SQRT((Summation((cumulative point profit - maximum realized point profit) ^2) / TickSize ) / Total # of trades)`

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`
Percent	`SUM(profit * quantity / entry price) of all trades / # of traded lots`
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 / # of months`
Percent	`(1 + cumulative profit)(1 / # of months) - 1`
Points	`cumulative profit / # of months`
Pips	`cumulative profit / # of months`
Ticks	`cumulative profit / # of months`

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 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 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 these 3 trades on a 10K account - 5%, 10%, 7.5% profit or in currency terms you made 500$, 1000$, 750$. The first trade added to the account gives us 10500$, adding the second trade in gives us 10550 (first one of 500$ plus the 10% second trade of 10500$ as base). The third trade of 7.5% of 11550$ gives us 866.25$, so a total of 2386.25$ if we reinvested. Or in % terms we made 23.8625% - while, if we had only summed the individual trades

% profits without reinvesting, we would have had 22.5% profit (5% + 10% + 7.5%)

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

11.26.4 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 * 0.1 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

11.26.5 Trade Performance Properties

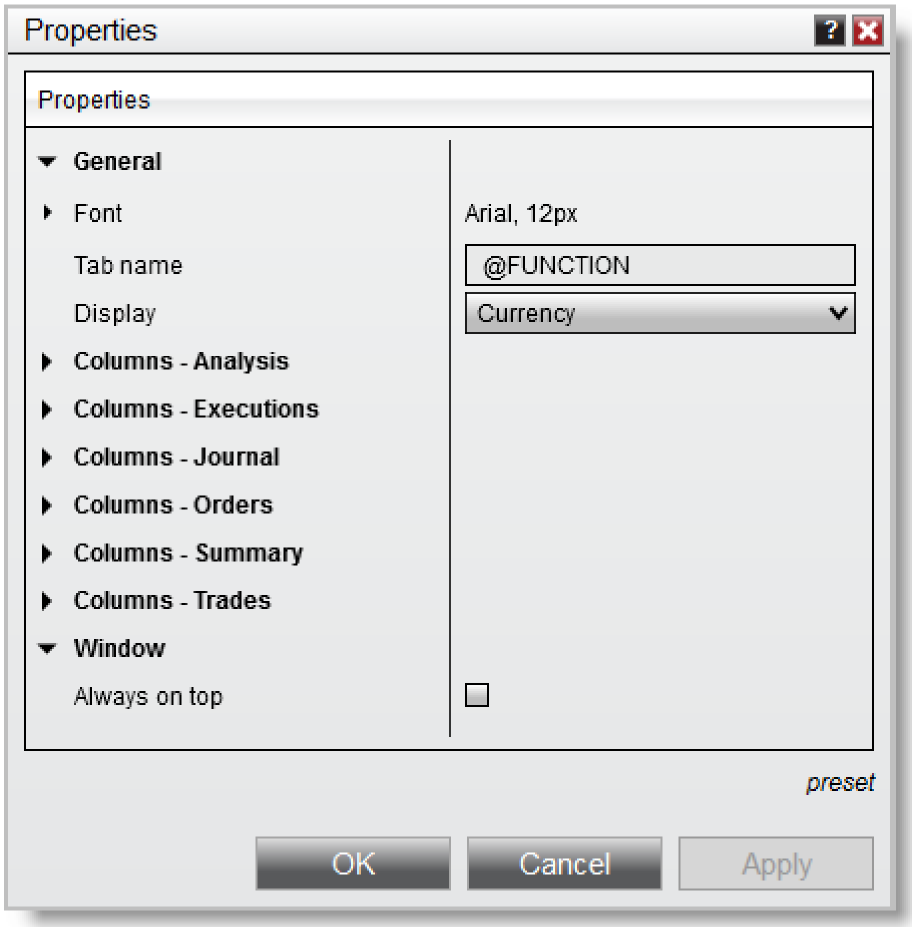
Many of the **Trade Performance** visual display settings can be customized using the **Trade Performance Properties** window.

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:



Font: Arial, 12px
Tab name: @FUNCTION
Display: Currency
Always on top: ☐

Property Definitions

General	
Display	Sets the currency display
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.

11.27 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

11.27.1 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 the Market Analyzer Properties window
- 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**:

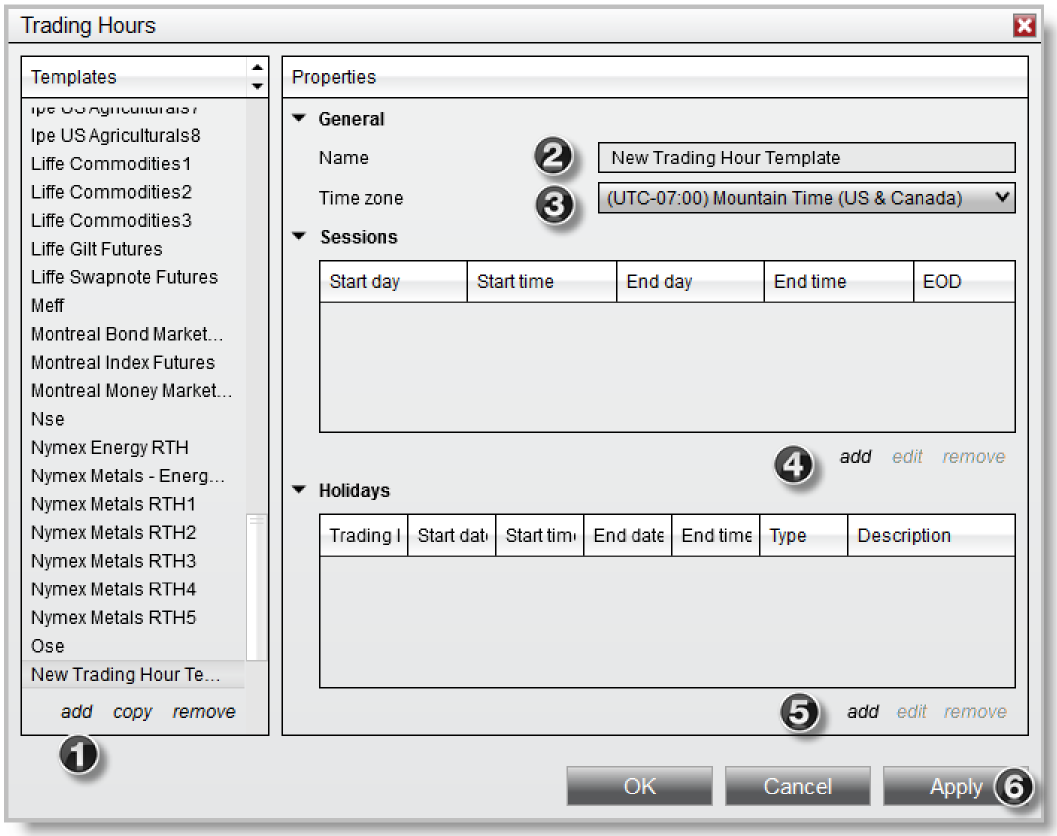
1. Left mouse click on "add"
2. Type in the name of the **Trading Hours Template**
3. Left mouse click on the time zone drop down menu and select the time zone that represents the time inputted in the session definitions
4. Select "add" to add a new session definition, see "**Understanding session definitions**" 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

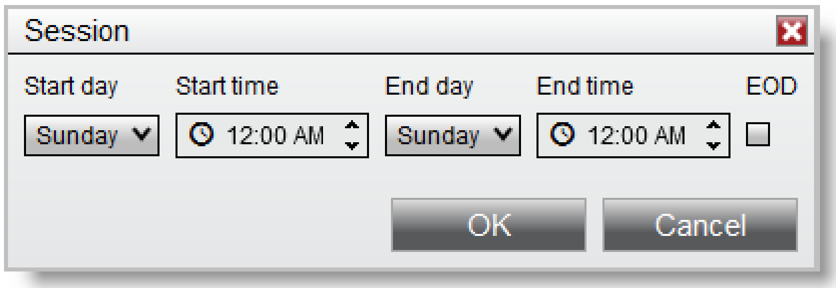
Individual holiday definitions can be edited in the following ways:

- 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.



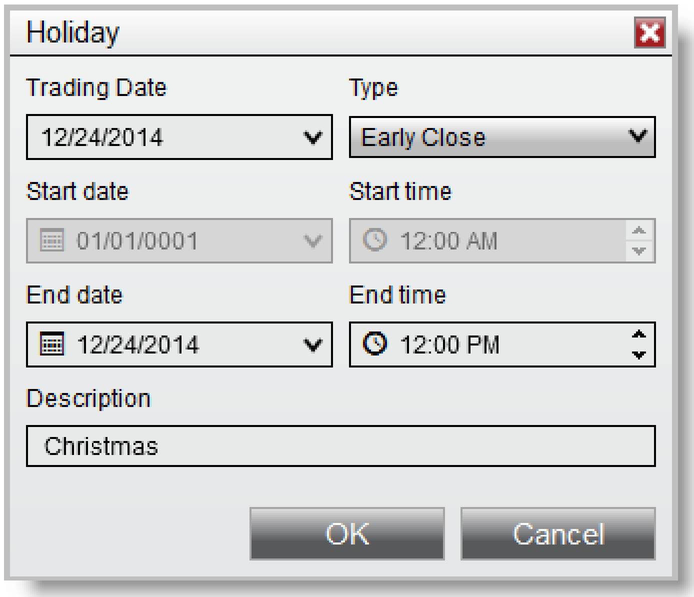
Start day	Sets the Start day of the session definition.
Start time	Sets the start time of the session definition.
End day	Sets the end day of the session definition.

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**.



Trading Date: 12/24/2014
Type: Early Close
Start date: 01/01/0001
Start time: 12:00 AM
End date: 12/24/2014
End time: 12:00 PM
Description: Christmas

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.
Repla	Replace all session definitions for the **Holidays**

ce	**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.

11.28 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.

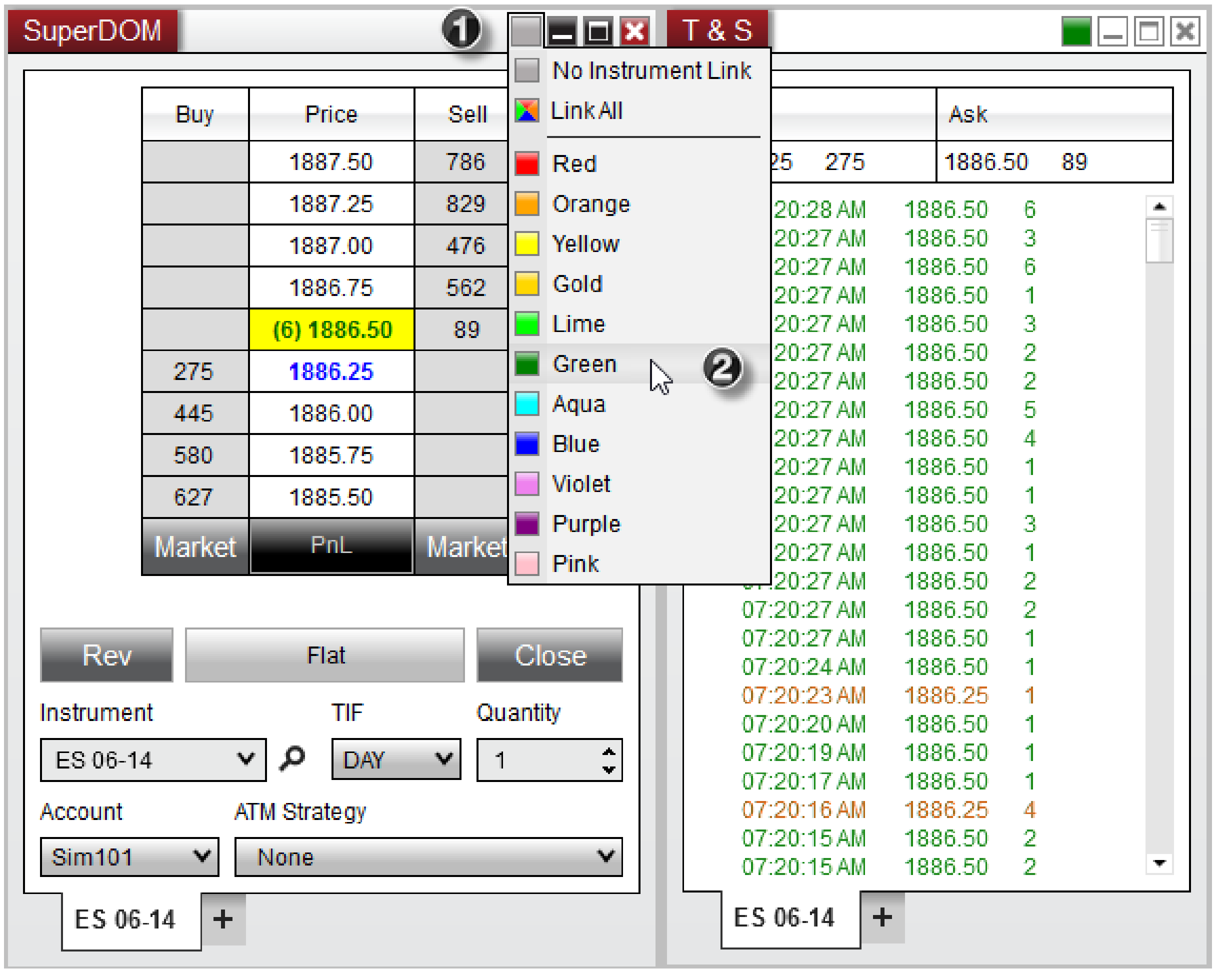
› Linking Windows
› Using the overlay instrument selector
› Using Tabs

Play Video

11.28.1 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 there interval.

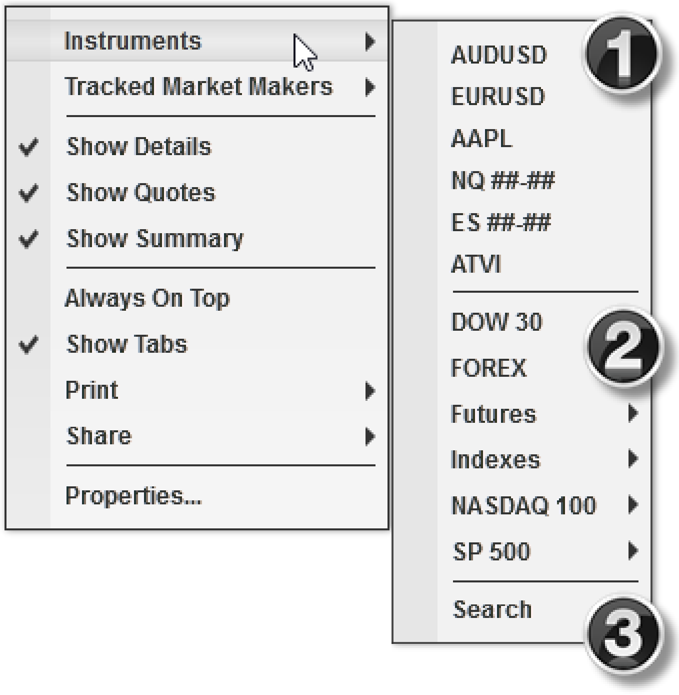
11.28.2 Using the Instrument Selector

Instrument Selector

All order-entry and market-data windows feature an **Instrument Selector**, which can be used to quickly apply any recently viewed instrument to the window, or any instrument saved in an instrument list. In addition, instruments can be pinned to the top of the **Instrument Selector**, allowing you to save any instrument you wish for quick access in any window.

Accessing Instruments and Instrument Lists

To access the **Instrument Selector**, first 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 **Instrument Selector** is separated into three sections:

1) Pinned or recently viewed instruments

2) Instrument Lists

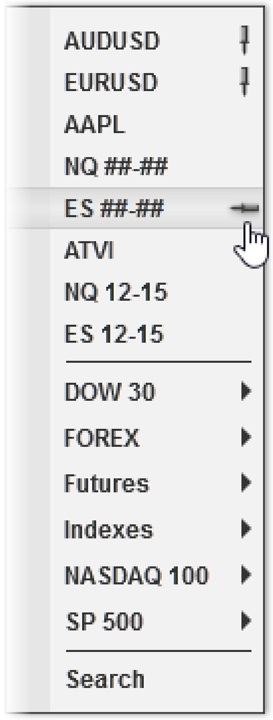
3) Search feature

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 pin an instrument in the **Instrument Selector**, 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 **Instrument Selector** 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.

11.28.3 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**.

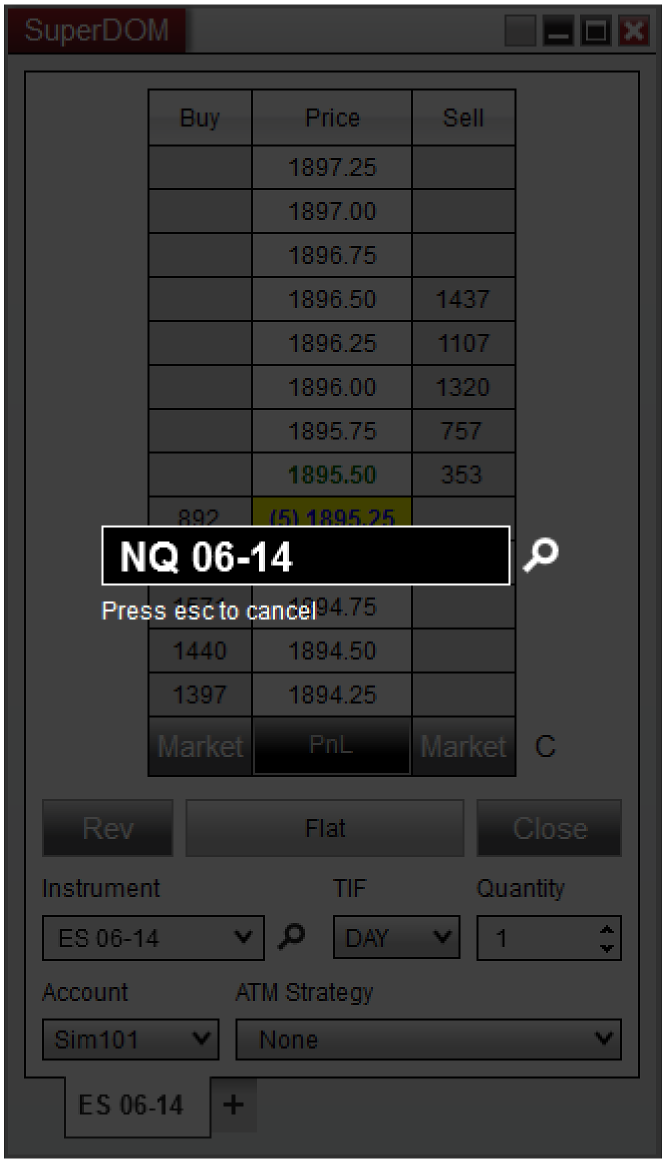
Using the overlay instrument selector

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 if you

need assistance selecting the right instrument symbol then select the magnifying class.

- 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.

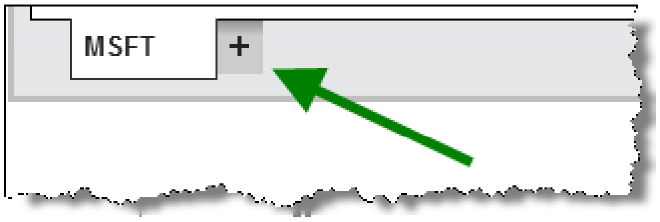
11.28.4 Using Tabs

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.

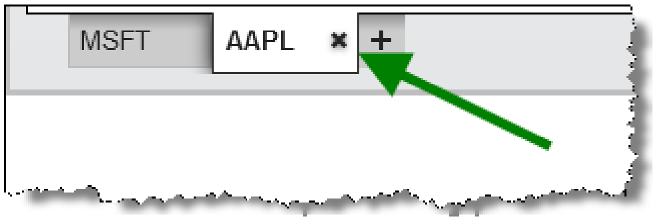


Note: With the **Time & Sales** window selected you may also start typing "++" followed by the instrument symbol into the **Overlay Instrument Selector** to quickly create a new tab with that instrument preselected. Example: "++MSFT"

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

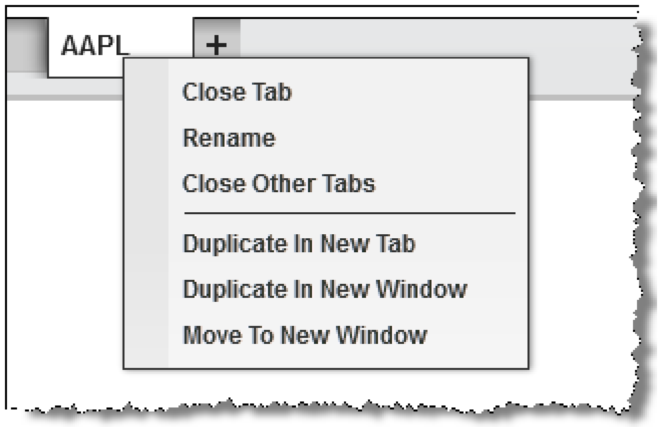
Reordering Tabs

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.

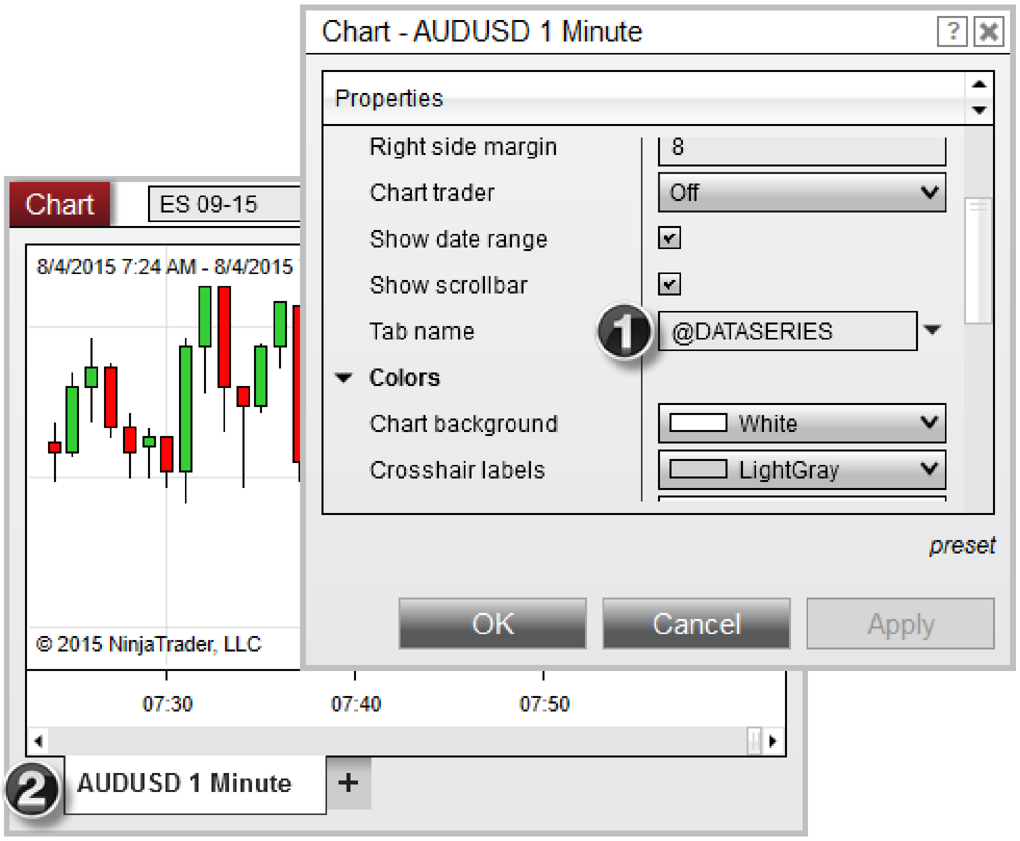


Close Tab	Removed the selected tab
Rename	Opens the properties window with the tab name property selected for direct editing
Close Other Tabs	Removed all tabs except for the selected tab
Duplicate In New Tab	Duplicates the selected tab into a new tab in the same window
Duplicate In New Window	Duplicates the selected tab into a new tab in a new window
Move To New Window	Moves the selected tab into a new window

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.



Right side margin	8
Chart trader	Off
Show date range	✓
Show scrollbar	✓
Tab name	@DATASERIES
Colors	
Chart background	White
Crosshair labels	LightGray

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
@FUNCTION	Displays the function of the tab (examples: "Chart" or "Log")	All Tabs
@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

11.28.5 Sharing Content

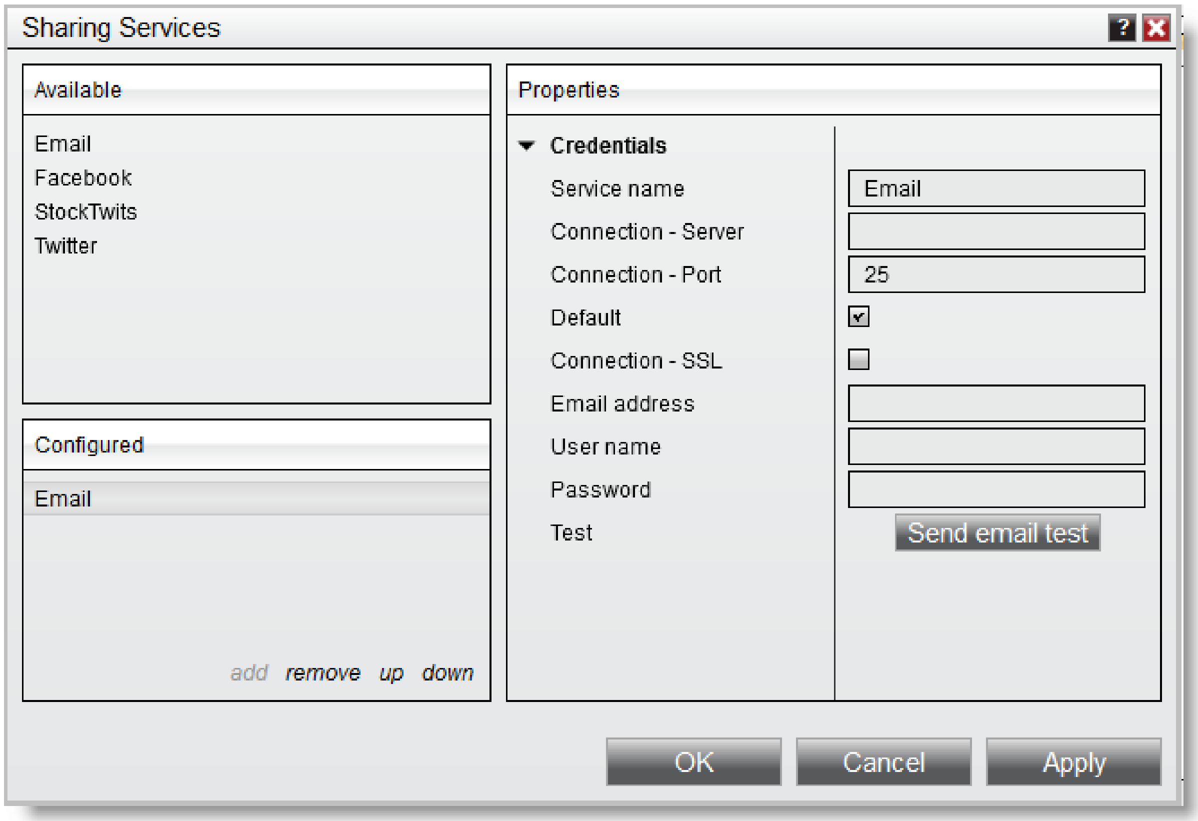
NinjaTrader support sharing messages and images to **Facebook**, **Twitter**, **StockTwits**, and via **Email**.

Play Video

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".


Sharing Services

Available

Email
Facebook
StockTwits
Twitter

Configured

Email

Properties

Credentials	
Service name	Email
Connection - Server	
Connection - Port	25
Default	☑
Connection - SSL	☐
Email address	
User name	
Password	
Test	Send email test

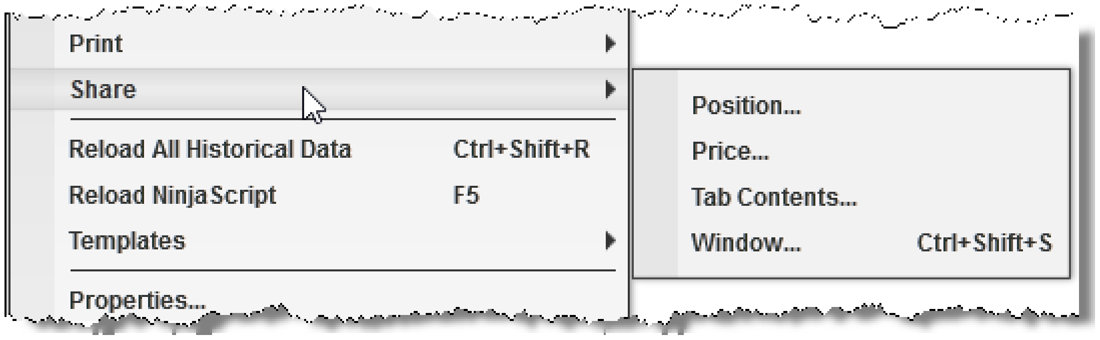
Select an available sharing service and double click or select "add" to configure that connection. You will add a name for the account and then for the **Facebook**, **StockTwits**, and **Twitter** connections select the "**Connect**" button to be walked through signing into your account.

When selecting connect a web dialog will pop up that you must follow the steps to finish the account setup.

Once the account is connected the connect button will change to "**Connected**" and 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.

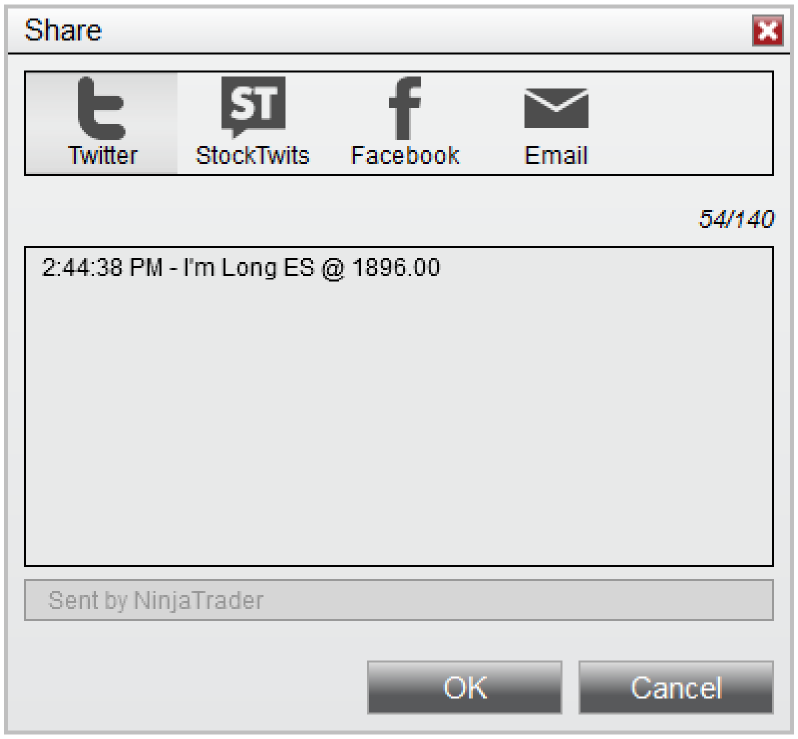
Sharing from a NinjaTrader Window



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.



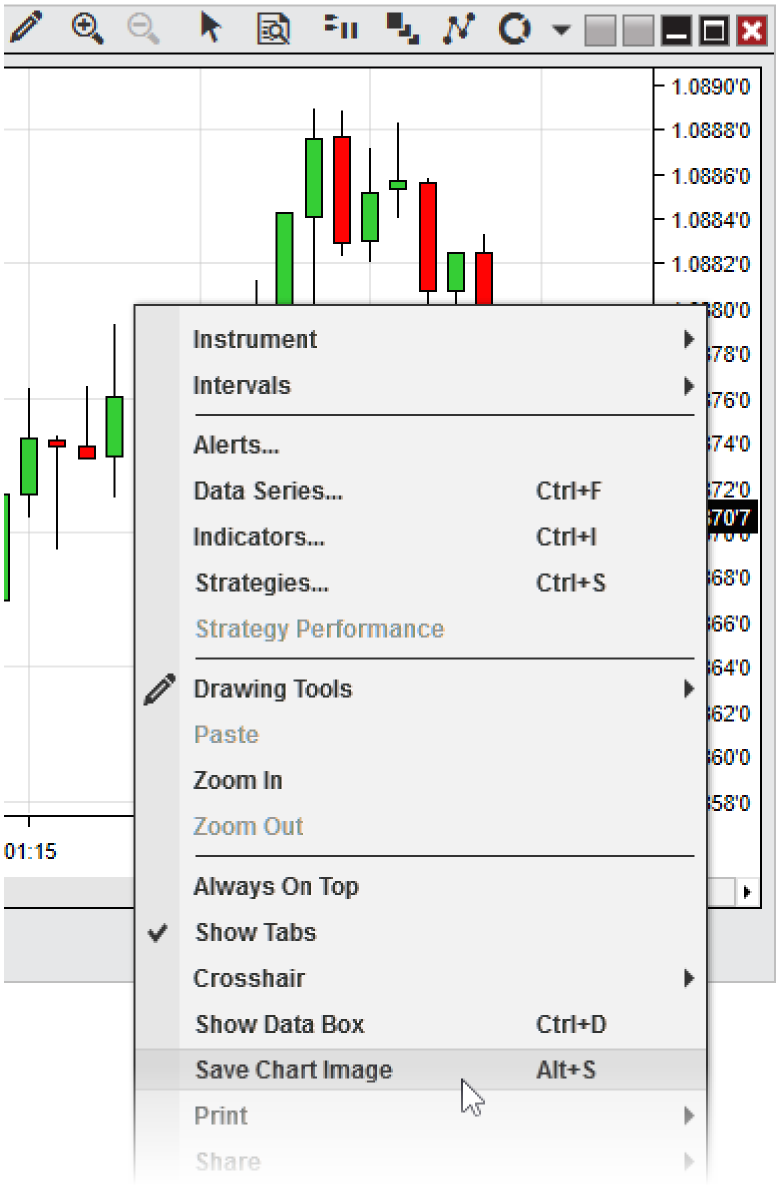
2:44:38 PM - I'm Long ES @ 1896.00

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.

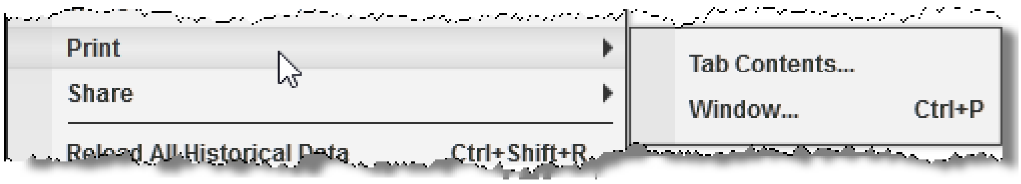
To save a chart image, first right click within the chart canvas area, then click "Save As Image," as seen in the screenshot below:



11.28.6 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.



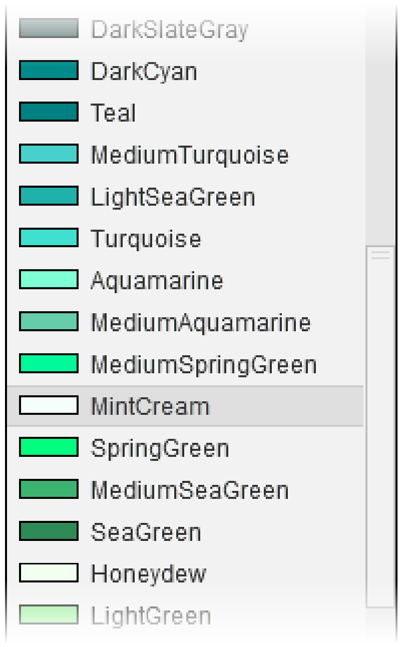
The print options available to you will vary depending on the window you choose to print from, in the screenshot above there are two options:

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.

11.28.7 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.

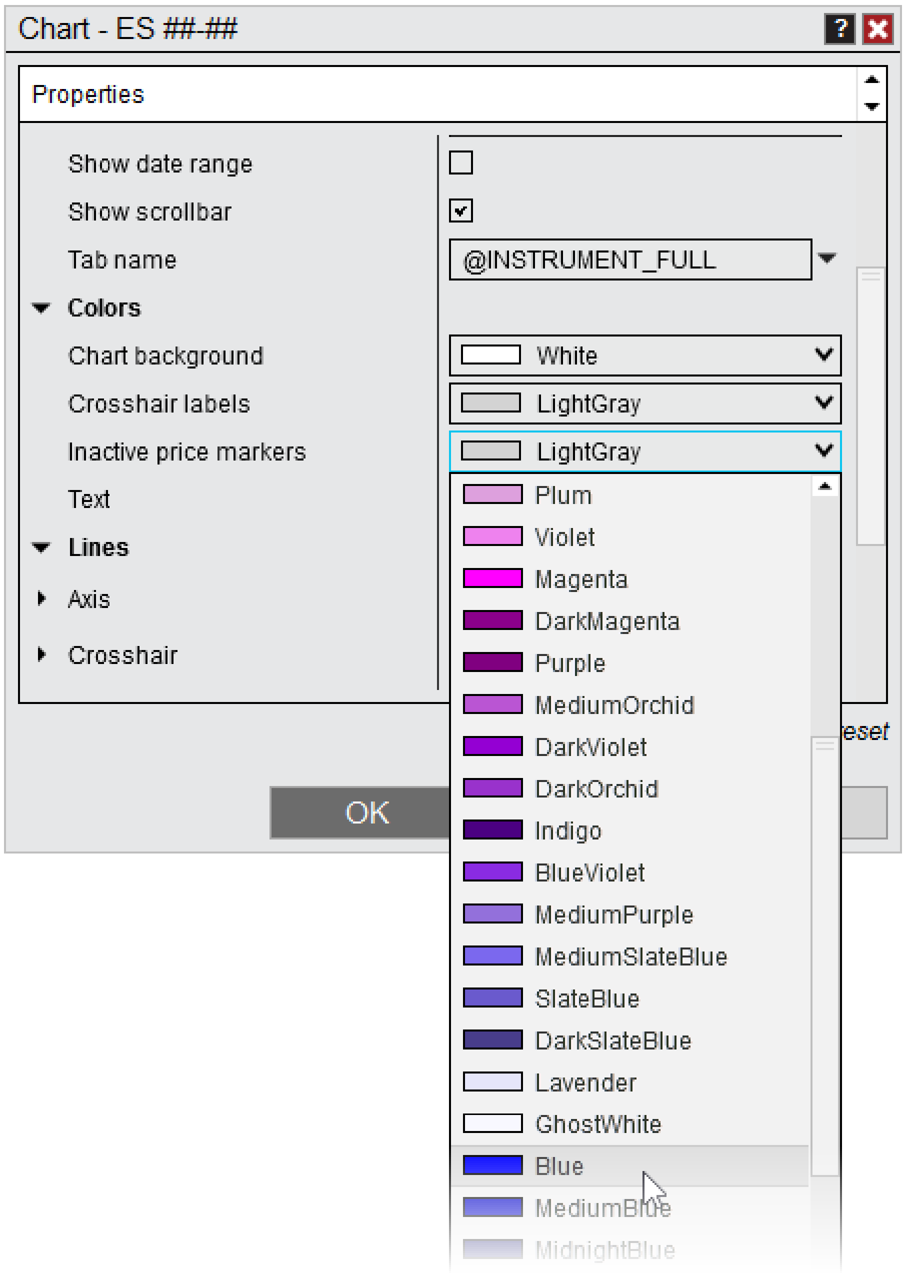


Chart - ES ##-##

Properties

Show date range	☐
Show scrollbar	☑
Tab name	@INSTRUMENT_FULL
Colors	
Chart background	White
Crosshair labels	LightGray
Inactive price markers	LightGray
Text	
Lines	
Axis	
Crosshair	

Using Custom Colors

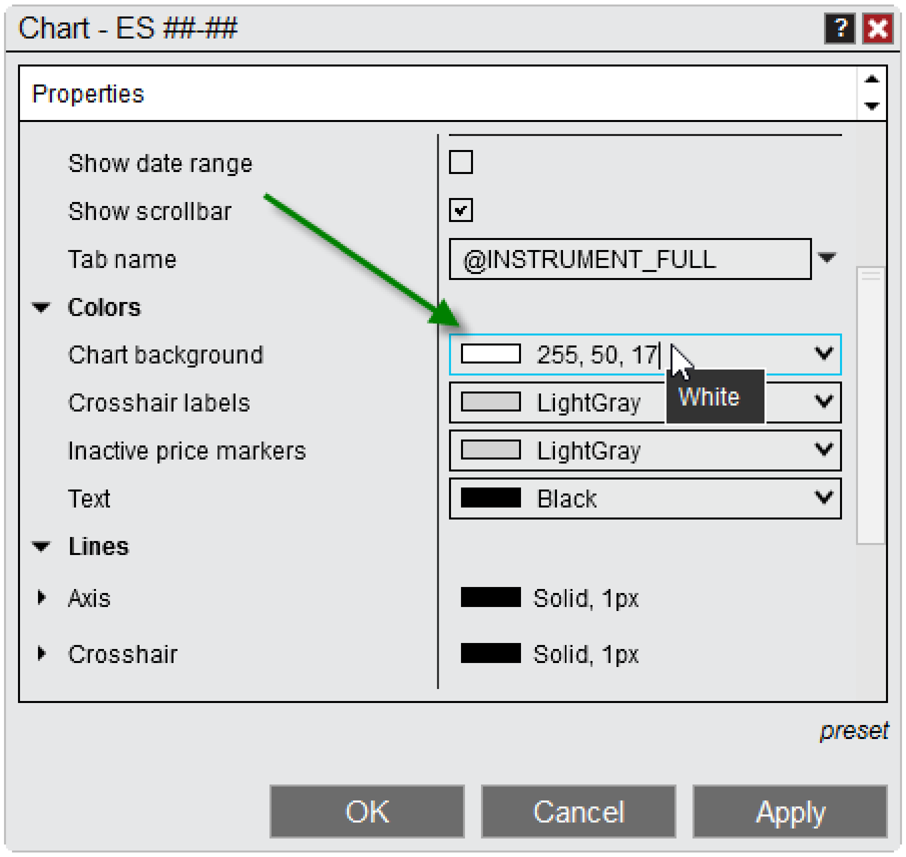
NinjaTrader's **Color Pickers** allow you to enter custom color values not defined in by default by clicking and typing directly into the field, rather than pulling down the menu. Values can be entered in one of 5 formats:

Method	Brush	Description

	Type	
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%



Show date range	☐
Show scrollbar	✓
Tab name	@INSTRUMENT_FULL
Colors	
Chart background	255, 50, 17
Crosshair labels	LightGray
Inactive price markers	LightGray
Text	Black
Lines	
Axis	Solid, 1px
Crosshair	Solid, 1px

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.

12 NinjaScript

NinjaScript Overview

NinjaScript is a C# based language that allows unlimited extensibility to NinjaTrader.

› Distribution
› Editor
› Educational Resources
› Language Reference

12.1 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 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 other than Indicator, replacing ": Indicator" with the name of your desired parent class will change the inheritance.

Note: Methods within partial classes should be use the "public" and "static" modifiers, to allow for any other classes to invoke the methods without requiring an instance of the partial class.

Partial Class Example Template

```
namespace NinjaTrader.NinjaScript.Indicators
{
    public partial class MyMethods : Indicator
    {
        //Sample method which calculates the delta of
two prices
        public static double calculateDelta(double
firstPrice, double secondPrice)
        {
            return Math.Abs(firstPrice - secondPrice);
        }

        //Sample method which prints Position
information
        public static 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.

12.2 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.

› Import
› Export
› Export Problems
› Protection/DLL Security
› Commercial Distribution

12.2.1 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

12.2.2 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

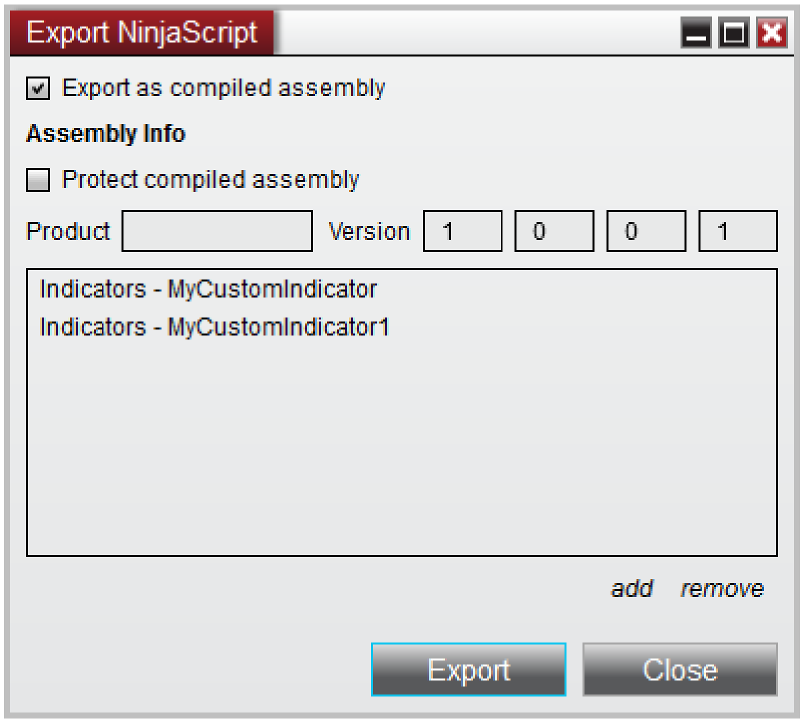
7. A NinjaScript Archive File (.zip) file will be created in My Documents \<NinjaTrader Folder>\bin\Custom\ExportNinjaScript. You can press **".."** and change the export directory if you wish

8. The file can be imported by another NinjaTrader application on a different PC

Note: The NinjaScript Archive File (.zip) generated through this process is compatible with both 32-bit and 64-bit versions of NinjaTrader.

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.



[✓] Export as compiled assembly

Assembly Info

[ ] Protect compiled assembly

Product | Version 1 0 0 1

Indicators - MyCustomIndicator
Indicators - MyCustomIndicator1

- 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

NinjaScript Archive File (.zip) file will be created in My Documents\<NinjaTrader Folder>\bin\Custom\ExportNinjaScript. You can press **".."** and change the export directory if you wish

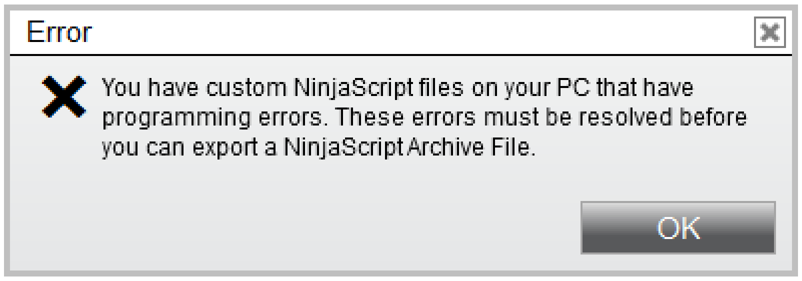
11. The file can be imported by another NinjaTrader application on a different PC

Note: When exporting a protected assembly the generated .zip is compatible with both 32-bit and 64-bit versions of NinjaTrader.

12.2.3 Export Problems

If you are having difficulties exporting NinjaScript it could be due to one of the following reasons:

▽ NinjaScript Compile Error



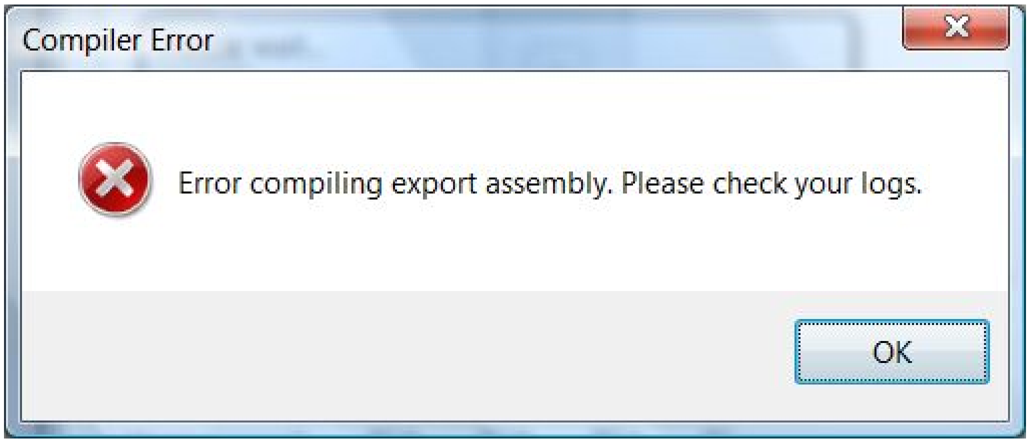
Error

You have custom NinjaScript files on your PC that have programming errors. These errors must be resolved before you can export a NinjaScript Archive File.

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. For more information on compile errors please see this article.

.NET Referencing



Compiler Error

Error compiling export assembly. Please check your logs.

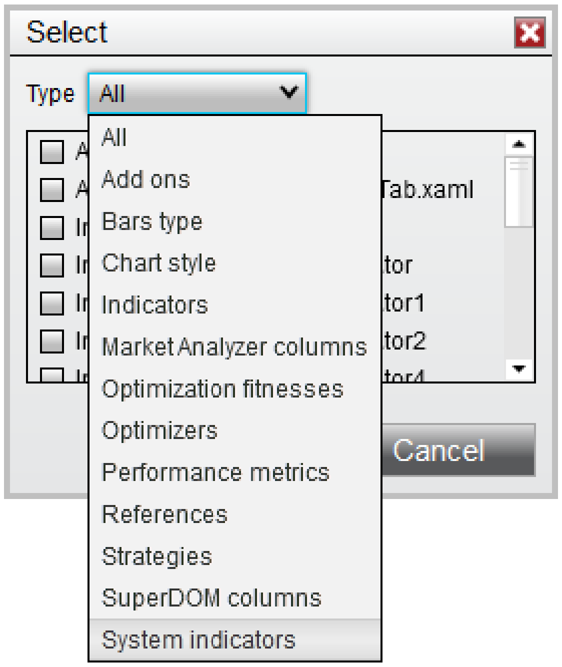
If you are able to compile without errors and still experience exporting difficulties like the one above, check to see if you receive an error similar to this in the Control Center logs:

```
"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"
```

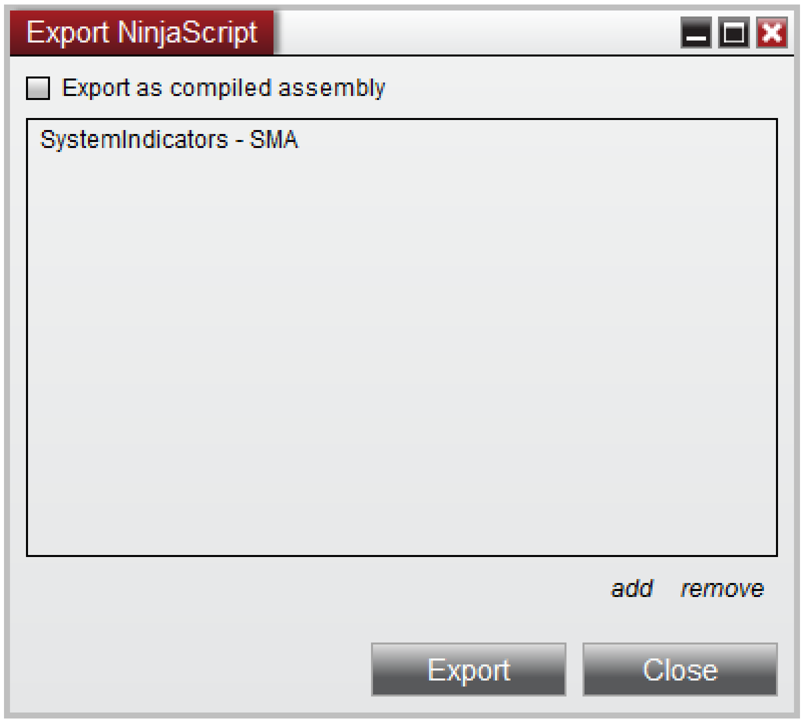
Note: This error may have a different error code and message depending on which variant of .NET you have installed. An error message indicative of this issue would include an indicator name without quotation marks.

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



SystemIndicators - SMA

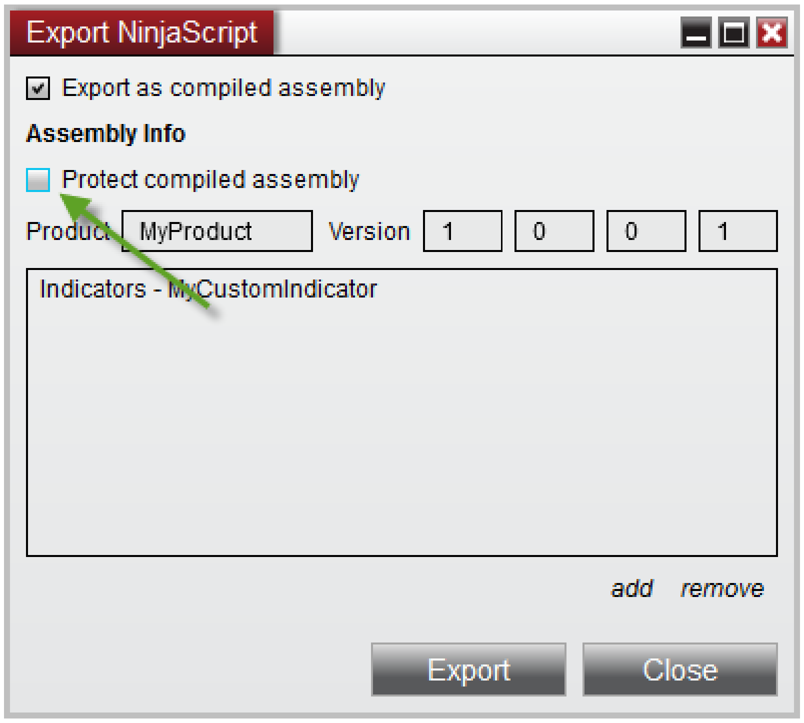
5. Press the “Export” button to create your NinjaScript Archive File. If you receive the same error again, repeat this procedure until you add all the referenced system indicators and are able to successfully export your custom NinjaScript.

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.

12.2.4 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 NinjaTraderand 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.


Export NinjaScript
Export as compiled assembly
Assembly Info
Protect compiled assembly
Product
MyProduct
Version
1
0
0
1
Indicators - MyCustomIndicator
add
remove
Export
Close

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.

12.2.5 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.

12.2.5.1 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.

12.2.5.2 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 Vista, 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.

12.2.5.3 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 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.

12.3 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

12.3.1 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.

12.3.1.1 CS0006

See CS0234.

12.3.1.2 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’

12.3.1.3 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];
```

12.3.1.4 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];
```

12.3.1.5 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;
```

12.3.1.6 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])
```

12.3.1.7 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];
```

12.3.1.8 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 Tools > Edit NinjaScript and selecting any indicator or strategy

2. Right mouse click in the NinjaScript Editor and select the menu name "References"
3. In the "References" dialog window press the button "Add"
4. Select the 3rd party DLL

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?)

12.3.1.9 CS0246

See CS0234.

12.3.1.10 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();
```

12.3.1.11 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];
```

12.3.1.12 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];
```

12.3.1.13 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];
```

12.3.1.14 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];
```

12.3.1.15 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];
```

12.3.1.16 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];
```

12.3.1.17 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
}
```

12.3.1.18 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
   }
}
```

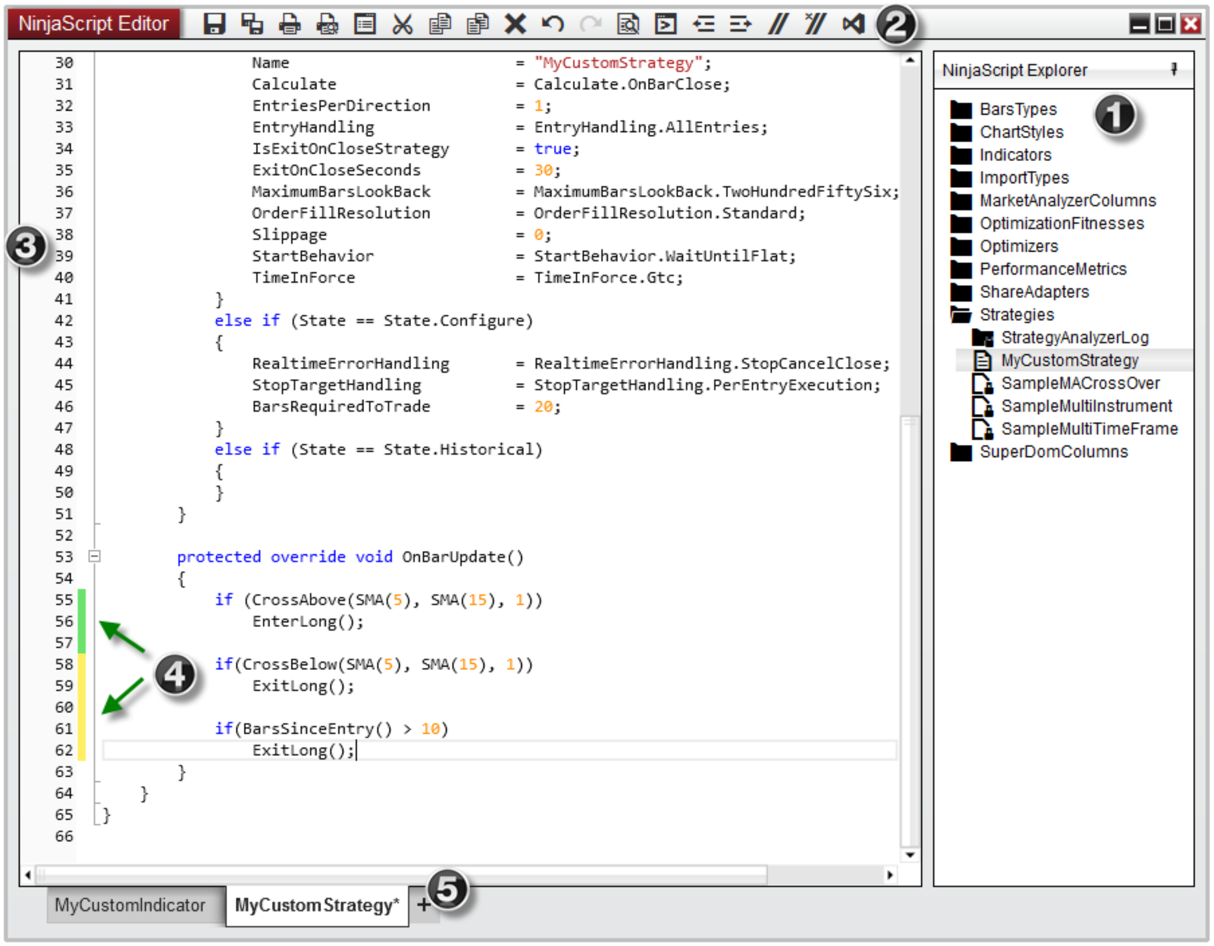
12.3.1.19 NoDoc

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.

12.3.2 NinjaScript Editor Components

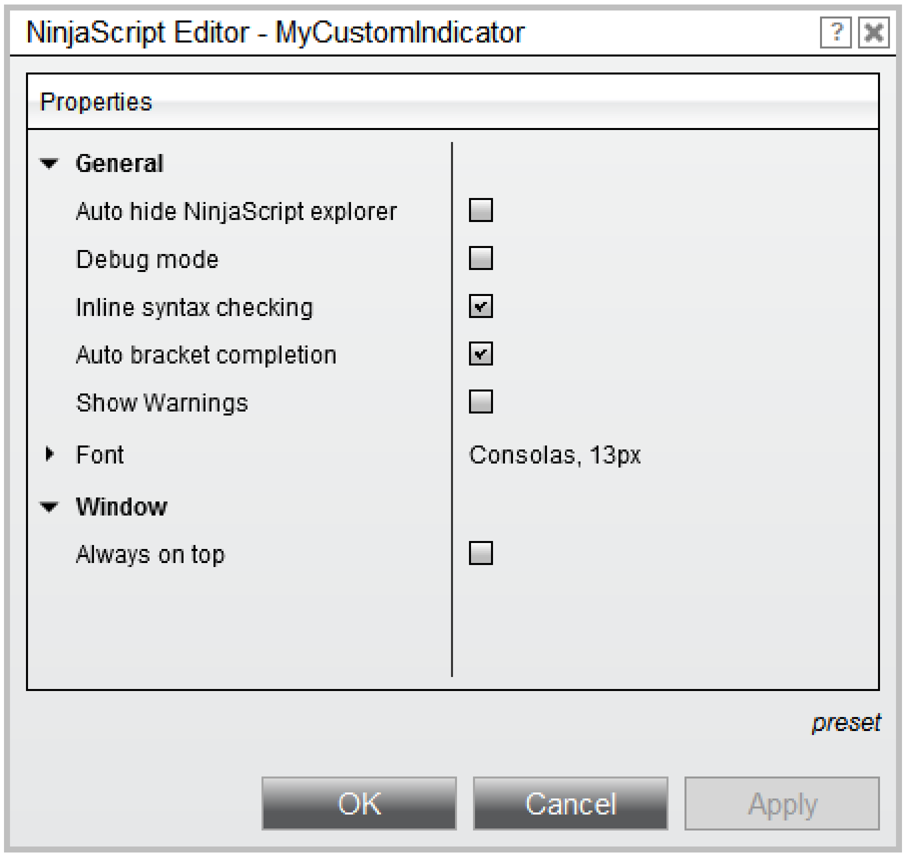
Overview
The NinjaScript Editor is a powerful scripting editor that allows you to create custom indicators, strategies, and any other custom NinjaScript types used to enhance the NinjaTrader platform. The NinjaScript Editor can be opened by selecting the **New** menu from the NinjaTrader Control Center. 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.



```
                Name                        = "MyCustomStrategy";
                Calculate                   = Calculate.OnBarClose;
                EntriesPerDirection         = 1;
                EntryHandling               = EntryHandling.AllEntries;
                IsExitOnCloseStrategy       = true;
                ExitOnCloseSeconds          = 30;
                MaximumBarsLookBack         = MaximumBarsLookBack.TwoHundredFiftySix;
                OrderFillResolution         = OrderFillResolution.Standard;
                Slippage                    = 0;
                StartBehavior               = StartBehavior.WaitUntilFlat;
                TimeInForce                 = TimeInForce.Gtc;
            }
            else if (State == State.Configure)
            {
                RealtimeErrorHandling       = RealtimeErrorHandling.StopCancelClose;
                StopTargetHandling          = StopTargetHandling.PerEntryExecution;
                BarsRequiredToTrade         = 20;
            }
            else if (State == State.Historical)
            {
            }
        }

        protected override void OnBarUpdate()
        {
            if (CrossAbove(SMA(5), SMA(15), 1))
                EnterLong();

            if(CrossBelow(SMA(5), SMA(15), 1))
                ExitLong();

            if(BarsSinceEntry() > 10)
                ExitLong();
        }
    }
}
```

Properties and Definitions



NinjaScript Editor - MyCustomIndicator

Properties

General

Auto hide NinjaScript explorer	☐
Debug mode	☐
Inline syntax checking	☑
Auto bracket completion	☑
Show Warnings	☐
Font	Consolas, 13px

Window

Always on top	☐

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 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.

12.3.3 NinjaScript Explorer

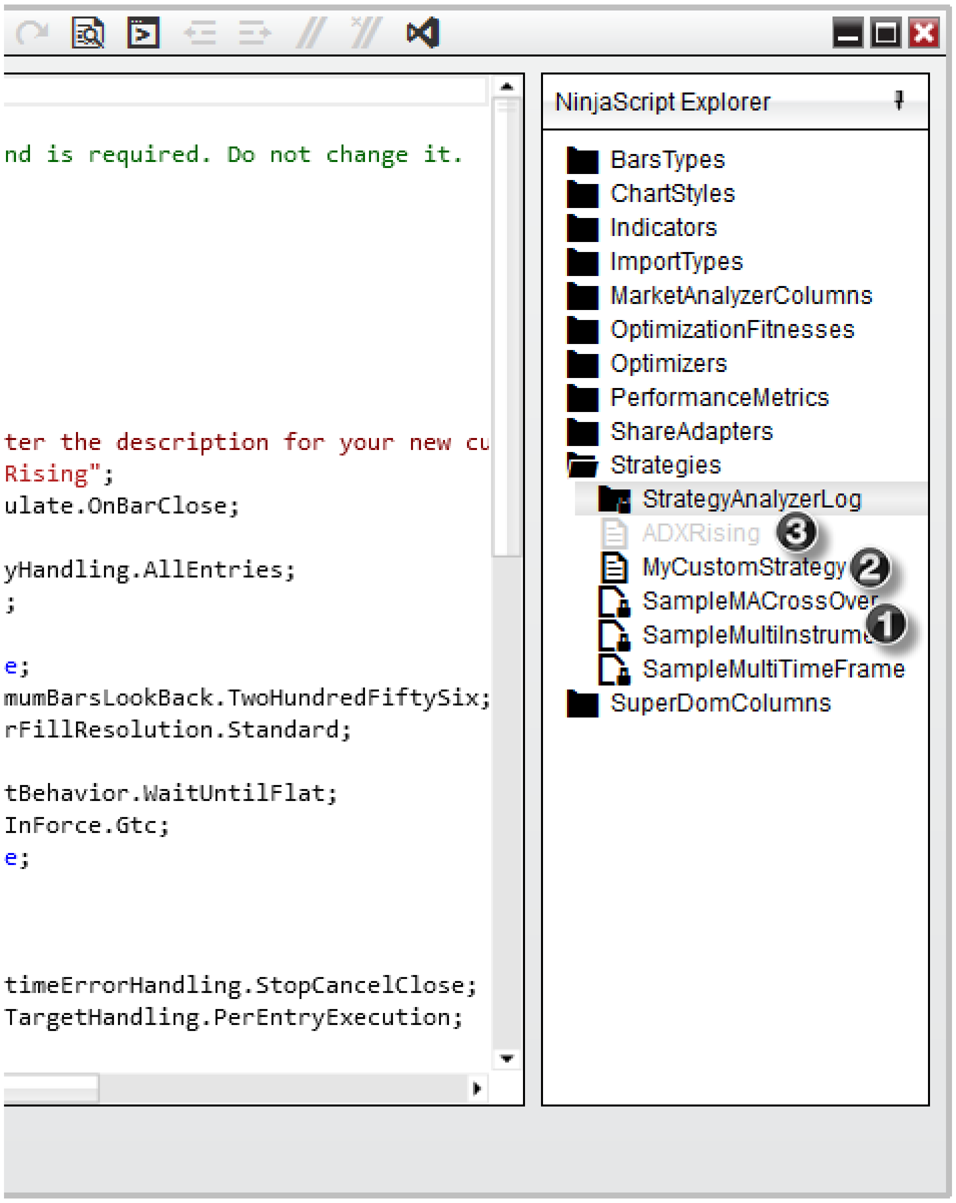
The **NinjaScript Explorer** provides a **Folder** view of all the supported NinjaScript categories that can be developed in NinjaTrader.

▽ 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:

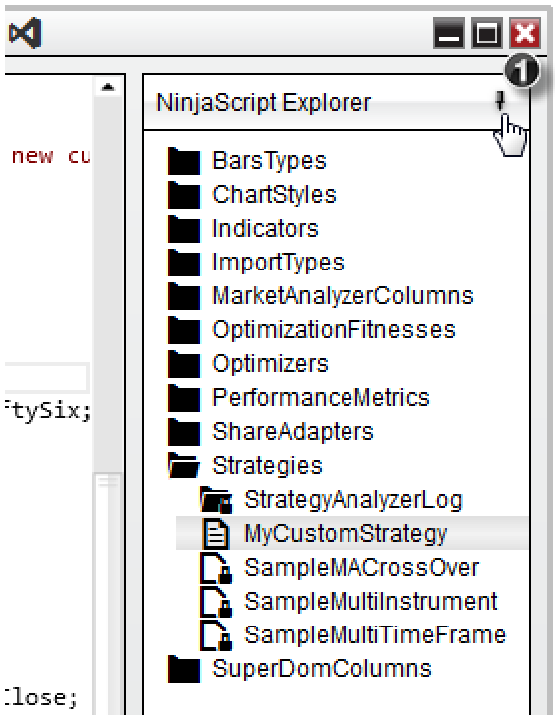
1. Locked scripts	Pre-built system scripts which come installed with NinjaTrader which can be viewed as read-only and are required for compilation (of course you can safe a custom copy of those to modify)
2. Custom scripts	Any script imported, or under development, which can be modified
3. Ignored custom scripts	Custom scripts which have been excluded from compilation (see the "*Excluding a script from compilation*" section below for more information)



```
nd is required. Do not change it.

ter the description for your new cu
Rising";
ulate.OnBarClose;

yHandling.AllEntries;
;

e;
mumBarsLookBack.TwoHundredFiftySix;
rFillResolution.Standard;

tBehavior.WaitUntilFlat;
InForce.Gtc;
e;

timeErrorHandling.StopCancelClose;
TargetHandling.PerEntryExecution;
```

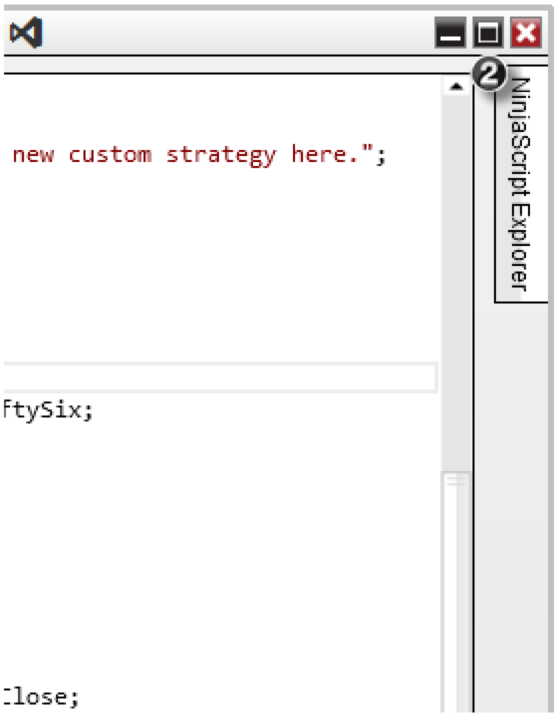
Pinning the NinjaScript Explorer

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.

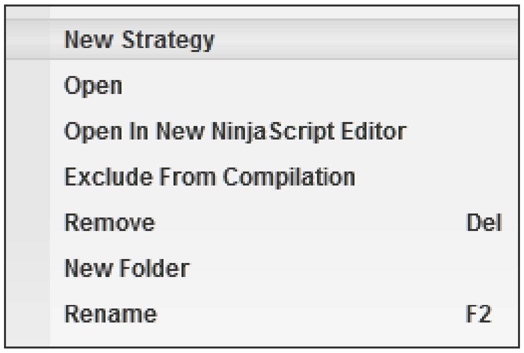
Selecting the pin icon again will re-pin the NinjaScript Explorer to the NinjaScript Editor.



```
new custom strategy here.";
ftySix;
Close;
```

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	Opens the selected script(s) in a new

NinjaScript Editor	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

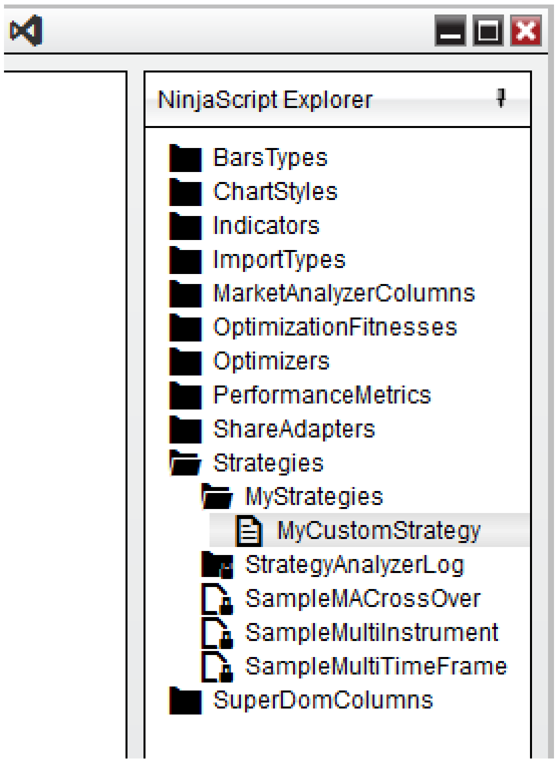
Right clicking on a NinjaScript category and selecting **New...** will open the **NinjaScript wizard** allowing you to create new custom scripts.

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.

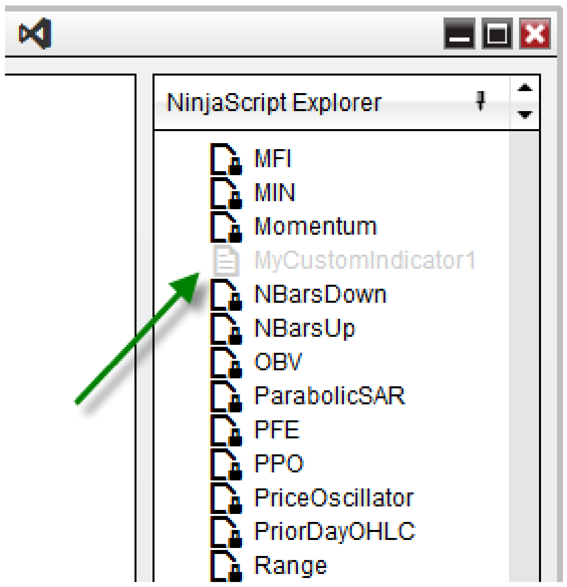
Excluding a script from compilation

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.

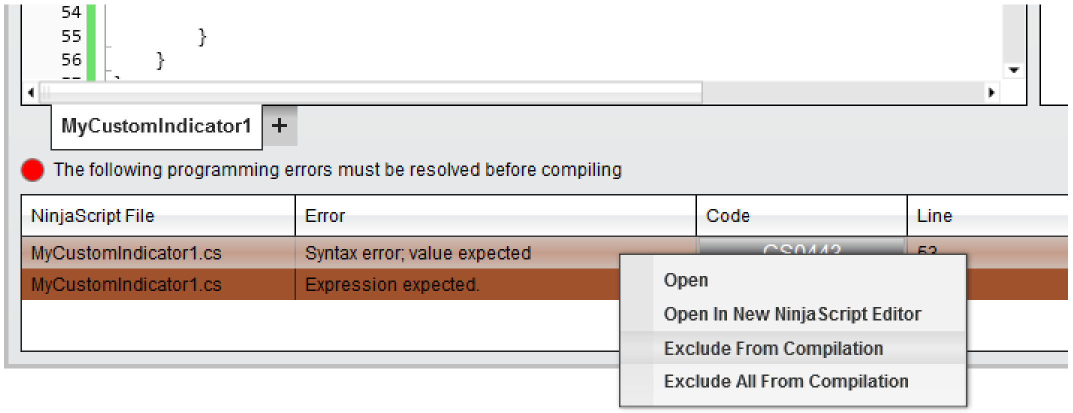


To include this script for the next compilation, simply right click on the script from the NinjaScript Explorer and uncheck **Exclude From Compilation**

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



MyCustomIndicator1 +

The following programming errors must be resolved before compiling

NinjaScript File	Error	Code	Line
MyCustomIndicator1.cs	Syntax error; value expected	CS0443	53
MyCustomIndicator1.cs	Expression expected.		

12.3.4 NinjaScript Wizard

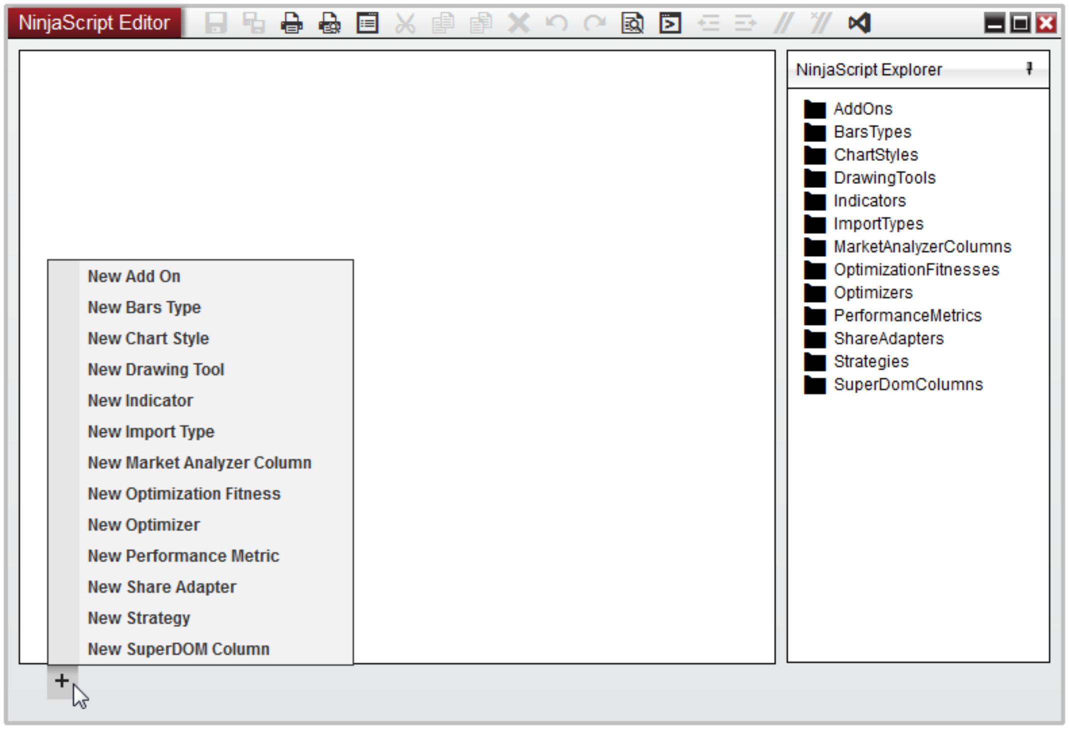
The **NinjaScript Wizard** is used to generate the minimum code to get started programming any supported NinjaScript type. This wizard will allow you to define any **default properties**, add **custom input parameters**, add **additional data series**, and add any relevant **event methods**. There are 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.

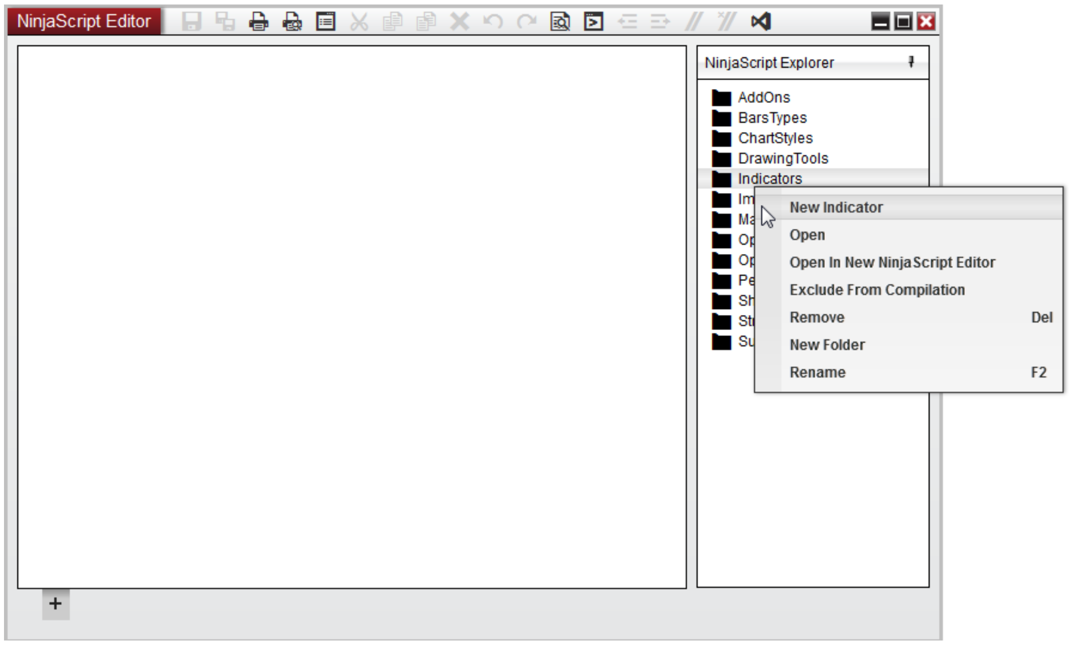
Opening the NinjaScript Wizard

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.

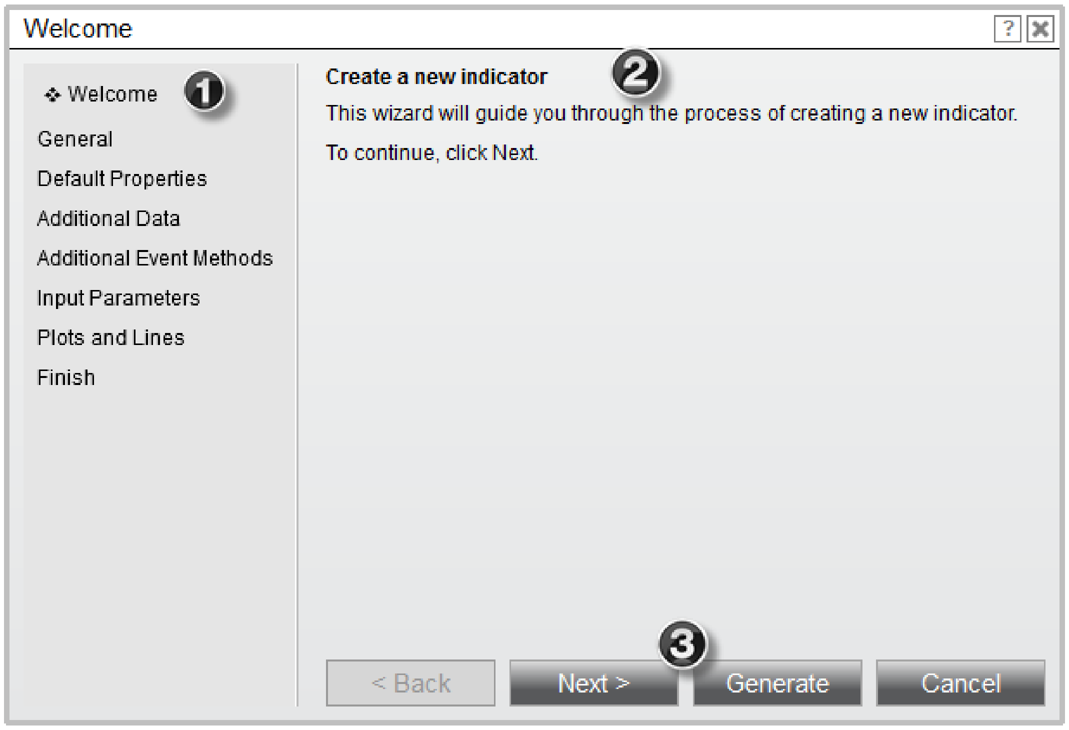


You can also right click on any of the NinjaScript categories listed in the NinjaScript Explorer and select "**New...**"



Understand the NinjaScript Wizard Display

Display Overview



Create a new indicator

This wizard will guide you through the process of creating a new indicator.

To continue, click Next.

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.

12.3.5 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)`
	`{`
	
	`}`

bars change	`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,`

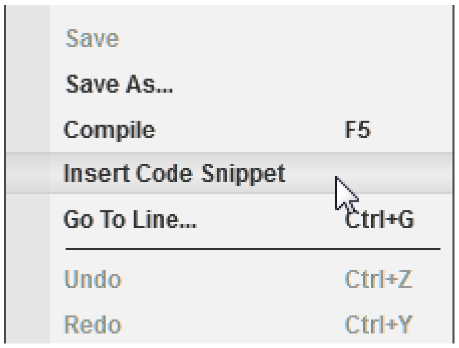
	`High[0], Brushes.Red);`
du	`Draw.ArrowUp(this, "MyArrowUp", false, 0, Low[0], Brushes.Red);`
ddi	`Draw.Diamond(this, "MyDiamond", false, 0, High[0] + 2 * TickSize, Brushes.Blue);`
dt	`Draw.Dot(this, "MyDot", false, 0,  High[0] + 2 * TickSize, Brushes.Blue);`
de	`Draw.Ellipse(this, "MyEllipse", 10, Low[10], 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);`

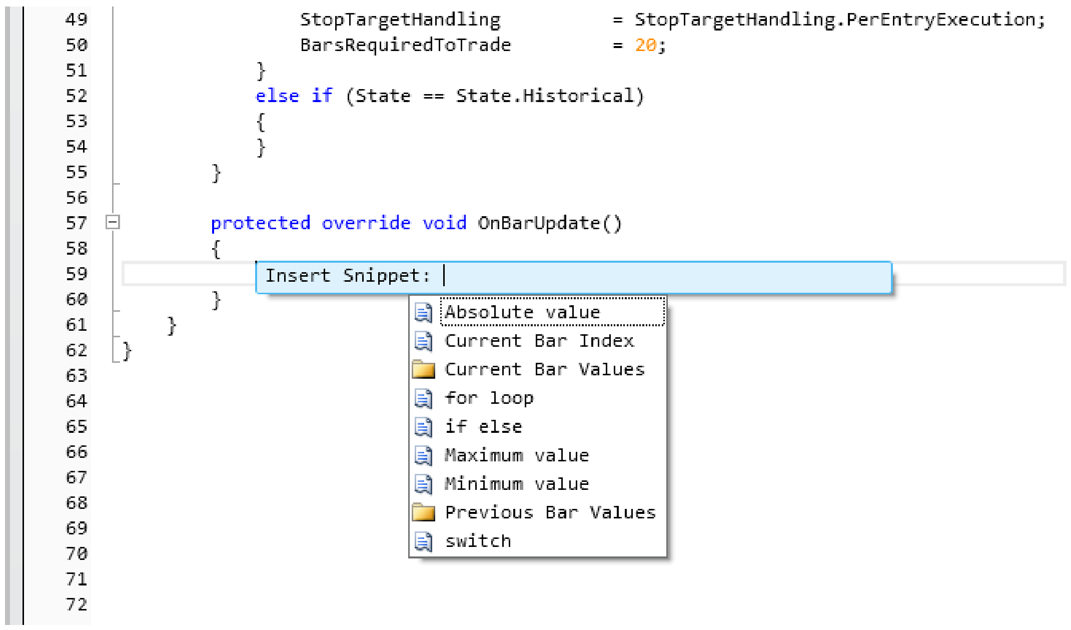
How to insert Code Snippets via the mouse or F2 key

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.



```
                StopTargetHandling          = StopTargetHandling.PerEntryExecution;
                BarsRequiredToTrade         = 20;
            }
            else if (State == State.Historical)
            {
            }
        }

        protected override void OnBarUpdate()
        {
            Insert Snippet: 
        }
    }
}
```

12.3.6 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

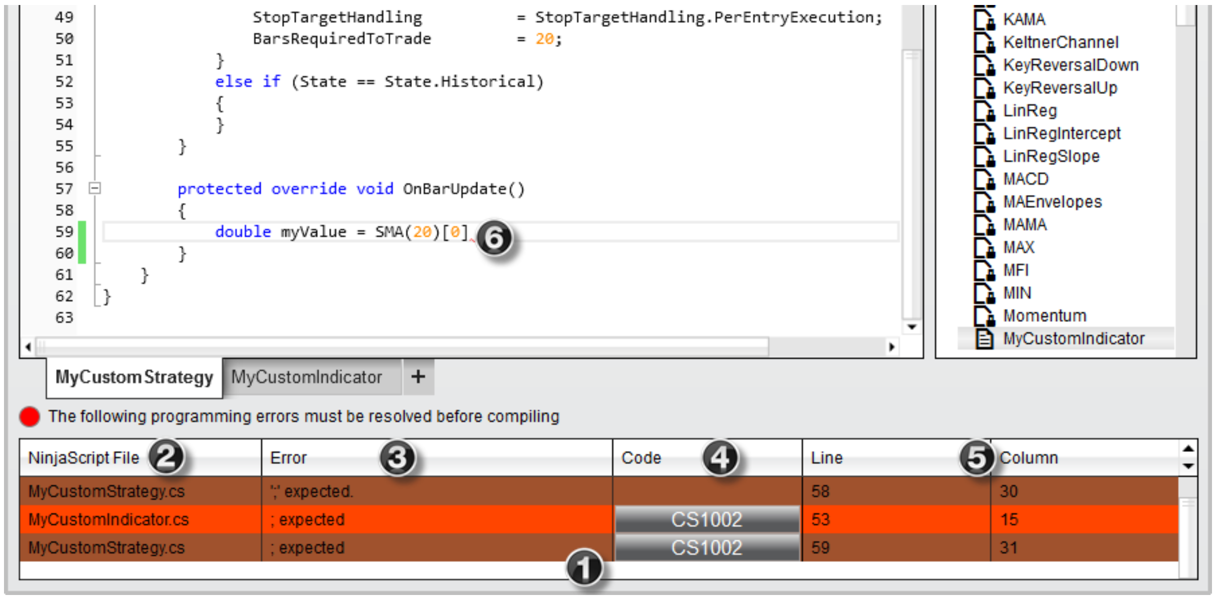
- Click on the error code to bring up Help Documentation on a specific error
- Right click on the error to exclude the problem file from compilation (see the section on Excluding a script from compilation for more information)

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];
```



```
49                  StopTargetHandling          = StopTargetHandling.PerEntryExecution;
50                  BarsRequiredToTrade         = 20;
51              }
52              else if (State == State.Historical)
53              {
54              }
55          }
56
57          protected override void OnBarUpdate()
58          {
59              double myValue = SMA(20)[0]
60          }
61      }
62  }
63
```

The following programming errors must be resolved before compiling

NinjaScript File	Error	Code	Line	Column
MyCustomStrategy.cs	';' expected.		58	30
MyCustomIndicator.cs	; expected	CS1002	53	15
MyCustomStrategy.cs	; expected	CS1002	59	31

12.3.7 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.

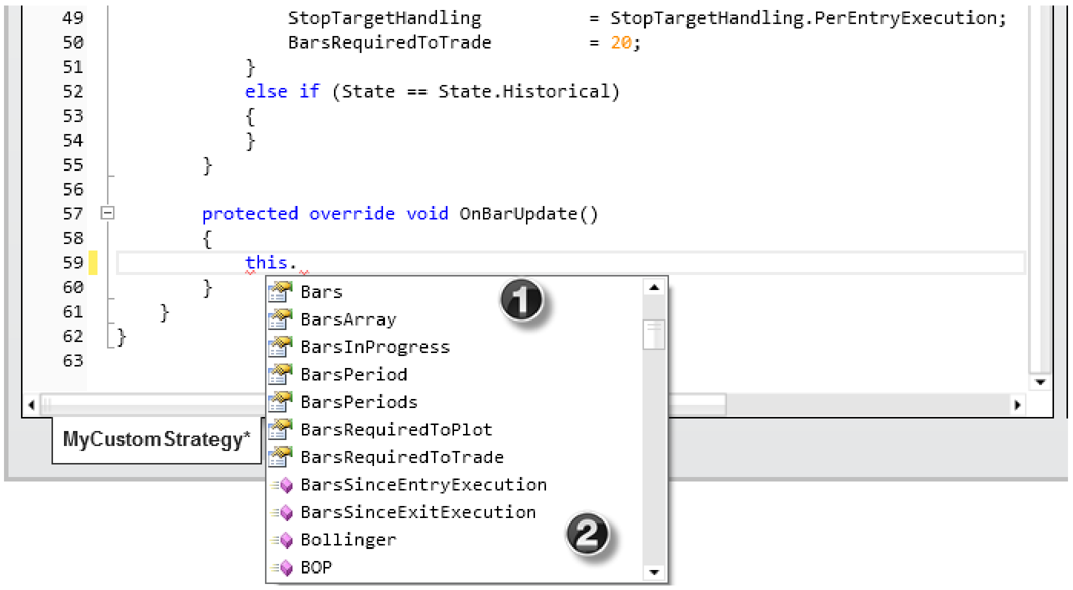
How to access the Intelliprompt list box

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.

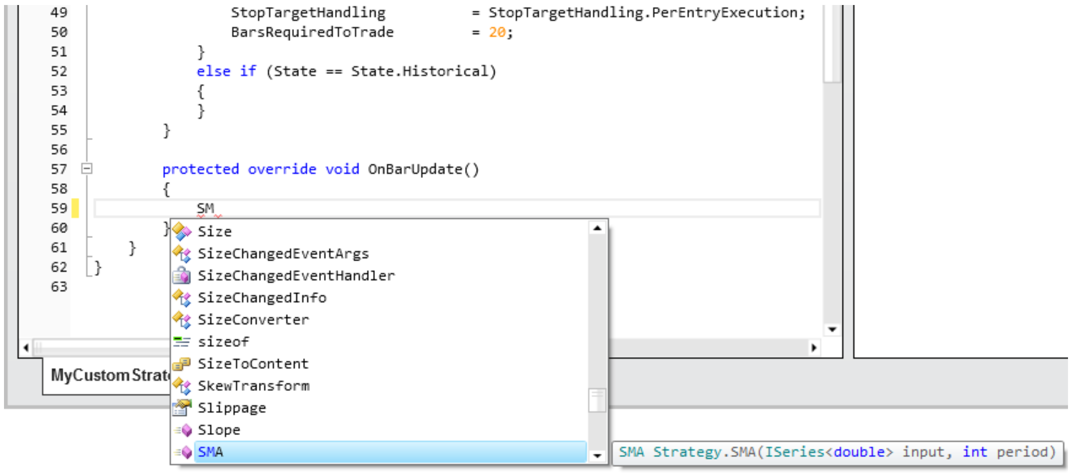
In the image below:

1. A property
2. A method



```
                StopTargetHandling          = StopTargetHandling.PerEntryExecution;
                BarsRequiredToTrade         = 20;
            }
            else if (State == State.Historical)
            {
            }
        }

        protected override void OnBarUpdate()
        {
            this.
        }
    }
}
```

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.



```
                StopTargetHandling          = StopTargetHandling.PerEntryExecution;
                BarsRequiredToTrade         = 20;
            }
            else if (State == State.Historical)
            {
            }
        }

        protected override void OnBarUpdate()
        {
            SM
        }
    }
}
```

```
SMA Strategy.SMA(ISeries<double> input, int period)
```

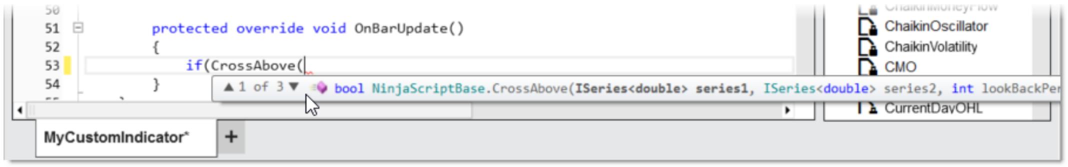
Pressing CTRL-space bar after any text will always either

- Bring up the Intelliprompt list box with related methods and properties
- Automatically insert code if the text can uniquely identify a method or property

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.



```
        protected override void OnBarUpdate()
        {
            if(CrossAbove(
        }
```

```
▲ 1 of 3 ▼  bool NinjaScriptBase.CrossAbove(ISeries<double> series1, ISeries<double> series2, int lookBackPer
```

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)
```

12.3.8 Output

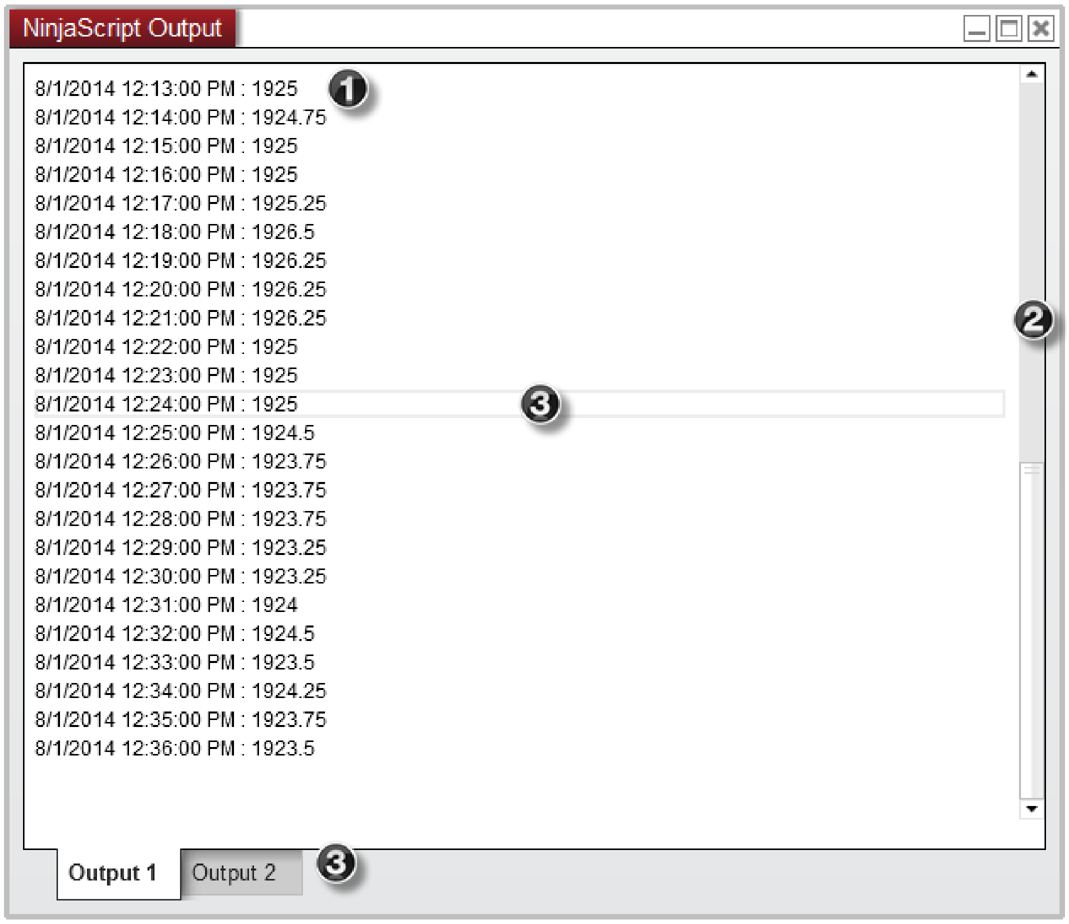
The NinjaScript Output is a powerful debugging tool which can be used to further analyze valuable information generated by your NinjaScript files. The **Output** window will only display data when other debugging methods such as the Print() or TraceOrders (for strategies) have been configured in a custom 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



```
8/1/2014 12:13:00 PM : 1925
8/1/2014 12:14:00 PM : 1924.75
8/1/2014 12:15:00 PM : 1925
8/1/2014 12:16:00 PM : 1925
8/1/2014 12:17:00 PM : 1925.25
8/1/2014 12:18:00 PM : 1926.5
8/1/2014 12:19:00 PM : 1926.25
8/1/2014 12:20:00 PM : 1926.25
8/1/2014 12:21:00 PM : 1926.25
8/1/2014 12:22:00 PM : 1925
8/1/2014 12:23:00 PM : 1925
8/1/2014 12:24:00 PM : 1925
8/1/2014 12:25:00 PM : 1924.5
8/1/2014 12:26:00 PM : 1923.75
8/1/2014 12:27:00 PM : 1923.75
8/1/2014 12:28:00 PM : 1923.75
8/1/2014 12:29:00 PM : 1923.25
8/1/2014 12:30:00 PM : 1923.25
8/1/2014 12:31:00 PM : 1924
8/1/2014 12:32:00 PM : 1924.5
8/1/2014 12:33:00 PM : 1923.5
8/1/2014 12:34:00 PM : 1924.25
8/1/2014 12:35:00 PM : 1923.75
8/1/2014 12:36:00 PM : 1923.5
```

Right click menu

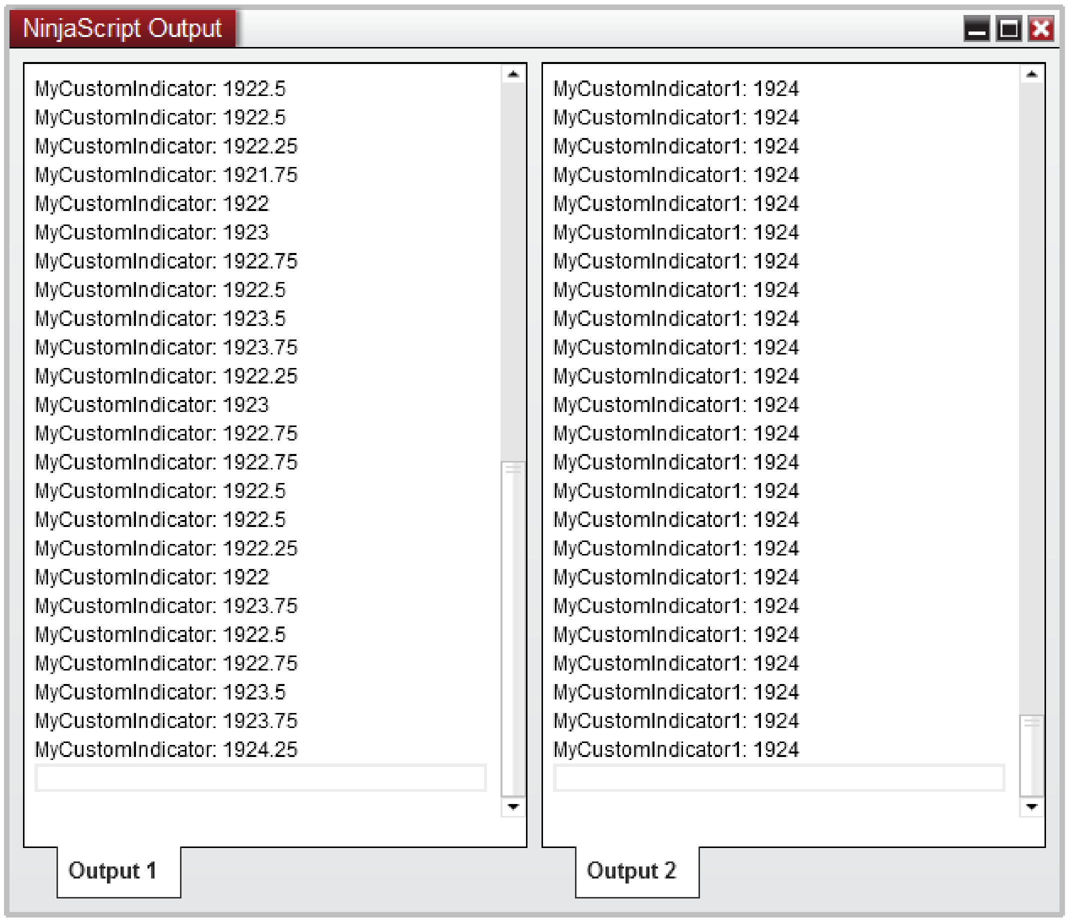
Clear	Clears the current content of the select Output window tab
Find...	Searches for a term in the Output window
Save As...	Saves the current content of the Output window in a text file
Always On Top	Sets the window to always be on top of other windows
Dual View	Enables/Disables the splitting of the 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
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`



```
MyCustomIndicator: 1922.5
MyCustomIndicator: 1922.5
MyCustomIndicator: 1922.25
MyCustomIndicator: 1921.75
MyCustomIndicator: 1922
MyCustomIndicator: 1923
MyCustomIndicator: 1922.75
MyCustomIndicator: 1922.5
MyCustomIndicator: 1923.5
MyCustomIndicator: 1923.75
MyCustomIndicator: 1922.25
MyCustomIndicator: 1923
MyCustomIndicator: 1922.75
MyCustomIndicator: 1922.75
MyCustomIndicator: 1922.5
MyCustomIndicator: 1922.5
MyCustomIndicator: 1922.25
MyCustomIndicator: 1922
MyCustomIndicator: 1923.75
MyCustomIndicator: 1922.5
MyCustomIndicator: 1922.75
MyCustomIndicator: 1923.5
MyCustomIndicator: 1923.75
MyCustomIndicator: 1924.25
```

```
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
MyCustomIndicator1: 1924
```

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

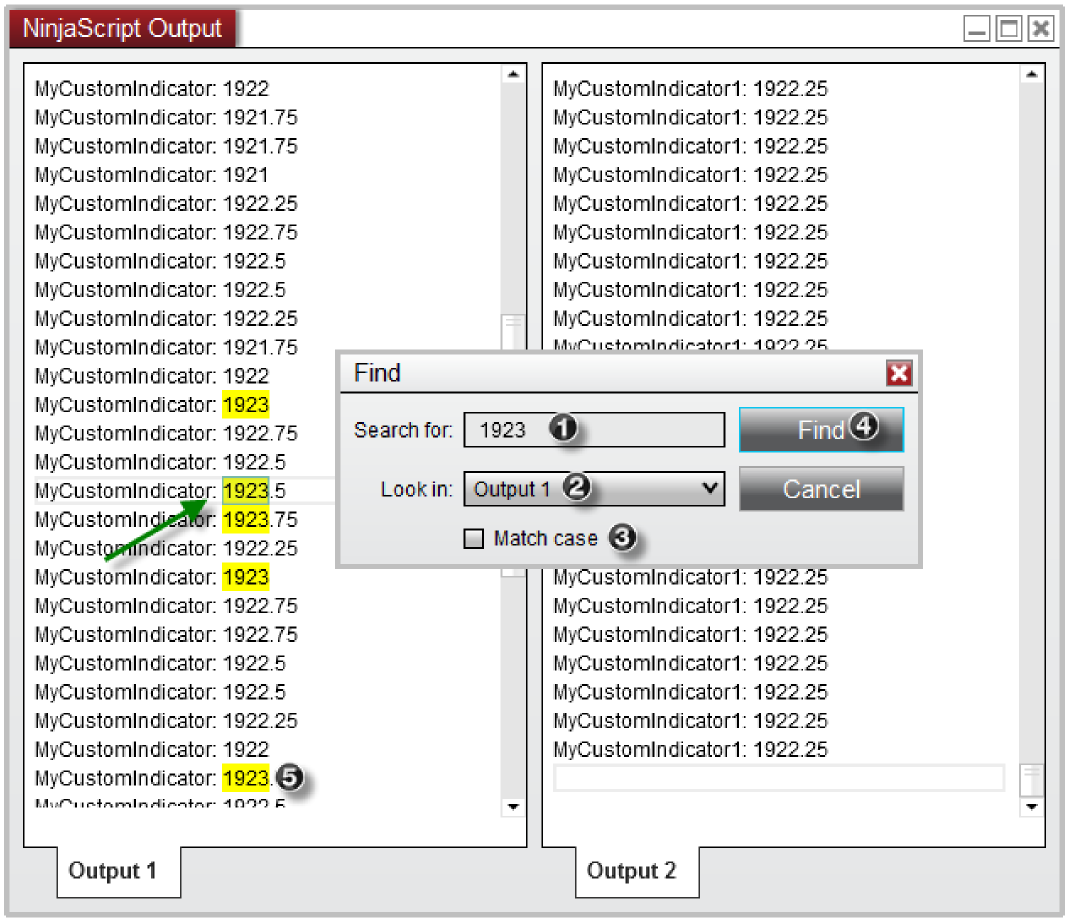
If you would like to search for a specific value or text displayed in your Output window, you can use the **Find** tool to both highlight and navigate any terms that match your search.

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.



```
MyCustomIndicator: 1922
MyCustomIndicator: 1921.75
MyCustomIndicator: 1921.75
MyCustomIndicator: 1921
MyCustomIndicator: 1922.25
MyCustomIndicator: 1922.75
MyCustomIndicator: 1922.5
MyCustomIndicator: 1922.5
MyCustomIndicator: 1922.25
MyCustomIndicator: 1921.75
MyCustomIndicator: 1922
MyCustomIndicator: 1923
MyCustomIndicator: 1922.75
MyCustomIndicator: 1922.5
MyCustomIndicator: 1923.5
MyCustomIndicator: 1923.75
MyCustomIndicator: 1922.25
MyCustomIndicator: 1923
MyCustomIndicator: 1922.75
MyCustomIndicator: 1922.75
MyCustomIndicator: 1922.5
MyCustomIndicator: 1922.5
MyCustomIndicator: 1922.25
MyCustomIndicator: 1922
MyCustomIndicator: 1923.
```

```
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
MyCustomIndicator1: 1922.25
```

Search for: 1923
Look in: Output 1
☐ Match case

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**".

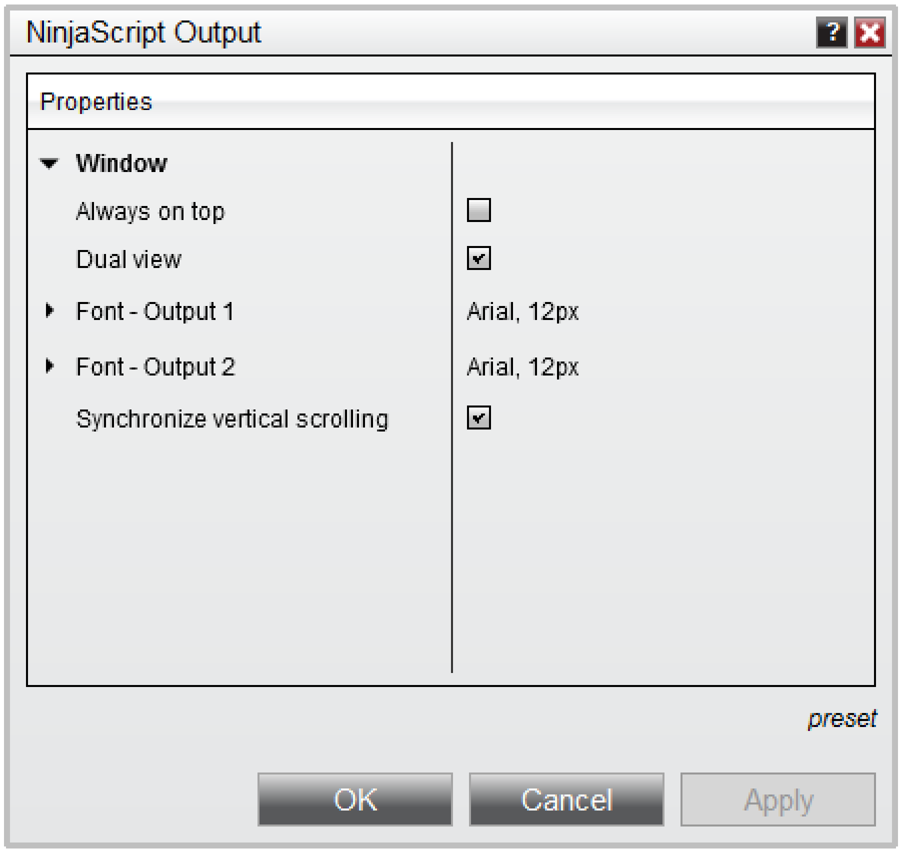
Tip: You can also use the ClearOutputWindow() method in directly your script to automatically clear the output content at a specific event or interval

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:


NinjaScript Output

Properties

Window
- Always on top: ☐
- Dual view: ✓
- Font - Output 1: Arial, 12px
- Font - Output 2: Arial, 12px
- Synchronize vertical scrolling: ✓

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

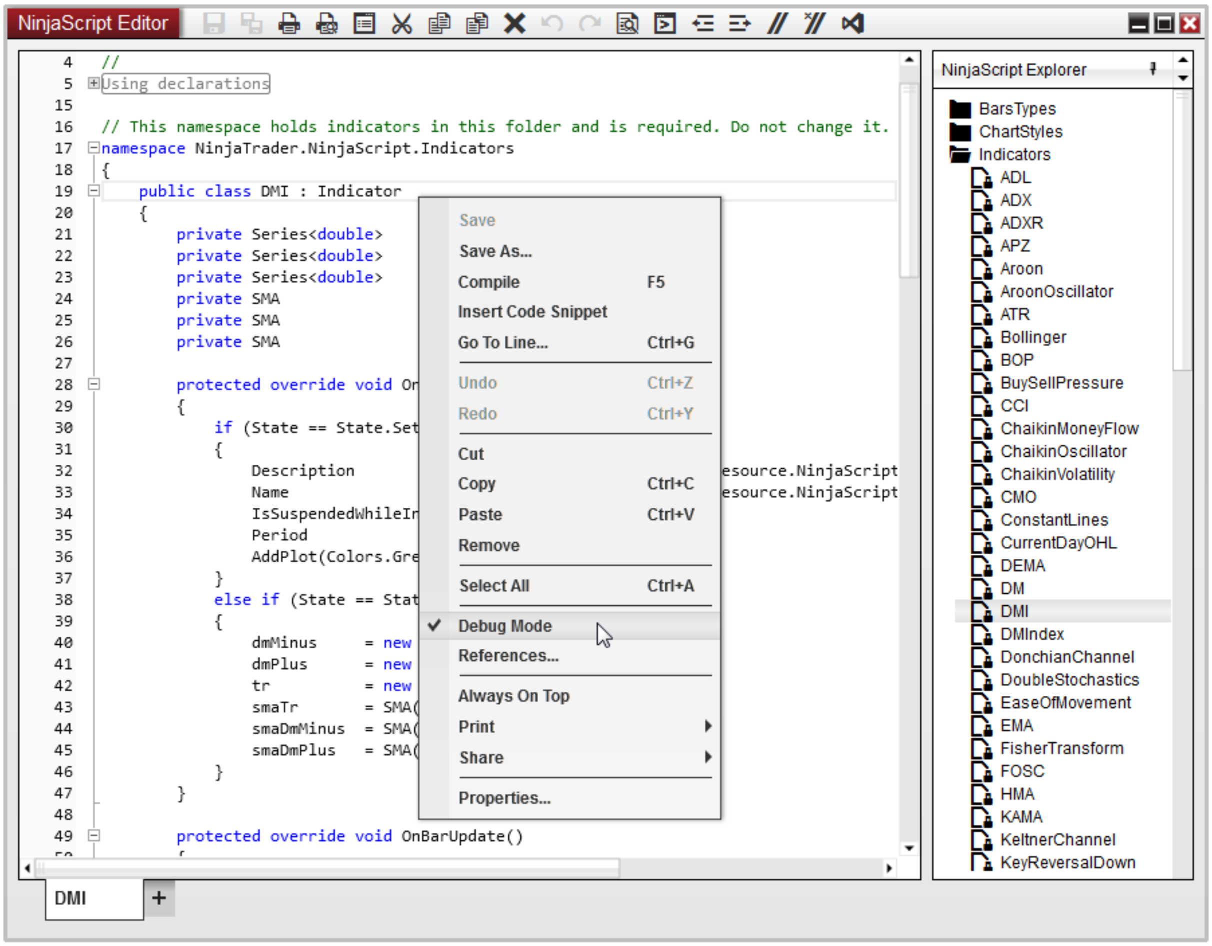
12.3.9 Visual Studio Debugging

You can debug your NinjaScript objects using Microsoft Visual Studio 2012 or later. 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.

Note: 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.

Using Visual Studio Debugging

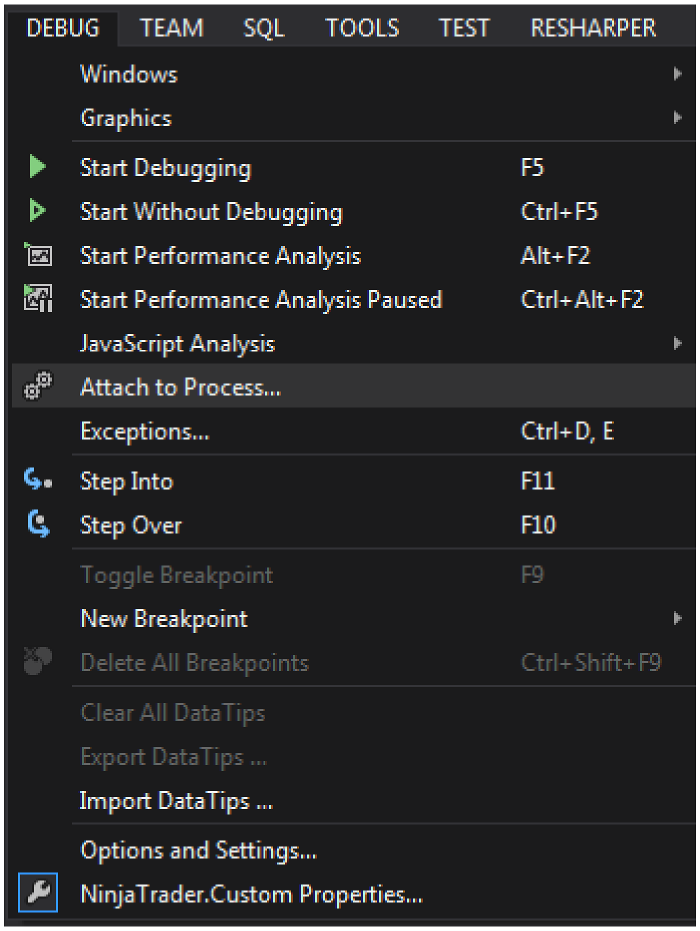
1. In the NinjaScript Editor, enable "Debug Mode" via the right-click menu, as seen in the image below. After this, compile your scripts to create the debug DLL.



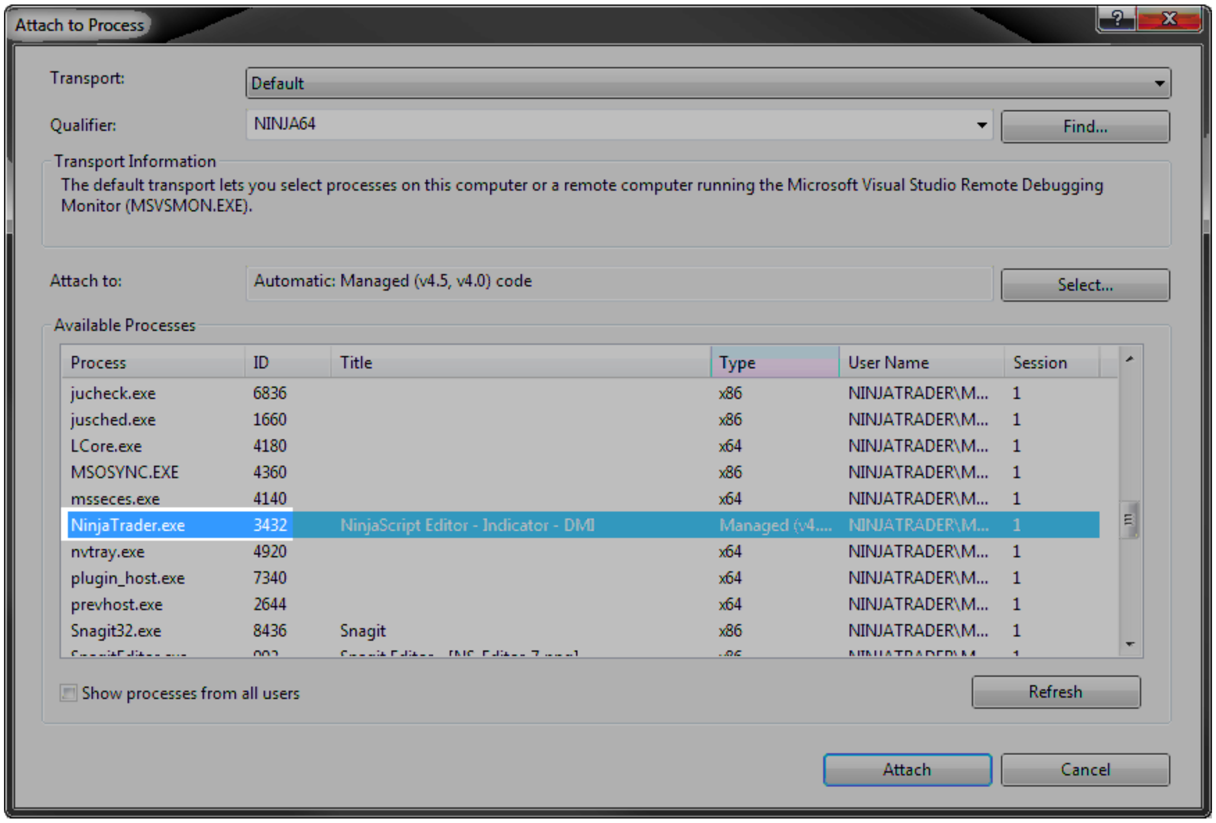
```
//
Using declarations

// This namespace holds indicators in this folder and is required. Do not change it.
namespace NinjaTrader.NinjaScript.Indicators
{
    public class DMI : Indicator
    {
        private Series<double>
        private Series<double>
        private Series<double>
        private SMA
        private SMA
        private SMA

        protected override void Or
        {
            if (State == State.Set
            {
                Description                            esource.NinjaScript
                Name                                   esource.NinjaScript
                IsSuspendedWhileIr
                Period
                AddPlot(Colors.Gre
            }
            else if (State == Stat
            {
                dmMinus     = new
                dmPlus      = new
                tr          = new
                smaTr       = SMA(
                smaDmMinus  = SMA(
                smaDmPlus   = SMA(
            }
        }

        protected override void OnBarUpdate()
```

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".



Transport: Default

Qualifier: NINJA64 [Find...]

Transport Information

The default transport lets you select processes on this computer or a remote computer running the Microsoft Visual Studio Remote Debugging Monitor (MSVSMON.EXE).

Attach to: Automatic: Managed (v4.5, v4.0) code [Select...]

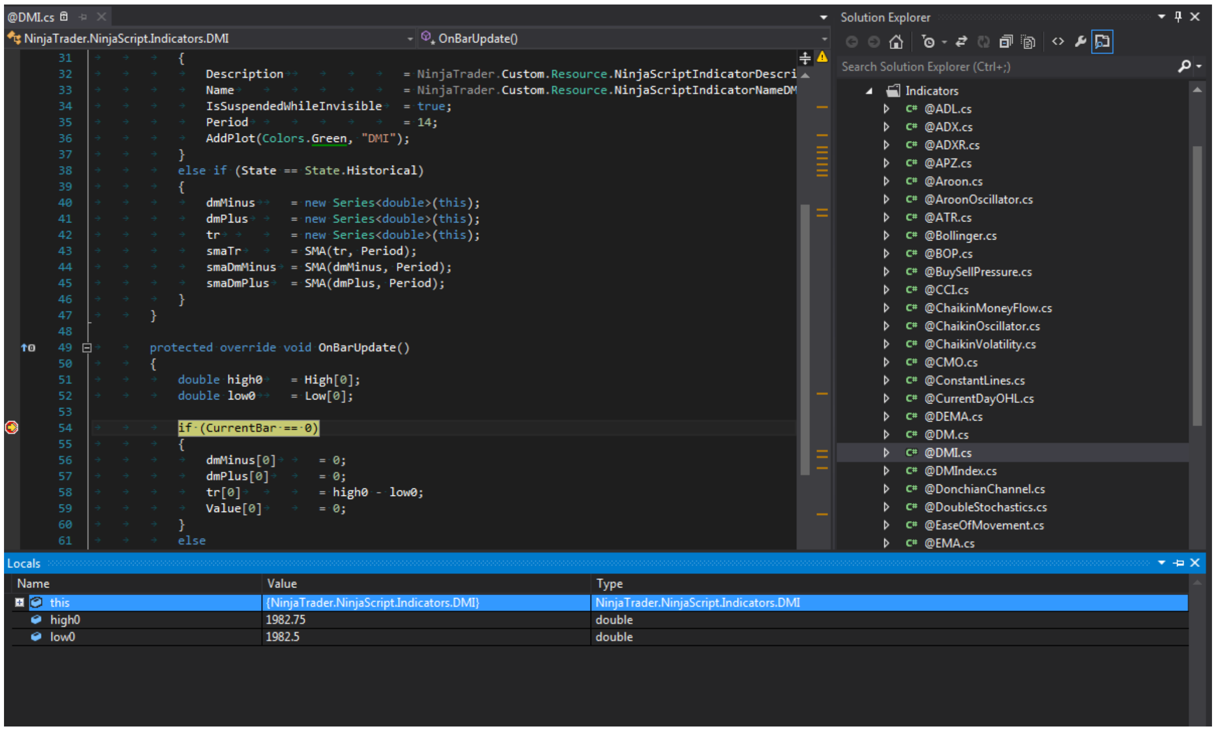
Available Processes

Process	ID	Title	Type	User Name	Session
jucheck.exe	6836		x86	NINJATRADER\M...	1
jusched.exe	1660		x86	NINJATRADER\M...	1
LCore.exe	4180		x64	NINJATRADER\M...	1
MSOSYNC.EXE	4360		x86	NINJATRADER\M...	1
msseces.exe	4140		x64	NINJATRADER\M...	1
NinjaTrader.exe	3432	NinjaScript Editor - Indicator - DMI	Managed (v4....	NINJATRADER\M...	1
nvtray.exe	4920		x64	NINJATRADER\M...	1
plugin_host.exe	7340		x64	NINJATRADER\M...	1
prevhost.exe	2644		x64	NINJATRADER\M...	1
Snagit32.exe	8436	Snagit	x86	NINJATRADER\M...	1

☐ Show processes from all users [Refresh]

[Attach] [Cancel]

4. Open the NinjaScript source file within Microsoft Visual Studio and set your break point(s)



```
            {
                Description                 = NinjaTrader.Custom.Resource.NinjaScriptIndicatorDescri
                Name                        = NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameDM
                IsSuspendedWhileInvisible   = true;
                Period                      = 14;
                AddPlot(Colors.Green, "DMI");
            }
            else if (State == State.Historical)
            {
                dmMinus     = new Series<double>(this);
                dmPlus      = new Series<double>(this);
                tr          = new Series<double>(this);
                smaTr       = SMA(tr, Period);
                smaDmMinus  = SMA(dmMinus, Period);
                smaDmPlus   = SMA(dmPlus, Period);
            }
        }

        protected override void OnBarUpdate()
        {
            double high0    = High[0];
            double low0     = Low[0];

            if (CurrentBar == 0)
            {
                dmMinus[0]      = 0;
                dmPlus[0]       = 0;
                tr[0]           = high0 - low0;
                Value[0]        = 0;
            }
            else
```

Name	Value	Type
this	{NinjaTrader.NinjaScript.Indicators.DMI}	NinjaTrader.NinjaScript.Indicators.DMI
high0	1982.75	double
low0	1982.5	double

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.

12.3.10 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

12.4 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

12.4.1 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 the NinjaTrader File Sharing forum. The heavily commented code in the example can supplement the information on these pages to provide deeper insight.

AddOn Development Environment

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.

1. Use Visual Studio or a comparable IDE to create a solution linking all project files together
2. Use your IDE to build a DLL, rather than exporting through NinjaTrader
    a. This will allow you to bundle XAML and other files into the DLL
3. Set a post-build event to place the DLL into the appropriate subfolder (NinjaTrader 8/bin/Custom/AddOns, or a subfolder in this directory)
4. Set a Debug Start Action to launch NinjaTrader

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.

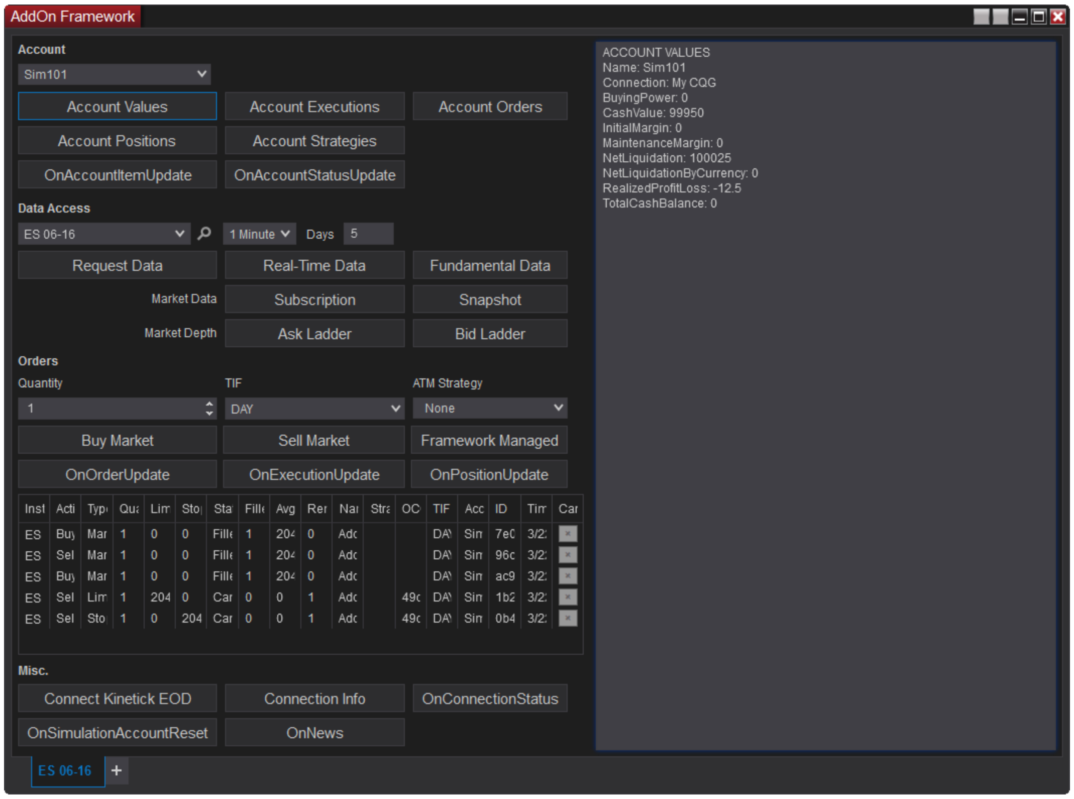
Download Visual Studio Solution for AddOn Development

Note: This Visual Studio solution uses the following path in the Start Action: C:\Program Files (x86)\NinjaTrader 8\bin64\NinjaTrader.exe. If you have installed NinjaTrader in a different directory, you will need to adjust the file path accordingly.

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.



Inst	Acti	Typ	Qu	Lim	Sto	Sta	Fille	Avg	Rer	Nar	Stra	OC	TIF	Acc	ID	Tim	Car
ES	Buy	Mar	1	0	0	Fille	1	204	0	Adc			DAY	Sim	7e0	3/2:	×
ES	Sel	Mar	1	0	0	Fille	1	204	0	Adc			DAY	Sim	96c	3/2:	×
ES	Buy	Mar	1	0	0	Fille	1	204	0	Adc			DAY	Sim	ac9	3/2:	×
ES	Sel	Lim	1	204	0	Car	0	0	1	Adc		49c	DAY	Sim	1b2	3/2:	×
ES	Sel	Sto	1	0	204	Car	0	0	1	Adc		49c	DAY	Sim	0b4	3/2:	×

```
ACCOUNT VALUES
Name: Sim101
Connection: My CQG
BuyingPower: 0
CashValue: 99950
InitialMargin: 0
MaintenanceMargin: 0
NetLiquidation: 100025
NetLiquidationByCurrency: 0
RealizedProfitLoss: -12.5
TotalCashBalance: 0
```

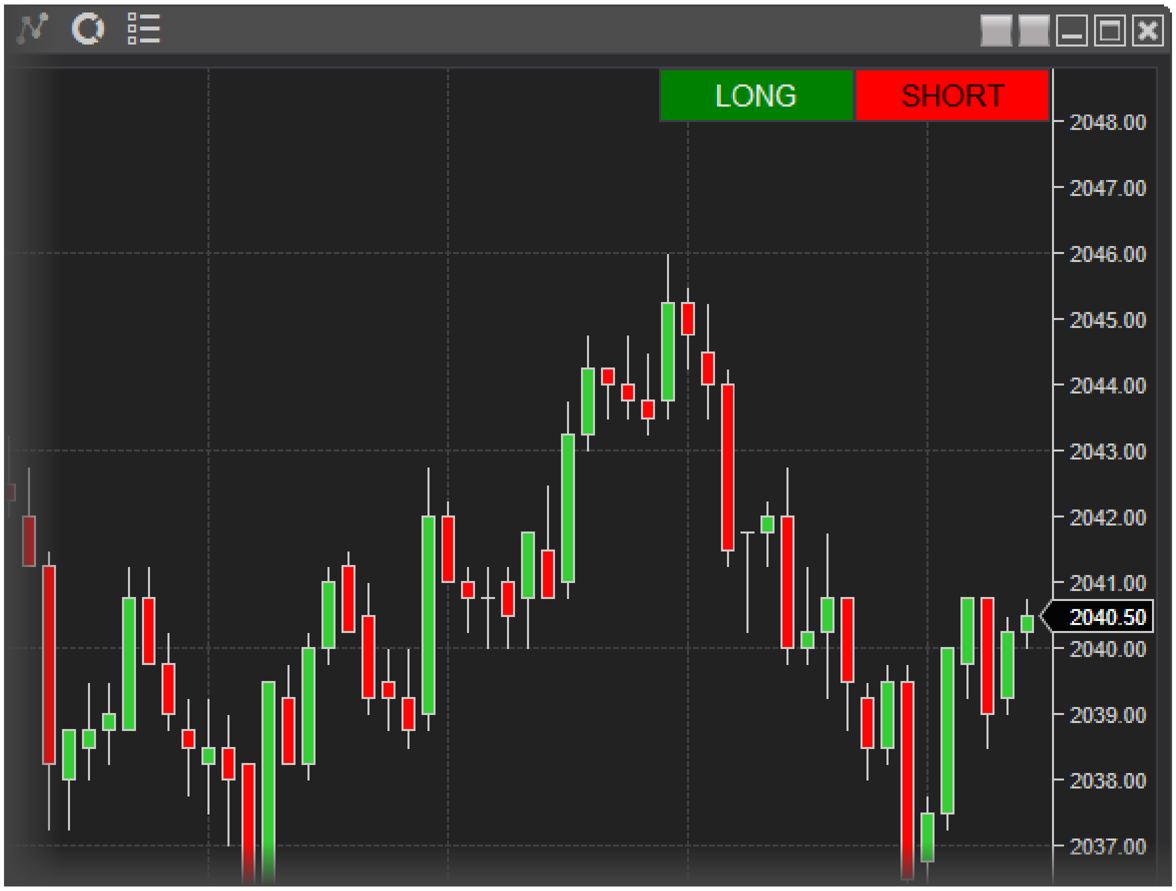
The image above shows a completely new window created by a custom AddOn.

Note: A working example of an independent AddOn window is available in the File Sharing section of the NinjaTrader forums, implementing a variety of features for demonstration purposes while guiding your through the underlying code with detailed comments. Download and import the AddOn Framework Example to learn more.

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, custom "Long" and "Short" buttons have been drawn on a chart window using an AddOn.

12.4.1.1 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.

12.4.1.2 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()
    {
        Content =
AddOnFrameworkWindowFactory.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 = 55;

ColumnDefinition col2 = new ColumnDefinition();
col2.Width = 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(addOnFrameworkMe
nuItem);

    // 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()));
}
```

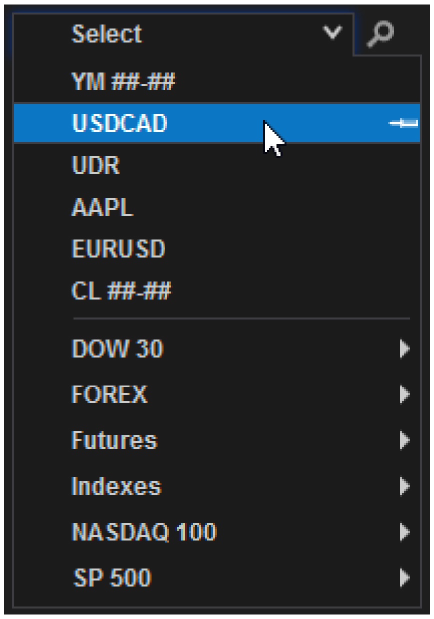
As always, it is important to unsubscribe from event handlers and dispose of unused resources when they are no longer needed. The OnWindowDestroyed() method can be used to clean up our work from the examples above:

```
// 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(addOnFrameworkM
enuItem))
            existingMenuItemInControlCenter.Items.Remove(addOn
FrameworkMenuItem);

        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



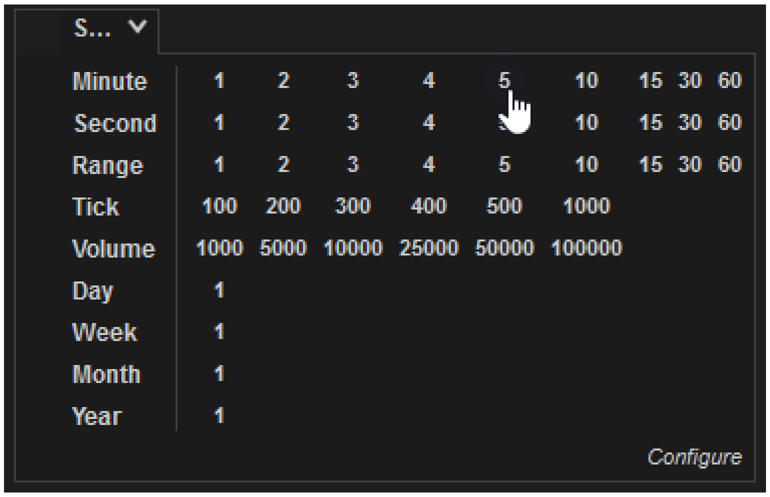
XAML - Instrument Selector Definition

```
<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



Minute	1	2	3	4	5	10	15	30	60
Second	1	2	3	4	5	10	15	30	60
Range	1	2	3	4	5	10	15	30	60
Tick	100	200	300	400	500	1000			
Volume	1000	5000	10000	25000	50000	100000			
Day	1								
Week	1								
Month	1								
Year	1								

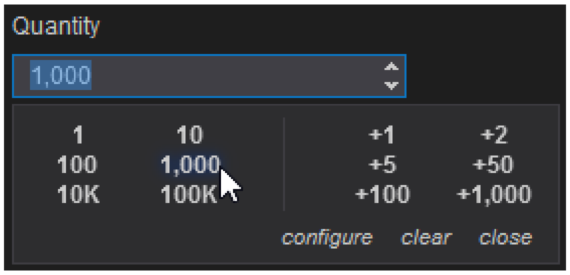
XAML - Interval Selector Definition

```
<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



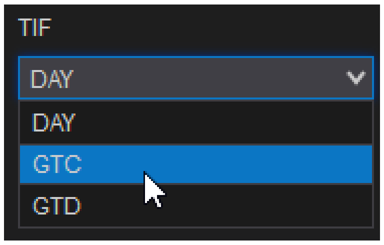
XAML - Quantity Up/Down Selector Definition

```
<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



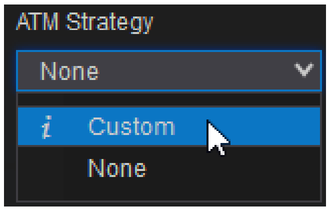
XAML - Time-in-Force Selector Definition

```
<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
.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)
    {
        // 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);
}
```

12.4.1.3 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(
"ChartWindowChartTraderControl") 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.SetAutomati
onId(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;
            }
        }));
    }));
```

12.4.2 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()););
```

12.4.3 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.

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. Define a variable of Type 'dynamic'
2. Assign a reference to the needed object to the dynamic variable
3. Access the dynamic variable as if it were of the expected Type

dynamic variables as an alternative to typecasting inside of a compiled assembly

```
foreach (dynamic line in DrawObjects)
{
    // Use ToString().Equals() to detect the object's Type
    if
(line.ToString().Equals("NinjaTrader.NinjaScript.DrawingTools.
HorizontalLine"))
    {
        // Access the object by assuming that it is the Type
we expect
        Print(String.Format("Horizontal Line {0} detected!",
line.Tag));
    }
}
```

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.

12.4.4 Developing for Tick Replay

Tick Replay Overview

Tick Replay 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 "inside" of your primary bar input series without needing to program a custom tick series. 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.

Tick Replay introduces advanced techniques to calculate your indicators and as a result, the following notes should be taken into consideration.

Notes:

1. Tick replay **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**"
2. 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
3. If you are calling an indicator which relies on historical Tick Replay data, you **MUST** ensure that the strategy or indicator which is hosting the desired indicator is aware of the hosted Tick Replay requirement *before* it has reached **State.Historical**.   Please see the topic section on this page below on "*Calling a Tick Replay indicator from another Indicator or Strategy*" for details

4. 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
5. 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()
6. If you would like to backtest OnMarketData() updates with bid/ask events, you will need to use the Playback connection with Market Replay
7. 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
8. With Tick Replay enabled on the primary Data Series, the chart's bars will be build off available tick data and thus tick data availability will restrict the range bars could be built for - this is especially important to keep in mind for any bars types that would per default have another base type to build from, for example minute bars.
9. 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
10. Running **Calculate.OnEachTick** with TickReplay can generate thousands of events per bar and may take an excessive amount of time to load. Often it is satisfactory to calculate your indicator or strategy **OnPriceChange** or **OnBarClose** while using TickReplay
11. 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

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 150-tick input series:

Calculate.OnBarClose	**Calculate.OnPriceChange**	**Calculate.OnEachTick**
OnMarketData calls for each tick used to build the bar, and OnBarUpdate only calls as the primary bar is closed	OnMarketData calls for each tick used to build the bar, and OnBarUpdate calls only as the last price changes	OnMarketData and OnBarUpdate call for each tick used to build the bar
• 120 OnMarketData events* • Followed by 1 OnBarUpdate on CurrentBar 0 • 150 OnMarketData events • Followed by 1 OnBarUpdate on CurrentBar 1 • 150 OnMarketData events • Followed by 1 OnBarUpdate on CurrentBar 2 • 150 OnMarketData events • Followed by 1 OnBarUpdate on CurrentBar 3 • ...etc.	• 1 OnBarUpdate event on CurrentBar 0 • Followed by 7 OnMarketData() events • (then price changes) • 1 OnBarUpdate event on CurrentBar 0 • Followed by 100 OnMarketData() events • (then price changes) • 1 OnBarUpdate event on CurrentBar 0 • Followed by 43 OnMarketData events • (then price changes) • 1 OnBarUpdate event on CurrentBar 1 • Followed by 10 OnMarketData events • ...etc.	• 1 OnBarUpdate event on CurrentBar 0 • Followed by 1 OnMarketData event • 1 OnBarUpdate event on CurrentBar 0 • Followed by 1 OnMarketData event • 1 OnBarUpdate event on CurrentBar 0 • Followed by 1 OnMarketData event • 1 OnBarUpdate event on CurrentBar 0 • Followed by 1 OnMarketData event • ...etc.

**Since NinjaTrader uses end-of-bar timestamps, OnBarUpdate occurs only when enough market data events have been generated to close the bar. As a result, the first 150 market data events do not have access to Bars related information (e.g., CurrentBar is equal to -1)*		

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, but will always ensure that you receive an **OnMarketData** event for every tick in the primary bar series. This process repeats for every historical bar on the chart and would continue as the indicator or strategy transitions to real-time data.

Warning: 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.

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.

Note: In the table above, you will notice that "Volume" is only available under the general last trade event. This is because tick replay only provides Last volume information. Bid/ Ask volume information is not stored in this processing mode.

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. For example, if you are calling a hosted Tick Replay indicator in another indicator or strategy in-flight during **OnBarUpdate()**, without any reference to the Tick Replay indicator beforehand, the hosting indicator or strategy would not "peek ahead" to its **OnBarUpdate()** method to determine it needs to run Tick Replay. To achieve desired results, you either need to store the reference in **State.Configure** 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.

Note: If you are calling an indicator which relies on historical Tick Replay data, you **MUST** ensure that the strategy or indicator which is hosting the desired indicator is aware of the hosted Tick Replay requirement before it has reached State.Historical. Please see the topic section on this page below on "Calling a Tick Replay indicator from another Indicator or Strategy" for details

Examples

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

```
TickReplayIndicator myTickReplayIndicator = null;

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        Name    = "TestHost";
    }
    else if (State == State.Configure)
    {
        // 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]);
}
```

12.4.5 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
      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
      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
      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

12.4.6 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
        }
    }
}
```

12.4.7 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 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 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.

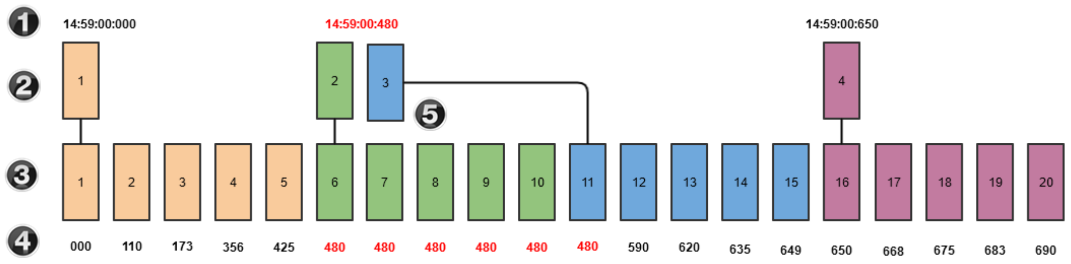


Figure 1. Bar processing with shared timestamps
1. Timestamps of primary series (hour, minute, second, millisecond)
2. Current bars numbered in series representing 5-tick primary series
3. Current bars numbered in series representing 1-tick secondary series
4. Millisecond time stamps of secondary series
5. A sequence of bars sharing the same time stamps

Adding Additional Bars Object 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 into 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.**

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.Historical)
    {
        // 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.Historical)
    {
        // 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.

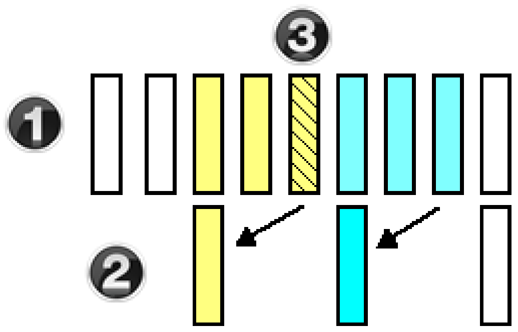


Figure 1. Bar processing on historical data using Calculate.OnBarClose
1. Primary 1-minute bar series
2. Secondary 3-minute bar series
3. Bar #5

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.

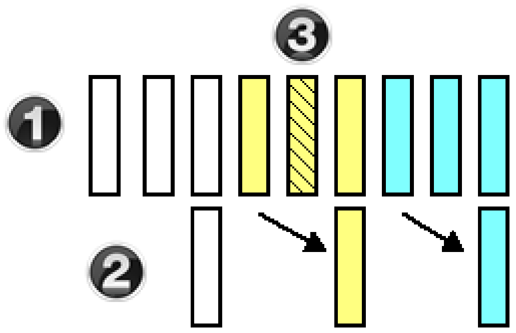


Figure 2. Bar processing in real-time using Calculate.OnEachTick or Calculate.OnPriceChange
1. Primary 1-minute bar series
2. Secondary 3-minute bar series

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. Now your chart looks different.

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.

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, or use High Order FIll Resolution which will handle this for you.

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();
     }
}
```

12.4.8 Understanding the lifecycle of your NinjaScript objects

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.

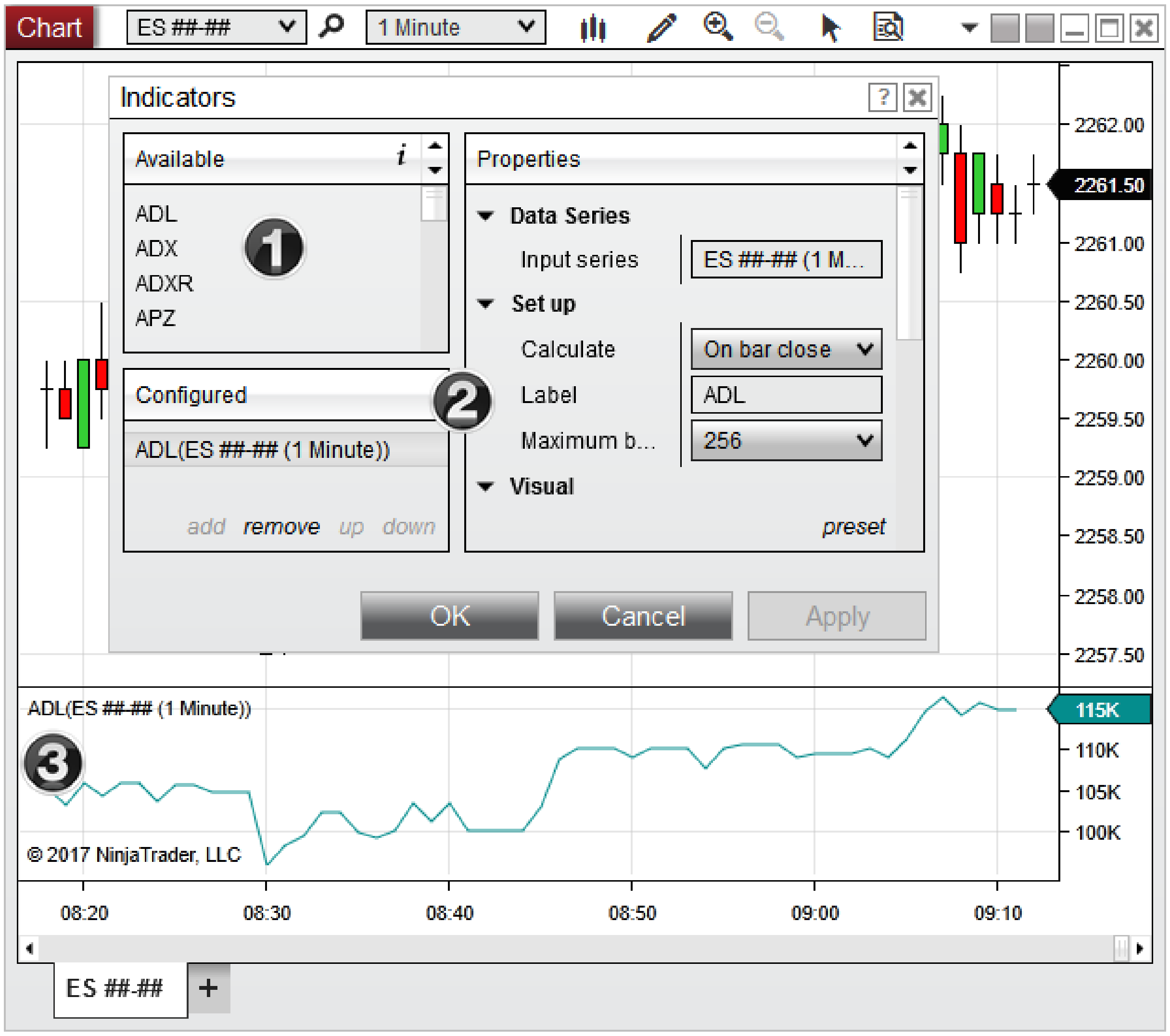
Note: It is important to understand that previous major versions of NinjaTrader were not so diligent in running termination logic for UI instances and the current major NinjaTrader 8 version has been changed to help properly address related issues.

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**"

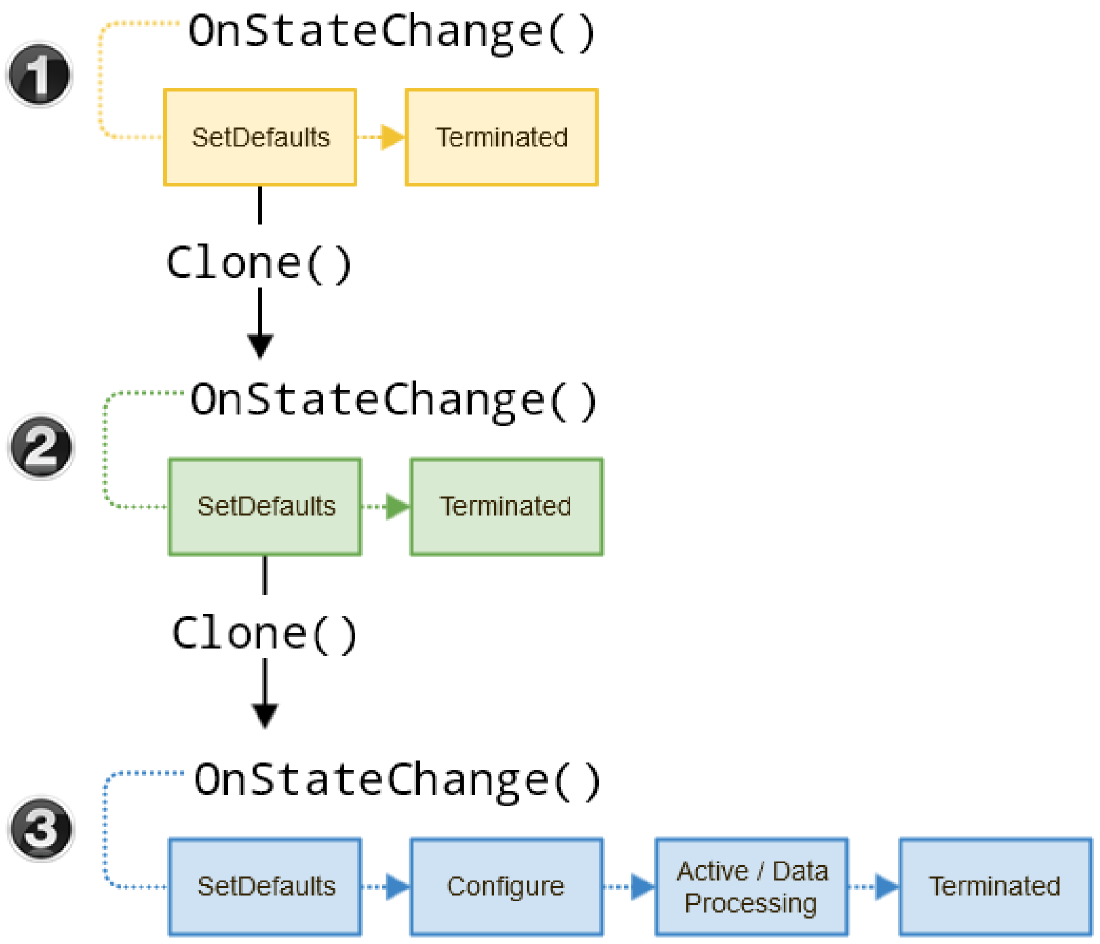
During this sequence, there are actually 3 instances of the same indicator created by NinjaTrader:

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 indicator executing on the chart.



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. AddOns are a special case which will run through their **SetDefaults/Terminated** states after each NinjaScript compile, since they run in the background and are not dependent on UI elements.
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**.

However - since the ChartControl object can be cloned back to UI instance, merely checking if ChartControl exits is not enough to determine when the remove logic should be called. To address this concern in the example below, you can see 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
      if (ChartControl != null)
      {
         AddToolBarButton(ChartControl); // 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 (ChartControl != null)
      {
         // with he help of our custom flag, we can then
ensure the custom method below
         // only runs when truly needed
         if (toolBarNeedsReset)
            RemoveToolBarButton(ChartControl);
      }
   }
}
```

12.4.9 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.

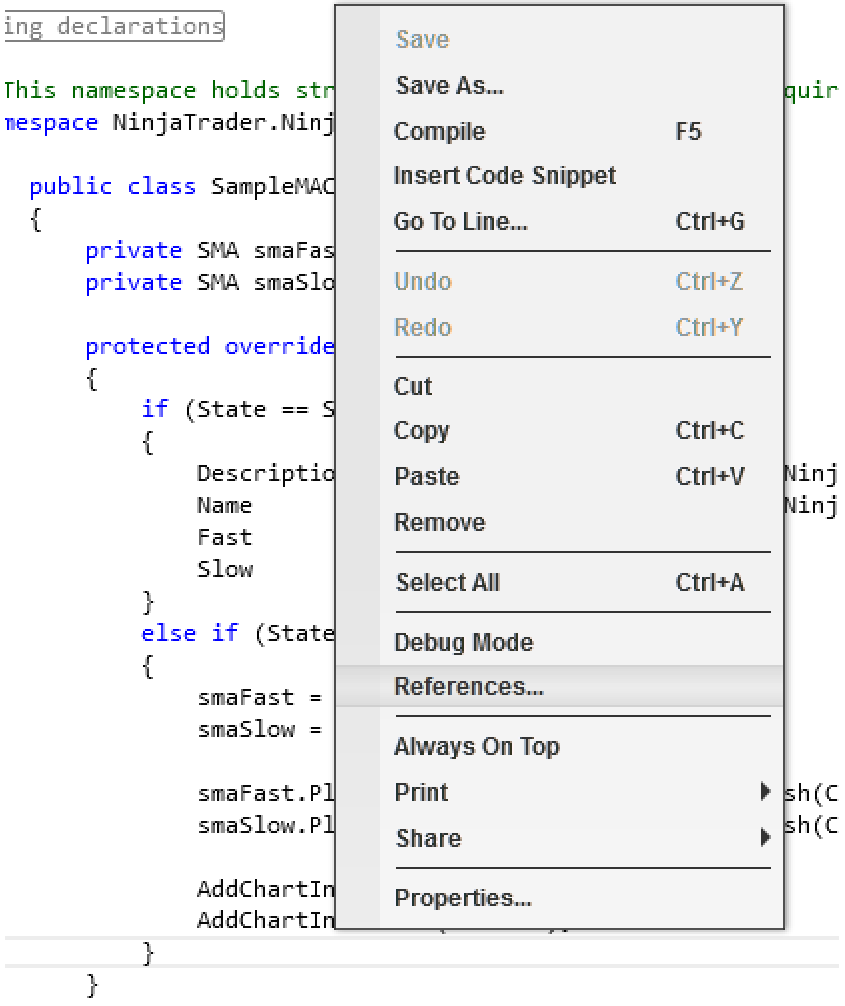
Vendor File

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.



```
ing declarations

This namespace holds str                                  quir
mespace NinjaTrader.Ninj

  public class SampleMAC
  {
      private SMA smaFas
      private SMA smaSlo

      protected override
      {
          if (State == S
          {
              Descriptio                                  Ninj
              Name                                        Ninj
              Fast
              Slow
          }
          else if (State
          {
              smaFast =
              smaSlow =

              smaFast.Pl                                  sh(C
              smaSlow.Pl                                  sh(C

              AddChartIn
              AddChartIn
          }
      }
```

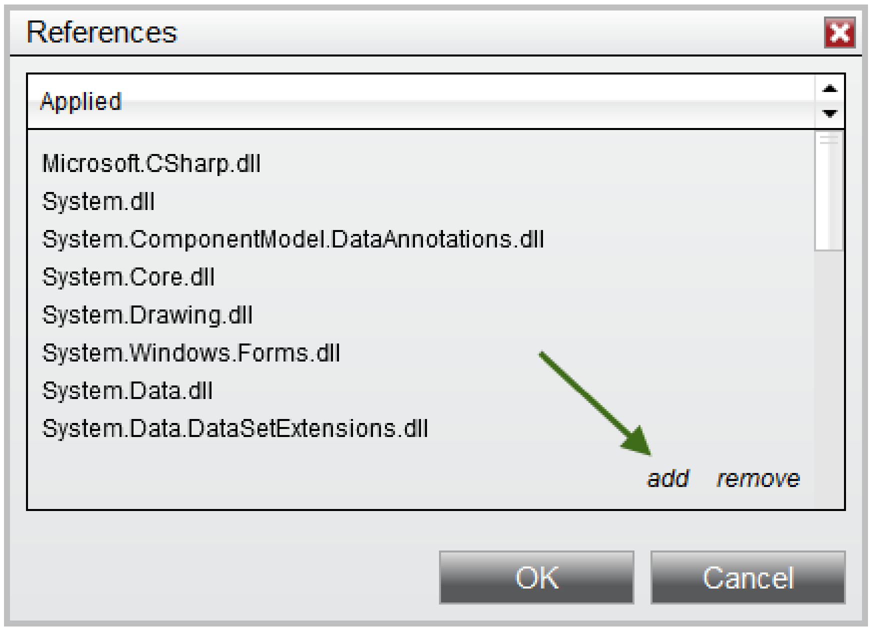
2. A **References** window will appear

3. Press the ***"add"*** and select the 3rd party vendor DLL file

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



Microsoft.CSharp.dll
System.dll
System.ComponentModel.DataAnnotations.dll
System.Core.dll
System.Drawing.dll
System.Windows.Forms.dll
System.Data.dll
System.Data.DataSetExtensions.dll

12.4.10 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
- **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)

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.

12.4.11 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();
```

12.4.12 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);
    }
}
```

Note:

12.4.13 Using Images and Geometry with Custom Icons

Custom Icon Overview

When overriding the Icon method in a Bars Type, 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;

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(p);
        return myCanvas;
    }
}
```

12.4.14 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 powerful the Microsoft DirectX API used for graphics processing 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 to which this reference is here to guide you through. 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.

Warning: Any **SharpDX Brush** 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.

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
SharpDX.Color3 dxColor3 = new SharpDX.Color3(255, 0,
0);

// create a 4 component color using rgb + alpha
(transparency)
SharpDX.Color4 dxColor4 = new SharpDX.Color4(dxColor3,
 50);

// 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 OnRenderTargetChange() 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.

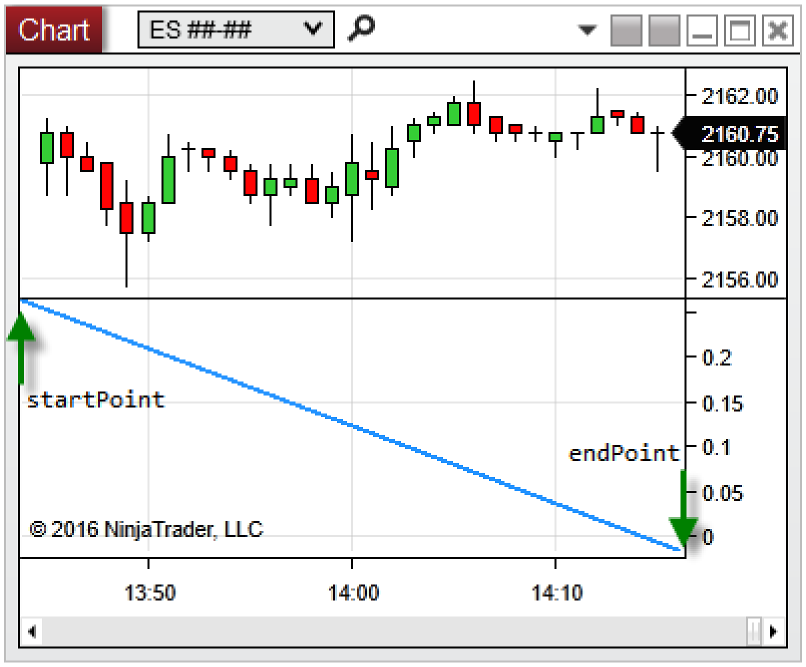
RenderTarget.DrawEllipse()	Draws the outline of the specified ellipse using the specified stroke style.
RenderTarget.DrawGeometry()	Draws the outline of the specified geometry using the specified stroke style.
RenderTarget.DrawLine()	Draws a line between the specified points.
RenderTarget.DrawRectangle()	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:

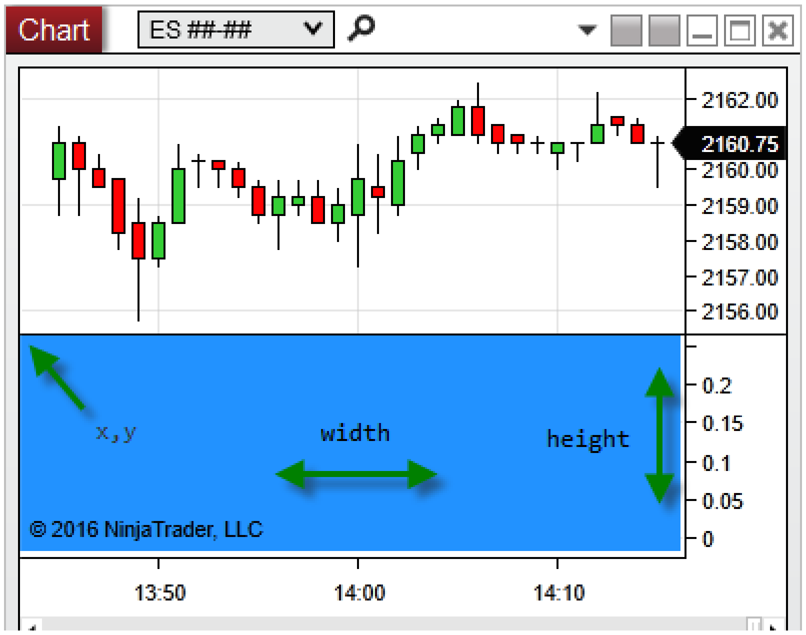
```
// create two vectors for the line to draw
SharpDX.Vector2 startPoint =  new
SharpDX.Vector2(ChartPanel.X, ChartPanel.Y);
SharpDX.Vector2 endPoint = new
SharpDX.Vector2(ChartPanel.X + ChartPanel.W, ChartPanel.Y
 + ChartPanel.H);

// 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.

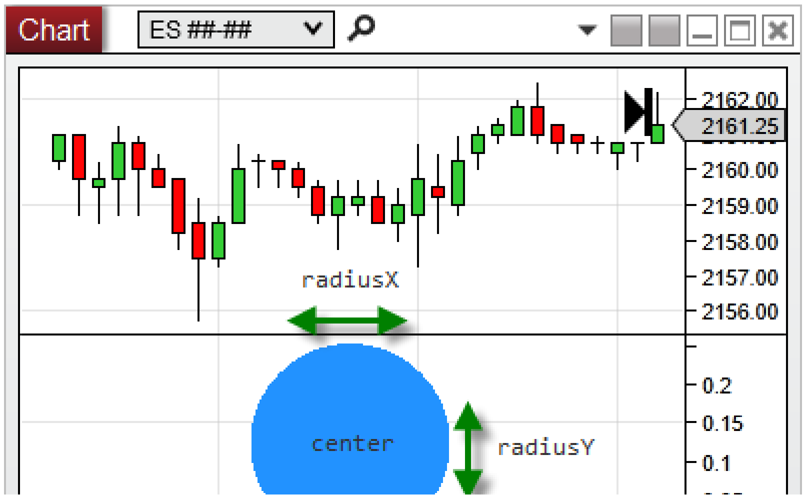
```
// create two vectors to position the rectangle
SharpDX.Vector2 startPoint =  new
SharpDX.Vector2(ChartPanel.X, ChartPanel.Y);
SharpDX.Vector2 endPoint = new
SharpDX.Vector2(ChartPanel.X + ChartPanel.W, ChartPanel.Y
 + ChartPanel.H);

// 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
```

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.

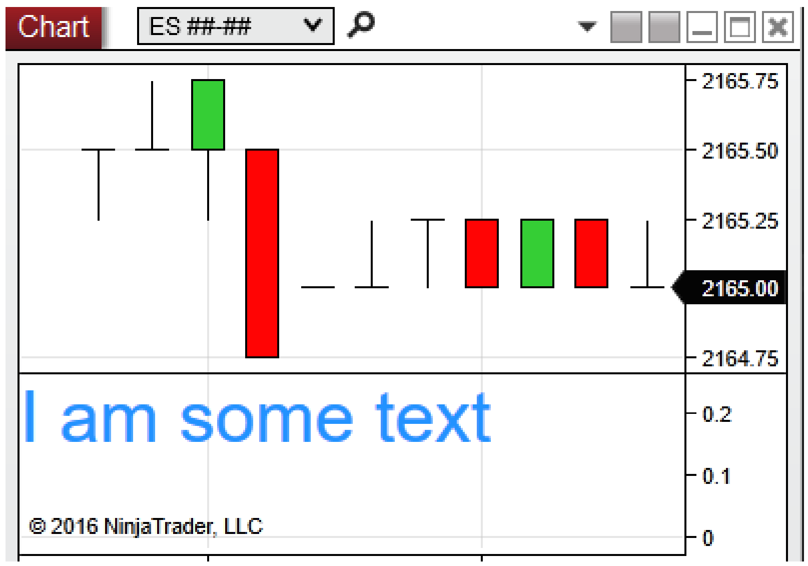
Warning: Any **SharpDX TextFormat** object used in your development must be disposed of after they have been used. NinjaTrader is **NOT** guaranteed to dispose of these resources for you! Please see the Best Practices for

SharpDX Resources section on this page for more information.

```
SharpDX.DirectWrite.TextFormat textFormat = new
SharpDX.DirectWrite.TextFormat(Core.Globals.DirectWrit
eFactory, "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.DirectWriteFa
ctory, "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.ToDirectWriteTextFor
mat();
```

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.

Warning: Any **SharpDX TextLayout** 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.

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.DirectWrit
eFactory, "I am also some text", textFormat,
ChartPanel.W, ChartPanel.H);
```

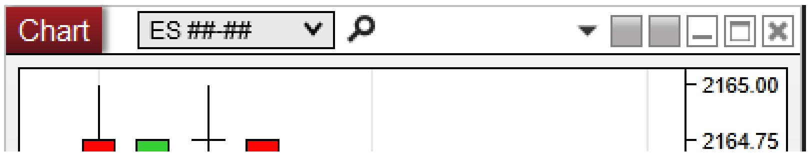
After the text has its format and layout, you can use the RenderTarget.DrawTextLayout() method to specify the exact location as a Vector2, as well as the Brush used to draw the text.

```
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.DirectWriteFa
ctory, "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.DirectWriteFa
ctory, "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
.D2DFactory, 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.

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.D2
DFactory,
              dxStrokeStyleProperties, dashes);
   }
   else if (State == State.Terminated)
   {
      // make sure to dispose of stroke style when
finished
      if (dxStrokeStyle != null)
      {
```

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 OnRenderTargetChange() 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(RenderTarge
t));
```

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(RenderTarg
et);
SharpDX.Direct2D1.Brush dxBrush2 =
System.Windows.Media.Brushes.Red.ToDxBrush(RenderTarge
t);
SharpDX.Direct2D1.Brush dxBrush3 =
System.Windows.Media.Brushes.Green.ToDxBrush(RenderTar
get);
SharpDX.Direct2D1.Brush dxBrush4 =
System.Windows.Media.Brushes.Purple.ToDxBrush(RenderTa
rget);
SharpDX.Direct2D1.Brush dxBrush5 =
System.Windows.Media.Brushes.Orange.ToDxBrush(RenderTa
rget);
SharpDX.Direct2D1.Brush dxBrush6 =
System.Windows.Media.Brushes.Yellow.ToDxBrush(RenderTa
rget);
```

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;
```

12.4.15 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.

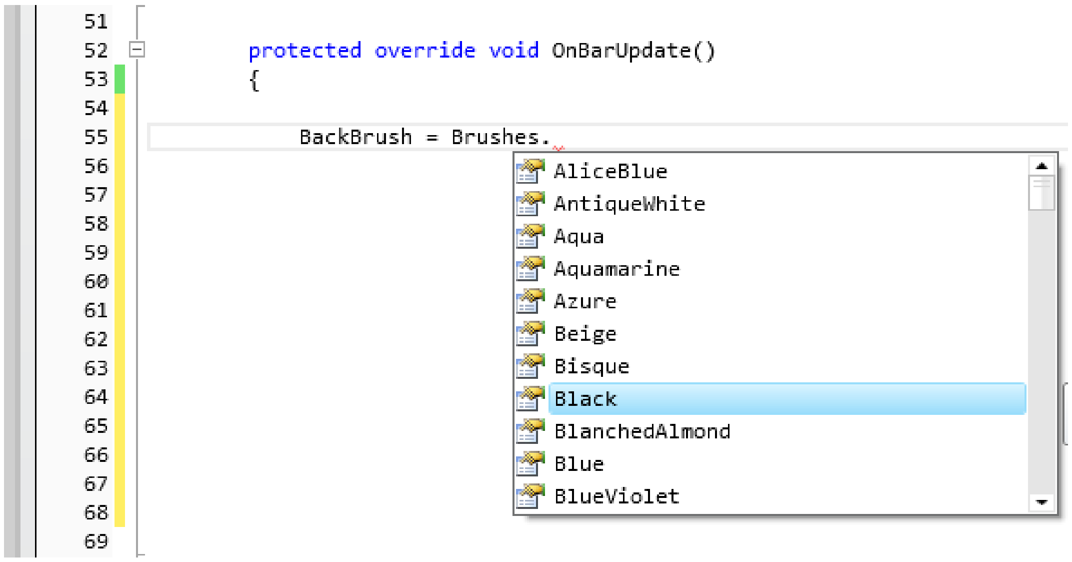
Notes: The following document is written in sequential fashion, starting with the most simple concepts, to the more advance topics. The majority of the brushes discussed in this document will be referred to as **"WPF" brushes** which exist in the System.Windows.Media namespace, however there are also **"SharpDX" brushes** which exist in the 3rd party SharpDX.Direct2D1 nampspace 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*.



```
        protected override void OnBarUpdate()
        {

            BackBrush = Brushes.
```

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.

```
// 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.

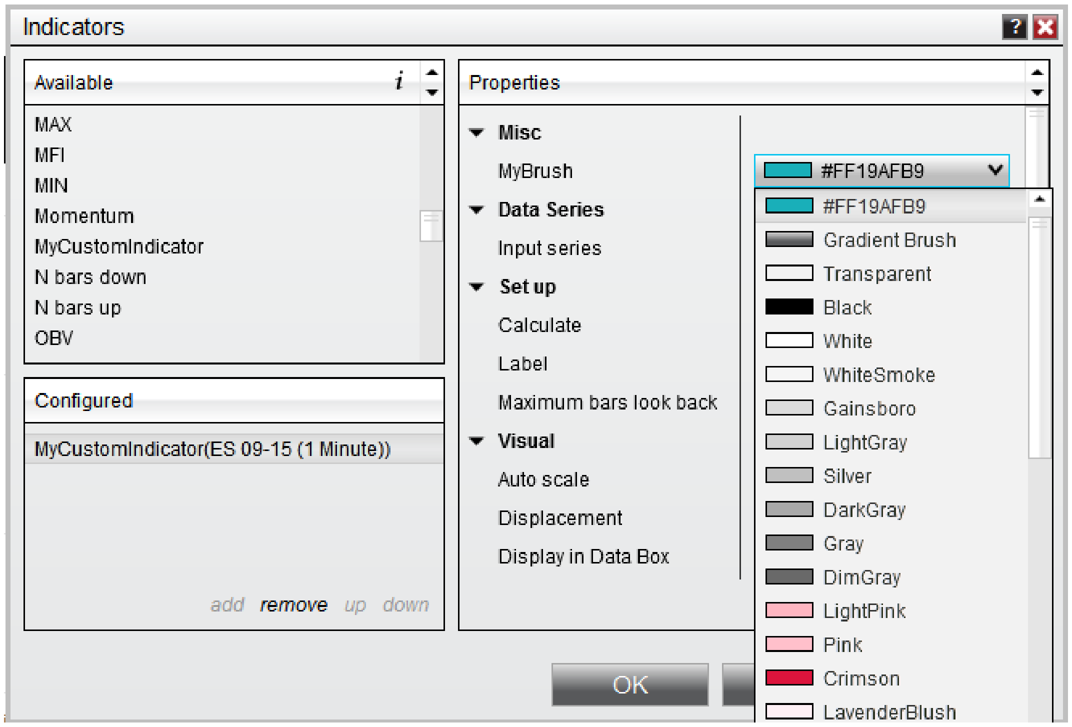
```
[Browsable(false)]
public string MyBrushSerialize
{
   get { return Serialize.BrushToString(MyBrush); }
   set { MyBrush = Serialize.StringToBrush(value); }
}
```

Tip: For a complete example of **User Definable Color Inputs**, please see the reference sample on our support forum

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

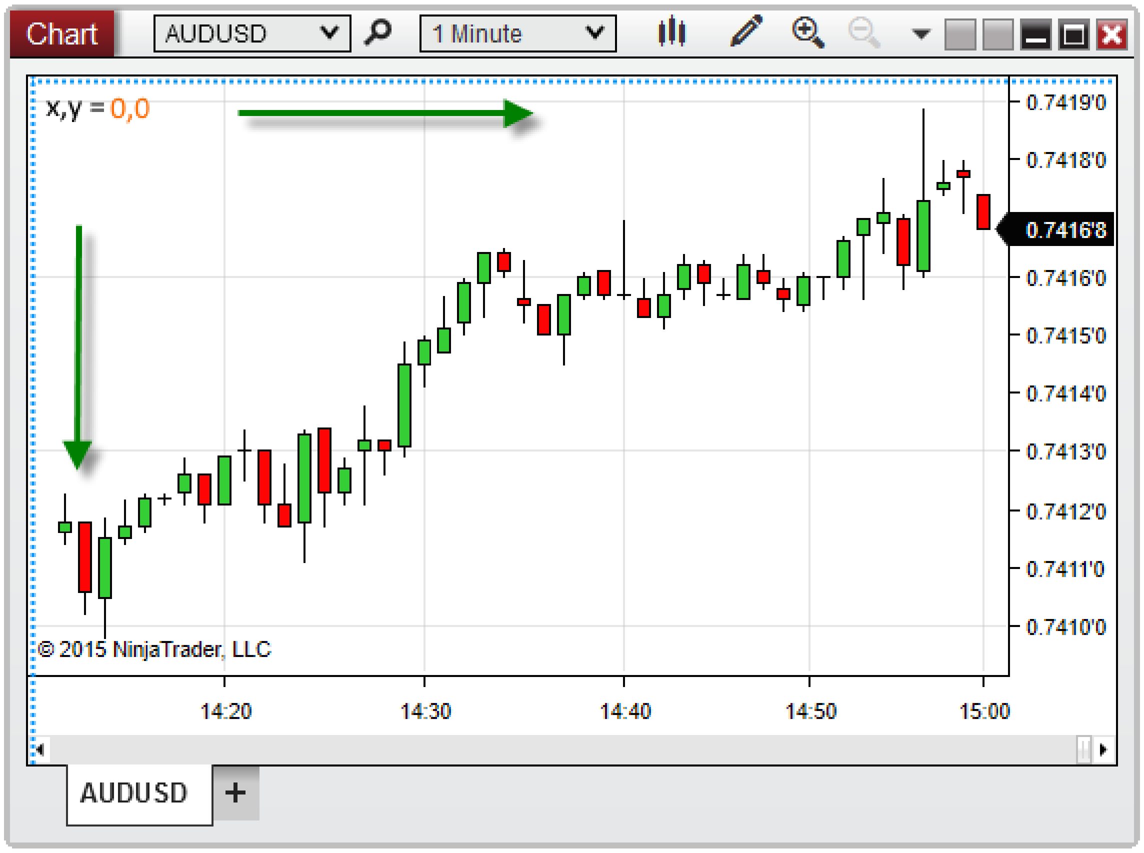
In addition to the SolidColorBrush object demonstrated on this page, the .NET Framework provides more complex brushes which have more attributes than just filling an area with a solid color. Information on these special types of brushes can be found on the MSDN website: LinearGradientBrush, RadialGradientBrush, ImageBrush.

These complex types also have an equivalent found in the **SharpDX SDK Reference**: SharpDX.Direct2D1.LinearGradientBrush, SharpDX.Direct2D1.RadialGradientBrush

12.4.16 Working with Chart Object Coordinates

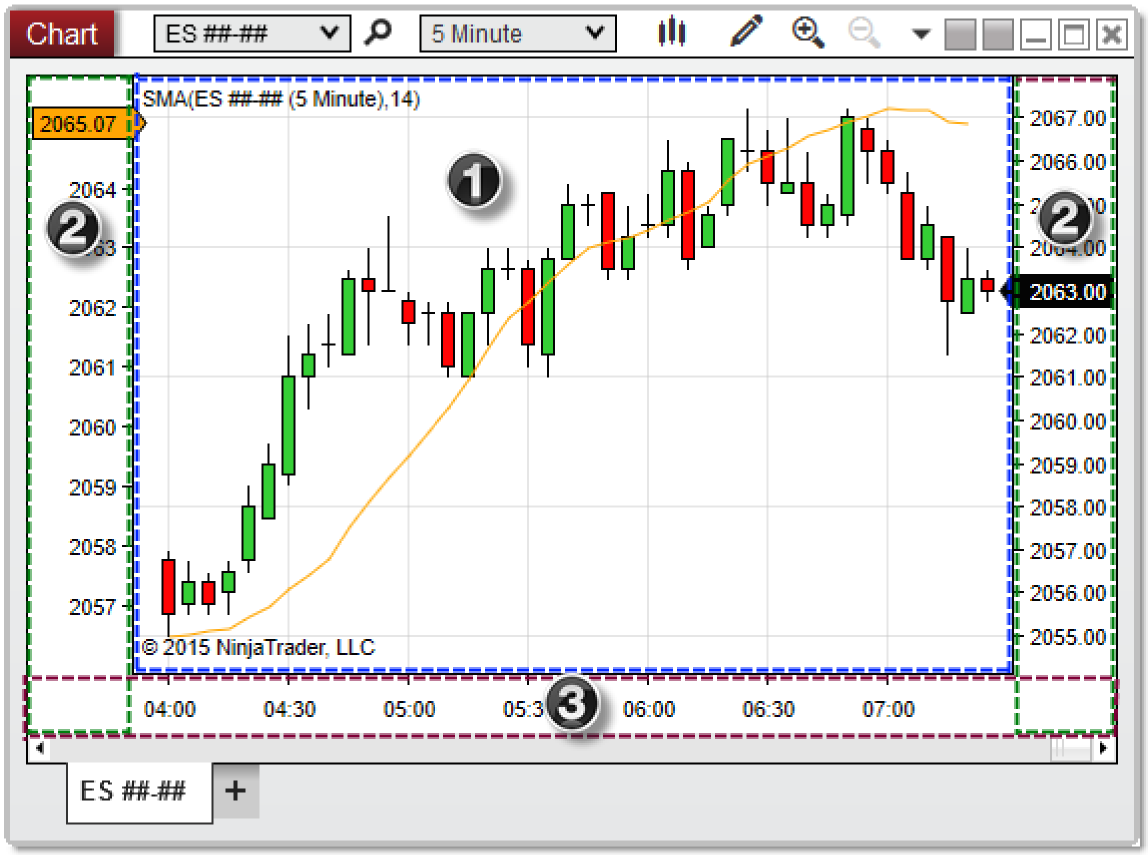
Understanding Chart Canvas Coordinates

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.

12.4.17 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);
}
```

12.4.18 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.

12.5 Language Reference

NinjaScript Language Reference

12.5.1 Alphabetical Reference

12.5.2 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,	Documents methods for triggering alerts, printing debug

Debug, Share	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
Instruments	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	Documents syntax and return values for system indicator methods

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

12.5.2.1 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()	Similar to the AddDataSeries() method for adding Bars objects, this method adds a Renko 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)
```

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 into 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"
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 });`

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 09-14
contract - BarsInProgress index = 2
          AddDataSeries("ES 09-14", 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();
}
```

12.5.2.1.1 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 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**
- 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 into 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", 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
09-14 contract - BarsInProgress index = 1
       AddHeikenAshi("ES 09-14", BarsPeriodType.Minute, 1,
MarketDataType.Last);
   }
}

protected override void OnBarUpdate()
{
     // Ignore the primary Bars object and only process the
Kagi Bars object
     if (BarsInProgress == 1)
     {
          // Do something;
     }
}
```

12.5.2.1.2 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 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**
- 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 into 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 09-14
contract - BarsInProgress index = 1
        AddKagi("ES 09-14",  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;
     }
}
```

12.5.2.1.3 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 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**
- 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 into 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 09-
14 - BarsInProgress index = 1
      AddLineBreak("ES 09-14", 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;
     }
}
```

12.5.2.1.4 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 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**
- 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 into 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 09-14 contract - BarsInProgress index = 1
          AddPointAndFigure("ES 09-14", 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;
     }
}
```

12.5.2.1.5 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 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**
- 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 into 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.
isTickReplay	Determines if the Bars object uses Tick Replay * Please see the article here on using Tick Replay

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 09-14
contract - BarsInProgress index = 1
            AddRenko("ES 09-14", 2, MarketDataType.Last);
      }
}

protected override void OnBarUpdate()
{
      // Ignore the primary Bars object and only process the
Renko Bars object
      if (BarsInProgress == 1)
      {
            // Do something;
      }
}
```

12.5.2.1.6 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();
}
```

12.5.2.1.7 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 the Initialize()
          // Do something within the context of the 5 minute
Bars
     }
}
```

12.5.2.1.8 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);
}
```

12.5.2.1.9 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;

     // Script logic calculation code...
}
```

Strategy (BarsRequiredToTrade)

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // 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;

          // Script logic calculation code...
          //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 strategy 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;

     // Script logic calculation code...
}
```

12.5.2.2 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.

12.5.2.2.1 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);
}
```

12.5.2.2.2 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();
   }
}
```

12.5.2.2.3 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);
```

12.5.2.2.4 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.
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"
```

12.5.2.2.5 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
}
```

12.5.2.2.6 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);
   }
}
```

12.5.2.2.7 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");
}
```

12.5.2.2.8 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.
> 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");
```

12.5.2.2.9 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

```
// sending a screen shot to Twitter
Share("Twitter", "Check out this setup!", @"C:\MyImages
\chart03.PNG");

// using "args" as the Mail "To" and "Subject" parameters
Share("Gmail", "Test Message", new object[]
{ "example@test.com", "Test Subject Line" });

// using "args" as the StockTwit "Sentiment" parameter
Share("StockTwits", "Test Message", new object[]
{ ShareServices.StockTwits.Sentiment.Bearish });
```

12.5.2.3 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

12.5.2.3.1 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();
```

12.5.2.3.2 CrossAbove()

Definition

Evaluates a cross above condition over the specified bar look-back period.

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

```
// 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();
```

12.5.2.3.3 CrossBelow()

Definition
Evaluates a cross below condition over the specified bar look-back period.

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 above
20 EMA within the last 5 bars
   if (Close[0] < Open[0] && CrossBelow(EMA(10), EMA(20), 5))
        EnterShort();
}
```

12.5.2.3.4 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
   }
}
```

12.5.2.3.5 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
   }
}
```

12.5.2.3.6 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
   }
}
```

12.5.2.3.7 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
   }
}
```

12.5.2.3.8 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);
}
```

12.5.2.3.9 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));

}
```

12.5.2.3.10 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();
}
```

12.5.2.3.11 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();
}
```

12.5.2.3.12 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 1.
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)

12.5.2.3.13 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));
}
```

12.5.2.3.14 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 1.
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)

12.5.2.3.15 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));

}
```

12.5.2.3.16 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.

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);
```

12.5.2.3.17 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
   }
}
```

12.5.2.3.18 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
}
```

12.5.2.4 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()**.

12.5.2.4.1 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
```

12.5.2.4.2 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
}
```

12.5.2.4.3 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
```

12.5.2.4.4 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.

> **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

```
[XmlIgnore] // cannot serialize type of TimeSpan, use the
BeginTimeSpanSerialize object to persist properties
[Browsable(false)] // prevents this property from showing up
on the UI
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
[NinjaScriptProperty]
[Display(Name = "Begin TimeSpan", GroupName =
"NinjaScriptStrategyParameters", Order = 1)]
public string BeginTimeSpanSerialize
{
   get { return BeginTimeSpan.ToString(); }
   set { BeginTimeSpan = TimeSpan.Parse(value); }
}
```

12.5.2.4.5 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
```

12.5.2.4.6 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

12.5.2.5 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

12.5.2.5.1 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
   }
}
```

12.5.2.5.2 GetAsk()

Definition

Returns the ask price value at a selected absolute bar index value.

Note: 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().

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));
    }
}
```

12.5.2.5.3 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());
}
```

12.5.2.5.4 GetBid()

Definition

Returns the bid price value at a selected absolute bar index value.

Note: 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().

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

© 2017 NinjaTrader, LLC

```
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));
   }
}
```

12.5.2.5.5 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);
   }
}
```

12.5.2.5.6 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);
}
```

12.5.2.5.7 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);
   }
}
```

12.5.2.5.8 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);
   }
}
```

12.5.2.5.9 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);
   }
}
```

12.5.2.5.10 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);
   }
}
```

12.5.2.5.11 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);
   }
}
```

12.5.2.5.12 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);
   }
}
```

12.5.2.5.13 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]));
}
```

12.5.2.5.14 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));
      }
   }
}
```

12.5.2.5.15 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.IsFirstBarOfSession
```

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]));
}
```

12.5.2.5.16 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
   }
}
```

12.5.2.5.17 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);
   }
}
```

12.5.2.5.18 PercentComplete

Definition

Returns a value indicating the percentage complete of the real-time bar processing.

Notes:
1. Since a historical bar is complete, a value of 1 always returns during **State.Historical** (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);
   }
}
```

12.5.2.5.19 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());
```

12.5.2.5.20 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)

}
```

12.5.2.6 Charts

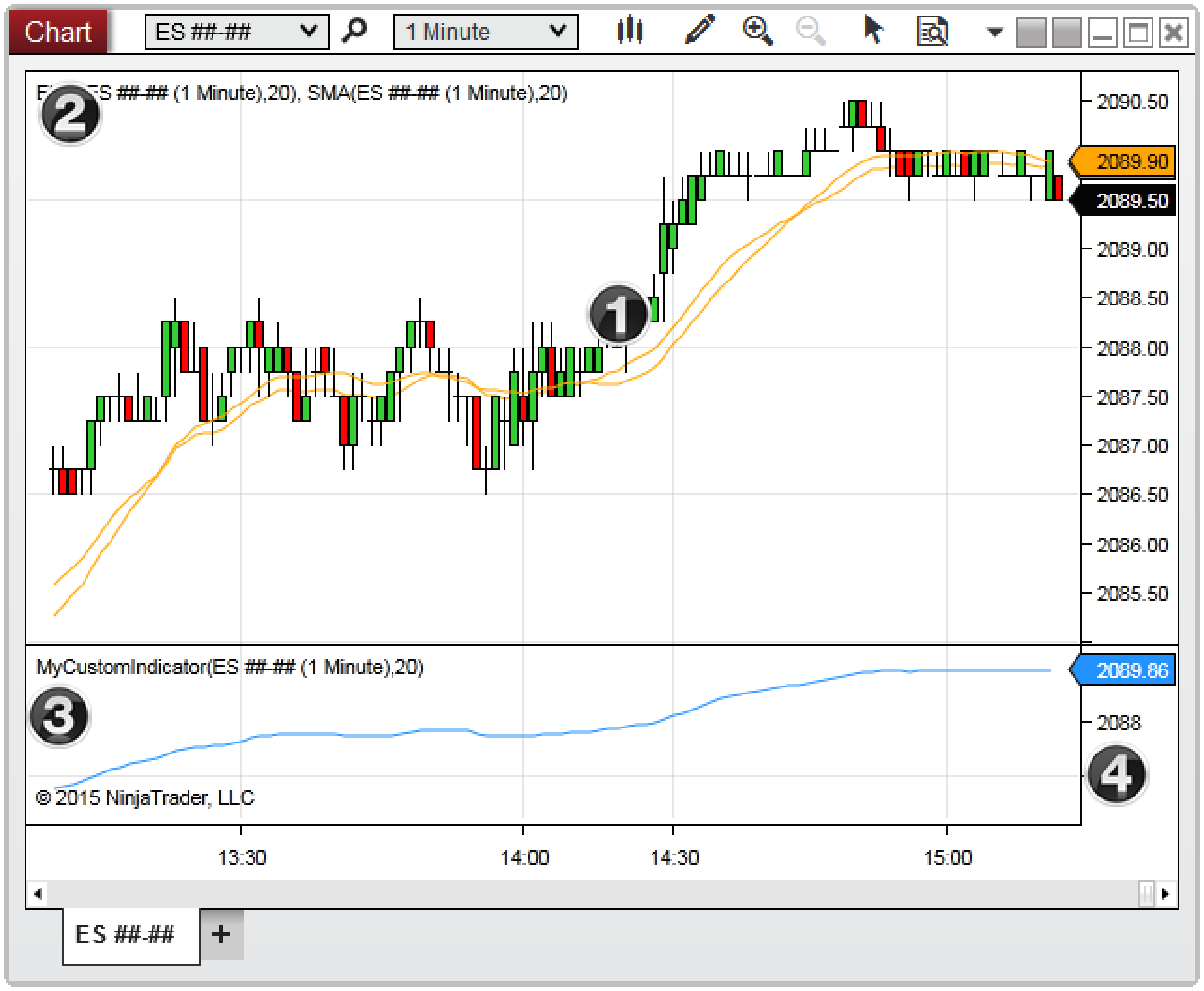
The following section covers information related to accessing chart related data, such as ChartControl, ChartBars, ChartScales, and ChartPanels, and advanced Indicator Rendering.

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	

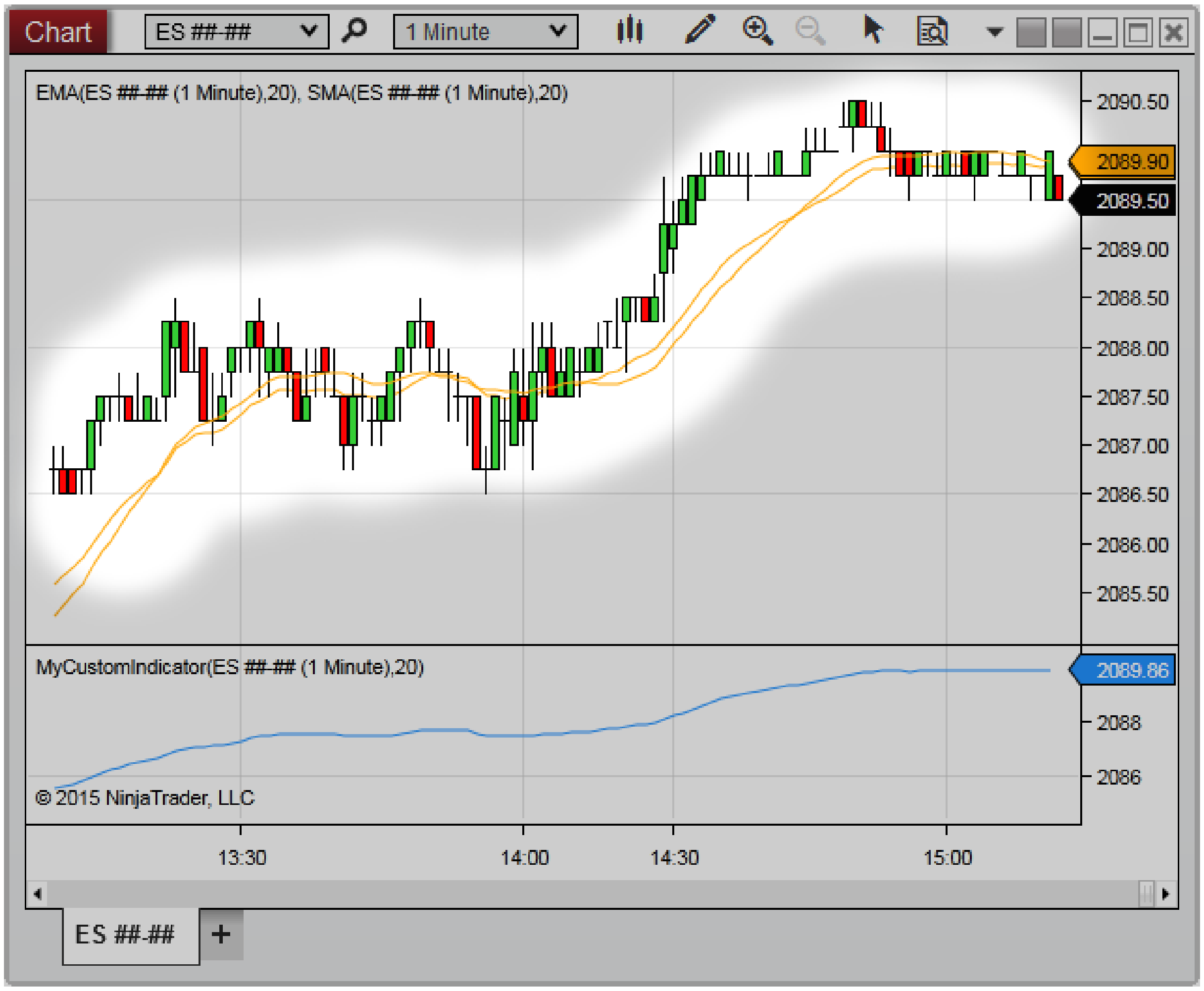
A chart's objects can be broken down into the four following areas:



12.5.2.6.1 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;
      }
   }
}
```

12.5.2.6.1.1 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);
   }
}
```

12.5.2.6.1.2 Count

Definition

The total number of ChartBars that exist on the chart.

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
   }
}
```

12.5.2.6.1.3 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));
      }
   }
}
```

12.5.2.6.1.4 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]));

   }
}
```

12.5.2.6.1.5 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(chartC
ontrol.PresentationSource);

   // convert mouse point to bar index
   int barIdx = ChartBars.GetBarIdxByX(chartControl,
mousePoint);

   Print("User clicked on Bar #" + barIdx);
}
```

12.5.2.6.1.6 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
   }
}
```

12.5.2.6.1.7 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`

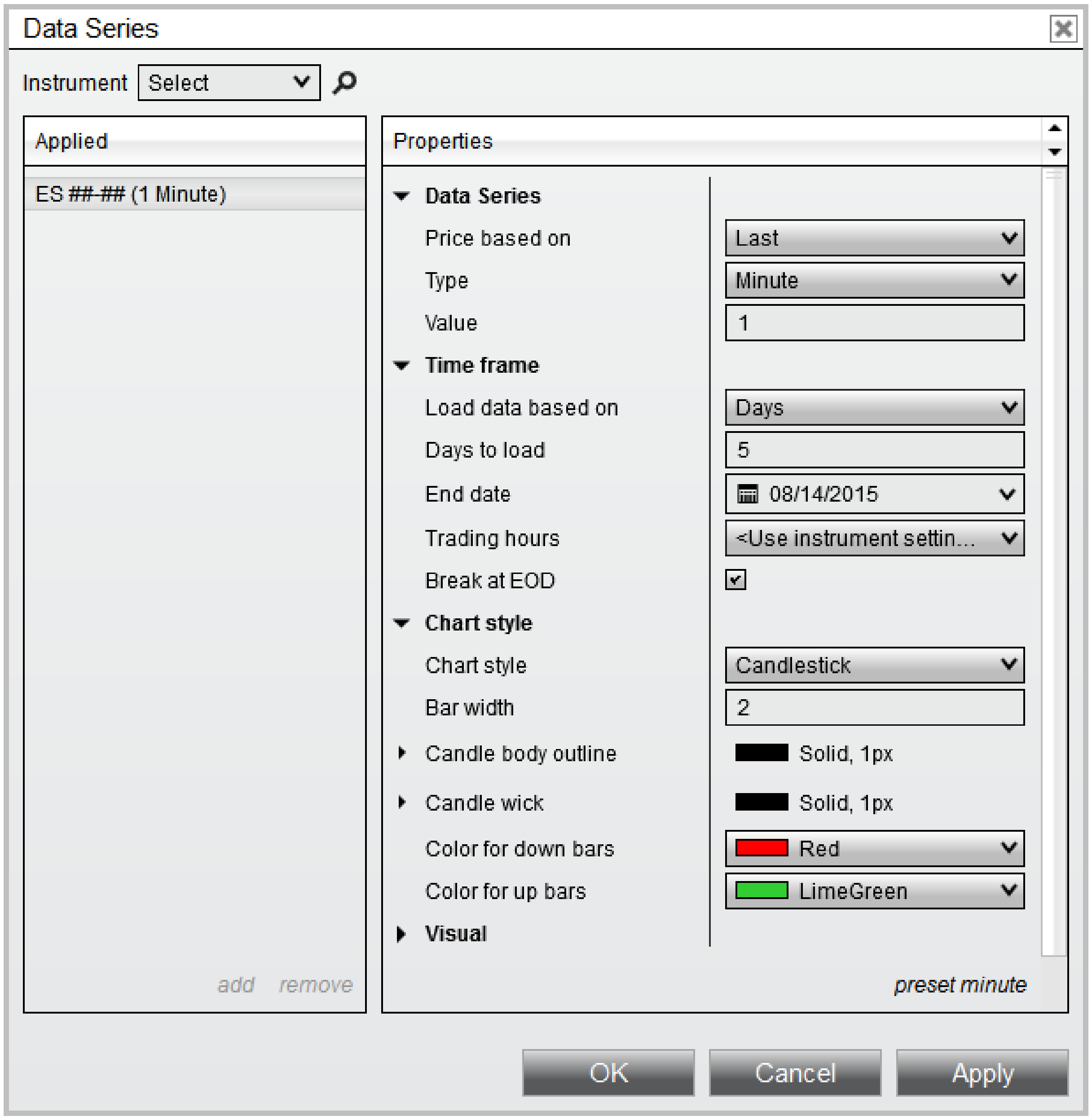
Examples

```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
   Print("ChartBars reside on panel index: " +
ChartBars.Panel);
   // Output:  ChartBars reside on panel index: 0
}
```

12.5.2.6.1.8 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.



ES ##-## (1 Minute)

Data Series	
Price based on	Last
Type	Minute
Value	1
Time frame	
Load data based on	Days
Days to load	5
End date	08/14/2015
Trading hours	<Use instrument settin...
Break at EOD	✓
Chart style	
Chart style	Candlestick
Bar width	2
Candle body outline	Solid, 1px
Candle wick	Solid, 1px
Color for down bars	🟥 Red
Color for up bars	🟩 LimeGreen
Visual	

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.TextAndMark

	er
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

12.5.2.6.1.9 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)
}
```

12.5.2.6.1.10 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`

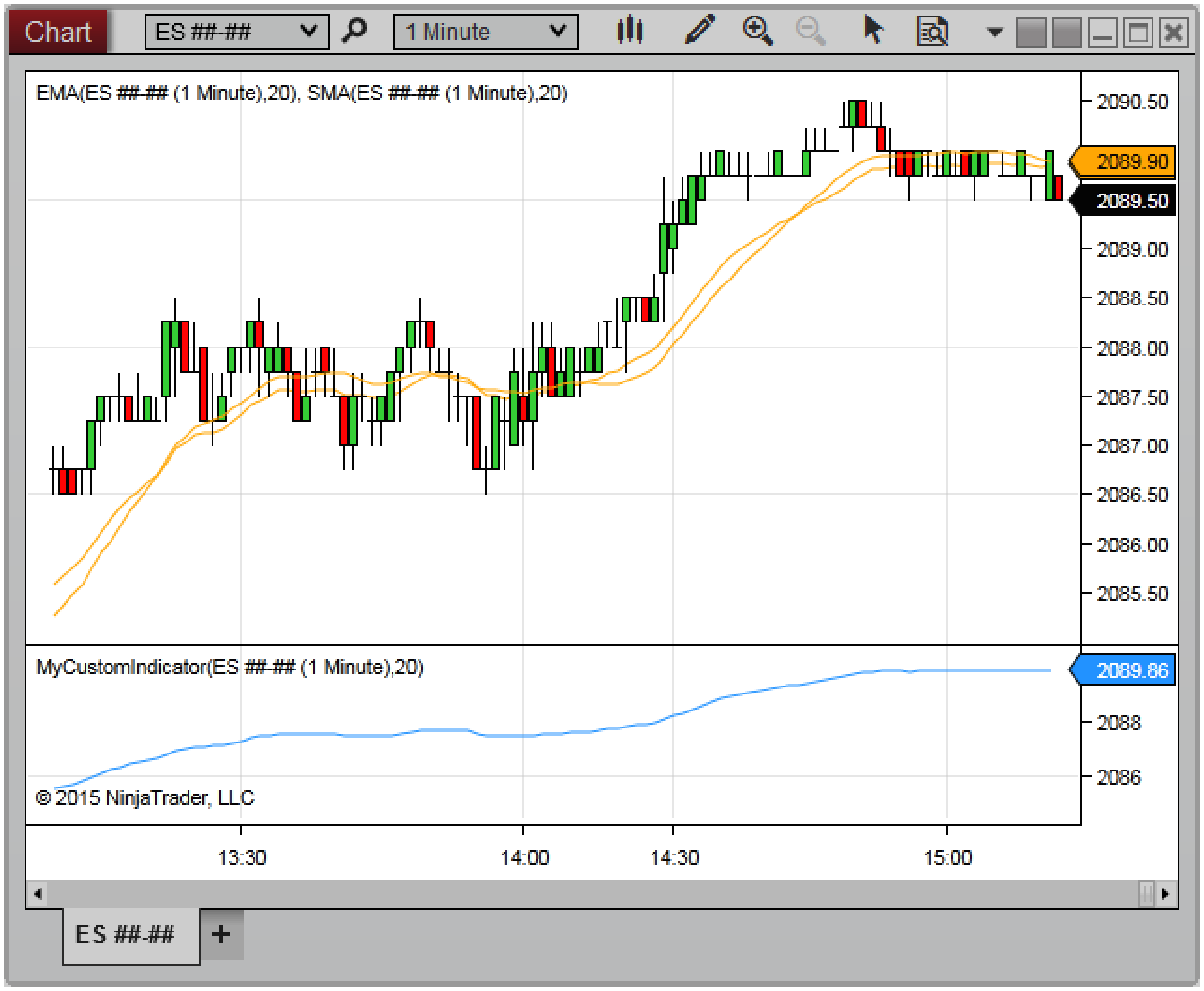
Examples

```
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));
      }
   }
}
```

12.5.2.6.2 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

12.5.2.6.2.1 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.

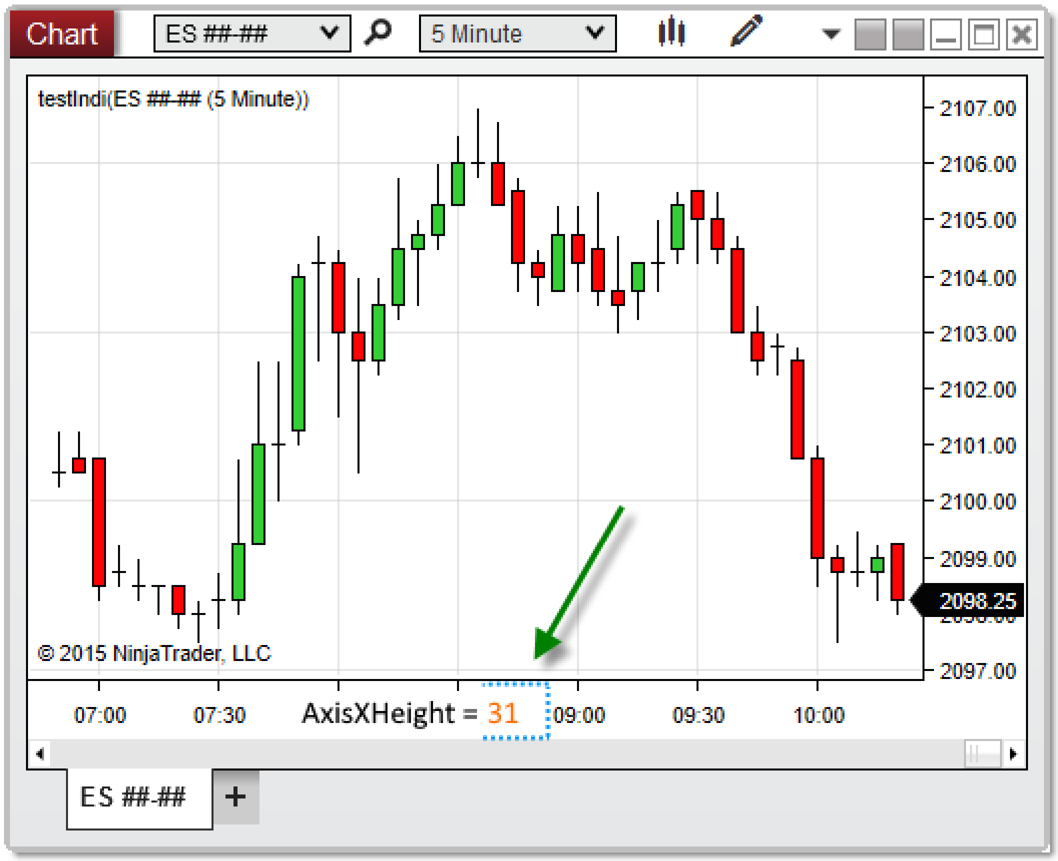
Property Value
A `double` representing the number of pixels separating the x-axis and the top of the horizontal scroll bar on the chart.

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.



12.5.2.6.2.2 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.

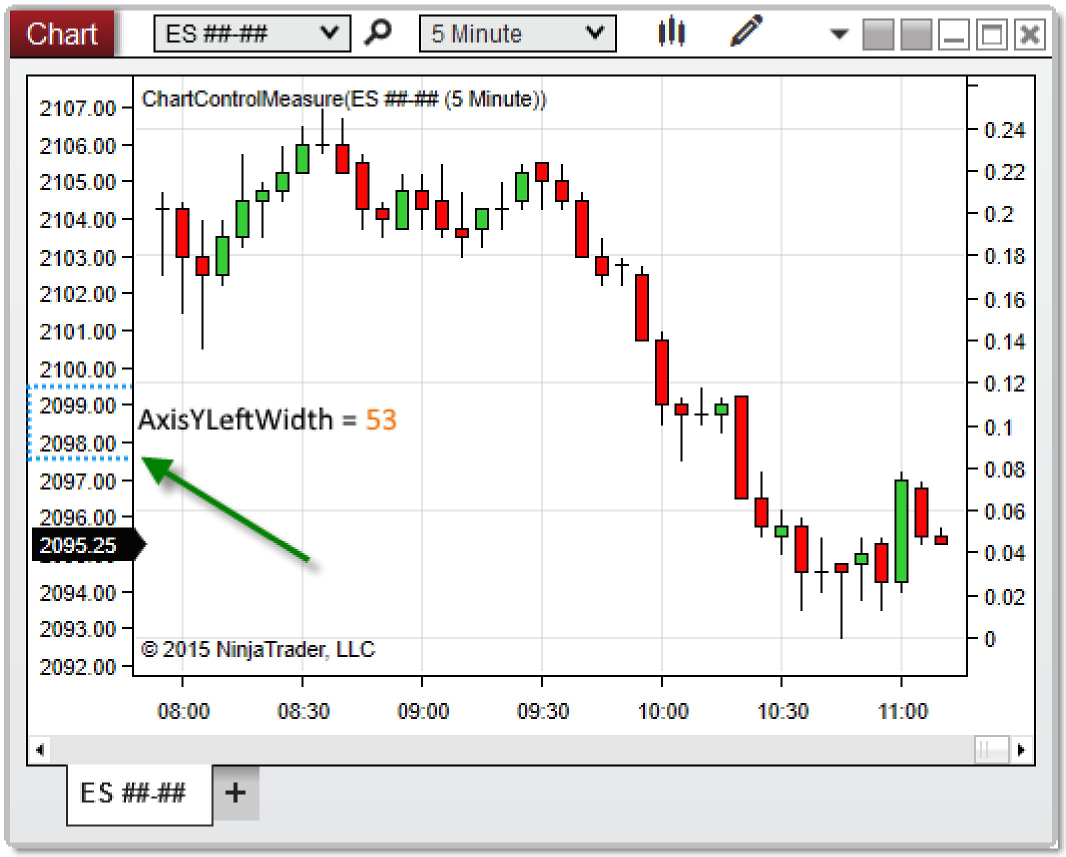
Syntax

```
<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.

12.5.2.6.2.3 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.

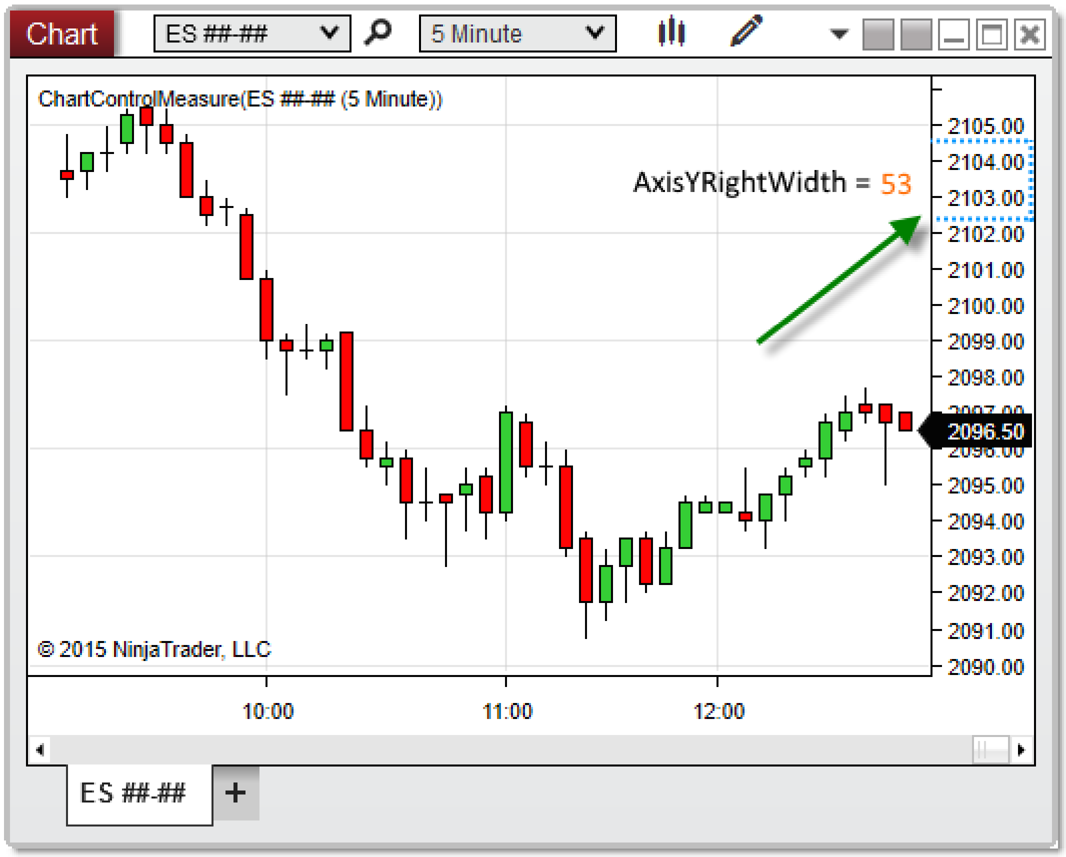
Syntax

```
<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.

12.5.2.6.2.4 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.

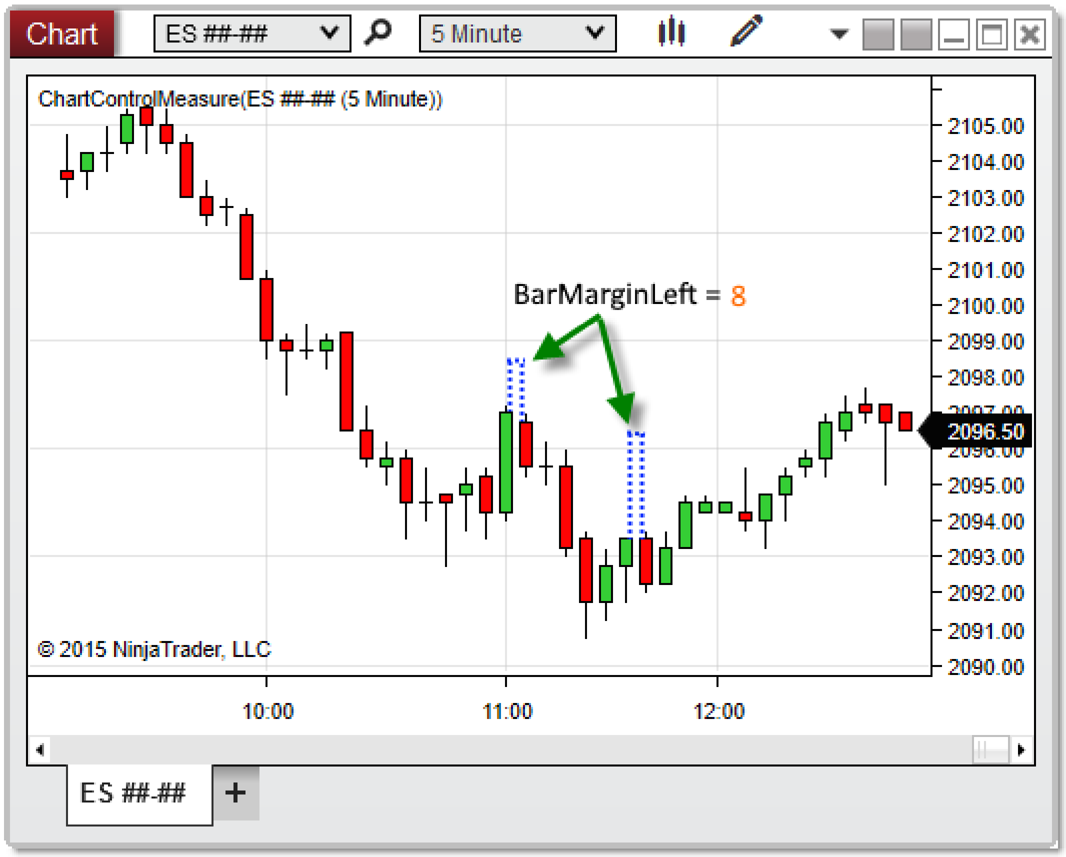
Syntax

<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.



12.5.2.6.2.5 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<ChartB
ars> 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);
    }
}
```

12.5.2.6.2.6 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:

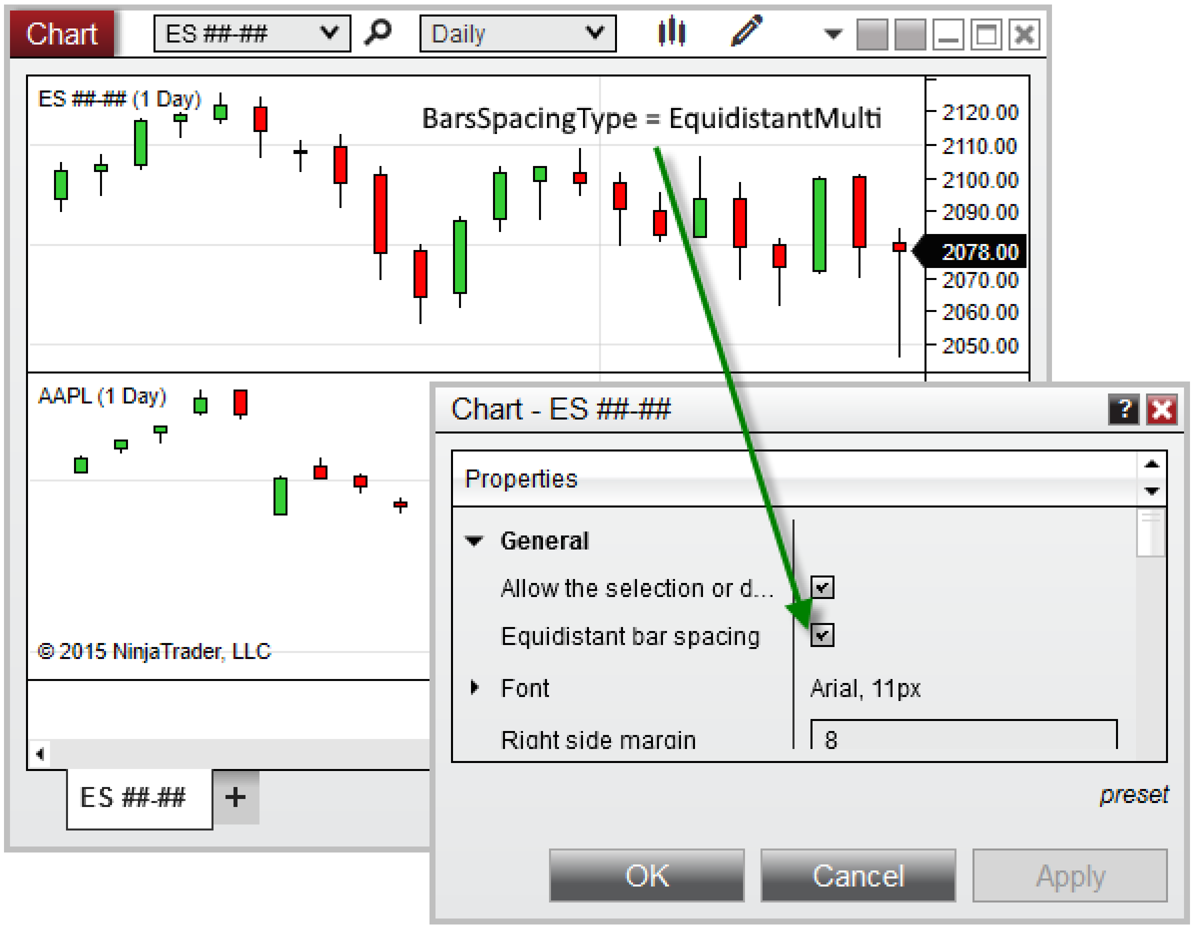
EquidistantSingle	Indicates Equidistant Bar Spacing is used, and only one Bars object exists on the chart
EquidistantMulti	Indicates Equidistant Bar Spacing is used, and more than one Bars objects exist on the chart
TimeBased	Indicates Time-Based bar spacing is used

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:



BarsSpacingType = EquidistantMulti

Allow the selection or d...	✓
Equidistant bar spacing	✓
Font	Arial, 11px
Right side margin	8

12.5.2.6.2.7 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.

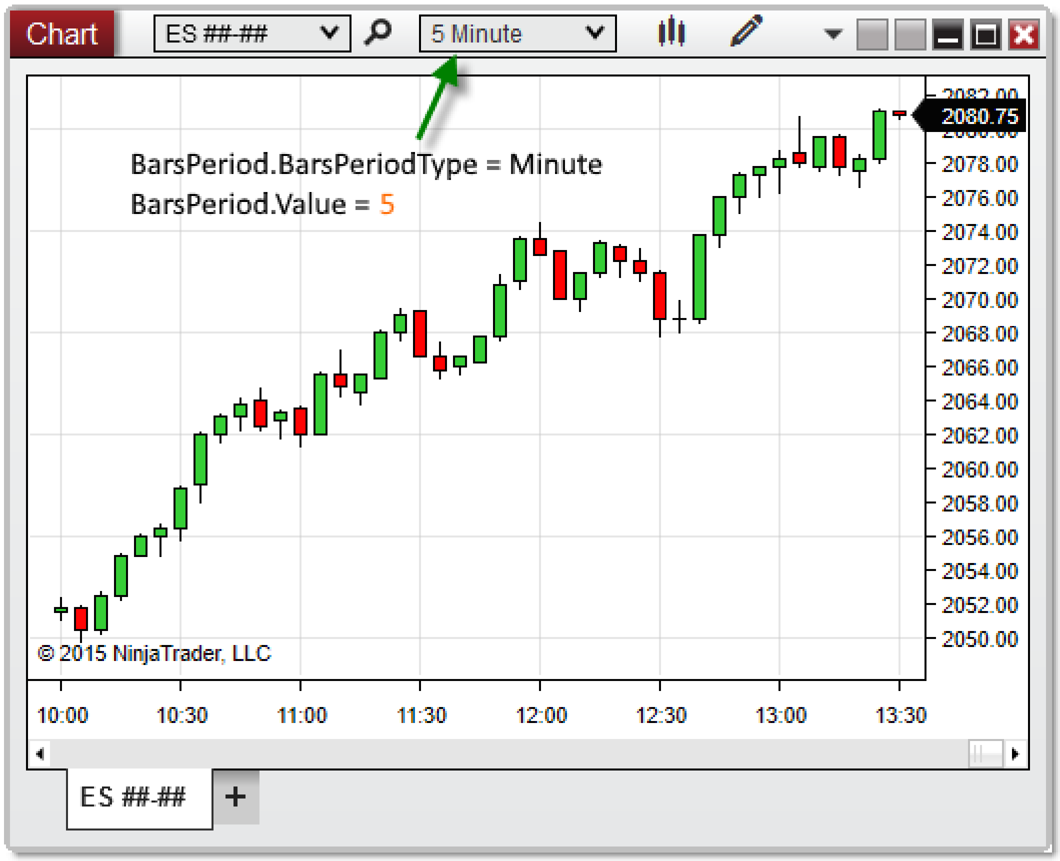
Syntax

`<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.


BarsPeriod.BarsPeriodType = Minute
BarsPeriod.Value = 5

12.5.2.6.2.8 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.

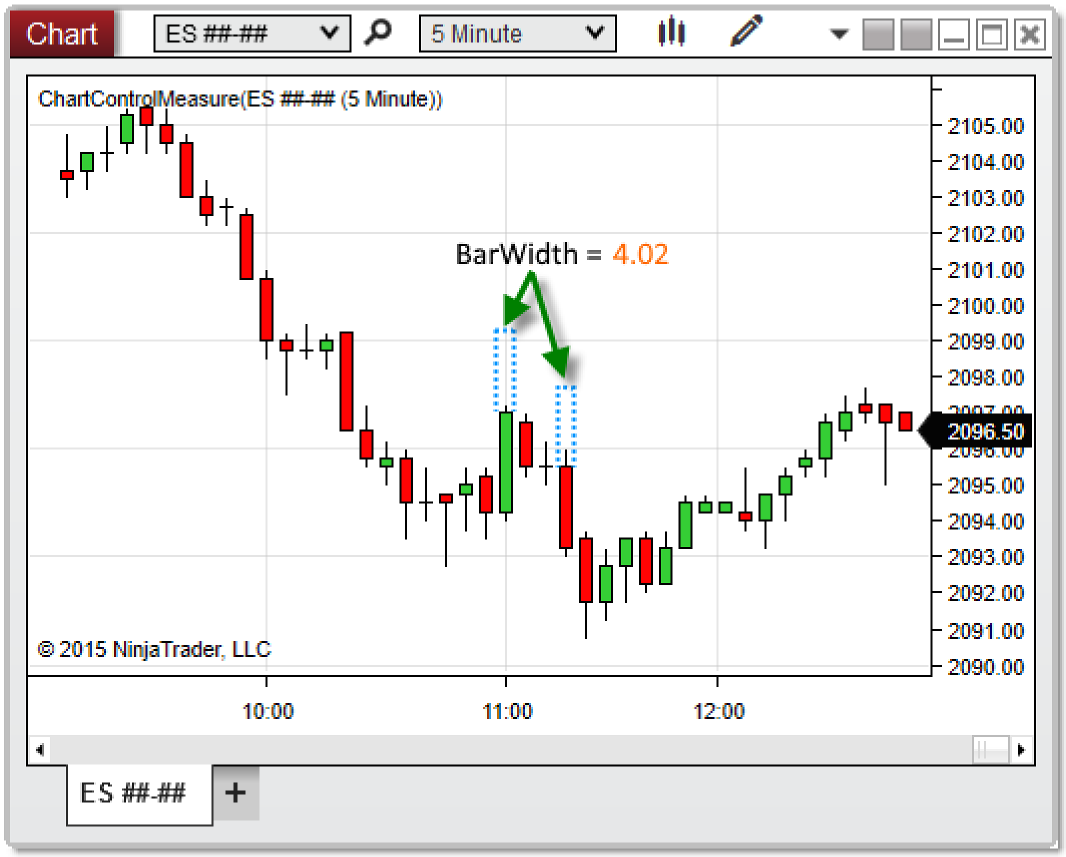
Syntax

`<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.



12.5.2.6.2.9 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);
    }
}
```

12.5.2.6.2.10 CanvasLeft

Definition
Indicates the x-coordinate (in pixels) of the beginning of the chart canvas area.

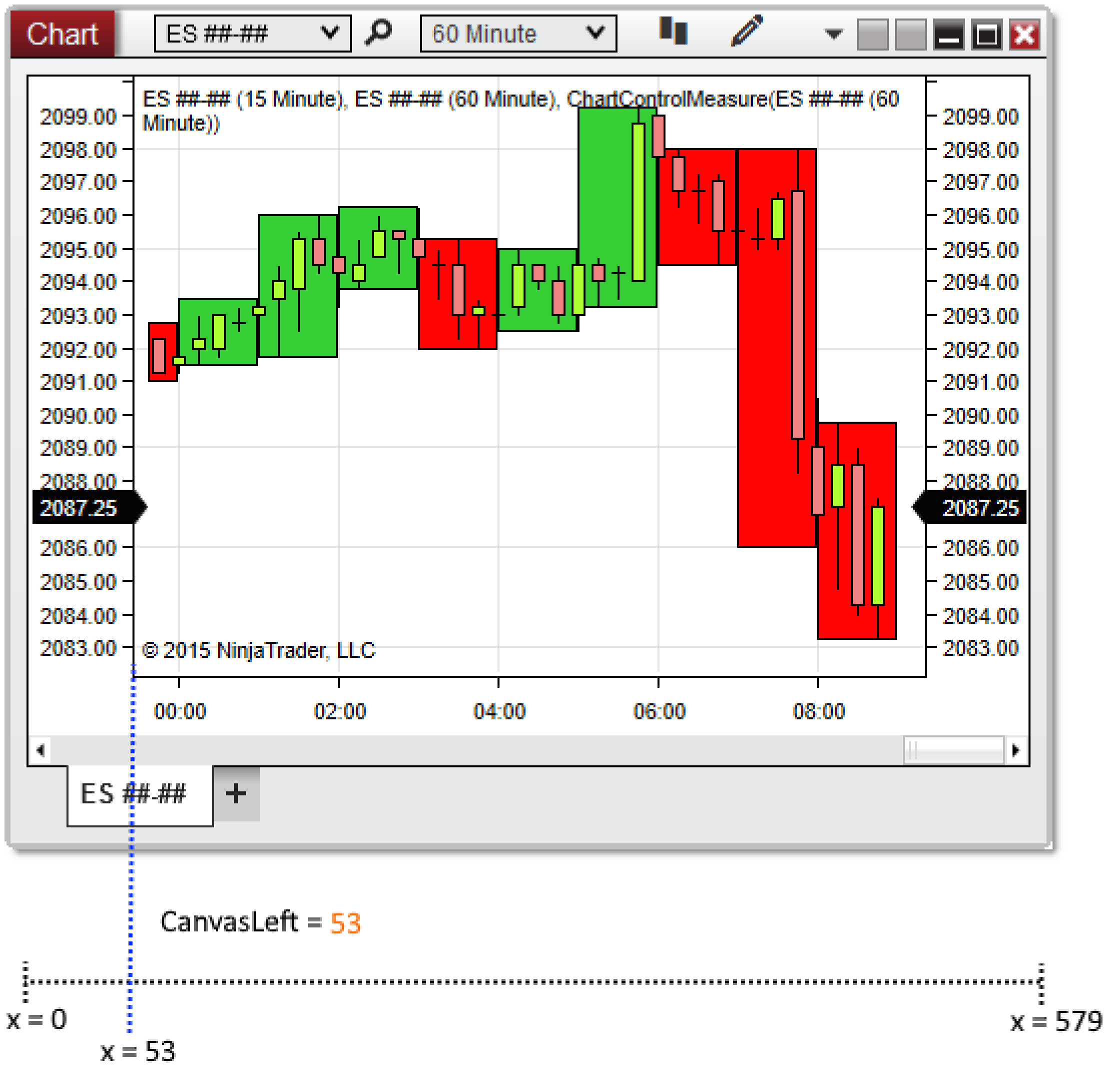
Property Value
A `double` representing the beginning of the chart canvas area.

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.

12.5.2.6.2.11 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.

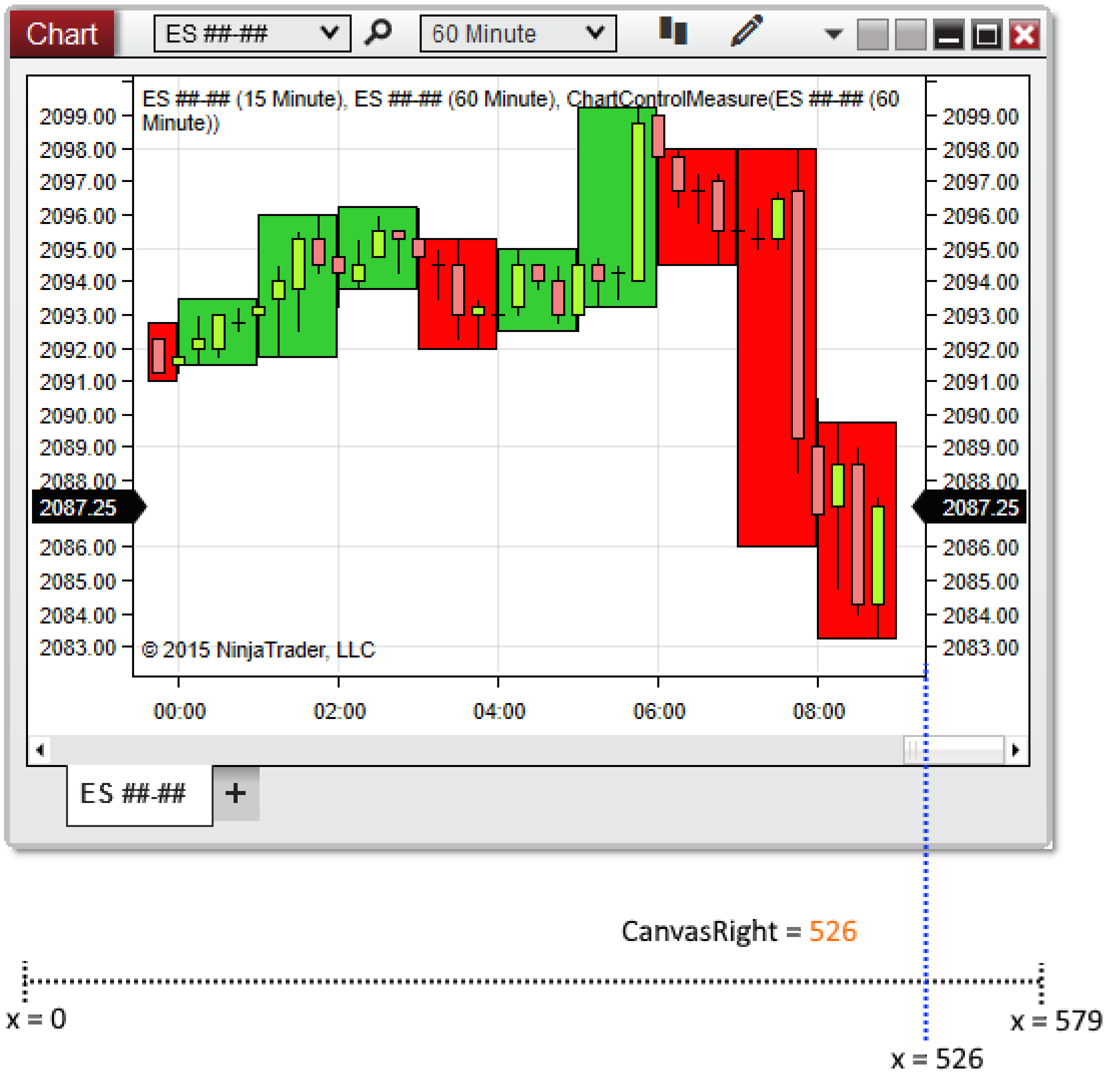
Syntax

<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.



12.5.2.6.2.12 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
Selecte	The Zoom tool is selected, but has not yet been used to

d	zoom in
Drawing Rectan gle	The Zoom tool is currently in use (User is currently drawing the rectangle in which to zoom)

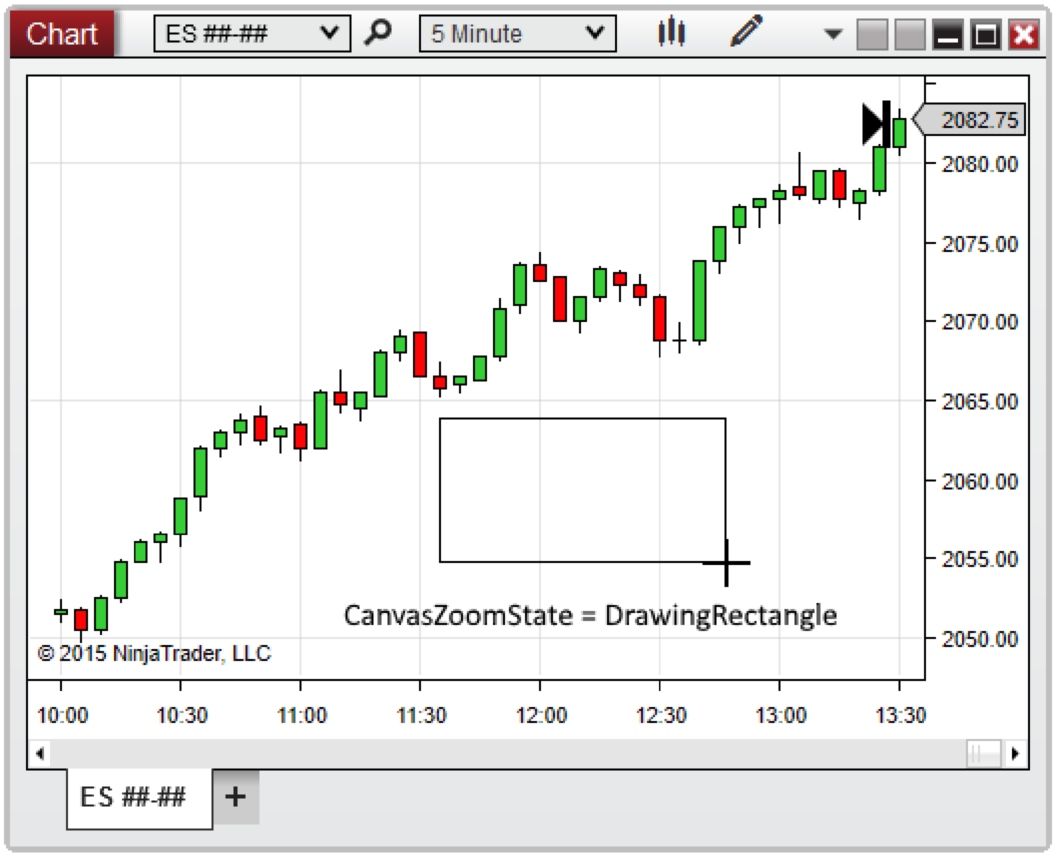
Syntax

<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:



12.5.2.6.2.13 ChartPanels

Definition

Holds a collection of ChartPanel objects containing information about the panels active on the chart.

Property Value

An ObservableCollection of `ChartPanel` objects

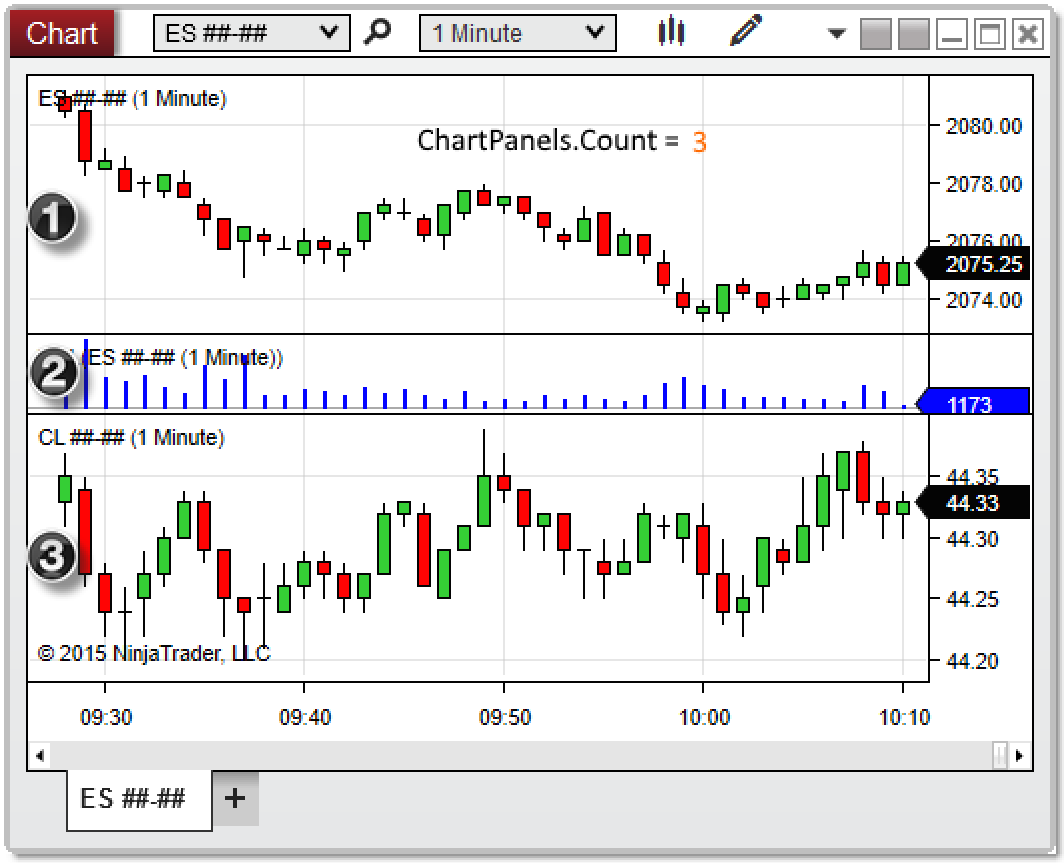
Syntax

<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.



12.5.2.6.2.14 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

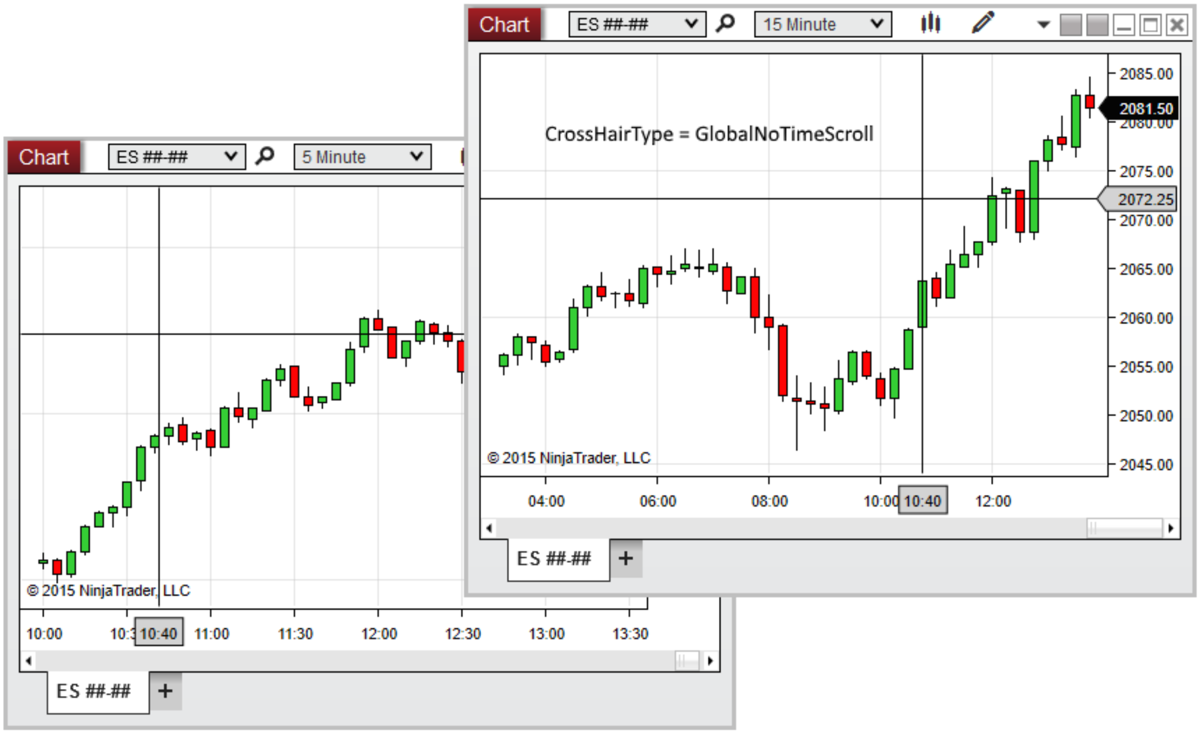
Syntax

`<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.



12.5.2.6.2.15 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.

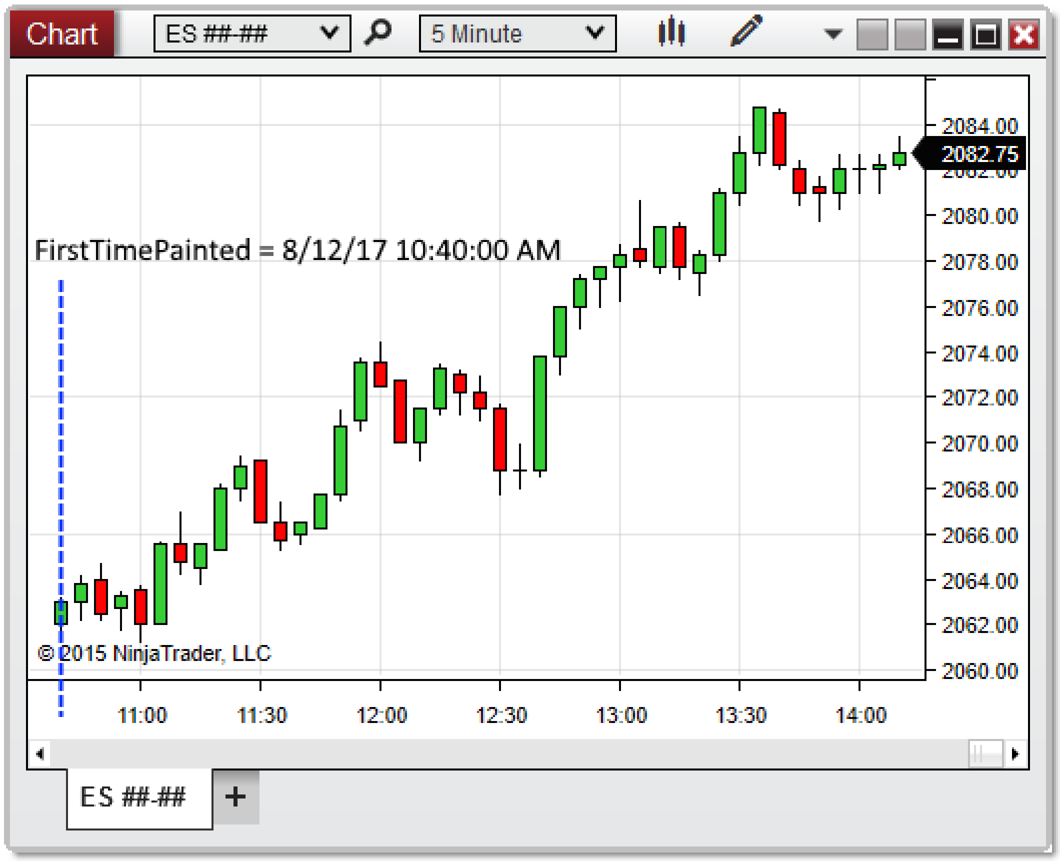
Syntax

<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.



12.5.2.6.2.16 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)
```

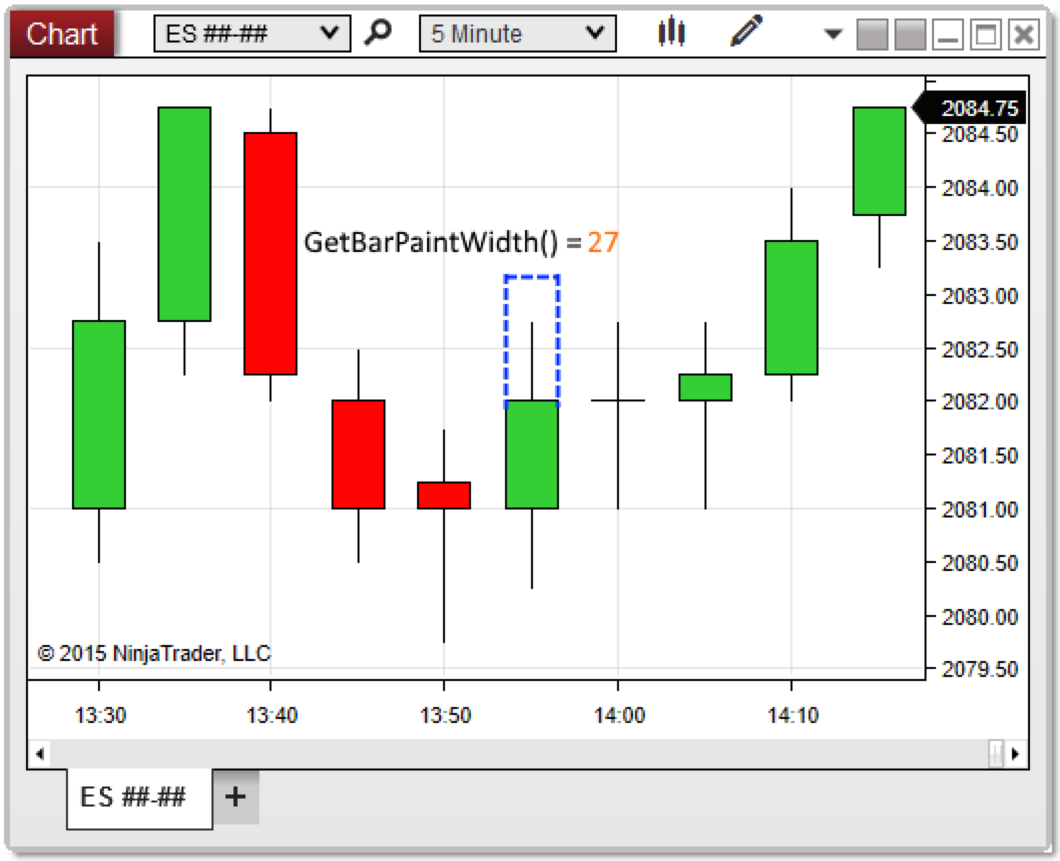
Method Parameters

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:



12.5.2.6.2.17 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);
   }
}
```

12.5.2.6.2.18 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);
}
```

12.5.2.6.2.19 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);
}
```

12.5.2.6.2.20 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);
}
```

12.5.2.6.2.21 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);
}
```

12.5.2.6.2.22 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);
}
```

12.5.2.6.2.23 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.Indicato
rRenderBase> 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()));
    }
}
```

12.5.2.6.2.24 IsScrollArrowVisible

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.

12.5.2.6.2.25 IsStayInDrawMode

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");
}
```

12.5.2.6.2.26 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**.

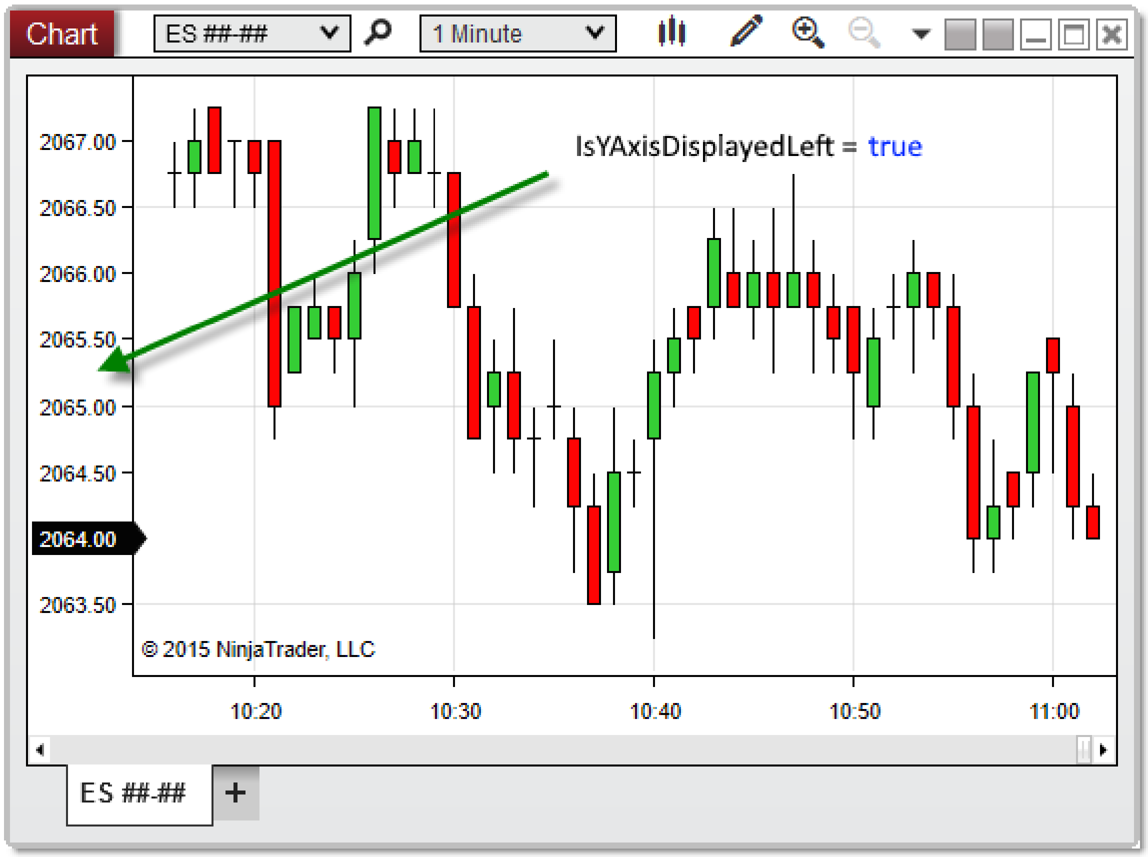
Syntax

```
<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.



12.5.2.6.2.27 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**.

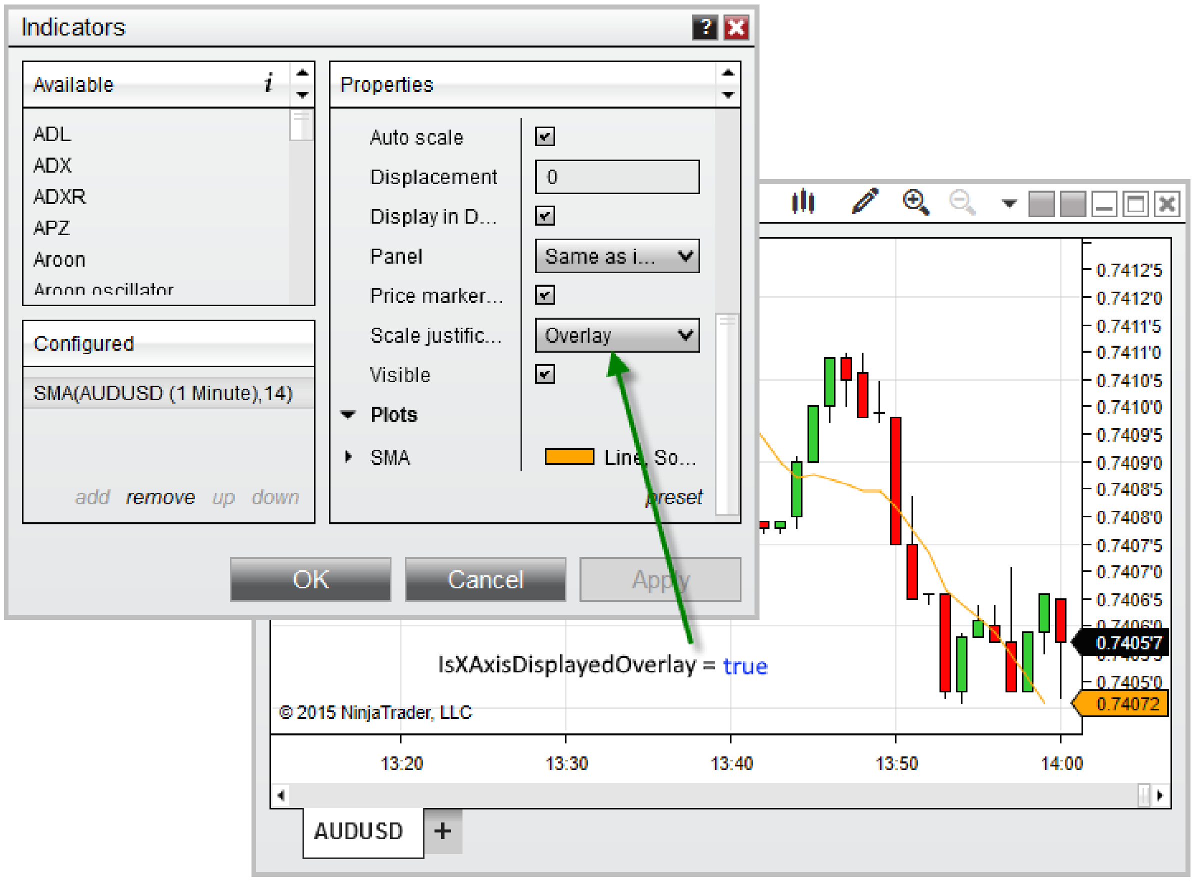
Syntax

```
<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.



Auto scale ✓
Displacement 0
Display in D... ✓
Panel Same as i...
Price marker... ✓
Scale justific... Overlay
Visible ✓
SMA Line, So...

IsXAxisDisplayedOverlay = true

© 2015 NinjaTrader, LLC

12.5.2.6.2.28 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**.

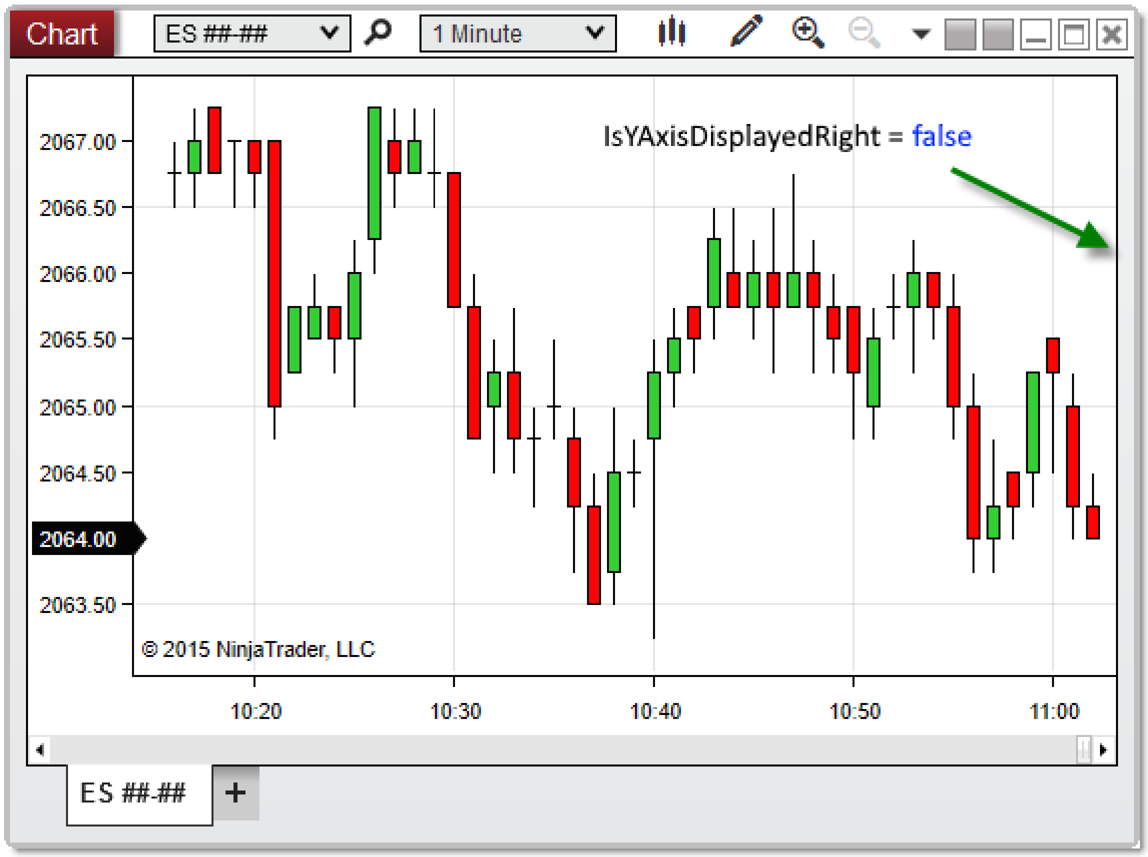
Syntax

```
<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.



12.5.2.6.2.29 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);
}
```

12.5.2.6.2.30 LastTimePainted

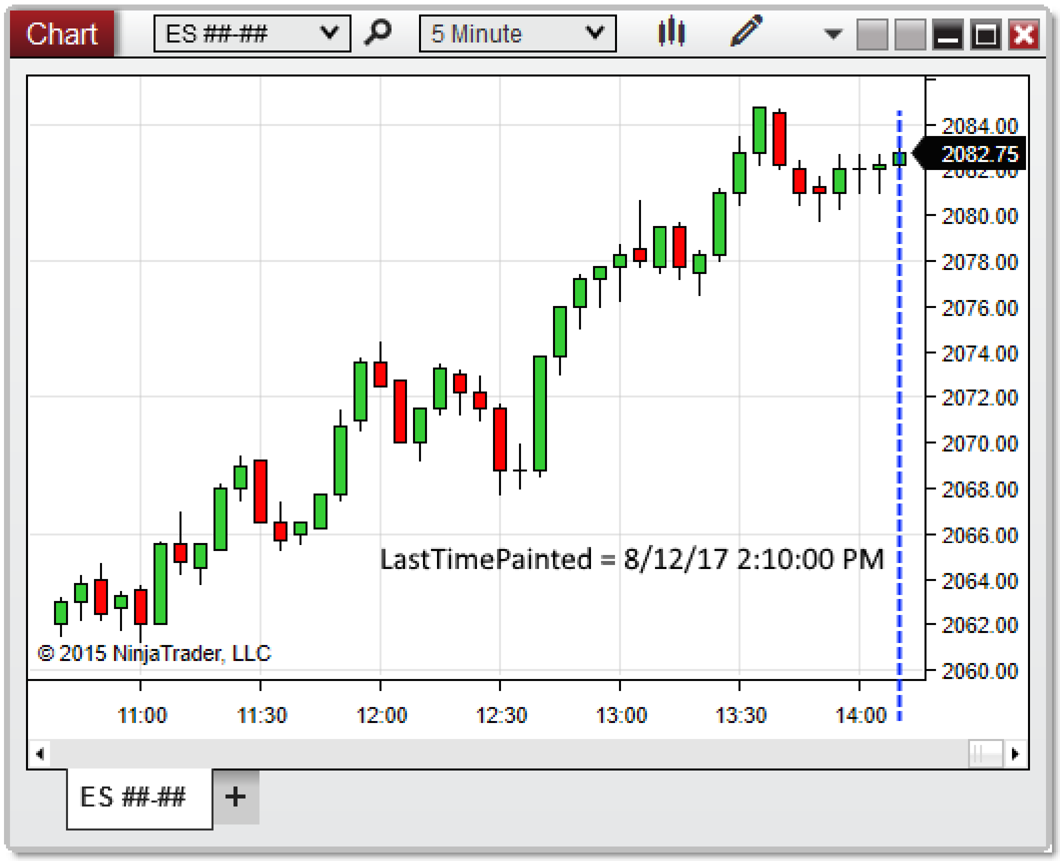
Definition
Indicates the time of the most recently painted bar on the primary Bars object configured on the chart.

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.



LastTimePainted = 8/12/17 2:10:00 PM

12.5.2.6.2.31 MouseDownPoint

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));
}
```

12.5.2.6.2.32 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);
}
```

12.5.2.6.2.33 Properties

Definition

A collection of properties related to the configuration of the Chart

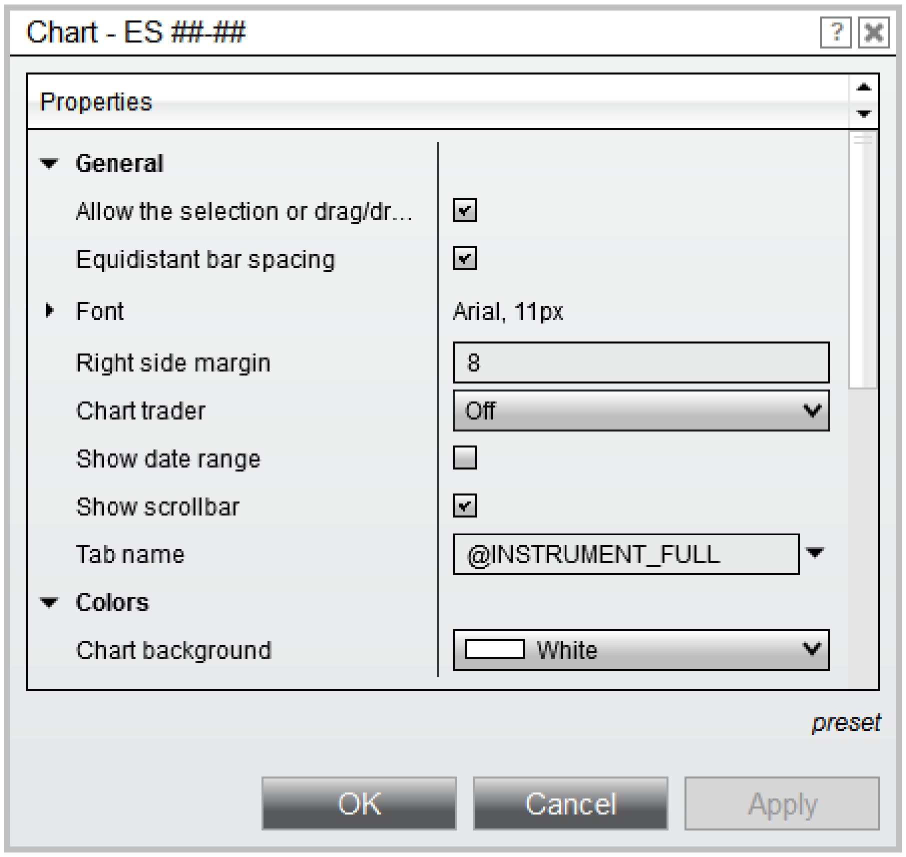


Chart - ES ##-##

Properties

General

Allow the selection or drag/dr...	✓
Equidistant bar spacing	✓
Font	Arial, 11px
Right side margin	8
Chart trader	Off
Show date range	☐
Show scrollbar	✓
Tab name	@INSTRUMENT_FULL

Colors

Chart background	White

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**
CrosshairSt	A `CrosshairStroke` object containing information on the

roke	**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;
}
```

12.5.2.6.2.34 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

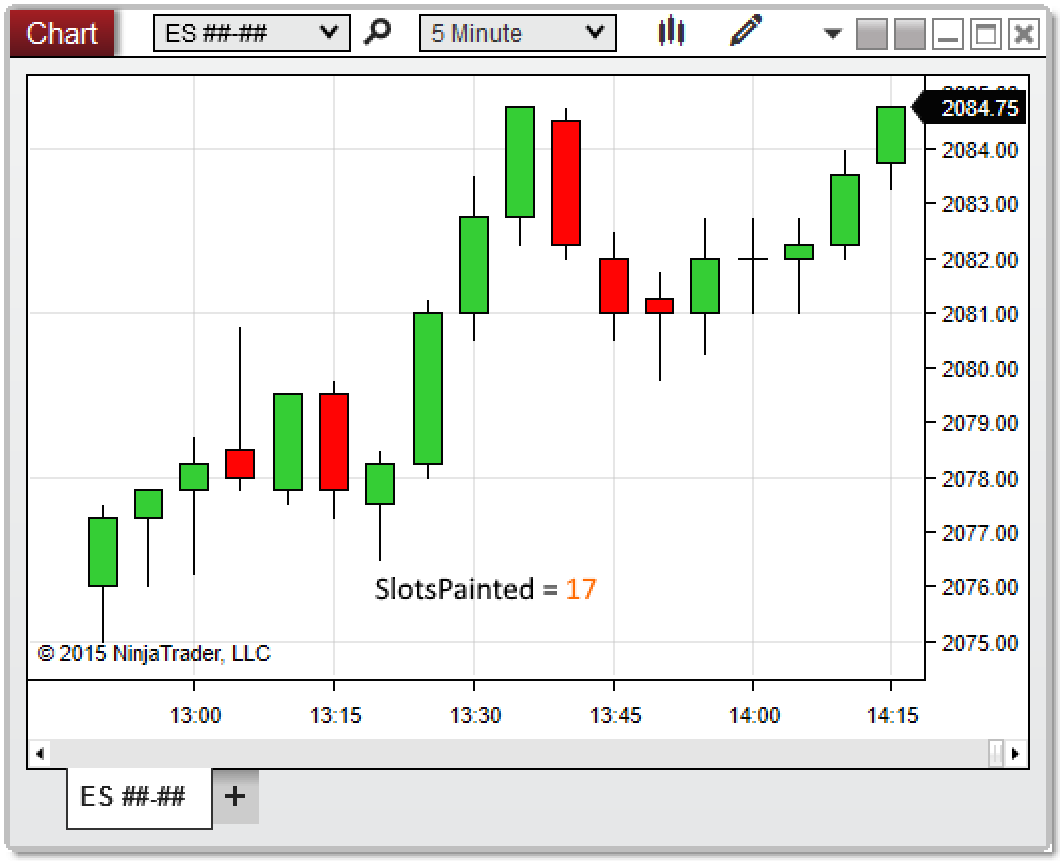
Syntax

`<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.



12.5.2.6.2.35 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>.TimePainted`

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);
}
```

12.5.2.6.2.36 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.

12.5.2.6.3 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

ConvertFromHorizontalPix	Converts a horizontal coordinate (x) from device pixels to application pixels

els	
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

12.5.2.6.3.1 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);
}
```

12.5.2.6.3.2 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);
}
```

12.5.2.6.3.3 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);
}
```

12.5.2.6.3.4 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)
```

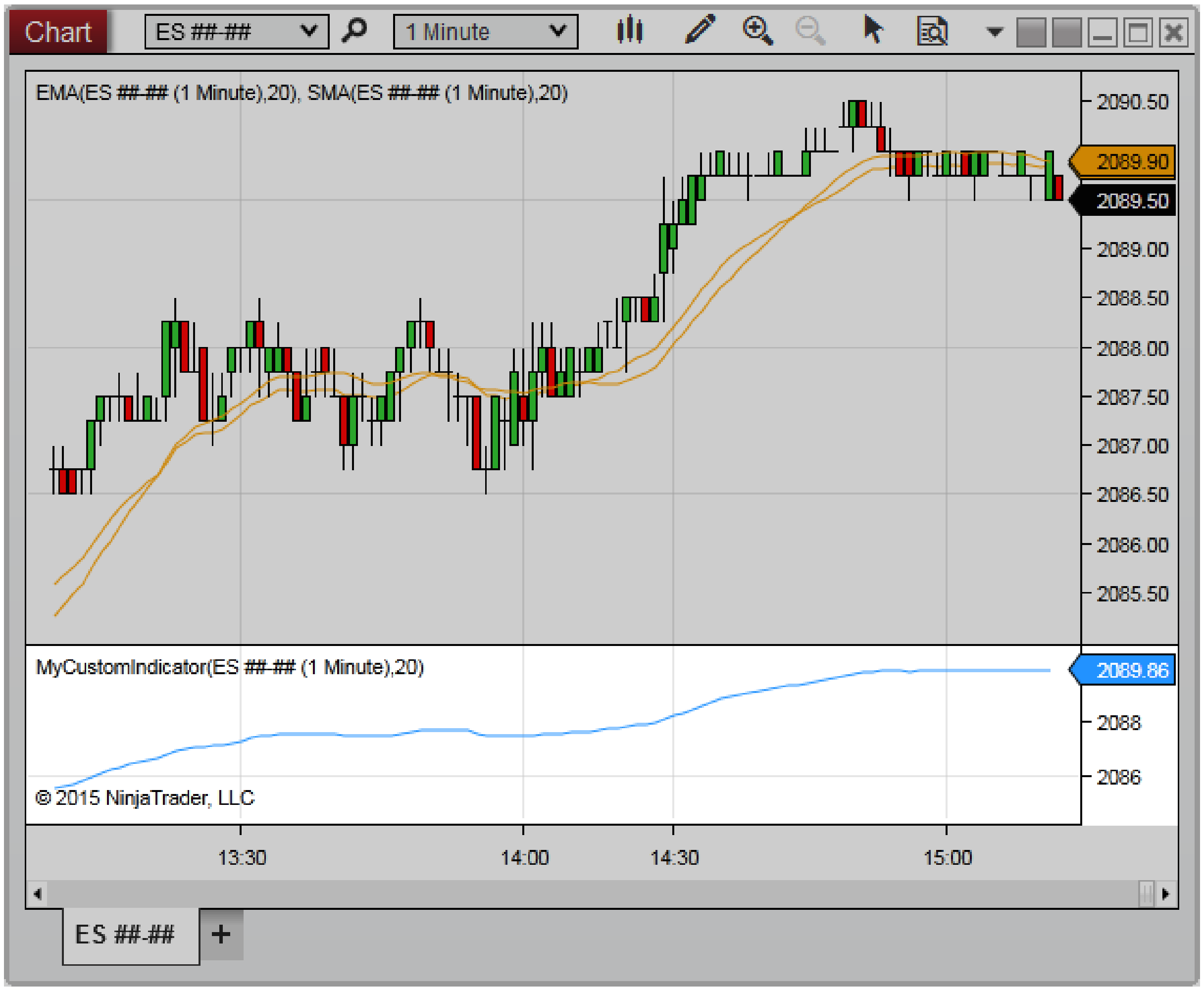
x	The vertical `double` coordinates in application pixels to convert
`target`	The PresenationSource representing the display surface used for the conversion **Note**: For Charts, see ChartControl.PresentationSource

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);
}
```

12.5.2.6.4 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

12.5.2.6.4.1 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

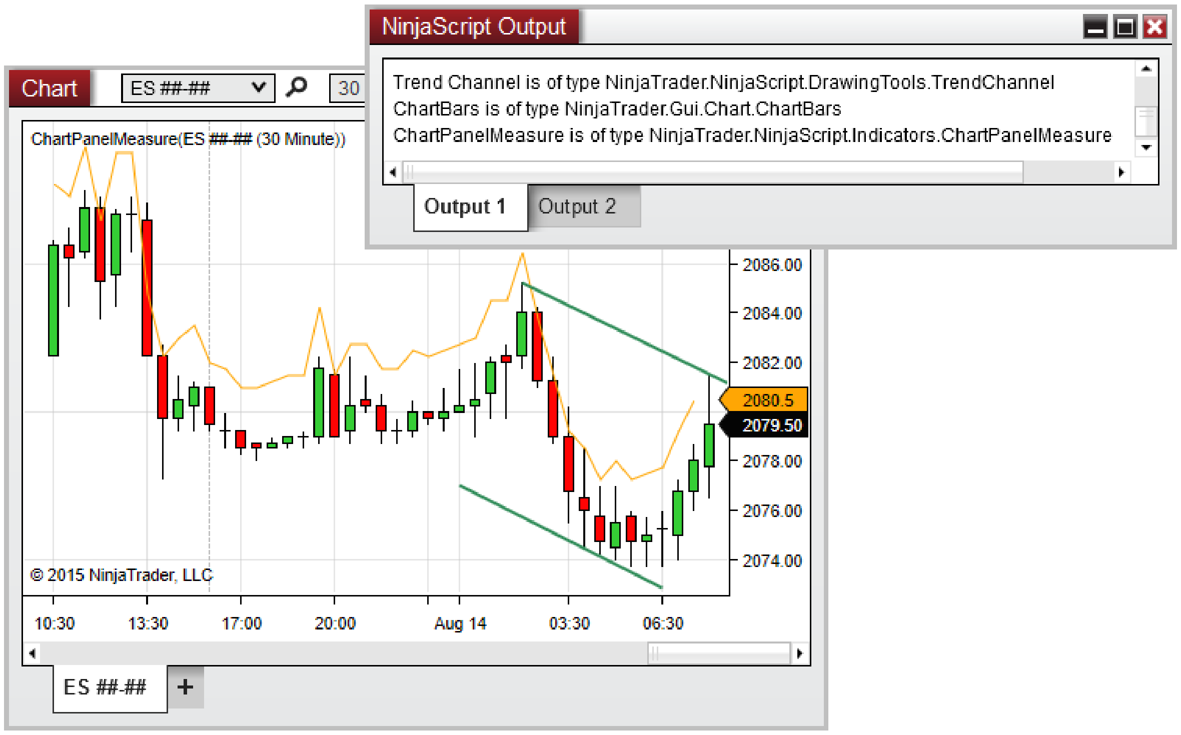
Syntax

```
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.



```
Trend Channel is of type NinjaTrader.NinjaScript.DrawingTools.TrendChannel
ChartBars is of type NinjaTrader.Gui.Chart.ChartBars
ChartPanelMeasure is of type NinjaTrader.NinjaScript.Indicators.ChartPanelMeasure
```

12.5.2.6.4.2 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

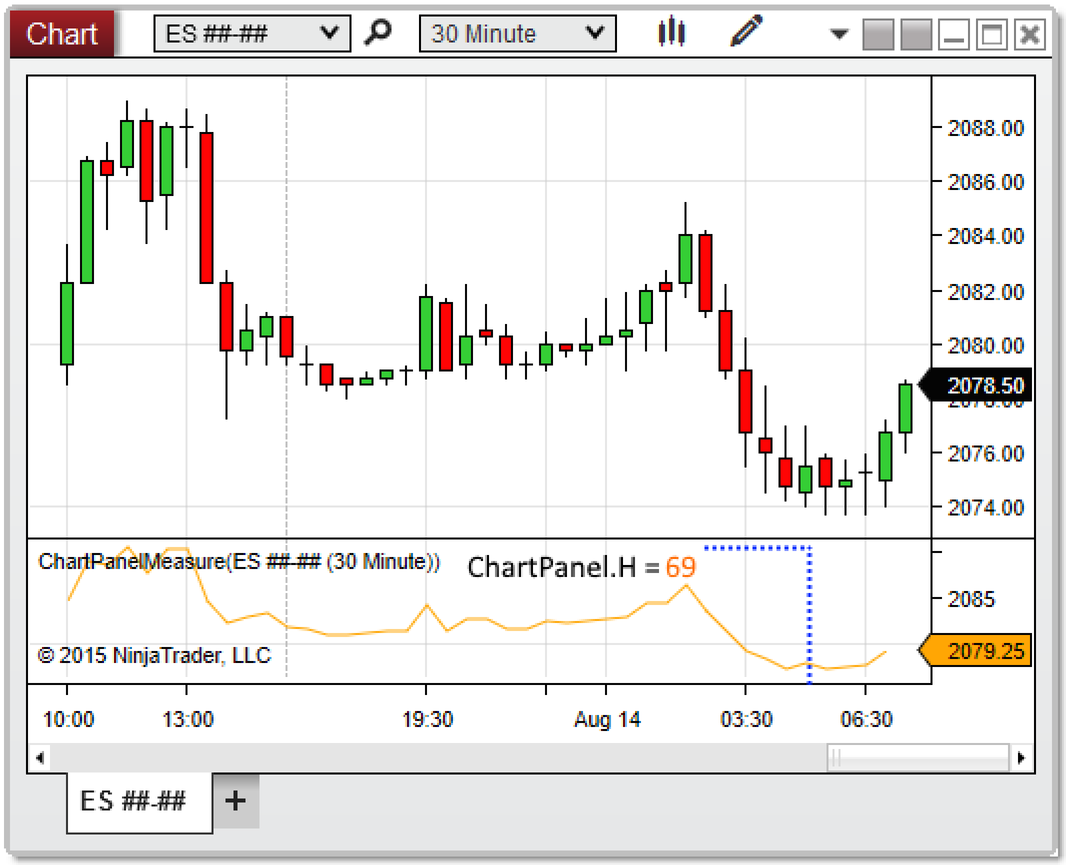
Syntax

`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.



12.5.2.6.4.3 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.

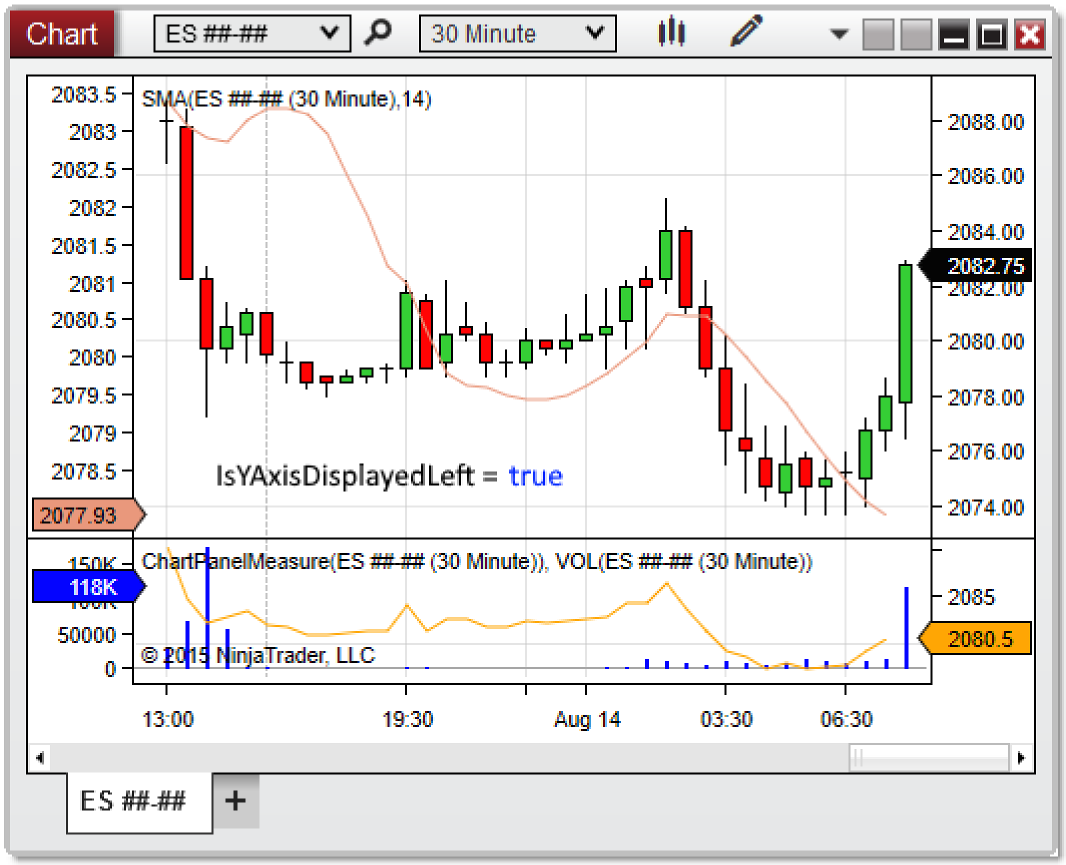


Chart
ES ##-##
30 Minute
SMA(ES ##-## (30 Minute),14)
IsYAxisDisplayedLeft = true
ChartPanelMeasure(ES ##-## (30 Minute)), VOL(ES ##-## (30 Minute))
© 2015 NinjaTrader, LLC
13:00
19:30
Aug 14
03:30
06:30
ES ##-##

12.5.2.6.4.4 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

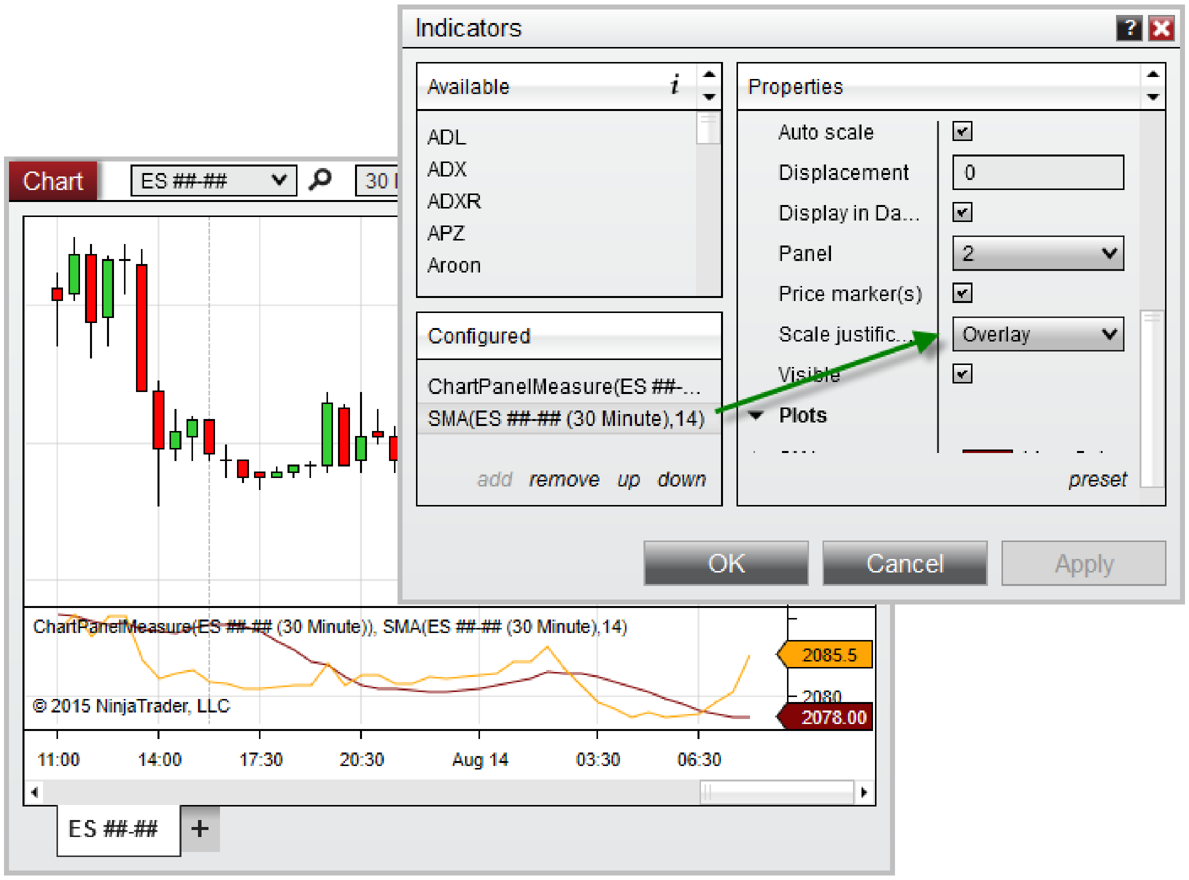
Syntax

```
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.



Auto scale ✓
Displacement 0
Display in Da... ✓
Panel 2
Price marker(s) ✓
Scale justific... Overlay
Visible ✓

12.5.2.6.4.5 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

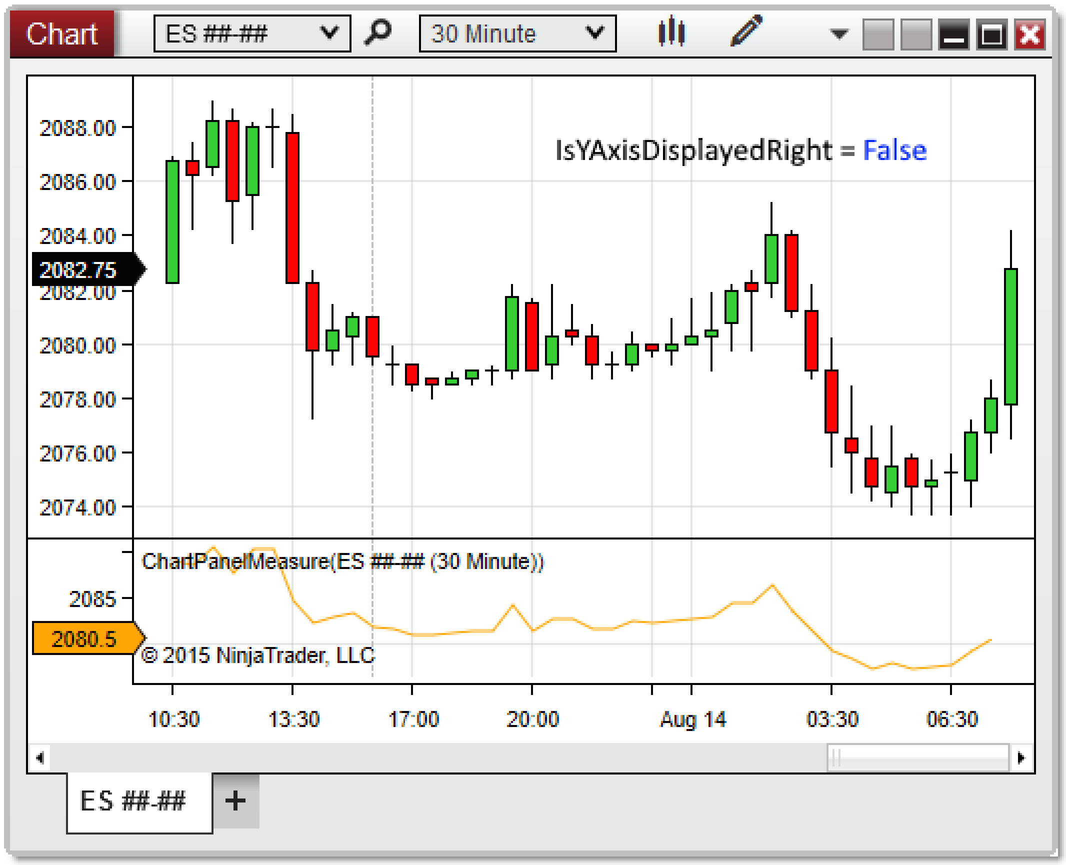
Syntax

```
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.



12.5.2.6.4.6 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));
}
```

12.5.2.6.4.7 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));
}
```

12.5.2.6.4.8 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

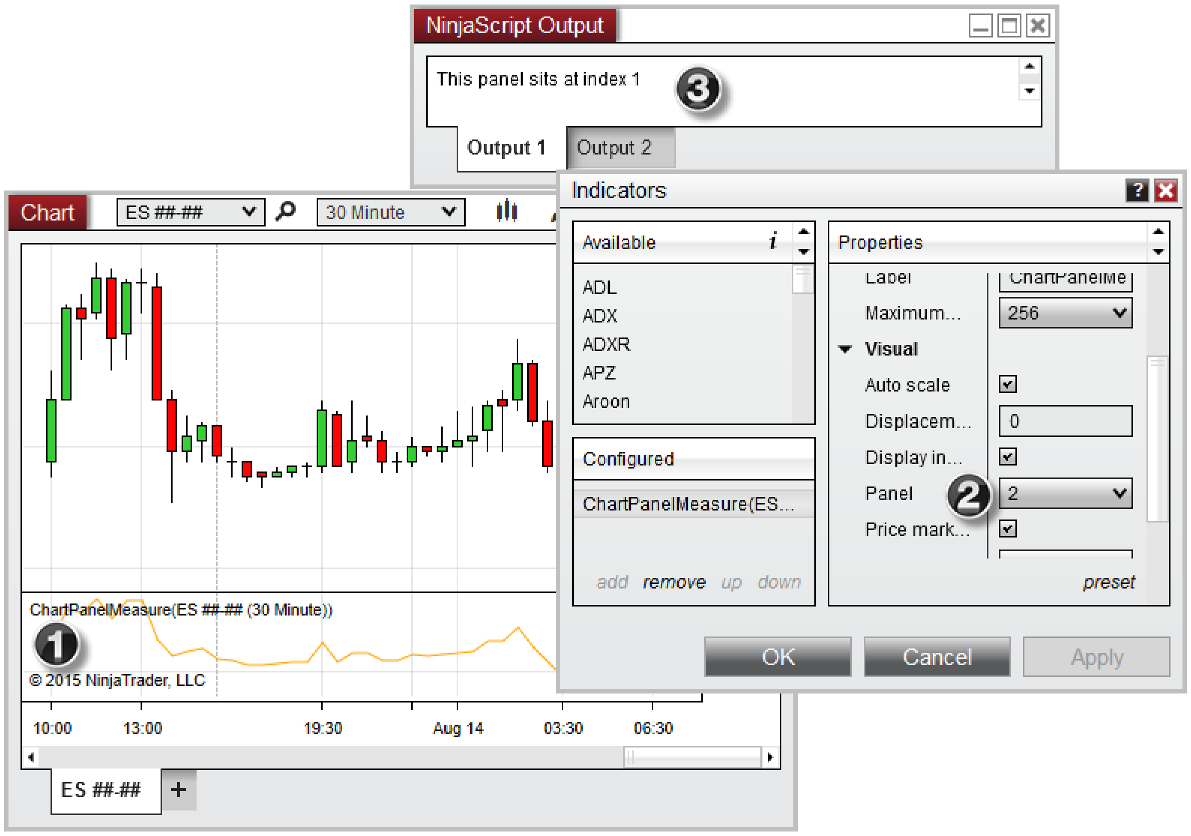
```
ChartPanel.PanelIndex
```

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



This panel sits at index 1

ChartPanelMeasure(ES ##-## (30 Minute))

© 2015 NinjaTrader, LLC

ChartPanelMeasure(ES...

Label	ChartPanelMe
Maximum...	256
Visual	
Auto scale	✓
Displacem...	0
Display in...	✓
Panel	2
Price mark...	✓

12.5.2.6.4.9 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]);
    }
}
```

12.5.2.6.4.10 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

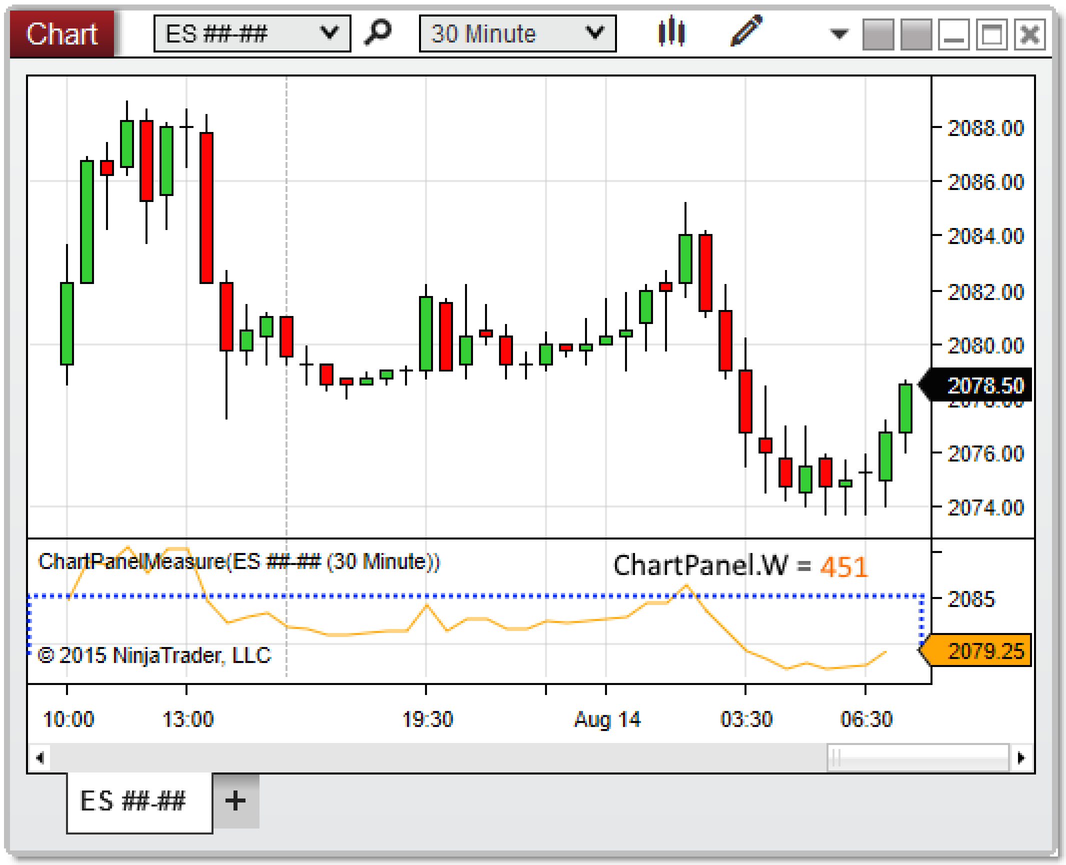
Syntax

`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.



12.5.2.6.4.11 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).

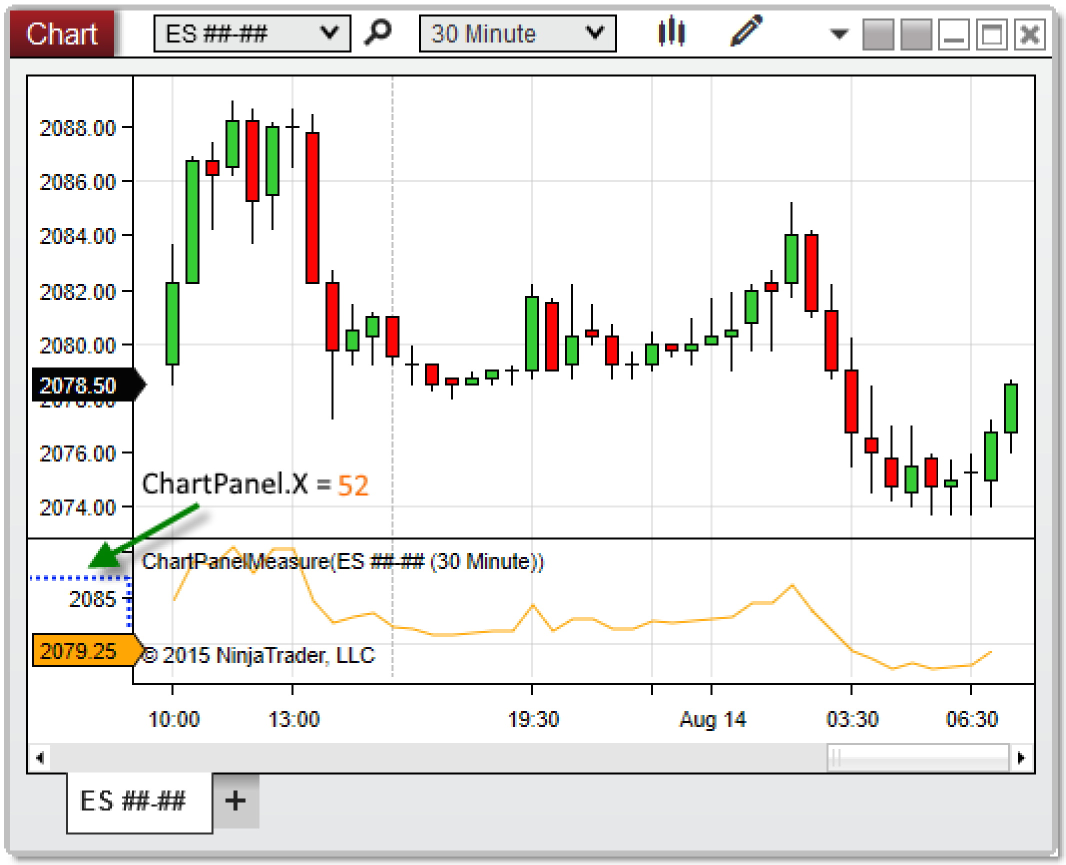
Syntax

```
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.



you

12.5.2.6.4.12 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.

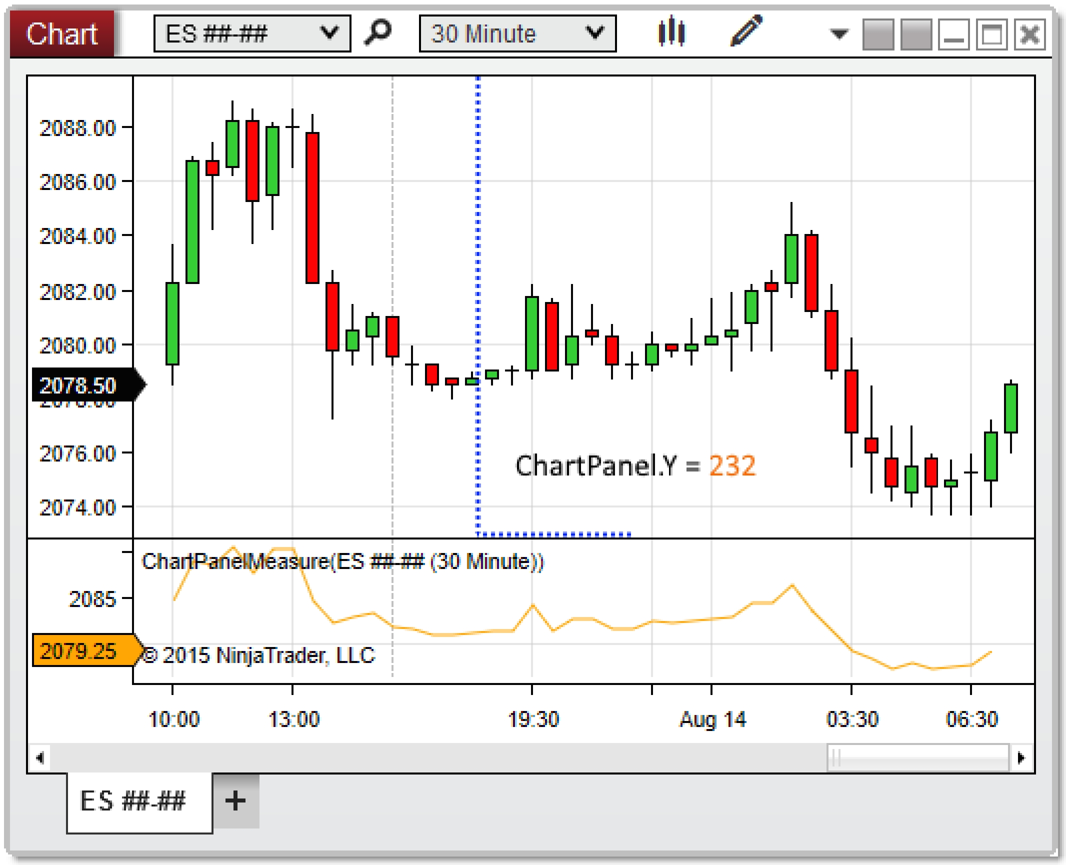
Syntax

```
ChartPanel.Y
```

Example

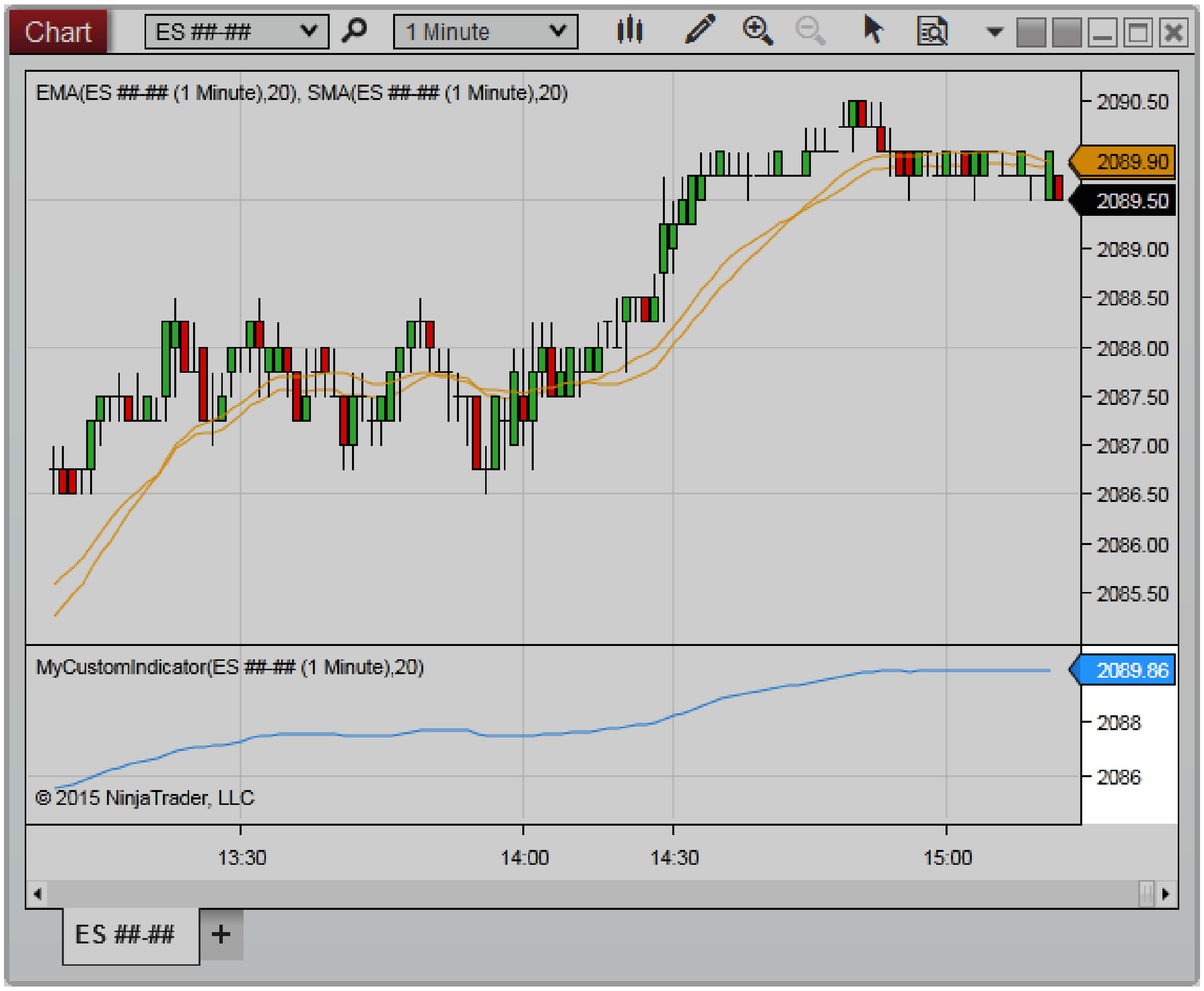
```
protected override void OnRender(ChartControl chartControl,
ChartScale chartScale)
{
    base.OnRender(chartControl, chartScale);

    // 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.



12.5.2.6.5 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

12.5.2.6.5.1 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)
```

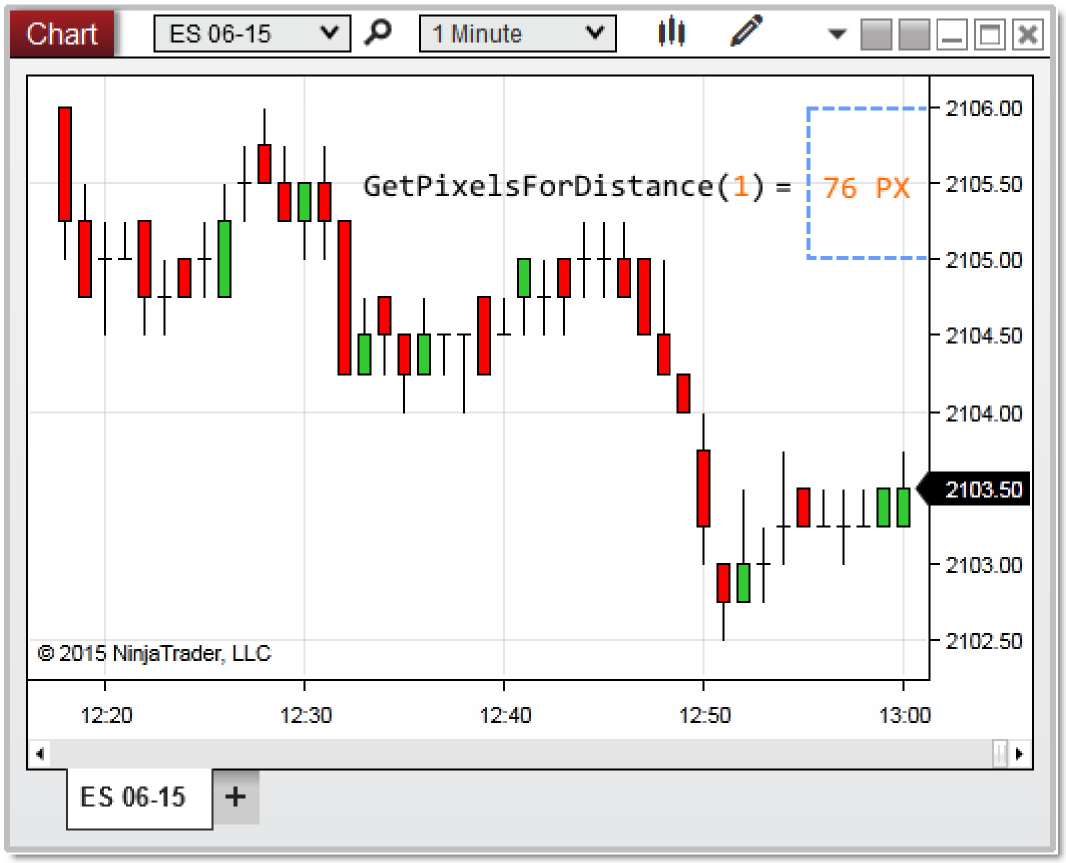
Method Parameters

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.



12.5.2.6.5.2 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)
```

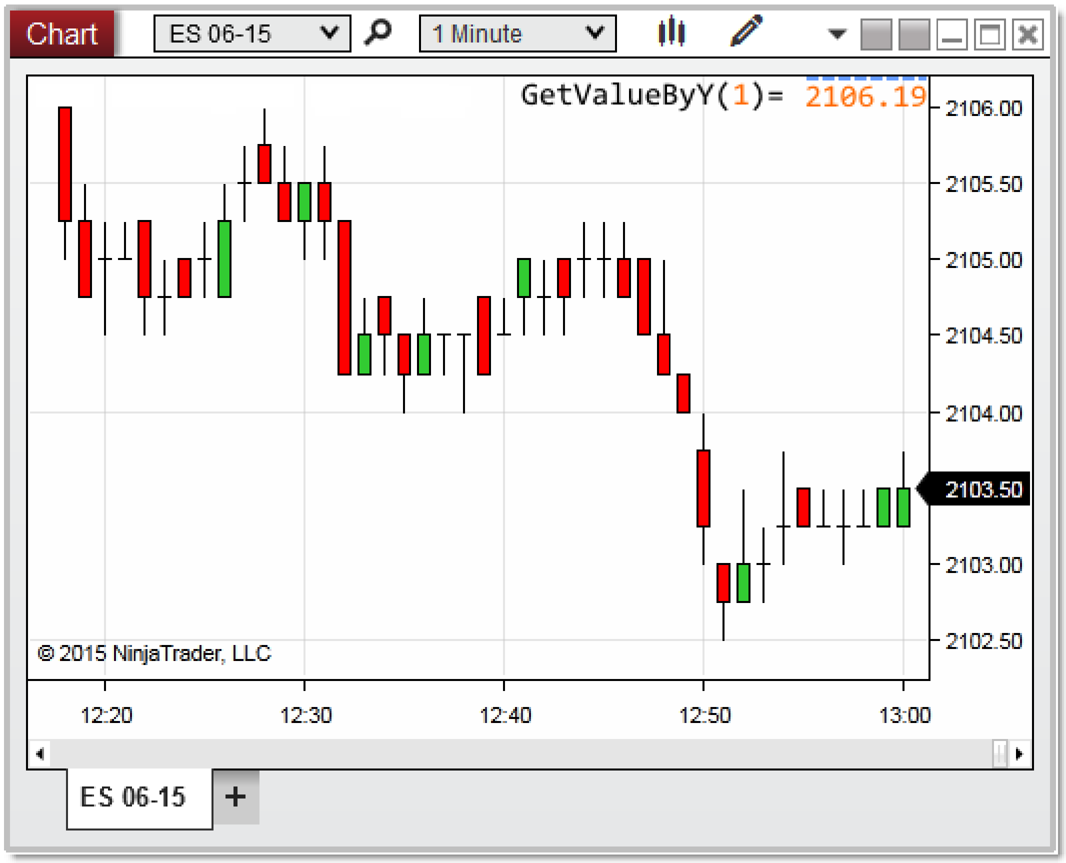
Method Parameters

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



12.5.2.6.5.3 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)
```

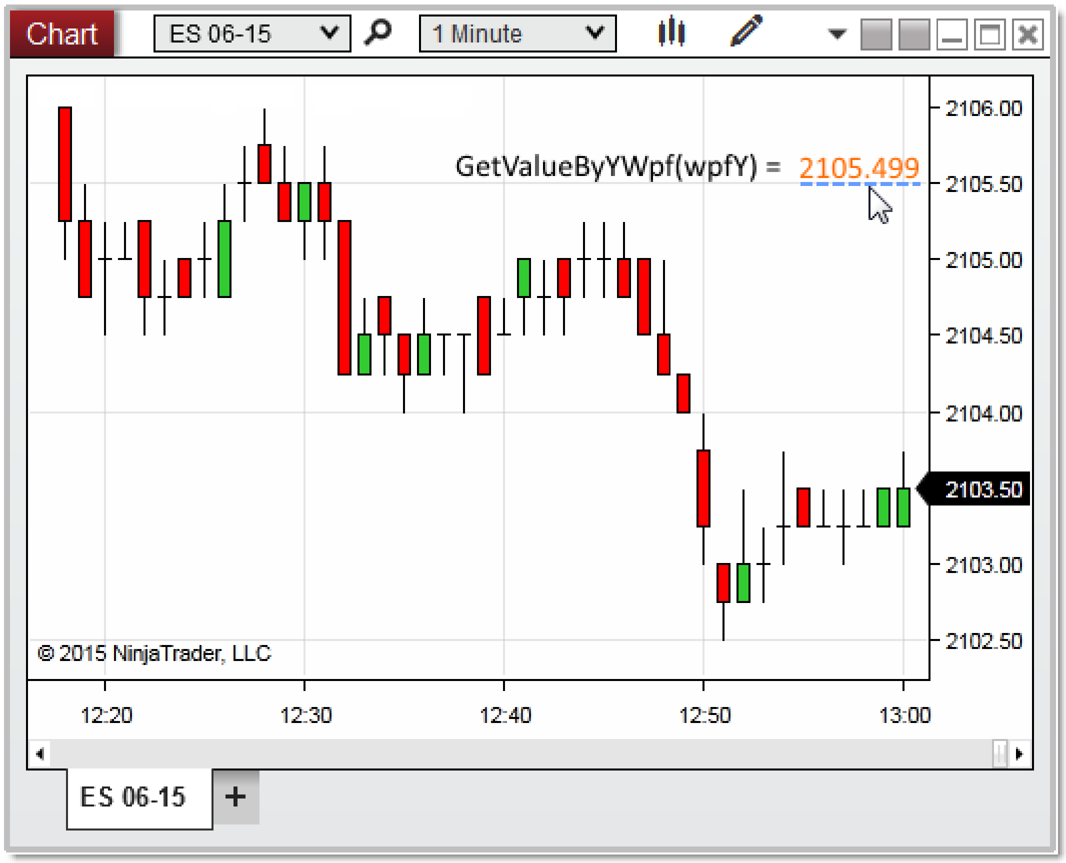
Method Parameters

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.



12.5.2.6.5.4 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)
```

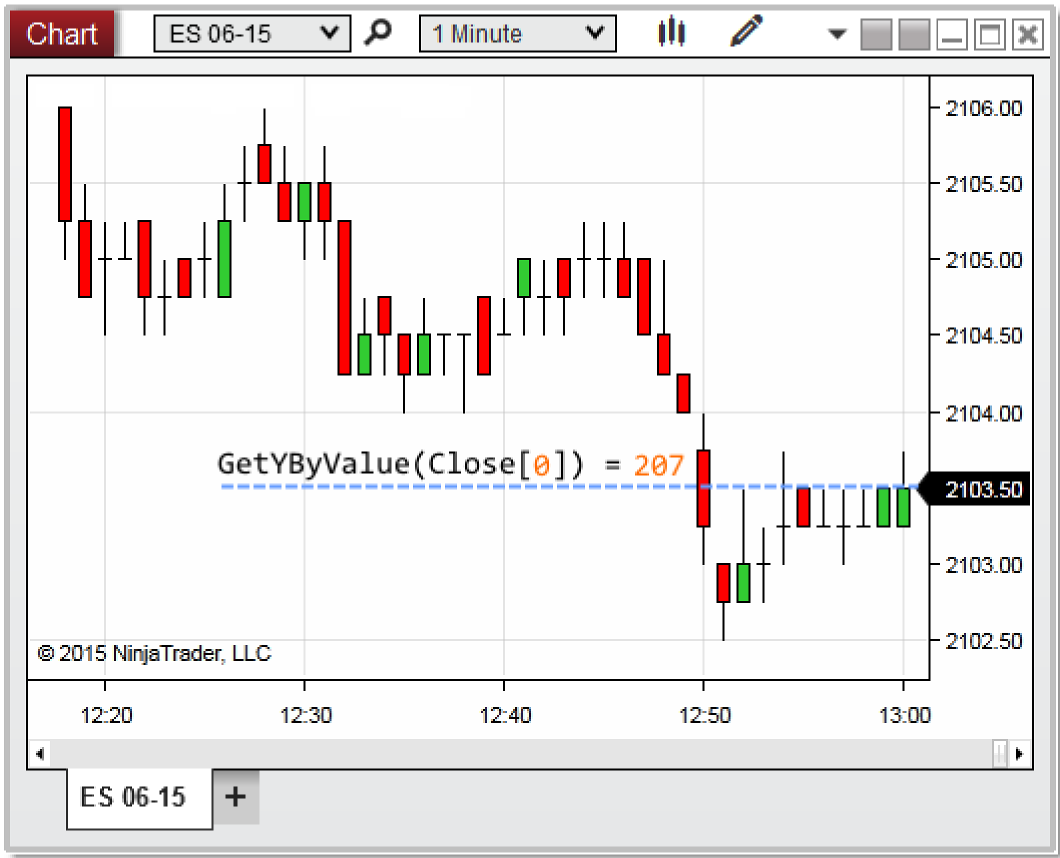
Method Parameters

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(Close[0]);

   Print("yByValue: " + yByValue); // 207
}
```

In the image below, we pass the `Close[0]` of the last bar as the value, which in return tells us the last price displayed on the chart is at a y location of 207 pixels.



12.5.2.6.5.5 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)
```

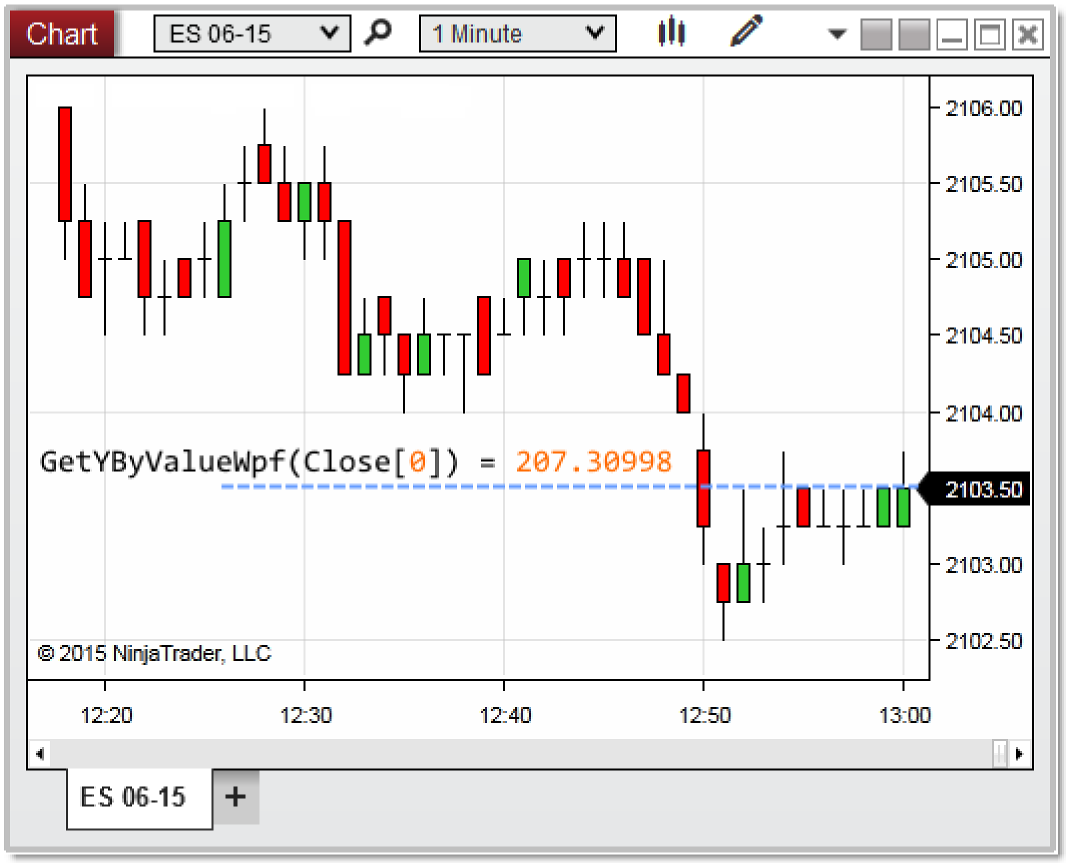
Method Parameters

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(Close[0]);

   Print("valueByYWpf: " + valueByYWpf); // 207
}
```

In the image below, we pass the Close[0] of the last bar as the value, which in return tells us the last price displayed on the chart is at a WPF location of 207.30998 pixels.



```
GetYByValueWpf(Close[0]) = 207.30998
```

12.5.2.6.5.6 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.

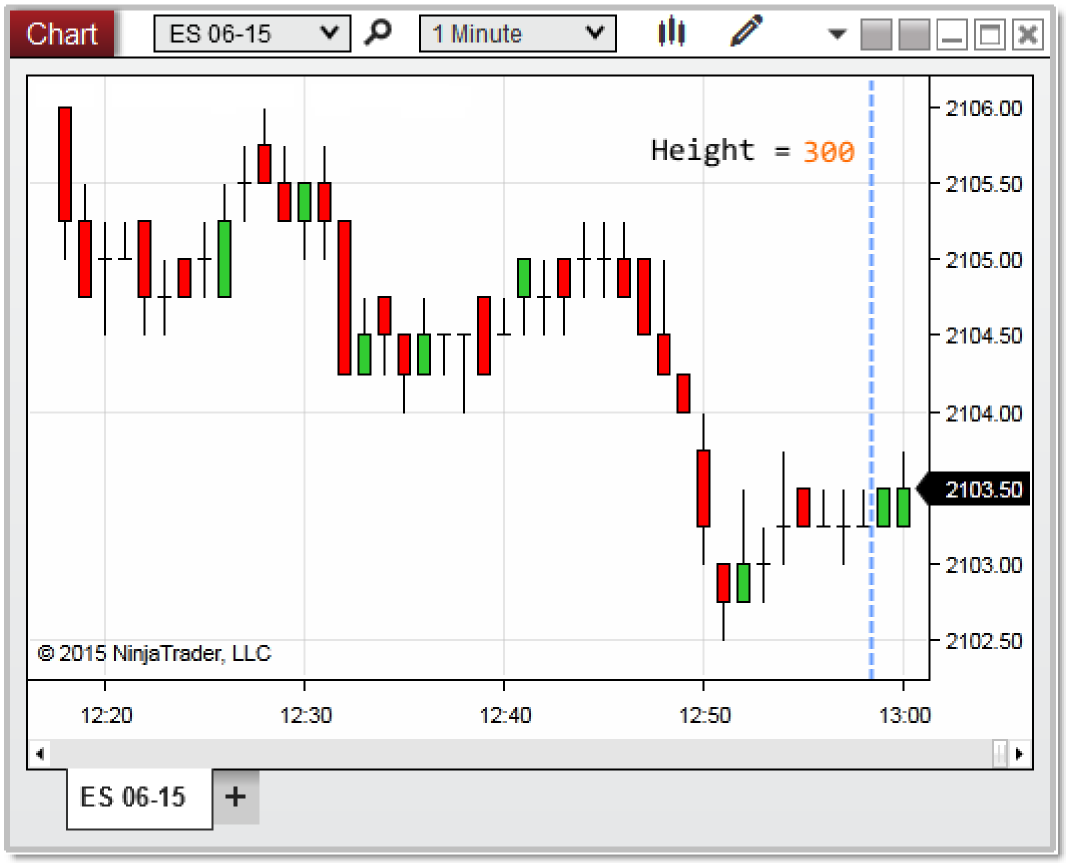
Syntax

```
<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.



12.5.2.6.5.7 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;
}
```

12.5.2.6.5.8 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.

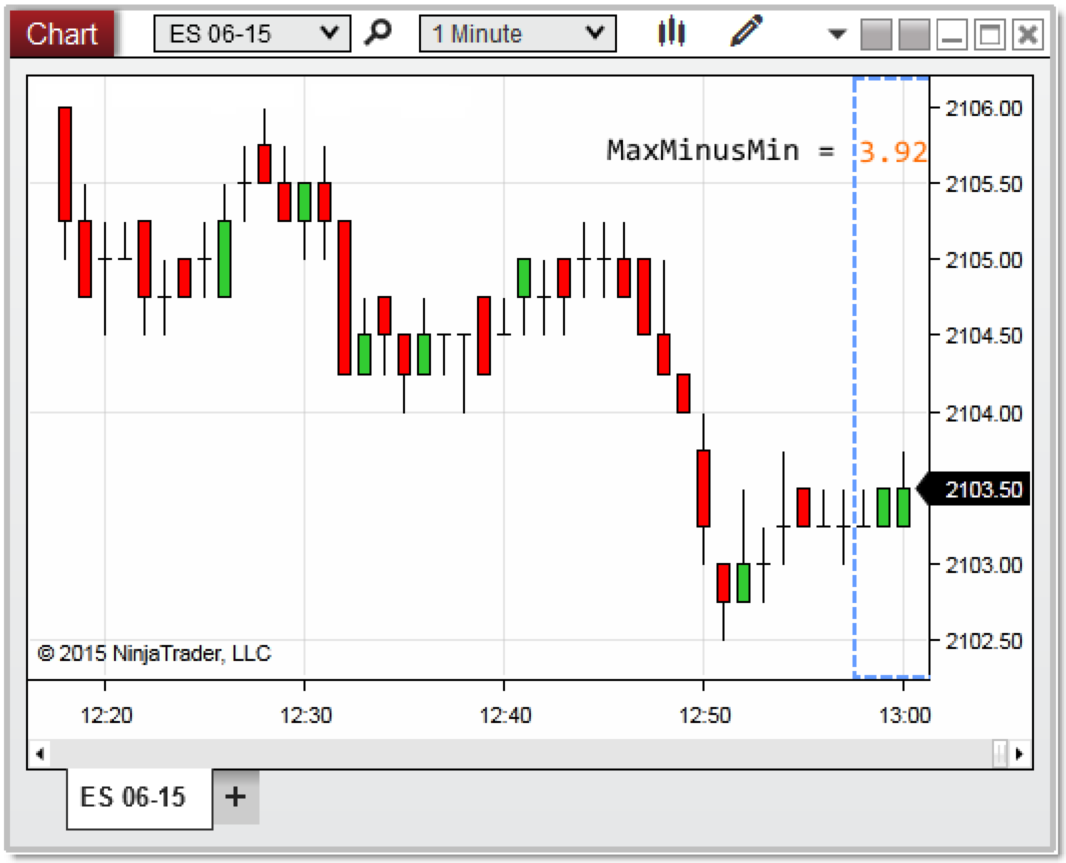
Syntax

```
<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.



12.5.2.6.5.9 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.

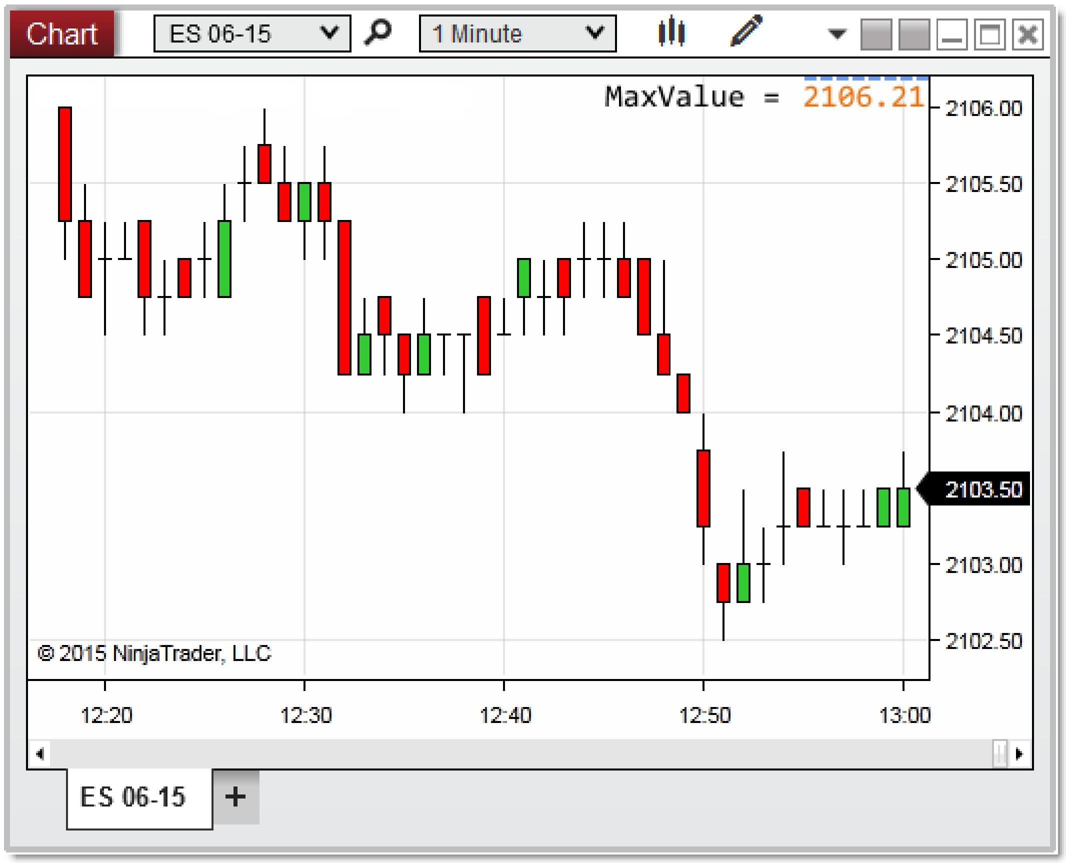
Syntax

```
<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



12.5.2.6.5.10 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.

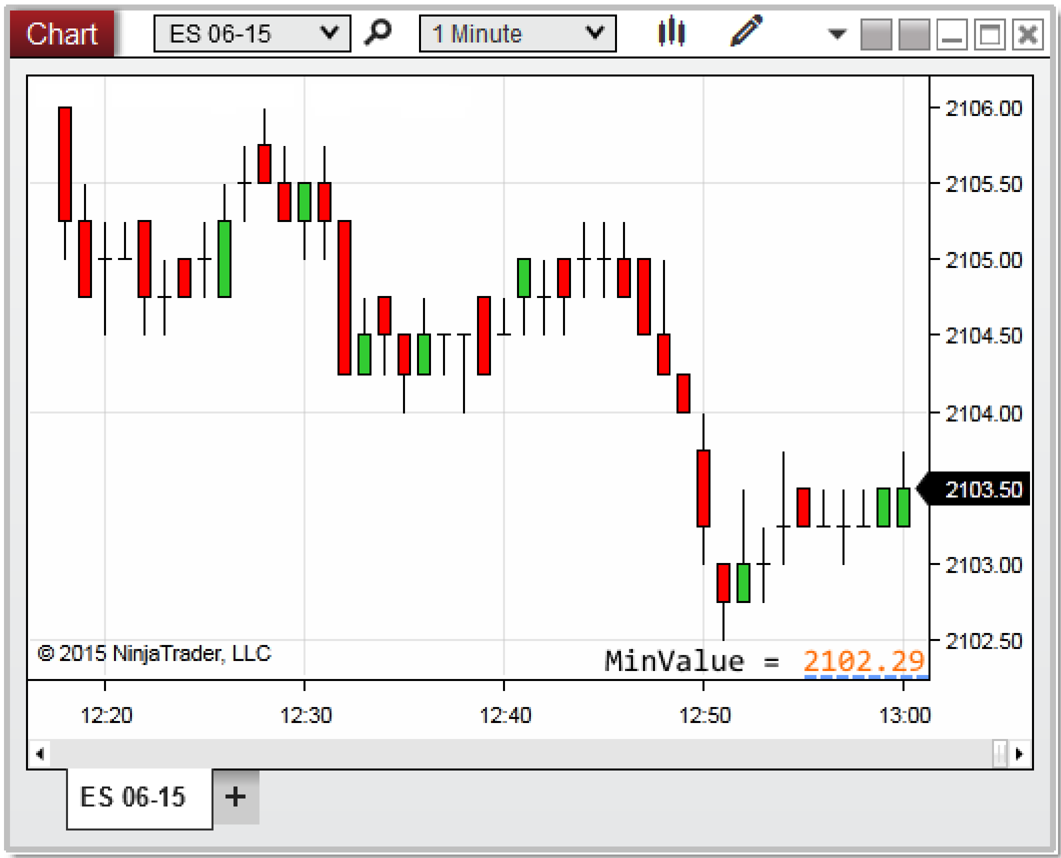
Syntax

```
<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.



MinValue = 2102.29

12.5.2.6.5.11 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);
}
```

12.5.2.6.5.12 Properties

Definition

Represents a number of properties available to the Chart Scale which can be configured to change the appearance of the scale.

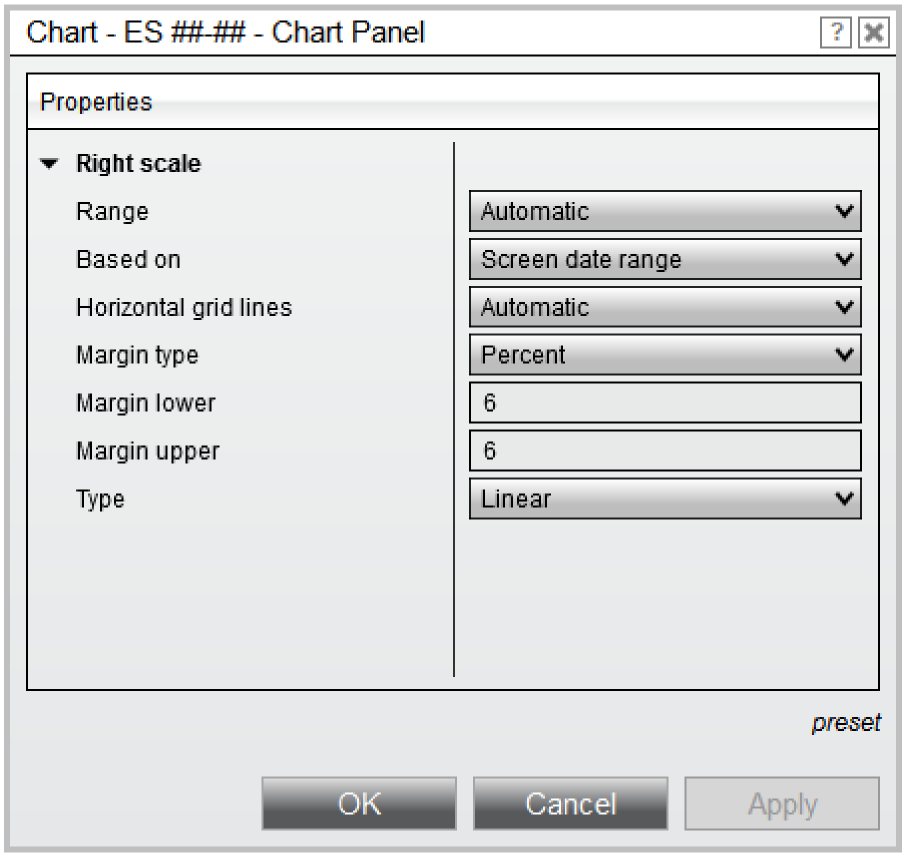


Chart - ES ##-## - Chart Panel

Properties

Right scale

Range	Automatic
Based on	Screen date range
Horizontal grid lines	Automatic
Margin type	Percent
Margin lower	6
Margin upper	6
Type	Linear

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
   }
}
```

12.5.2.6.5.13 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
   }
}
```

12.5.2.6.5.14 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.

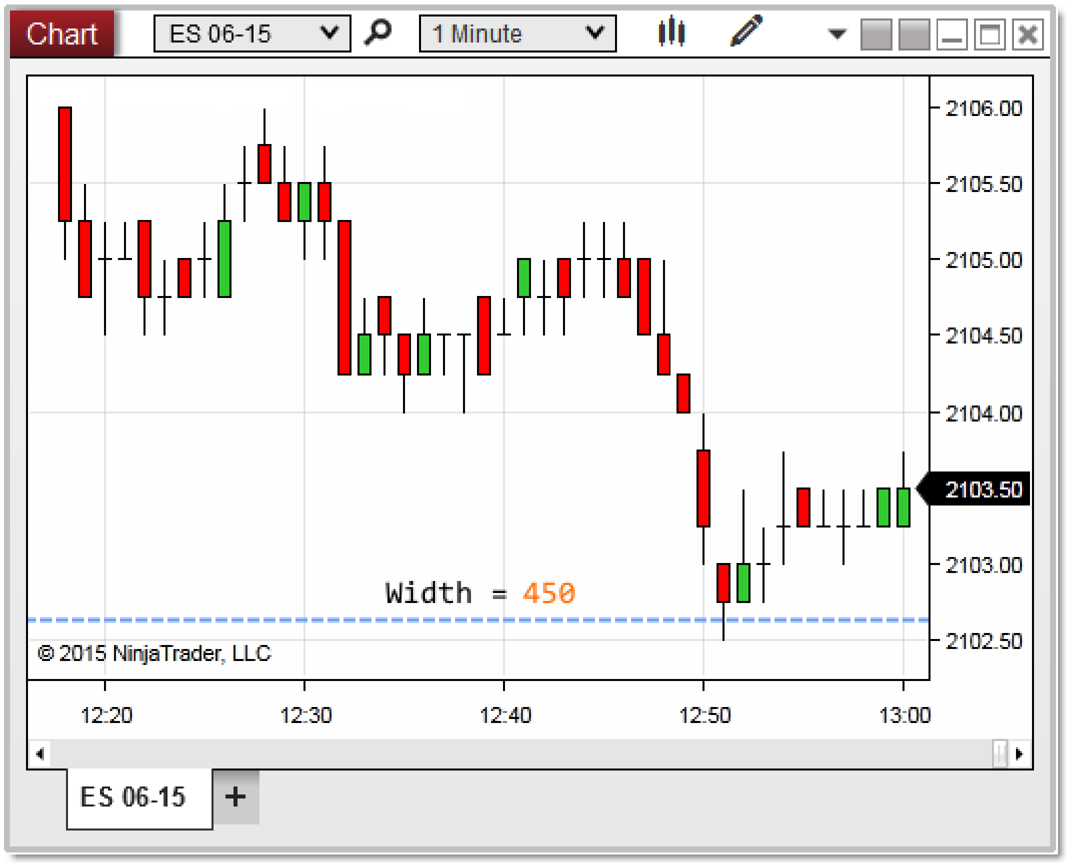
Syntax

`<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.



12.5.2.6.6 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

RenderTar get	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
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

12.5.2.6.6.1 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
```

```
// create a Direct2D1 PathGeometry format object with default
NinjaTrader D2DFactory factory
SharpDX.Direct2D1.PathGeometry pathGeometry = new
SharpDX.Direct2D1.PathGeometry(NinjaTrader.Core.Globals.D2DFac
tory);
```

12.5.2.6.6.2 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.Direct
WriteFactory,
   "Arial", 12f);

// create a text layout object with default NinjaTrader
DirectWrite factory
SharpDX.DirectWrite.TextLayout textLayout = new
SharpDX.DirectWrite.TextLayout(NinjaTrader.Core.Globals.Direct
WriteFactory,
   "text to render", textFormat, ChartPanel.W, ChartPanel.H);
```

12.5.2.6.6.3 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();
```

12.5.2.6.6.4 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.GlobalsNow.Subtract(lastTimeCalled).Seconds >= 1)
   {
      ForceRefresh();
      lastTimeCalled = Core.Globals.Now;
   }
}
```

12.5.2.6.6.5 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
   }
}
```

12.5.2.6.6.6 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;
      }
   }
}
```

12.5.2.6.6.7 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
}
```

12.5.2.6.6.8 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;
   }
}
```

12.5.2.6.6.9 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;
   }
}
```

12.5.2.6.6.10 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 and is only called when the chart object is set to IsAutoScale.

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;
}
```

12.5.2.6.6.11 OnRender()

Definition

Used to render custom drawing to a chart from various chart objects, such as an Indicator, DrawingTool or Strategy.

Notes:
1. Thie 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 OnRenderTargetChange() 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
```

12.5.2.6.6.12 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.

Note: 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.

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 **OnRenderTargetChange(**) 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 caluled 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
   }

   else if (State == State.Terminated)
   {
      // dispose of SharpDX brush when finished
      if (dxBrush != null)
      {
         dxBrush.Dispose();
         dxBrush = null;
      }
   }
}
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
```

12.5.2.6.6.13 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);
}
```

12.5.2.6.6.14 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.

12.5.2.6.6.15 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, default levels used by NinjaTrader can be seen here.

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);
    }
}
```

12.5.2.6.6.16 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);
}
```

12.5.2.6.7 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);
}
```

12.5.2.6.8 IsAutoScale

Definition

If true, the drawing tool will call CalculateMinMax() in order to determine the drawing tool's MinValue and MaxValue value used to scale the Y-axis of the chart.

Property Value

This property returns **true** if the drawing tool plot(s) are included in the y-scale; otherwise, **false**. Default set to **false**.

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)
   {
   }
}
```

12.5.2.6.9 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** or **State.Configure**

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");
     }
}
```

12.5.2.6.10 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;
}
```

12.5.2.6.11 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");
   }
}
```

12.5.2.6.12 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 dashtyle 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

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; }
```

12.5.2.6.13 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,
```

12.5.2.7 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.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.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
RemoveDrawObje	Removes a draw object from the chart based on

ct()	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.

12.5.2.7.1 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);
```

12.5.2.7.1.1 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.
HandleLineStroke	A Stroke object used to draw the handle (mid line) of the object
ExtensionLineStroke	A Stroke object used to draw the equidistant trend lines 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);
```

12.5.2.7.2 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);
```

12.5.2.7.2.1 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
Stroke	A Stroke object used to draw the object

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;
```

12.5.2.7.3 Draw .Arrow Dow n()

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)

**T
i
p**
:
T
h
e
s
i
z
e
o
f
t
h
e
a
r

r
o
w
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
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 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);
```

12.5.2.7.3.1 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
Stroke	A Stroke object used to draw the object

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;
```

12.5.2.7.4 Draw.ArrowLine()

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);
```

12.5.2.7.4.1 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;
```

12.5.2.7.5 Draw .Arrow Up()

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 to the chart'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 down arrow on the current bar 1 tick below the
low
Draw.ArrowUp(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);
```

12.5.2.7.5.1 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

Stroke	A Stroke object used to draw the object

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;
```

12.5.2.7.6 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 th

e
d
i
a
m
o
n
d
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
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 diamond on the current bar 1 tick below the
low
Draw.Diamond(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);
```

12.5.2.7.6.1 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
Stroke	A Stroke object used to draw the object

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;
```

12.5.2.7.7 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: The size of the dot is tied to the chart's BarWi

```
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 dot on the current bar 1 tick below the low
Draw.Dot(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);
```

12.5.2.7.7.1 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
Stroke	A Stroke object used to draw the object

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;
```

12.5.2.7.8 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);
```

12.5.2.7.8.1 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);

// Apply the ellipse as a Global Drawing Object
myEllipse.IsGlobalDrawingTool = true;
```

12.5.2.7.9 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);
```

12.5.2.7.9.1 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;
```

12.5.2.7.10 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]);
```

12.5.2.7.10.1 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;
```

12.5.2.7.11 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]);
```

12.5.2.7.11.1 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
IsExtendedLinesRi	A `bool` value determining if the draw object

ght	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;
```

12.5.2.7.12 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]);
```

12.5.2.7.12.1 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;
```

12.5.2.7.13 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]);
```

12.5.2.7.13.1 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;
```

12.5.2.7.14 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]);
```

12.5.2.7.14.1 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;
```

12.5.2.7.15 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);
```

12.5.2.7.15.1 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
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 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);
```

12.5.2.7.16 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);
```

12.5.2.7.16.1 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.

12.5.2.7.17 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);
```

12.5.2.7.17.1 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);
```

12.5.2.7.18 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);
```

12.5.2.7.18.1 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 Ray 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;
```

12.5.2.7.19 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.

12.5.2.7.19.1 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);
```

12.5.2.7.20 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);
```

12.5.2.7.20.1 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;
```

12.5.2.7.21 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", High[0], Low[0],
Brushes.Blue, Brushes.Green, 20);
```

12.5.2.7.21.1 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;
```

12.5.2.7.22 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);
```

12.5.2.7.22.1 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.

12.5.2.7.23 Draw .RiskRew ard()

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 `int` value determining the calculated ratio between the risk or reward based on the entry point
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.
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);
```

12.5.2.7.23.1 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
StopAnchor	An IDrawingTool's ChartAnchor representing the stop loss point of the drawing object
TargetAnchor	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;
```

12.5.2.7.24 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]);
```

12.5.2.7.24.1 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;
```

12.5.2.7.25 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)

T
i
p
:
T
h
e

```
s
i
z
e
o
f
t
h
e
s
q
u
a
r
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
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);
```

12.5.2.7.25.1 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;
```

12.5.2.7.26 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);
```

12.5.2.7.26.1 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";
```

12.5.2.7.27 Draw.TextFixed()

Definition

Draws text in one of 5 available pre-defined fixed locations on panel 1 (price panel) of a chart.

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);
```

12.5.2.7.27.1 TextFixed

Definition

Represents an interface that exposes information regarding a Text Fixed IDrawingTool.

Methods and Properties

Anchor	AAn 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;
```

12.5.2.7.28 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);
```

12.5.2.7.28.1 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

ParallelEndAnchor	An IDrawingTool's ChartAnchor representing the ending 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;
```

12.5.2.7.29 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);
```

12.5.2.7.29.1 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;
```

12.5.2.7.30 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)

T
i
p

:
T
h
e
s
i
z
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
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 triangle pointing down on the current bar 1
tick below the low
Draw.TriangleDown(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);
```

12.5.2.7.30.1 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;
```

12.5.2.7.31 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

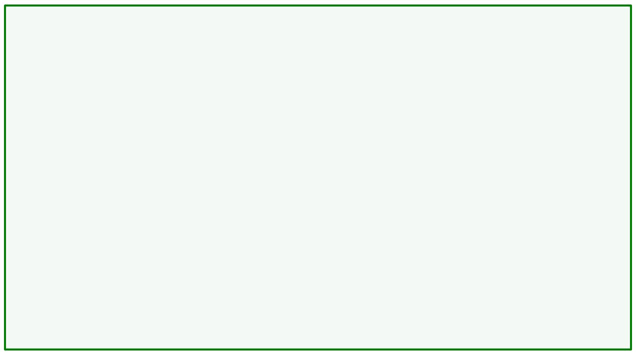
owner	The hosting NinjaScript object which is calling the draw method Typically will be the object which is calling the draw method (e.g., "`this`")
tag	A user defined unique id used to reference the draw object. For example, if you pass in a value of "myTag",

	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)

T
i
p
:
T
h
e
s
i
z
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
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 triangle pointing up on the current bar 1 tick
below the low
Draw.TriangleUp(this, "tag1", true, 0, Low[0] - TickSize,
Brushes.Red);
```

12.5.2.7.31.1 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;
```

12.5.2.7.32 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);
```

12.5.2.7.32.1 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);
```

12.5.2.7.33 Brushes

For detailed information on using Brushes for Drawing please see the Working with Brushes educational resource.

12.5.2.7.34 Allow RemovalOfDraw Objects

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
}
```

12.5.2.7.35 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.

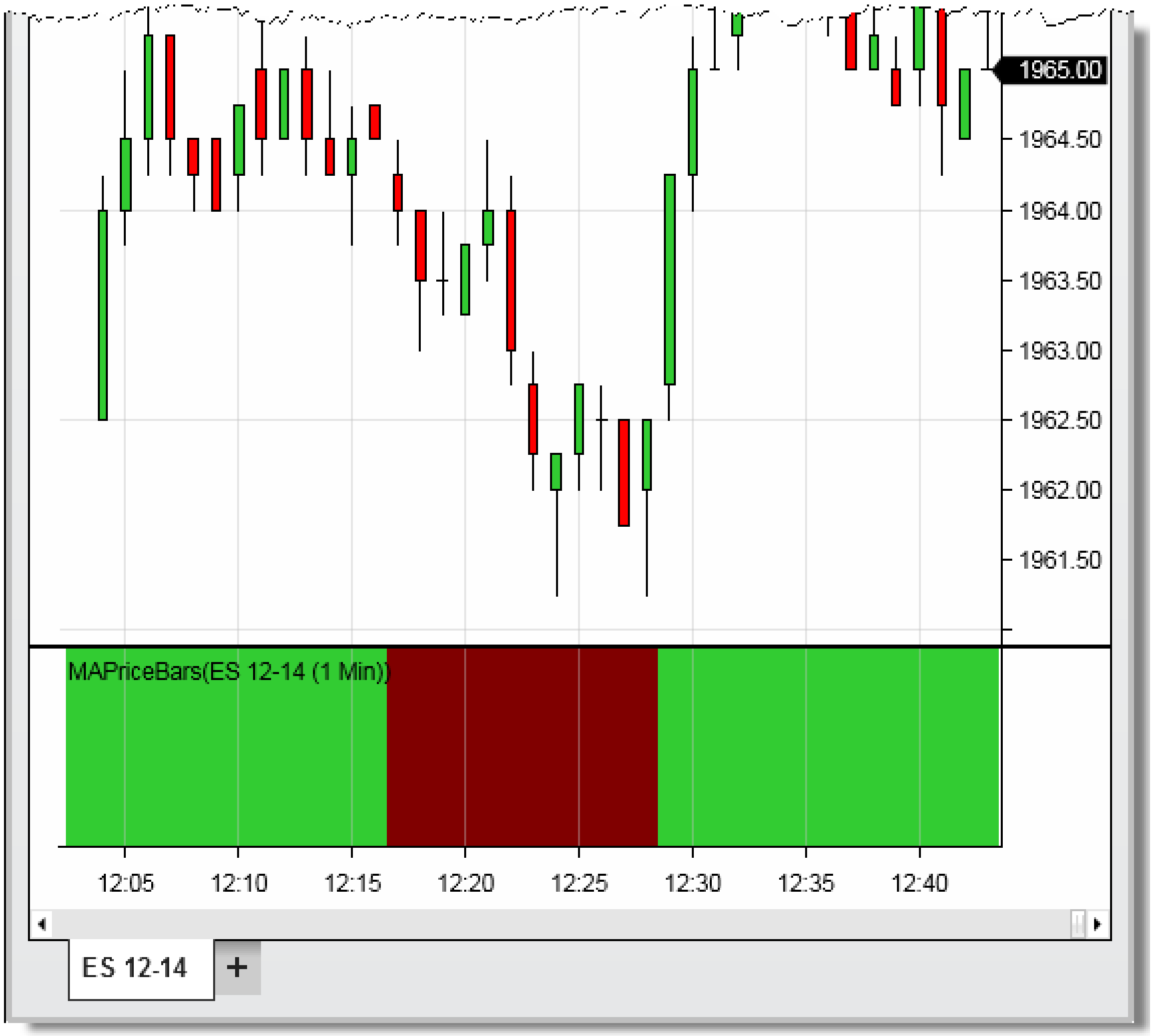
Syntax

```
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;
}
```



12.5.2.7.36 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 **BackBrush** to `null` for that bar.

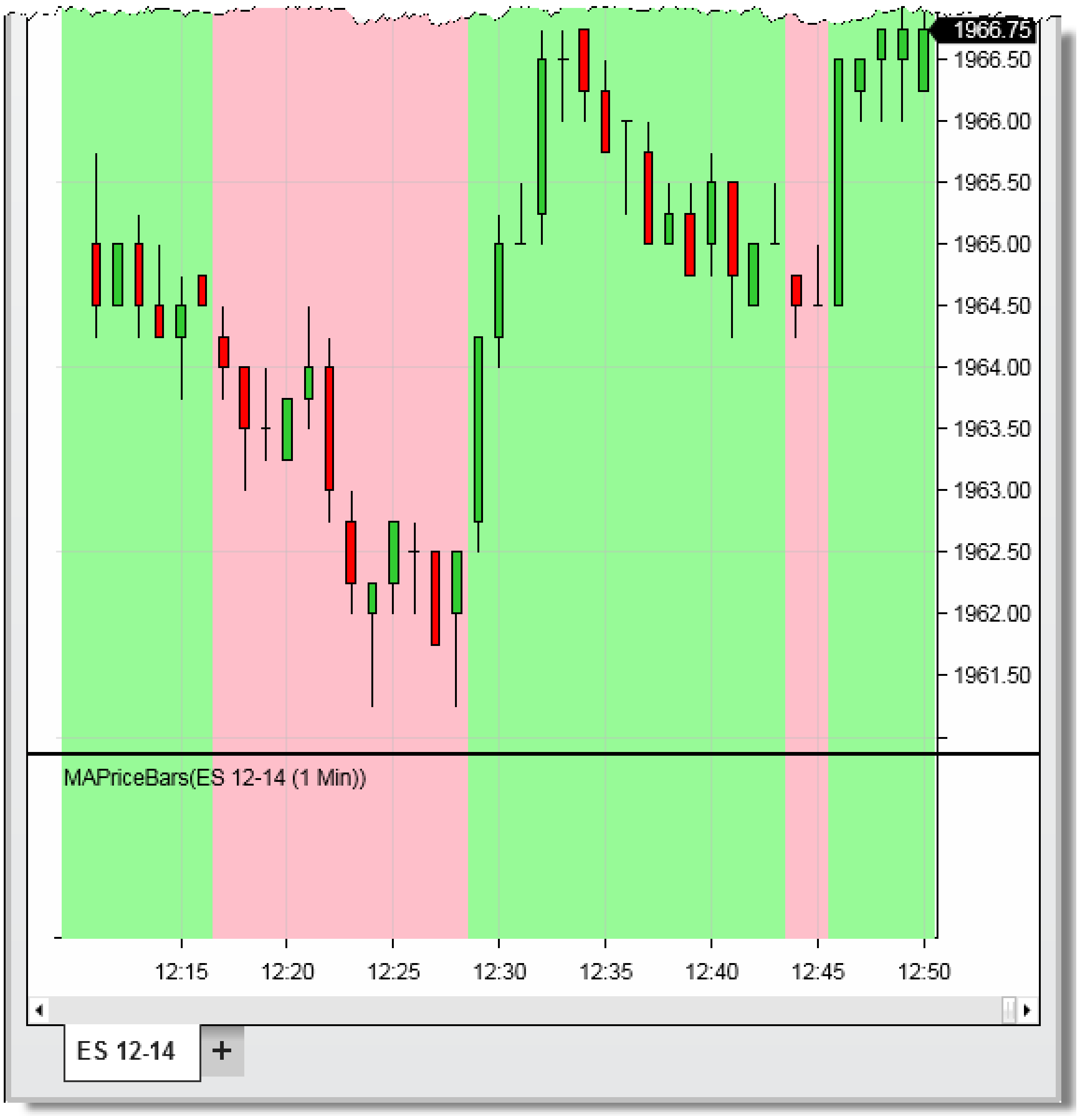
Syntax

```
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;
}
```



12.5.2.7.37 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;
}
```

12.5.2.7.38 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;
}
```

12.5.2.7.39 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 **BackBrush** 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;
}
```

12.5.2.7.40 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;

}
```

12.5.2.7.41 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;
```

12.5.2.7.42 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;
```

12.5.2.7.43 Draw Objects

Definition

A collection holding all of the drawn chart objects for the primary bar series. The draw objects can be manually drawn or script generated objects.

Note: 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.

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 tag a tag
   if (DrawObjects["someTag"].GetType().Name == "Line")
   {
      // Do something drawing tool line
   }
}
```

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
   foreach (DrawingTool draw in DrawObjects)
   {
      // 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.

12.5.2.7.44 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	A read-only 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
   }

}
```

12.5.2.7.45 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 in the PriceLevels collection
contain in myRetracements
   foreach (PriceLevel p in myRetracements.PriceLevels)
   {
      Print(p.Value);
   }
}
```

12.5.2.7.46 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");
```

12.5.2.7.47 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();
```

12.5.2.8 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);
            e++;
        }
    }
}
```

12.5.2.8.1 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.

12.5.2.8.1.1 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)

12.5.2.8.1.2 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)

12.5.2.8.1.3 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 09-14".

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)

12.5.2.8.1.4 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);
   }
}
```

12.5.2.8.1.5 MasterInstrument

Definition
An instrument's configuration settings. These are settings and properties which are defined in the Instrument window.

Warning: Properties in this class should **NOT** be accessed during **State.SetDefaults** from within the OnStateChange() method.

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.
NextExpiry()	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
```
!!! needsExample !!!
```

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
```
!!! needsExample !!!
```

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.

Parameters

price	A double value representing a price
round	An optional bool when true will round the price value to the nearest tick size

Syntax

```
Bars.Instrument.MasterInstrument.FormatPrice(double price)
```

Examples

```
protected override void OnBarUpdate()
{
   Print(Bars.Instrument.MasterInstrument.FormatPrice(Close[
0]));
}
```

Definition

Returns the type of instrument.

Property Value

An `InstrumentType` representing the type of an instrument.

Possible values are:
InstrumentType.Forex
InstrumentType.Future
InstrumentType.Index
InstrumentType.Option
InstrumentType.Stock

Syntax

```
Instrument.MasterInstrument.InstrumentType
```

Examples

```
!!! needsExample !!!
```

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

```

```

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
```
!!! needsExample !!!
```

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 next futures expiry for compared to the time of the input value used for the method.

Method Return Value
A `DataTime` structure

Syntax
```
Bars.Instrument.MasterInstrument.GetNextExpiry(DateTime afterDate)
```

Parameters
afterDate	A DateTime value representing to be compared

Examples
```
```

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

```
!!! needsExample !!!
```

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
```
!!! needsExample !!!
```

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

```
!!! needsExample !!!
```

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
```
!!! needsExample !!!
```

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
```
!!! needsExample !!!
```

12.5.2.9 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];
}
```

12.5.2.9.1 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.

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, 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.
// Create a DataSeries object and assign it to the variable
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // MaximumBarsLookBack determines how many values
the Series<double> will have access to
          myDoubleSeries = new Series<double>(this,
MaximumBarsLookBack.Infinite);
     }
}
```

Tip: `Series<T>` objects can be used on supplementary series in a multi-time frame and instrument strategy. Please see our support forum NinjaScript reference samples section for further information.

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]);
}
```

12.5.2.9.1.1 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]);
}
```

12.5.2.9.2 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 typica 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.

12.5.2.9.2.1 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");
}
```

12.5.2.9.2.2 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");
}
```

12.5.2.9.2.3 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());
}
```

12.5.2.9.2.4 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");
}
```

12.5.2.9.2.5 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 for example 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");
```

Tips

1. When working with multi-series indicators, Input always references BarsInProgress 0. Please be mindful as to when you access `Input[0]` as you will only be able to do so after BarsInProgress 0 has bars. To check to ensure BarsInProgress 0 has some bars you can use CurrentBars[0] to check.

12.5.2.9.2.6 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");
}
```

12.5.2.9.2.7 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];
```

12.5.2.9.2.8 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");
}
```

12.5.2.9.2.9 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];
```

12.5.2.9.2.10 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");
}
```

12.5.2.9.2.11 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];
```

12.5.2.9.2.12 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");
}
```

12.5.2.9.2.13 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];
```

12.5.2.9.2.14 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");

}
```

12.5.2.9.2.15 Value

Definition

A collection of historical references to the first ISeries object Values[0] in the indicator. This is the primary indicator value.

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];
}
```

12.5.2.9.2.16 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.

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];
}
```

12.5.2.9.2.17 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];
```

12.5.2.9.2.18 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");
}
```

12.5.2.9.3 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

12.5.2.9.3.1 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");
```

12.5.2.9.3.2 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");
}
```

12.5.2.9.4 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 times

12.5.2.9.4.1 Volume

Definition
A collection of historical bar volume values.

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");
}
```

12.5.2.9.4.2 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.

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");
}
```

12.5.2.9.5 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);
}
```

12.5.2.9.6 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
```

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));
   }
}
```

12.5.2.9.7 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>, IsValidPlot() 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

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];
}
```

12.5.2.9.8 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>, IsValidPlot() 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];
}
```

12.5.2.9.9 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.Configure)
   {
      // 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.Configure)
   {
      // 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)
   }
}
```

12.5.2.10 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.

Note: 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.

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");
}
```

12.5.2.10.1 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. Possible values:
	`BarsPeriodType.Tick`
	`BarsPeriodType.Volume`
	`BarsPeriodType.Second`
	`BarsPeriodType.Range`
	`BarsPeriodType.Minute`
	`BarsPeriodType.Day`
	`BarsPeriodType.Week`
	`BarsPeriodType.Month`
	`BarsPeriodType.Year`
	`BarsPeriodType.Kagi`
	`BarsPeriodType.LineBreak`
	`BarsPeriodType.PointAndFigure`
	`BarsPeriodType.Renko`

BarsPeriod.BaseBarsPeriodTyp	Only relevant for Kagi, LineBreak, and PointAndFigure Bars objects. Same possible

e	values as BarsPeriod.BarsPeriodType
BarsPeriod.BaseBarsPeriodValue	Only relevant for Kagi, LineBreak, and PointAndFigure 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
      }
}
```

12.5.2.10.2 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 which does not necessarily occur the end of the close of the bar

Property Value

An `enum` value determining the how frequently OnBarUpdate() will be called. Default value is set to Calculate.OnBarClose

Warning: 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 Calculate property since it is already utilizing the Calculate property of the parent script (i.e. the strategy your indicator is called from).
4. Scripts that require Calculate to be set by the developer must set this property in **State.Historical** in its OnStateChange() in order to ensure that if this script is a child (hosted) that the parent.Calculate property which is adopted by the child is overridden again.

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Calculate on the close of each bar
          Calculate = Calculate.OnBarClose;
     }
}
```

12.5.2.10.3 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

```
!!! needsExample !!!
```

Tips
1. 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.

12.5.2.10.4 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...
}
```

12.5.2.10.5 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;
     }
}
```

12.5.2.10.6 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`

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();
}
```

12.5.2.10.7 IsResetOnNew TradingDays

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
}
```

12.5.2.10.8 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);

      // Setting this option here for Primary does not take
affect the data series
      // Primary series must be configured from UI
      // IsTickReplays[0] = true;

     // 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; // AAPL, do not run tick
replay (saves CPU)

   }

}

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]);

   //Output:
   //ES 03-15 (1 Minute) True
   //AAPL (1 Minute) False
}
```

12.5.2.10.9 Update()

Definition

Forces the OnBarUpdate() method to be called so that indicator values are updated to the current bar. If the values are already up to date, the Update() method will not be run.

Note: This method is only relevant in specific use cases and should only used by advanced programmers

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()`

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;
     }
}
```

12.5.2.11 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.

Examples

```
protected override void
OnConnectionStatusUpdate(ConnectionStatusEventArgs
connectionStatusUpdate)
{
   if(connectionStatusUpdate.Status ==
ConnectionStatus.Connected)
   {
      Print("Connected at " + DateTime.Now);
   }

   else if(connectionStatusUpdate.Status ==
ConnectionStatus.ConnectionLost)
   {
      Print("Connection lost at: " + DateTime.Now);
   }
}
```

12.5.2.11.1 ConnectionStatusEventArgs

Definition

ConnectionStatusEventArgs contains Connection-related information to be passed as an argument to the OnConnectionStatusUpdate() event.

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));
    });
}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.2.12 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.

12.5.2.12.1 FundamentalDataEventArgs

Definition

Represents a change in fundamental data and is passed as a parameter in the OnFundamentalData() method.

Methods and Parameters

DateTimeValue	A `DateTime` structure 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
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. == .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.

12.5.2.13 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 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 guaranteed to always 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));
}
```

12.5.2.13.1 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);
}
```

12.5.2.14 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.

12.5.2.14.1 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);
}
```

Tips

1. For an example of how to use IsReset please see \MarketAnalyzerColumns\AskPrice.cs

12.5.2.15 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
State.Active	**Active** is called once after the object is configured and is ready to process data.	• Used for objects such as Share Service which do not process price series data • Indicate the object is ready to being processing information
State.Dat	**DataLoaded** is called only once	• Use for logic that needs to access

aLoaded	after all data series have been loaded.	data related objects like Bars, Instruments, BarsPeriod, TradingHours • Notify that all data series have been loaded
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 false (default behavior)	• Initialize any class level variables (including custom `Series<T>` objects) • 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 **false** (default behavior).
- Scripts that require Calculate to be set by the developer must set this property in **State.Historical** in order to ensure that if this script is a child (hosted) that the parent.Calculate property which is adopted by the child is overridden again.

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);
   }
}
```

12.5.2.15.1 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)
```

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;
   }
}
```

12.5.2.15.2 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
}
```

12.5.2.16 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.Historical)
   {
      //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
   }
}
```

12.5.2.16.1 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);
   }
}
```

12.5.2.16.2 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);
   }
}
```

12.5.2.16.3 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);
   }
}
```

12.5.2.16.4 ActualTradingDayExchange

Definition

Obtains the date of a trading session defined by the exchange.

Notes:
1. In order to obtain historical **ActualTradingDayEndLocal** 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);
   }
}
```

12.5.2.16.5 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);
}
```

12.5.2.16.6 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

```
rthSessionIterator 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
rthSessionIterator(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);
   }
}
```

12.5.2.16.7 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).ToStri
ng());
}
```

12.5.2.16.8 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

tradingDayExchang e	The DateTime value used to calculate the trading day.

Example

```
protected override void OnBarUpdate()
{
    // Only process strategy logic starting three hours after
trading begins at the exchange
    if (DateTime.Now >=
Bars.SessionIterator.GetTradingDayBeginLocal(Bars.SessionItera
tor.ActualTradingDayExchange).AddHours(3))
    {
        // Strategy logic here
    }
}
```

12.5.2.16.9 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 GetTradingDayBeginLocal() 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

tradingDayExchang e	The DateTime value used to calculate the trading day.

Example

```
protected override void OnBarUpdate()
{
    // Only process strategy logic up until three hours prior
to the end of the trading day at the exchange
    if (DateTime.Now <=
Bars.SessionIterator.GetTradingDayEndLocal(Bars.SessionIterato
r.ActualTradingDayExchange).AddHours(-3))
    {
        // Strategy logic here
    }
}
```

12.5.2.16.10 IsInSession()

Definition

Indicates a specified date is within the bounds of the current session, according to the configured Trading Hours template.

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);
}
```

12.5.2.16.11 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;
}
```

12.5.2.16.12 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));
```

12.5.2.17 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);
```

12.5.2.17.1 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);
```

12.5.2.17.2 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();
}
```

12.5.2.18 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)
- Adaptive Price Zone (APZ)
- Aroon
- Aroon Oscillator
- Average Directional Index (ADX)
- Average Directional Movement Rating (ADXR)
- Average True Range (ATR)
- Balance of Power (BOP)
- Bollinger Bands
- BuySell Pressure
- BuySell Volume
- CandleStickPattern
- Chaikin Money Flow
- Chaikin Oscillator
- Chaikin Volatility
- Chande Momentum Oscillator (CMO)
- Commodity Channel Index (CCI)
- Current Day OHL
- Darvas

- Directional Movement (DM)
- Directional Movement Index (DMI)
- Donchian Channel
- Double Stochastics
- Dynamic Momentum Index (DMIndex)
- Ease of Movement
- Fisher Transform
- Forecast Oscillator (FOSC)
- Keltner Channel
- KeyReversalDown
- KeyReversalUp
- Linear Regression
- Linear Regression Intercept
- Linear Regression Slope
- MA Envelopes
- Maximum (MAX)
- Minimum (MIN)
- Momentum
- Money Flow Index (MFI)
- Moving Average - Double Exponential (DEMA)
- Moving Average - Exponential (EMA)
- Moving Average - Hull (HMA)
- Moving Average - Kaufman's Adaptive (KAMA)
- Moving Average - Mesa Adaptive (MAMA)
- Moving Average - Simple (SMA)
- Moving Average - T3 (T3)
- Moving Average - Triangular (TMA)
- Moving Average - Triple Exponential (TEMA)
- Moving Average - Triple Exponential (TRIX)
- Moving Average - Variable (VMA)
- Moving Average - Volume Weighted (VWMA)
- Moving Average - Weighted (WMA)
- Moving Average - Zero Lag Exponential (ZLEMA)
- Moving Average Convergence-Divergence (MACD)
- n Bars Down
- n Bars Up
- On Balance Volume (OBV)
- Parabolic SAR
- Percentage Price Oscillator (PPO)
- Pivots
- Polarized Fractal Efficiency (PFE)
- Price Oscillator

- Prior Day OHLC
- Range
- Range Indicator (RIND)
- Rate of Change (ROC)
- Regression Channel
- Relative Spread Strength (RSS)
- Relative Strength Index (RSI)
- Relative Volatility Index (RVI)
- R-squared
- Standard Deviation (StdDev)
- Standard Error (StdError)
- Stochastics
- Stochastics Fast
- Stochastics RSI (StochRSI)
- Summation (SUM)
- Swing
- Time Series Forecast (TSF)
- True Strength Index (TSI)
- Ultimate Oscillator
- Volume (VOL)
- Volume Moving Average (VOLMA)
- Volume Oscillator
- Volume Rate of Change (VROC)
- Volume Up Down
- Williams %R
- Woodies CCI
- Woodies Pivots
- ZigZag

12.5.2.18.1 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 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];
```

12.5.2.18.2 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.

12.5.2.18.3 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.

12.5.2.18.4 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.

12.5.2.18.5 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.

12.5.2.18.6 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 using default
price type
double value = ADX(20)[0];
Print("The current ADX value is " + value.ToString());

// Prints the current value of a 20 period ADX using high
price type
double value = ADX(High, 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.

12.5.2.18.7 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.

12.5.2.18.8 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.

12.5.2.18.9 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.

12.5.2.18.10 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.

12.5.2.18.11 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)
    {
        // You have to set this specific indicator's Calculate
property
        // to Calculate.OnEachTick if the strategy, indicator
or Quote Board Column's Calculate
        // property is set to true, otherwise the indicator
will not calculate it's values
        BuySellPressure().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.

12.5.2.18.12 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)
    {
        // You have to set this specific indicator's Calculate
property
        // to Calculate.OnEachTick if the strategy, indicator
or Quote Board Column's Calculate
        // property is set to true, otherwise the indicator
will not calculate it's values
        BuySellVolume().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().Sellss[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.

12.5.2.18.13 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

`double`; 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.

12.5.2.18.14 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.

12.5.2.18.15 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.

12.5.2.18.16 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.

12.5.2.18.17 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.

12.5.2.18.18 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.

12.5.2.18.19 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 (?)
period	Number of bars used in the calculation

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.

12.5.2.18.20 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.

12.5.2.18.21 Directional Movement (DM)

Description

Same as the ADX indicator with the addition of the +DI and -DI values.

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.

12.5.2.18.22 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 Investopedia

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.

12.5.2.18.23 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());

// Prints the current lower value of a 20 period DonchianChannel using high
price type
double value = DonchianChannel(High, 20).Lower[0];
Print("The current DonchianChannel 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.

12.5.2.18.24 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.

12.5.2.18.25 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.

12.5.2.18.26 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.

12.5.2.18.27 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.

12.5.2.18.28 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.

12.5.2.18.29 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.

12.5.2.18.30 KeyReversalDow n

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.

12.5.2.18.31 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.

12.5.2.18.32 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.

12.5.2.18.33 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.

12.5.2.18.34 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.

12.5.2.18.35 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.

12.5.2.18.36 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());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

12.5.2.18.37 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());
```

Source Code

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

12.5.2.18.38 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.

12.5.2.18.39 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.

12.5.2.18.40 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.

12.5.2.18.41 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.

12.5.2.18.42 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.

12.5.2.18.43 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.

12.5.2.18.44 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.

12.5.2.18.45 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.

12.5.2.18.46 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 vfactorof 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.

12.5.2.18.47 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.

12.5.2.18.48 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.

12.5.2.18.49 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.

12.5.2.18.50 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.

12.5.2.18.51 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.

12.5.2.18.52 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.

12.5.2.18.53 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.

12.5.2.18.54 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.

12.5.2.18.55 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.

12.5.2.18.56 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.

12.5.2.18.57 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 (?)
period	Number of bars used in the calculation

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.

12.5.2.18.58 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.

12.5.2.18.59 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.

12.5.2.18.60 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());
}
```

12.5.2.18.61 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.

12.5.2.18.62 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.

12.5.2.18.63 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 (?)
period	Number of bars used in the calculation

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.

12.5.2.18.64 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 current value of a 20 period ROC using default price type
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.

12.5.2.18.65 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.

12.5.2.18.66 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.

12.5.2.18.67 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.

12.5.2.18.68 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.

12.5.2.18.69 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.

12.5.2.18.70 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.

12.5.2.18.71 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.

12.5.2.18.72 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.

12.5.2.18.73 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.

12.5.2.18.74 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(3, 14, 7).D[0];
Print("The current Stochastics %D value is " + value.ToString());

// Prints the current %K value
double value = Stochastics(3, 14, 7).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.

12.5.2.18.75 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.

12.5.2.18.76 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.

12.5.2.18.77 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.

12.5.2.18.78 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.

12.5.2.18.79 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.

12.5.2.18.80 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.

12.5.2.18.81 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.

12.5.2.18.82 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.

12.5.2.18.83 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.

12.5.2.18.84 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.

12.5.2.18.85 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.

12.5.2.18.86 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.

12.5.2.18.87 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.

12.5.2.18.88 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());
```

12.5.2.18.89 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());
}
```

12.5.2.18.90 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.

12.5.2.19 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

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

12.5.2.19.1 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
}
```

12.5.2.19.2 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);
}
```

12.5.2.19.3 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()
{
   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

}
```

12.5.2.19.4 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);
}
```

12.5.2.19.5 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].DayOfWeek = Monday,
Bars.TradingHours.Sessions[1].DayOfWeek = Tuesday,
Bars.TradingHours.Sessions[2].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));
}
```

12.5.2.19.6 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));
}
```

12.5.2.20 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
```
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
}
```

12.5.2.21 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";
   }
}
```

12.5.2.22 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. The default behavior of this property will including 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
}
```

12.5.2.23 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.

Property Value

A `bool` value when **true** will be displayed on the chart; otherwise **false**; default value is **true**.

Syntax

`IsVisible`

Examples

```
public override void OnRender(ChartControl chartControl, ChartScale
chartScale)
{
  // only render the currently as visible selected price levels in our
NinjaScript drawing object
  foreach (PriceLevel priceLevel in PriceLevels.Where(pl => pl.IsVisible
&& pl.Stroke != null)
  {
       Vector startDir              = priceLevel.Value / 100 *
(startPoint2 - startPoint);
       Vector lineVector            = endPoint - startPoint;
       Point newStartPoint          = new Point(startPoint.X + startDir.X,
startPoint.Y + startDir.Y);
       Point newEndPoint            = new Point(newStartPoint.X +
lineVector.X, newStartPoint.Y + lineVector.Y);

       RenderTarget.DrawLine(newStartPoint.ToVector2(),
newEndPoint.ToVector2(), priceLevel.Stroke.BrushDX,
priceLevel.Stroke.Width, priceLevel.Stroke.StrokeStyle);
  }
}
```

12.5.2.24 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";
   }
}
```

12.5.2.25 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 during or after **State.Terminated**. In effect, attempting to trigger custom events may be unavailable 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

Tip: 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.

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
   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);
}

private System.Windows.Forms.Timer myTimer = new
System.Windows.Forms.Timer();

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name  = "SampleTriggerCustomEventTimer";
   }
   else if (State == State.Configure)
   {
      // Adds the event and the event handler for the method
that will
      // process the timer event to the timer.
      myTimer.Tick += new EventHandler(TimerEventProcessor);

      // Sets the timer interval to 5 seconds.
      myTimer.Interval = 5000;
      myTimer.Start();
   }
```

Using TiggerCustomEvent 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)
```

12.5.3 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.

12.5.3.1 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.

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.

12.5.3.1.1 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
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;
```

C#

```
          accountSelector.SelectedAccount.PositionUpdate
+= OnPositionUpdate;
               }
          };
     }

     // Called by TabControl when tab is being removed or
window is closed
     public override void Cleanup()
     {
          // Clean up our resources
          accountSelector.Cleanup();
     }

     // Other required NTTabPage members left out for
demonstration purposes. Be sure to add them in your own code.
}
```

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=NinjaTra
der.Gui"
   xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui
"
   xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=Nin
jaTrader.Gui">

<Grid>
     <Tools:AccountSelector x:Name="accountSelector"
HorizontalAlignment="Left" VerticalAlignment="Top"/>
</Grid>
```

12.5.3.1.2 AtmStrategySelector

Definition

AtmStrategySelector is an UI element users can interact with for selecting ATM Strategies.

Events and Properties

Cleanup()	Disposes of the AtmStrategySelector
CustomProperties Changed	Event handler for when properties have changed on the ATM strategy
Id	A string identifying the ATM Strategy selector
SelectedAtmStrate gy	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

Example: 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.TimeInF
orce = 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.AtmStrategySel
ector.AccountProperty,
          new Binding { Source = accountSelector, Path = new
PropertyPath("SelectedAccount") });

     // When our ATM selector's selection changes
     atmStrategySelector.SelectionChanged += (o, args) =>
     {
          if (atmStrategySelector.SelectedItem == null)
```

Examples

Example: 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=NinjaTra
der.Gui"
       xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui
"
       xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=Nin
jaTrader.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>
```

12.5.3.1.3 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
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
     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();
          instrumentSelector.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=NinjaTrad
er.Gui"
       xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui"

       xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=Ninj
aTrader.Gui">

<Grid>
     <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition Width="*"/>
     </Grid.ColumnDefinitions>
     <Tools:InstrumentSelector x:Name="instrumentSelector"
Grid.Column="0" LastUsedGroup="MyAddOn"/>
</Grid>
```

12.5.3.1.4 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

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();
}
```

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=NinjaTrad
er.Gui"
       xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui"

       xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=Ninj
aTrader.Gui">

<Grid>
     <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition Width="*"/>
     </Grid.ColumnDefinitions>

     <Tools:IntervalSelector x:Name="intervalSelector"
```

```
Grid.Column="0" HorizontalAlignment="Left"/>
</Grid>
```

12.5.3.1.5 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
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.TimeInFo
rce = 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.AtmStrategySele
ctor.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)
          {
               // 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
     atmStrategySelector.Cleanup();
     tifSelector.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.Gui"
       xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui"
       xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=NinjaTrader.G
ui">

<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>
```

12.5.3.1.6 QuantityUpDow n

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.TimeInFo
rce = 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.AtmStrategySele
ctor.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)
          {
               // 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
     atmStrategySelector.Cleanup();
     tifSelector.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=NinjaTrad
er.Gui"
       xmlns:AccountData="clr-
namespace:NinjaTrader.Gui.AccountData;assembly=NinjaTrader.Gui"

       xmlns:AtmStrategy="clr-
namespace:NinjaTrader.Gui.NinjaScript.AtmStrategy;assembly=Ninj
aTrader.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>
```

12.5.3.2 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;
     }
}

private void OnAccountStatusUpdate(object sender,
AccountStatusEventArgs e)
{
```

12.5.3.2.1 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

12.5.3.2.2 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("Accou
nt: {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.
}
```

12.5.3.2.3 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("Accou
nt: {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.
}
```

12.5.3.2.4 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));
    }
}
```

12.5.3.2.5 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 });
}
```

12.5.3.2.6 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);
}
```

12.5.3.2.7 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 });
}
```

12.5.3.2.8 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)
    {
        // Initialize myAccount
    }
}

private void OnAccountStatusUpdate(object sender,
AccountStatusEventArgs e)
{
    Print(String.Format("{0} connection updated",
myAccount.Connection.Options.Name));
}
```

12.5.3.2.9 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.

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;
        }
}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.3.2.10 CreateOrder()

Definition

Creates an Order to be submitted via Submit().

Syntax

CreateOrder(Instrument *instrument*, OrderAction *action*, OrderType *orderType*, TimeInForce *timeInForce*, int *quantity*, double *limitPrice*, double *stopPrice*, string *oco*, string *name*, 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
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"
customOrder	Custom order if it is being used

Examples

```
Order stopOrder;
stopOrder = myAccount.CreateOrder(myInstrument,
OrderAction.Sell, OrderType.StopMarket, TimeInForce.Day, 1, 0,
1400, "myOCO", "stopOrder", null);

myAccount.Submit(new[] { stopOrder });
```

12.5.3.2.11 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);
   }
}
```

12.5.3.2.12 Executions

Definition
A collection of `Execution` objects generated for the specified account

Property Value

An Collection of `Execution` objects

Syntax

```
<Account>.Executions
```

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));
    }
}
```

12.5.3.2.13 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.
}
```

12.5.3.2.14 Flatten()

Definition

Flattens the account on an instrument.

Syntax

```
Flatten(ICollection<Instrument> instruments)
```

Parameters

instruments	A collection of Instrument of the orders to be cancelled

Examples

Flatten a single instrument

```
Account.Flatten(new [] { Instrument.GetInstrument("ES 12-
15") });
```

Flatten a list of instruments

```
// instantiate a list of instruments
Collection<Cbi.Instrument> instrumentsToCancel = new
Collection<Instrument>();

// add instruments to the collection
instrumentsToCancel.Add(Instrument.GetInstrument("AAPL"));

instrumentsToCancel.Add(Instrument.GetInstrument("MSFT"));

// pass the instrument collection to the Flatten() method to
be flattened
Account.Flatten(instrumentsToCancel);
```

12.5.3.2.15 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

Examples

```
// Evaluates to see if the account has more than $25000
if (Account.Get(AccountItem.CashValue, Currency.UsDollar) >
25000)
{
   // Do something;
}
```

12.5.3.2.16 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));
}
```

12.5.3.2.17 Orders

Definition
A collection of `Order` objects generated for the specified account

Property Value

An Collection of `Order` objects

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));
    }
}
```

12.5.3.2.18 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, TimeInForce.Day,
                         e.Quantity, e.AverageFillPrice + 10 *
e.Order.Instrument.MasterInstrument.TickSize, 0, oco, "Profit Target",
null);
                    stopLoss     =
account.CreateOrder(e.Order.Instrument, OrderAction.Sell,
OrderType.StopMarket, TimeInForce.Day,
                         e.Quantity, 0, e.AverageFillPrice - 10 *
e.Order.Instrument.MasterInstrument.TickSize, oco, "Stop Loss", 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.
}
```

12.5.3.2.19 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

```
protected override void OnStateChange()
{
   if (State == State.DataLoaded)
   {
        lock (Account.Positions)
        {
            Print("Positions in State.DataLoaded:");

            foreach(Position position in Account.Positions)
            {
                Print(String.Format("Position: {0} at {1}",
position.MarketPosition, position.AveragePrice));
            }
        }
    }
}
```

12.5.3.2.20 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.
}
```

12.5.3.2.21 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.
}
```

12.5.3.2.22 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));
    }
}
```

12.5.3.2.23 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 });
```

12.5.3.3 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. Remember to unsubscribe from the .Update Event handler if you are no longer using the subscription.

Syntax

```
BarsRequest(Cbi.Instrument instrument, int barsBack)
BarsRequest(Cbi.Instrument instrument, DateTime fromLocal, DateTime toLocal)
```

Parameters

Instrument	The Instrument to request
bars	An `int` value determining the number of bars to request from

Back	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),
```

12.5.3.3.1 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);
     }
}));
```

12.5.3.3.2 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();
```

12.5.3.4 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)
         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.
   }
}
```

12.5.3.4.1 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);
}
```

12.5.3.4.2 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 void 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.
}
```

12.5.3.4.3 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 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()
     {
          // Subscribe to execution 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. */
          ConnectStatusEventArgs 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.
}
```

12.5.3.4.4 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();
}
```

12.5.3.4.5 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.
}
```

12.5.3.4.6 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));
}
```

12.5.3.4.7 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));
}
```

12.5.3.4.8 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;
   }

}
```

12.5.3.5 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
}
```

12.5.3.5.1 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();
     }
}
```

12.5.3.6 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
}
```

12.5.3.6.1 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; }
```

12.5.3.7 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();
      }
}
```

12.5.3.7.1 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();
}
```

12.5.3.7.2 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();
}
```

12.5.3.8 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; }
}
```

12.5.3.8.1 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);
}
```

12.5.3.8.2 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);
}
```

12.5.3.8.3 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, download the AddOn Framework Example located on our File Sharing forum.

Examples

```
// IWorkspacePersistence member
public WorkspaceOptions WorkspaceOptions { get; set; }
```

12.5.3.9 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.

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; }
}
```

12.5.3.9.1 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;
}
```

12.5.3.9.2 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
}
```

12.5.3.9.3 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
}
```

12.5.3.10 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.In
strument.GetInstrument("MSFT"), this, "someId",
NinjaTrader.Core.Globals.Now, Priority.High, "message", null,
new SolidColorBrush(Colors.Blue), new
SolidColorBrush(Colors.White), 0);

// Instead of ResetAlert()
NinjaTrader.NinjaScript.Alert.ResetAlertRearmById("someId");
```

Miscellaneous

Instead of Log(), use Log.Process() to send a message to NinjaTrader logs.
Instead of PlaySound(), use Globals.PlaySound() to play a sound.
Instead of SendMail(), use Globals.SendMail() to send a mail.

Examples

```
// Instead of Log()
NinjaTrader.Cbi.Log.Process(typeof(Resource), "someName", new
object[] { "My log message" }, LogLevel.Error,
LogCategories.Default, null)

// 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

12.5.3.10.1 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 AletCallBack() 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.In
strument.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);
```

12.5.3.10.2 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;
}
```

12.5.3.11 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 the AddOn Framework Example located on our File Sharing forum.
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;
        }
    }
};
```

12.5.3.12 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.

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;
}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.3.13 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

CalendarYearHigh Date	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
FiveYearsGrowthP ercentage	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
NextYearsEarning sPerShare	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 FundamentalData fundamentalData;

     public MyAddOnTab()
     {
          // Subscribe to fundamental data. Snapshot data is
provided right on subscription
          fundamentalData = new FundamentalData(value);
          fundamentalData.Update += OnFundamentalData;

          // Printing snapshot fundamental data for average
daily volume
          NinjaTrader.Code.Output.Process(fundamentalData.Aver
ageDailyVolume, 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 (fundamentalData != null)
               fundamentalData.Update -= OnFundamentalData;
     }

     // Other required NTTabPage members left out for
demonstration purposes. Be sure to add them in your own code.
}
```

12.5.3.14 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 even 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 MarketData marketData;

     public MyAddOnTab()
     {
          // 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()
          marketData = new MarketData(instrument);
          marketData.Update += OnMarketData;

          // Printing snapshot market data for the last price
and time
          NinjaTrader.Code.Output.Process(marketData.Last.Pric
e.ToString() + " " + 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 (marketData != null)
               marketData.Update -= OnMarketData;
     }

     // Other required NTTabPage members left out for
demonstration purposes. Be sure to add them in your own code.
}
```

12.5.3.15 MarketDepth

Definition

MarketDepth can be used to access snapshot market depth and for subscribing to market depth events.

Note: Remember to unsubscribe if you are no longer using the subscription.

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. The concept can be carried over
to any NinjaScript object you may be working on. */
public class MyAddOnTab : NTTabPage
{
     private MarketDepth<MarketDepthRow> marketDepth;

     public MyAddOnTab()
     {
          // 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()
          marketDepth = new
MarketDepth<MarketDepthRow>(instrument);
          marketDepth.Update += OnMarketDepth;
     }

     // This method is fired on market depth events and after
the snapshot data is updated.
     private void OnMarketDepth(object sender,
MarketDepthEventArgs e)
     {
          // Print the Ask's price ladder
          for (int i = 0; i < marketDepth.Asks.Count; i++)
          {
               NinjaTrader.Code.Output.Process(string.Format("
Position: {0} Price: {1} Volume: {2}", i,
                    marketDepth.Asks[i].Price,
marketDepth.Asks[i].Volume), PrintTo.OutputTab1);
          }
     }

     // 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 (marketDepth != null)
               marketDepth.Update -= OnMarketDepth;
     }

     // Other required NTTabPage members left out for
demonstration purposes. Be sure to add them in your own code.
}
```

12.5.3.16 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.
}
```

12.5.3.17 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.
}
```

12.5.3.18 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.

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;
}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.3.19 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 the AddOn Framework Example located on our File Sharing forum.

```
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)
```

12.5.3.20 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.

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;
}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.3.21 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 += ;
     }

     // 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(()=> new MyWindow().Show()));
     }
}
```

12.5.3.22 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;
          }
     }
}
```

12.5.3.23 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);
}
```

12.5.3.24 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);
}
```

12.5.3.25 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 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;

     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.StartAtmStr
ategy("myAtmStrategyName", entryOrder);
          }
     }

     // Other required NTTabPage members left out for
demonstration purposes. Be sure to add them in your own code.
}
```

12.5.3.26 StrategyBase

StrategyBase contains properties and methods for managing a Strategy object, and is the base class from which AtmStrategy derives.

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));
    }
}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.3.27 PropagateInstrumentChange()

Definition

In an NTWindow, PropagateInstrumentChange() sends an Instrument to other windows with the same Instrument Linking color configured.

Note: A public Instrument property must be defined in order to use

PropagateInstrumentChange(), as in the example below

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);
    }
}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.3.28 PropagateIntervalChange()

Definition

In an NTWindow, PropagateIntervalChange() sends an interval to other windows with the same Interval Linking color configured.

Note: A public Instrument property must be defined in order to use PropagateInstrumentChange(), as in the example below

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);
}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.3.29 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.

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();
    }
}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.3.30 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.

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 Simulation Color selected in the Options menu is visible in the tab background when a simulation account is selected in the tab
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());

}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.4 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.
ApplyDefaultBasePeriodValue	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.

12.5.4.1 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));
```

12.5.4.2 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;
    }

}
```

12.5.4.3 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;
}
```

12.5.4.4 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) 14, BarsPeriodTypeName =
"MyCustomBarsType(14)", Value = 1 };
      BuiltFrom     = BarsPeriodType.Minute;  // update
OnDataPoint() every minute on historical data
      DaysToLoad     = 5;
   }

   else if (State == State.Configure)
   {
   }
}
```

12.5.4.5 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.

12.5.4.6 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).TotalDay
s / barsSeries.BarsPeriod.Value)) /
               daysInMonth; // an estimate
     }
     return 1;
}
```

12.5.4.7 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;
   }
}
```

12.5.4.8 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; } }
```

12.5.4.9 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));
}
```

12.5.4.10 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

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 the incoming data is a bar from the data provider source
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;
}
```

12.5.4.11 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);
```

12.5.4.12 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);

}
```

12.5.4.13 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);
```

12.5.5 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
BarWidth UI	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
SetProper	Sets a default property name to a custom string to be

tyName()	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
UpBrushDX	A SharpDX.Brush object used to paint the up bars for the ChartStyle

12.5.5.1 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;
   }
}
```

12.5.5.2 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);

}
```

12.5.5.3 ChartStyleType

Definition

Defines a unique identifier value used to register a custom ChartStyle. There are 8 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.

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	Mountain

Examples

```
protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name = "Example ChartStyle";
      ChartStyleType    = (ChartStyleType) 80;
      BarWidth       = 1;
   }
}
```

12.5.5.4 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");
    }
}
```

12.5.5.5 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;
        }
}
```

12.5.5.6 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);
}
```

12.5.5.7 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;
   }
}
```

12.5.5.8 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");
    }
}
```

12.5.5.9 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
}
```

12.5.5.10 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.

12.5.5.11 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);
}
```

12.5.5.12 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");
    }
}
```

12.5.5.13 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;
        }
}
```

12.5.6 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
IgnoresUserI	Determines if the drawing tool can be clicked on by the

nput	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

12.5.6.1 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);
   }
}
```

12.5.6.2 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);
   }
}
```

12.5.6.3 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
```

12.5.6.4 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);
      }
   }
}
```

12.5.6.4.1 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);
}
```

12.5.6.4.2 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";
    }
}
```

12.5.6.4.3 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.SampleDrawingTo
          ol
     }
     else if (State == State.Configure)
     {

     }
}
```

12.5.6.4.4 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
```

12.5.6.4.5 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);
```

12.5.6.4.6 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)
     {

     }
}
```

12.5.6.4.7 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;

        }
    }
}
```

12.5.6.4.8 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`

12.5.6.4.9 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)
     {

     }
}
```

12.5.6.4.10 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)
     {

     }
}
```

12.5.6.4.11 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(Point startPoint, Point deltaPoint, ChartControl
chartControl, ChartScale)
<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,
chartScale);
```

12.5.6.4.12 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);
```

12.5.6.4.13 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);
```

12.5.6.4.14 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
}
```

12.5.6.4.15 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
}
```

12.5.6.4.16 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
}
```

12.5.6.4.17 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);
```

12.5.6.4.18 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);
```

12.5.6.4.19 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);
```

12.5.6.5 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
}
```

12.5.6.6 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
}
```

12.5.6.7 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;
   }
}
```

12.5.6.8 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();
}
```

12.5.6.9 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
   }

}
```

12.5.6.10 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);
}
```

12.5.6.11 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 bar series used for input will be returned.

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)
}
```

12.5.6.12 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;
}
```

12.5.6.13 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;
   }
}
```

12.5.6.14 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 } ;
}
```

12.5.6.15 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;
   }
}
```

12.5.6.16 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)
     {

     }
}
```

12.5.6.17 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)
     {

     }
}
```

12.5.6.18 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;
}
```

12.5.6.19 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;
}
```

12.5.6.20 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;
}
```

12.5.6.21 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
}
```

12.5.6.22 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
}
```

12.5.6.23 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.

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 apoint 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;
   }
}
```

12.5.6.24 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.

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);
   }
}
```

12.5.6.25 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

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 released 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;
}
```

12.5.6.26 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; } }
```

12.5.6.27 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)
   {
   }
}
```

12.5.7 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

12.5.7.1 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)
        {
            Cbi.Log.Process(typeof (Custom.Resource),
"ImportTypeNinjaTraderUnableReadData", new object[]
{FileNames[currentInstrumentIdx], exp.Message},
Cbi.LogLevel.Error, Cbi.LogCategories.Default);
            continue;
        }
}
```

12.5.7.2 OnNextDataPoint()

Definifition

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
    }
}
```

12.5.8 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.
DisplayInDa	Determines if plot(s) display in the chart data box.

taBox	
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.
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.

12.5.8.1 AddLine()

Definition
Adds line objects on a chart.

Note: Line configurations 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 set the line configuration. 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

AreLinesConfigurable	Determines if the line(s) used in an indicator are 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; }
```

12.5.8.1.1 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
     }
}
```

12.5.8.1.2 Line Class

Definition
Objects derived from the Line class are used to characterize how an oscillator line is visually displayed (plotted) on a chart.

Syntax
`Line(Stroke stroke)`

Parameters
stroke	The stroke style

Examples
See the AddLine() method for examples.

12.5.8.1.3 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;
   }
}
```

12.5.8.2 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: Plot configurations 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 set the plot configuration. Alternatively, you may use custom public Brush, Stroke, or **PlotStyle** properties which are accessible in **State.SetDefault** 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.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

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,
"myPlot");
   }
}
```

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; }
```

12.5.8.2.1 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
     }
}
```

12.5.8.2.2 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");
     }
}
```

12.5.8.2.3 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 (Falling(Upper))
          PlotBrushes[0][0] = Brushes.Red;
     else
          PlotBrushes[0][0] = Brushes.Yellow;

     // Color the Lower plot based on plot value conditions
     if (Rising(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]; }
}
```

12.5.8.2.4 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;
   }
}
```

12.5.8.3 BarsRequiredToPlot

Definition

The number of bars on a chart required before the script plots.

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");
     }
}
```

12.5.8.4 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");
     }
}
```

12.5.8.5 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** bet 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");
     }
}
```

12.5.8.6 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**.

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");
     }
}
```

12.5.8.7 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");
     }
}
```

12.5.8.8 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
     }
}
```

12.5.8.9 IsSuspendedWhileInactive

Definition

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. Enabling this property in your indicator helps save CPU cycles while the hosting feature is suspended and not in use by a user. Once the hosting feature is in a state that would no longer be considered suspended, the historical OnBarUpdate() events will be replayed allowing the indicator to catch up to current real-time values.

Suspension occurs when the indicator resides on:

- 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;
      }
}
```

12.5.8.10 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");
     }
}
```

12.5.8.11 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");
     }
}
```

12.5.9 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.

12.5.9.1 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";
}
```

12.5.9.2 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;
}
```

12.5.9.3 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;
     }
}
```

12.5.9.4 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;
}
```

12.5.9.5 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;
     }
}
```

12.5.9.6 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);
    }
}
```

12.5.10 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.

12.5.10.1 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.
Drawdown;
}
```

12.5.10.2 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.
Drawdown;
}
```

12.5.11 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

IsAborted	Gets a value indicating if the optimization was aborted.

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.

12.5.11.1 IsAborted

Definition
Gets a value indicating if the optimization was aborted.

Property Value
This property returns **true** if the optimization was aborted; otherwise, **false**.

Syntax
```
IsAborted
```

Examples
```
!!! needsExample !!!
```

12.5.11.2 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;
}
```

12.5.11.3 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
}
```

12.5.11.4 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];
}
```

12.5.11.5 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();
}
```

12.5.11.6 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;
     }
}
```

12.5.12 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.

12.5.12.1 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();
}
```

12.5.12.2 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 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;
}
```

12.5.12.3 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);
}
```

12.5.12.4 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);
}
```

12.5.12.5 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
```

12.5.12.6 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;
}
```

12.5.13 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 8 comes pre-configured with **Share Services** for Facebook, Twitter, StockTwits, and an E-Mail 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.
IsAuthorizationRequired	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, Twitter, Facebook, etc).
IsImageAttachmentSupported	Determines if the Share Service will allow for images as attachments.
OnAuthorizeAccount()	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.
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.

12.5.13.1 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.

Syntax

`CharacterLimit`

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        CharacterLimit      = 140;
    }
}
```

12.5.13.2 CharactersReservedPerMedia

Definition

Sets the number of characters allowed when attaching an image to ensure that character count is properly calculated.

Note: Social networks like Twitter or StockTwits, 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.

Syntax

`CharactersReservedPerMedia`

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        CharactersReservedPerMedia  = 40;
    }
}
```

12.5.13.3 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;
   }
}
```

12.5.13.4 IsAuthorizationRequired

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.

Syntax

```
UseOAuth
```

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        IsAuthorizationRequired     = true;
    }
}
```

12.5.13.5 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 by 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;
}
```

12.5.13.6 IsDefault

Definition
Sets the default Share Service used when the type of sharing service is selected (email, Twitter, Facebook, etc).

For example, if you are using two different Twitter adapters, you may set one to be the default when the user selects the Twitter sharing service. Setting this property as the default would only apply to any Twitter 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.

Syntax
`Default`

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        Default       = false;
    }
}
```

12.5.13.7 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.

Syntax
`IsImageAttachmentSupported`

Examples

```
protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        Default       = false;
    }
}
```

12.5.13.8 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!");
}
```

12.5.13.9 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
}
```

12.5.13.10 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 sent from NinjaTrader 8"
at the end of the message".

protected override void OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Signature     = "This message sent from NinjaTrader 8";
   }
}
```

```
//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;
    }
}
```

12.5.14 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.
ChartIndicato	Contains a collection of Indicators which have been

rs	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.
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.
OnPositionUp	An event driven method which is called each time the

date()	position of a strategy changes state.
Optimization Period	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.
OrderFillRes olution	Determines how strategy orders are filled during historical states.
OrderFillRes olutionType	Determines the bars type which will be used for historical fill processing.
OrderFillRes olutionValue	Determines the bars period interval value which will be used for historical fill processing.
Performance Metrics	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.
PositionAcco unt	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.
PositionsAcc ount	Holds an array of PositionAccount objects that represent positions managed by the strategy's account.
RealtimeErro	Defines the behavior of a strategy when a strategy

rHandling	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.
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.

12.5.14.1 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
```
!!! needsExample !!!
```

12.5.14.2 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.
  - 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.

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.Configure**

Parameters

indicator	An indicator object

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          // 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.Configure)
   {
      // 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];
}
```

12.5.14.3 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.SetDefaults)
     {
          AddPerformanceMetric(new
NinjaTrader.NinjaScript.PerformanceMetrics.SampleCumProfit());
     }
}
```

12.5.14.4 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()

12.5.14.4.1 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[3] == "Working")
   {
      AtmStrategyCancelEntryOrder("orderId");
   }
}
```

12.5.14.4.2 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");
}
```

12.5.14.4.3 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");
}
```

12.5.14.4.4 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");
}
```

12.5.14.4.5 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 on the Hosting NinjaScripts UI Thread. A callback is provided to check when the ATM Strategy is started on that thread.
- 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
    }
}
```

12.5.14.4.6 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());
     }
}
```

12.5.14.4.7 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");
}
```

12.5.14.4.8 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());
}
```

12.5.14.4.9 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());
}
```

12.5.14.4.10 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());
}
```

12.5.14.4.11 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());
          }
     }
}
```

12.5.14.4.12 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());
}
```

12.5.14.4.13 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();
}
```

12.5.14.5 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
}
```

12.5.14.6 BarsSinceEntryExecution()

Definition

Returns the number of bars that have elapsed since the last specified entry.

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();

}
```

12.5.14.7 BarsSinceExitExecution()

Definition

Returns the number of bars that have elapsed since the last specified exit.

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();
}
```

12.5.14.8 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.Configure)
{
   AddChartIndicator(SMA(20));

   // Set the plots color for the added indicator
   ChartIndicators[0].Plots[0].Brush = Brushes.Blue;

   // Set the added indicator to new panel
   ChartIndicators[0].Panel = 1;
}
```

12.5.14.9 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.

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

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);
}
```

12.5.14.10 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;
     }
}
```

12.5.14.11 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;
     }
}
```

12.5.14.12 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;
     }
}
```

12.5.14.13 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;
     }
}
```

12.5.14.14 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);
}
```

12.5.14.1!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);
}
```

12.5.14.10 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 = 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());
}
```

12.5.14.17 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;
     }
}
```

12.5.14.18 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;
     }
}
```

12.5.14.19 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 set to **true** by default when a strategy is applied to a chart. However, in any other window (Strategies tab, Strategy Analyzer), it will be set to **false** by default. If you are working with a strategy outside of a chart, and would like to save orders, trades, and execution history for reference in your code, you will need to set **IncludeTradeHistoryInBacktest** to **true** in your script.
- During Strategy Analyzer Optimization in a 32-bit process, the **IncludeTradeHistoryInBacktest** property is forced to **false** due to the limited resources available in a 32-bit environment. You must use a 64-bit process if trade history is needed during optimization.

Property Value
This property returns **true** if the strategy will include trade history; otherwise, **false**. Default is set to **true**. Always **false** during a strategy analyzer optimization on a 32-bit process.

Warning: This property should **ONLY** bet set from the OnStateChange() method during **State.SetDefaults** or **State.Configure**

Syntax
```
IncludeTradeHistoryInBacktest
```

Examples

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          // Explicitly include trade history in a backtest
          IncludeTradeHistoryInBacktest = true;
     }
}

protected override void OnBarUpdate()
{
    // Stop taking trades after 10 trades have been taken
since the strategy was enabled
    if(SystemPerformance.AllTrades.Count >= 10)
        return;
}
```

12.5.14.2(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;
     }
}
```

12.5.14.2IsExitOnSessionCloseStrategy

Definition

Determines if the strategy will cancel all strategy generated orders and close all open strategy positions at the close of the session. 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)

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;
     }
}
```

12.5.14.22 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;
     }
}
```

12.5.14.2 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, resulting in a performance increase concerning the time required and computer resources used to run an optimization. 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 state 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();
```

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
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
      myTrades = new Dictionary<DateTime, string>();

      //Any strategy defaults which are maintained do not need
to be reset if that 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;
      myOrderObject = null;
      myBrushObject = null;
```

12.5.14.24 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");
     }
}
```

12.5.14.25 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;
}
```

12.5.14.20 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;
     }
}
```

12.5.14.27 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
}
```

12.5.14.27.1 AccountItemEventArgs

Definition

AccountItemEventArgs contains Account-related information to be passed as an argument to the OnAccountItemUpdate() event.

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));
        });

}
```

Note: For a complete, working example of this class in use, download the AddOn Framework Example located on our File Sharing forum.

12.5.14.28 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 always called after OnOrderUpdate() is called
- Only orders which have been submitted and managed by the strategy will call OnExecutionUpdate()

Notes:

- If you are relying on the OnExecutionUpdate() method to trigger actions such as the submission of a stop loss order when your entry order is filled, **ALWAYS** reference the

properties on the Order object property attached to the Execution object passed into the OnExecutionUpdate() method.
- 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.

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 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

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 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 (execution.Order.Name == "myEntryOrder")
        entryOrder = 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());
}
```

Additional Reference Samples

Additional reference code samples are available the NinjaScript Educational Resources section of our support forum.

12.5.14.29 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.

12.5.14.3(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().
- If you are relying on the OnOrderUpdate() method to trigger actions such as the submission of a stop loss order when your entry order is filled **ALWAYS** reference the properties on the Order object passed into the OnOrderUpdate() method.
- 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.

Multi-threaded consideration

NinjaTrader is a multi-threaded application and therefore it is extremely important to understand the following concepts:

- This method guarantees that you will see each order state change in sequence
- This method does not provide an update for the most current state of an order but instead

provides you an event notifying you of each state change in sequence and the relevant information on the order at the time the state changed

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" and 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 Order object passed in will reflect the "Working" state although the actual order is really in a state of "Part Filled" which is truly reflected in the original Order object returned in any of the order methods such as EnterLong(). Of course, the OnOrderUpdate() method will subsequently receive the event for "PartFilled" state.

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 representing the order
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, cancelled, 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.BrokerOrderError ErrorCode.InvalidInstrument ErrorCode.LoginFailed ErrorCode.NoError ErrorCode.NotConnected ErrorCode.NotSupported ErrorCode.OrderRejected ErrorCode.Panic ErrorCode.ServerConnectionIsBroken 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.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

```
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.

12.5.14.3 OnPositionUpdate()

Definition

An event driven method which is called each time the position of a strategy changes state.

- This method is called after OnExecutionUpdate()
- OnPositionUpdate() is guaranteed to be called for every change in strategy position

NOTE: 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.

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 representing the most recent position update
averageFillPrice	A `double` value representing the average fill price of an order
quantity	An `int` value representing the quantity of an order
marketPosition	A MarketPosition object representing the position provided directly from the broker. Possible values are: • MarketPosition.Flat • MarketPosition.Long • MarketPosition.Short

Examples

```
protected override void OnPositionUpdate(Position position,
double averagePrice, int quantity, MarketPosition
marketPosition)
{
    if (position.MarketPosition == MarketPosition.Flat)
    {
        // Do something like reset some variables here
    }
}
```

Additional Reference Samples

Additional reference code samples are available the NinjaScript Educational Resources section of our support forum.

12.5.14.3: 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;
     }
}
```

12.5.14.3 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
IsLiveUntilCancelled	A `bool` when true indicates the order will be canceled by managed order handling at

	expiration, otherwise false
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
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

StopPrice	A `double` value representing the stop price of an order
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.Cancel	Order cancellation is submitted to the broker

Submitted	
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;
    }
}
```

12.5.14.33.1 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(entryOrder.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;
   }
}
```

12.5.14.3.4 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()
- SetProfitTarget()
- SetStopLoss()
- SetTrailStop()

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()
- IgnoreOverfill
- IsUnmanaged
- SubmitOrderUnmanaged()

12.5.14.34.1 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

* Via the SetProfitTarget(), SetStopLoss() and SetTrailStop() 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()
{
     Add("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() 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. 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 an 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
- 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 an 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 short 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.
EnterSho	Generates a sell MIT order to enter a short position.

rtMIT()	
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.
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.

12.5.14.34.1.1 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;

     if (entryOrder != null && entryOrder == order)
     {
          Print(order.ToString());
          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.

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 OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name    = "Example Strategy";
   }
   // one time only, as we transition from historical
to real-time
   else if (State == State.Realtime)
   {
      // convert any old historical order object
references
      // to the new live order submitted to the real-
time account
      if (entryOrder != null)
        entryOrder = GetRealtimeOrder(entryOrder);
   }
}

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)
{

   // the code below is added to illustrate how an
order ID may change through out the lifetime of the
order.
   // It is critical you update your order references
in these scenarios.
   if (order.Name == "myEntryOrder")
   {
      if (State == State.Historical)
      {
         Print(order.OrderId) // e.g., NT-0001
      }

      else if (State == State.Realtime)
      {
         Print(order.OrderId) // e.g., 3259392555
      }
   }
}
```

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. Foe 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 == "myEntryOrder" && orderState !=
 OrderState.Filled)
         entryOrder = order;
}
```

12.5.14.34.1.2 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" && orderState ==
OrderState.Filled)
        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;
        }
    }
}
```

12.5.14.34.1.3 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);
}
```

12.5.14.34.1.4 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.

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");
}
```

12.5.14.34.1.5 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.
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");
}
```

12.5.14.34.1.6 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.
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");
}
```

12.5.14.34.1.7 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.
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");
}
```

12.5.14.34.1.8 EnterLongStopMarket()

Definition

Generates a buy stop 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

```
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.
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");
}
```

12.5.14.34.1.9 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.
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");
}
```

12.5.14.34.1.10 EnterShortLimit()

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

`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.
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");
}
```

12.5.14.34.1.11 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.
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");
}
```

12.5.14.34.1.12 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.
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");
}
```

12.5.14.34.1.13 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.
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");
}
```

12.5.14.34.1.14 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

12.5.14.34.1.15 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

12.5.14.34.1.16 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

12.5.14.34.1.17 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];
     }

     // 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

12.5.14.34.1.18 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

12.5.14.34.1.19 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) && CrossAbove(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 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

12.5.14.34.1.20 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) && CrossAbove(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 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

12.5.14.34.1.21 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) && CrossAbove(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 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

12.5.14.34.1.22 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) && CrossAbove(SMA(10),
SMA(20), 1))
     {
          EnterShort("SMA Cross Entry");
          stopPrice = Low[0];
     }

     // Exits position
     ExitShortStopLimit(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

12.5.14.34.1.23 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) && CrossAbove(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 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

12.5.14.34.1.24 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). You only need to call this method once (e.g., during OnStateChange() as the strategy State transitions from **State.Historical** to **State.Realtime**). 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 OnStateChange()
{
   if (State == State.SetDefaults)
   {
      Name     = "Example Strategy";
   }

   // one time only, as we transition from historical to real-
time
   else if (State == State.Realtime)
   {
      // convert any old historical order object references
      // to the new live order submitted to the real-time
account
      if (myOrder != null)
         myOrder = GetRealtimeOrder(myOrder);
   }
}
```

12.5.14.34.1.25 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
- 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 include:

	• CalculationMode.Currency • CalculationMode.Percent • CalculationMode.Price • CalculationMode.Ticks • CalculationMode.Pips
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);
     }
}
```

12.5.14.34.1.26 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:

- The SetStopLoss() method can **NOT** be used concurrently with the SetTrailStop() method for the same position, if both methods are called for the same position (fromEntrySignal) the SetStopLoss() will always take precedence. You can however, use both methods in the same strategy if they reference different signal names.
- Stop loss orders are submitted in real-time on incoming executions from entry orders
- 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.

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 • CalculationMode.Percent • CalculationMode.Price • CalculationMode.Ticks • CalculationMode.Pips
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);
     }
}
```

12.5.14.34.1.27 SetTrailStop()

Definition

Generates a trail stop order with the signal name "Trail stop" to exit a position. Trail stops are amended on a tick by tick basis. 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() method for the same position, if both methods are called for the same position (fromEntrySignal) the SetStopLoss() will always take precedence. You can however, use both 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
- 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.

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 • CalculationMode.Ticks • CalculationMode.Pips
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

12.5.14.34.2 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.

s

```
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.AvgPrice && Close[0] >= Position.AvgPrice + 4
 * TickSize)
          ChangeOrder(stopOrder, stopOrder.Quantity,
0, Position.AvgPrice);
}
```

Order Cancellation

Similar to the live until cancelled technique from the Managed approach, cancelling 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

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 cancelled. 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 overfillcan 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.
SubmitO	Generates an Unmanaged order.

rderUnmanaged()	

12.5.14.34.2.1 CancelOrder()

Please see the "CancelOrder()" section under the "Managed Approach".

12.5.14.34.2.2 ChangeOrder()

Please see the "ChangeOrder()" section under the "Managed Approach".

12.5.14.34.2.3 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;
    }
}
```

12.5.14.34.2.4 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;
     }
}
```

12.5.14.34.2.5 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 grouop 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 OnExecutionUpdate(Execution execution,
 string executionId, double price, int quantity,
MarketPosition marketPosition, string orderId, DateTime time)
{
    // 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;
}
```

12.5.14.3! 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;
   }
}
```

12.5.14.30 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;
   }

}
```

12.5.14.37 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;
   }

}
```

12.5.14.38 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]);
}
```

12.5.14.39 Plots

See the Plots page under the Indicators section.

12.5.14.40 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);
}
```

12.5.14.40.1 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);
}
```

12.5.14.40.2 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, double.MinValue will be used.

Parameters

unit	Possible values: PerformanceUnit.Currency PerformanceUnit.Percent PerformanceUnit.Pips PerformanceUnit.Points PerformanceUnit.Ticks
price	Price 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]));
}
```

12.5.14.40.3 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
    }
}
```

12.5.14.40.4 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]));
}
```

12.5.14.40.5 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());
}
```

12.5.14.41 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);
}
```

12.5.14.41.1 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);
}
```

12.5.14.41.2 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, double.MinValue will be used.

Parameters

unit	Possible values: PerformanceUnit.Currency PerformanceUnit.Percent PerformanceUnit.Pips PerformanceUnit.Points PerformanceUnit.Ticks
price	Price 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]));
}
```

12.5.14.41.3 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
    }
}
```

12.5.14.41.4 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]));
}
```

12.5.14.41.5 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());
}
```

12.5.14.42 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);
}
```

12.5.14.4 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);
}
```

12.5.14.44 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
      }
   }
}
```

12.5.14.45 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;
     }
}
```

12.5.14.46 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

Syntax
`SetOrderQuantity`

Examples

```
protected override void OnStateChange()
{
     if (State == State.Configure)
     {
          SetOrderQuantity =
SetOrderQuantity.DefaultQuantity; // calculate orders based
off default size
     }
}
```

12.5.14.47 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;
     }
}
```

12.5.14.48 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;
     }
}
```

12.5.14.49 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 may not match strategy position under a partial fill scenario)

Syntax
`StopTargetHandling`

Tip: The default strategy behavior is to match the order quantity used for the stops and targets to overall strategy position. However in cases where the strategy's entry order is partially filled, StopTargetHandling.ByStrategyPosition 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.PerEntryExecution. However this may result in more stops and targets submitted than the overall strategy position in a scenario in which the strategy's entire entry orders are not filled.

Example

```
protected override void OnStateChange()
{
     if (State == State.SetDefaults)
     {
          StopTargetHandling =
StopTargetHandling.PerEntryExecution;
     }
}
```

12.5.14.50 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.");
}
```

12.5.14.50.1 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.");
}
```

12.5.14.50.2 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.");
}
```

12.5.14.50.3 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.");
}
```

12.5.14.50.4 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.");
}
```

12.5.14.51 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;
     }
}
```

12.5.14.57 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;
       }
     }
}
```

12.5.14.5 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.

12.5.14.54 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.RealTimeTra
des.Count - 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));
    }
}
```

12.5.14.5 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);
        }
    }
}
```

12.5.14.55.1 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.");
}
```

12.5.14.55.2 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.AllTr
ades.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);
     }
}
```

12.5.14.55.3 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.");
}
```

12.5.14.55.4 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.All
Trades.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);
     }
}
```

12.5.14.55.5 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
AverageEntryEfficien	A `double` value representing the average entry

cy	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
Commission	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
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.MaxConsecu
tiveLoser < 5)
    {
        // Trade logic here
    }
}
```

12.5.14.55.5.1 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.AverageBarsInTra
de);
}
```

12.5.14.55.5.2 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.AverageEntryEffi
ciency);
}
```

12.5.14.55.5.3 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.AverageExitEffic
iency);
}
```

12.5.14.55.5.4 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.AverageTimeInMar
ket);
}
```

12.5.14.55.5.5 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.AverageTotalEffi
ciency);
}
```

12.5.14.55.5.6 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.Average
Profit);
}
```

12.5.14.55.5.7 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);
}
```

12.5.14.55.5.8 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);
}
```

12.5.14.55.5.9 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.LongestFlatPerio
d);
}
```

12.5.14.55.5.10 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.MaxConsecutiveLo
ser);
}
```

12.5.14.55.5.11 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.MaxConsecutiveWi
nner);
}
```

12.5.14.55.5.12 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
);
}
```

12.5.14.55.5.13 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);
}
```

12.5.14.55.5.14 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);
}
```

12.5.14.55.5.15 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);
}
```

12.5.14.55.5.16 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.AverageP
rofit);
}
```

12.5.14.55.5.17 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.PerformanceMetri
cs.Length);

     // Find a the value of a specific custom Performance
Metric named "MyPerformanceMetric"
     for (int i = 0; i <
SystemPerformance.AllTrades.TradesPerformance.PerformanceMetri
cs.Length; i++)
     {
          if
(SystemPerformance.AllTrades.TradesPerformance.PerformanceMetr
ics[i] is
               NinjaTrader.NinjaScript.PerformanceMetrics.MyPe
rformanceMetric)
          {
               Print((SystemPerformance.AllTrades.TradesPerfor
mance.PerformanceMetrics[i] as
                    NinjaTrader.NinjaScript.PerformanceMetrics
.MyPerformanceMetric).Values[0]);
          }
     }
}
```

12.5.14.55.5.18 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.AverageProf
it);
}
```

12.5.14.55.5.19 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.AveragePr
ofit);
}
```

12.5.14.55.5.20 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);
}
```

12.5.14.55.5.21 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);
}
```

12.5.14.55.5.22 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.RiskFreeRet
urn = 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);
}
```

12.5.14.55.5.23 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.RiskFreeRet
urn = 0;

     // Print out the Sharpe ratio of all trades based on a
zero risk free return
     Print("Sharpe ratio is: " +
SystemPerformance.AllTrades.TradesPerformance.SharpeRatio);
}
```

12.5.14.55.5.24 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.RiskFreeRet
urn = 0;

     // Print out the Sortino ratio of all trades based on a
zero risk free return
     Print("Sortino ratio is: " +
SystemPerformance.AllTrades.TradesPerformance.SortinoRatio);
}
```

12.5.14.55.5.25 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.AveragePro
fit);
}
```

12.5.14.55.5.26 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)
;
}
```

12.5.14.55.5.27 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);
}
```

12.5.14.55.5.28 TotalSlippage

Definition
Returns the total slippage.

Property Value
A `double` value that represents the total slippage.

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);
}
```

12.5.14.55.5.29 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);
}
```

12.5.14.55.5.30 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);
}
```

12.5.14.55.6 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.Al
lTrades.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);
     }
}
```

12.5.14.5(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.CumProfit >
1000)
          return;
}
```

12.5.14.56.1 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>.AvgEtd`

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.AvgEtd)
;
}
```

12.5.14.56.2 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>.AvgMae`

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.AvgMae)
;
}
```

12.5.14.56.3 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>.AvgMfe`

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.AvgMfe)
;
}
```

12.5.14.56.4 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>.AvgProfit`

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.AvgProf
it);
}
```

12.5.14.56.5 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.CumProf
it);
}
```

12.5.14.56.6 Draw dow n

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.DrawDow
n);
}
```

12.5.14.56.7 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.Largest
Loser);
}
```

12.5.14.56.8 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.Largest
Winner);
}
```

12.5.14.56.9 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.ProfitP
erMonth);
}
```

12.5.14.56.10 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)
;
}
```

12.5.14.56.11 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.Turnaro
und);
}
```

12.5.14.56.12 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);
}
```

12.5.14.57 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.

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;
     }
}
```

12.5.15 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).

12.5.15.1 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));
}
```

12.5.15.2 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
   }
}
```

12.5.15.3 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
}
```

12.5.15.4 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
}
```

12.5.15.5 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();
```

12.5.15.6 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
}
```

12.5.15.7 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()
}
```

12.6 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-install 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.

12.6.1 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.

12.6.1.1 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

12.6.1.2 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

12.6.1.3 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

12.6.1.4 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())

12.6.1.4.1 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()
```

12.6.1.4.2 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
```

12.6.1.5 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.

12.6.1.6 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.

12.6.1.7 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.

12.6.1.8 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.Media.Brush** to a **SharpDX.Direct2D1.Brush**

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.

12.6.2 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.

12.6.2.1 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.

12.6.2.2 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.

12.6.2.3 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.

12.6.2.4 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.

12.6.2.4.1 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
```

12.6.2.4.2 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
```

12.6.2.5 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.

12.6.2.6 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.

12.6.2.7 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.

12.6.2.8 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

12.6.2.9 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.

12.6.2.10 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.

12.6.2.11 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.

12.6.2.12 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.

12.6.2.12.1 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.

12.6.2.12.2 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.

12.6.2.12.3 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.

12.6.2.12.4 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.

12.6.2.12.5 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

12.6.2.12.6 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().

12.6.2.12.7 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.

12.6.2.13 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.

12.6.2.14 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,`	Creates an **GradientStopCollection** from the specified gradient stops,

`GradientStop[] gradientStops, ExtendMode extendMode)`	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.)

12.6.2.14.1
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
```

12.6.2.14.2 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
```

12.6.2.14.3 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
```

12.6.2.15 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.)

12.6.2.15.1 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
```

12.6.2.15.2 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
```

12.6.2.15.3 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
```

12.6.2.16 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.

12.6.2.17 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_QUALIT Y.

12.6.2.18 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.

12.6.2.18.1 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
```

12.6.2.18.2 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.

12.6.2.18.3 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

12.6.2.18.4 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

12.6.2.18.5 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
```

12.6.2.18.6 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.

12.6.2.19 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`	Creates an RadialGradientBrush

`RadialGradientBrush(RenderTarget renderTarget, RadialGradientBrushProperties radialGradientBrushProperties, GradientStopCollection gradientStopCollection)`	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 radialGradientBrushProperties, Nullable<BrushProperties> brushProperties, GradientStopCollection gradientStopCollection)`	Creates an RadialGradientBrush that contains the specified gradient stops and has the 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.)

12.6.2.19.1 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
```

12.6.2.19.2 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
```

12.6.2.19.3 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
```

12.6.2.19.4 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
```

12.6.2.19.5 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
```

12.6.2.20 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.

12.6.2.21 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.

12.6.2.21.1 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
```

12.6.2.21.2 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.

12.6.2.21.3 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.

12.6.2.21.4 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.

12.6.2.21.5 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.

12.6.2.21.6 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.MeasuringMo

	de 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.

12.6.2.21.7 DrawTextLayout()

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.

12.6.2.21.8 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.

12.6.2.21.9 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

12.6.2.21.10 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.

12.6.2.21.11 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
```

12.6.2.22 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.)

12.6.2.22.1 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
```

12.6.2.23 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.

12.6.2.23.1 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
```

12.6.2.23.2 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
```

12.6.2.23.3 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`

12.6.2.23.4 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
```

12.6.2.23.5 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
```

12.6.2.23.6 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.

12.6.2.23.7 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
```

12.6.2.23.8 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
```

12.6.2.23.9 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
```

12.6.2.24 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 MeterLImit 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.

12.6.2.25 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.

12.6.3 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.

12.6.3.1 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.

12.6.3.1.1 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`

12.6.3.1.2 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
```

12.6.3.1.3 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
```

12.6.3.1.4 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).

12.6.3.1.5 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
```

12.6.3.1.6 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).

12.6.3.1.7 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
```

12.6.3.1.8 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
```

12.6.3.1.9 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
```

12.6.3.1.10 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
```

12.6.3.2 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.

12.6.3.3 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.)

12.6.3.3.1 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

12.6.3.3.2 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
```

12.6.3.3.3 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 height.

Syntax

```
<TextLayout>.MaxHeight
```

12.6.3.3.4 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)