```
#region Using declarations
using System;
using System.ComponentModel;
using System.Diagnostics;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Xml.Serialization;
using NinjaTrader.Cbi;
using NinjaTrader.Data;
using NinjaTrader.Gui.Chart;
#endregion

// This namespace holds all indicators and is required. Do not change it.
namespace NinjaTrader.Indicator
{
    /// anaSuperTrendM1 Indicator
    /// </summary>
    [Description("Modified SuperTrend Indicator based on a Moving Median")]
    public class anaSuperTrendM1 : Indicator
    {
        #region Variables
                private int             periodMedian      = 3; // Default setting
for Median Period
            private int             periodATR         = 3; // Default setting for
Range Period
            private double          multiplier        = 2; // Default setting for
Multiplier
                private bool            candles                = false;
                private bool            gap                    = false;
                private bool            paintBars         = false;
                private bool            showArrows        = true;
                private bool            showStopDots      = true;
                private bool            showStopLine      = true;
                private bool            currentUpTrend    = true;
                private bool            priorUpTrend      = true;
                private double          offset                 = 0;
                private double          newStop                = 0;
                private double          priorStop         = 0;
                private int             plot0Width        = 1;
                private PlotStyle       plot0Style        = PlotStyle.Dot;
                private DashStyle dash0Style        = DashStyle.Dot;
                private int             plot1Width        = 1;
                private PlotStyle       plot1Style        = PlotStyle.Line;
                private DashStyle       dash1Style        = DashStyle.Solid;
                private Color           upColor                = Color.RoyalBlue;
                private Color           downColor         = Color.Red;
                private Color           neutralColor      = Color.Transparent;
                private Color           priorColor        = Color.Empty;
                private BoolSeries      upTrend;
                private anaMovingMedian MM;
                private ATR MAE;
        #endregion

        /// <summary>
        /// This method is used to configure the indicator and is called once
before any bar data is loaded.
        /// </summary>
        protected override void Initialize()
```

```
        {
                Add(new Plot(Color.Gray, PlotStyle.Dot, "StopDot"));
            Add(new Plot(Color.Gray, PlotStyle.Line, "StopLine"));
                CalculateOnBarClose    = false;
            Overlay                    = true;
            PriceTypeSupported     = false;
                PlotsConfigurable = false;
                upTrend = new BoolSeries(this);
        }

            protected override void OnStartUp()
            {
                Plots[0].Pen.Width = plot0Width;
                Plots[0].PlotStyle = plot0Style;
                Plots[0].Pen.DashStyle = dash0Style;
                Plots[1].Pen.Width = plot1Width;
                Plots[1].PlotStyle = plot1Style;
                Plots[1].Pen.DashStyle = dash1Style;
                candles = false;
                if (ChartControl != null && ChartControl.ChartStyleType ==
ChartStyleType.CandleStick)
                    candles = true;
                gap = (plot1Style == PlotStyle.Line)||(plot1Style ==
PlotStyle.Square);
                if (ShowStopDots)
                    Plots[0].Pen.Color = Color.Gray;
                else
                    Plots[0].Pen.Color = Color.Transparent;
                if (ShowStopLine)
                    Plots[1].Pen.Color = Color.Gray;
                else
                    Plots[1].Pen.Color = Color.Transparent;
                MM = anaMovingMedian(Medians[0], periodMedian);
                MAE = ATR(Closes[0],periodATR);
            }

            /// <summary>
            /// Called on each bar update event (incoming tick)
            protected override void OnBarUpdate()
        {
                if (CurrentBar == 0)
                {
                    upTrend.Set(true);
                    StopDot.Set (Close[0]);
                    StopLine.Set (Close[0]);
                    PlotColors[0][0] = neutralColor;
                    PlotColors[1][0] = neutralColor;
                    return;
                }
                if (FirstTickOfBar)
                {
                    priorUpTrend = upTrend[1];
                    priorStop = StopLine[1];
                    priorColor = PlotColors[0][1];
                    offset = MAE[1];
                }
                if (Close[0] > priorStop)
                {
```

```
                        upTrend.Set(true);
                        newStop = MM[0] - Multiplier * offset;
                        currentUpTrend = true;
                        if (!priorUpTrend) // trend change up
                        {
                              StopDot.Set(newStop);
                              StopLine.Set(newStop);
                        }
                        else
                        {
                              StopDot.Set(Math.Max(newStop,priorStop));
                              StopLine.Set(Math.Max(newStop,priorStop));
                        }
                  }
                  else if (Close[0] < priorStop)
                  {
                        upTrend.Set(false);
                        newStop = MM[0]+ Multiplier * offset;
                        currentUpTrend = false;
                        if (priorUpTrend) // trend change down
                        {
                              StopDot.Set(newStop);
                              StopLine.Set(newStop);
                        }
                        else
                        {
                              StopDot.Set(Math.Min(newStop,priorStop));
                              StopLine.Set(Math.Min(newStop,priorStop));
                        }
                  }
                  else
                  {
                        upTrend.Set(priorUpTrend);
                        currentUpTrend = priorUpTrend;
                        StopDot.Set(priorStop);
                        StopLine.Set(priorStop);
                  }

                  if(PaintBars)
                  {
                        if (currentUpTrend)
                        {
                              CandleOutlineColor = upColor;
                              BarColor = upColor;
                        }
                        else
                        {
                              CandleOutlineColor = downColor;
                                    BarColor  = downColor;
                        }
                        if(Open[0] < Close[0] && candles)
                              BarColor  = Color.Transparent;
                  }
                  if(ShowArrows)
                  {
                        if(currentUpTrend && !priorUpTrend)
                              DrawArrowUp("arrow" + CurrentBar, true, 0, newStop -
0.5*offset, upColor);
```

```
                        else if(!currentUpTrend && priorUpTrend)
                              DrawArrowDown("arrow" + CurrentBar, true, 0, newStop
+ 0.5*offset, downColor);
                        else
                              RemoveDrawObject("arrow" + CurrentBar);
                  }
                  if (ShowStopDots)
                  {
                        if(currentUpTrend)
                              PlotColors[0][0] = upColor;
                        else
                              PlotColors[0][0] = downColor;
                  }
                  if(ShowStopLine)
                  {
                        if(currentUpTrend && !priorUpTrend)
                        {
                              if (gap)
                                    PlotColors[1][0]= neutralColor;
                              else
                                    PlotColors[1][0] = upColor;
                        }
                        else if (currentUpTrend)
                              PlotColors[1][0] = upColor;
                        else if(!currentUpTrend && priorUpTrend)
                        {
                              if (gap)
                                    PlotColors[1][0]= neutralColor;
                              else
                                    PlotColors[1][0] = downColor;
                        }
                        else
                              PlotColors[1][0] = downColor;
                  }
            }

            #region Properties
                /// <summary>
                /// </summary>
                [Browsable(false)]
                [XmlIgnore]
                public DataSeries StopDot
                {
                      get { return Values[0]; }
                }

                /// <summary>
                /// </summary>
                [Browsable(false)]
                [XmlIgnore]
                public DataSeries StopLine
                {
                      get { return Values[1]; }
                }

                [Browsable(false)]
            [XmlIgnore()]
            public BoolSeries UpTrend
```

```
{
    get { return upTrend; }
}

    [Description("Median Period")]
[Category("Parameters")]
    [Gui.Design.DisplayName("Median Period")]
public int PeriodMedian
{
    get { return periodMedian; }
    set { periodMedian = Math.Max(1, value); }
}

    [Description("ATR Period")]
[Category("Parameters")]
    [Gui.Design.DisplayName("ATR Period")]
public int PeriodATR
{
    get { return periodATR; }
    set { periodATR = Math.Max(1, value); }
}

[Description("ATR Multiplier")]
[Category("Parameters")]
    [Gui.Design.DisplayName("ATR Multiplier")]
public double Multiplier
{
    get { return multiplier; }
    set { multiplier = Math.Max(0.0, value); }
}

    [Description("Color the bars in the direction of the trend?")]
[Category("Options")]
    [Gui.Design.DisplayName ("Paint Bars")]
public bool PaintBars
{
    get { return paintBars; }
    set { paintBars = value; }
}

    [Description("Show Arrows when Trendline is violated?")]
[Category("Options")]
    [Gui.Design.DisplayName ("Show Arrows")]
public bool ShowArrows
{
    get { return showArrows; }
    set { showArrows = value; }
}

    [Description("Show Stop Dots")]
[Category("Options")]
    [Gui.Design.DisplayName ("Show Stop Dots")]
public bool ShowStopDots
{
    get { return showStopDots; }
    set { showStopDots = value; }
}
```

```
            [Description("Show Stop Line")]
        [Category("Options")]
            [Gui.Design.DisplayName ("Show Stop Line")]
        public bool ShowStopLine
        {
            get { return showStopLine; }
            set { showStopLine = value; }
        }

            /// <summary>
            /// </summary>
        [XmlIgnore()]
            [Description("Select color for Rising Trend")]
            [Category("Plot Colors")]
            [Gui.Design.DisplayName("Uptrend")]
            public Color UpColor
            {
                  get { return upColor; }
                  set { upColor = value; }
            }

            // Serialize Color object
            [Browsable(false)]
            public string UpColorSerialize
            {
                  get { return
NinjaTrader.Gui.Design.SerializableColor.ToString(upColor); }
                  set { upColor =
NinjaTrader.Gui.Design.SerializableColor.FromString(value); }
            }

            /// <summary>
            /// </summary>
        [XmlIgnore()]
            [Description("Select color for downtrend")]
            [Category("Plot Colors")]
            [Gui.Design.DisplayName("Downtrend")]
            public Color DownColor
            {
                  get { return downColor; }
                  set { downColor = value; }
            }

            // Serialize Color object
            [Browsable(false)]
            public string DownColorSerialize
            {
                  get { return
NinjaTrader.Gui.Design.SerializableColor.ToString(downColor); }
                  set { downColor =
NinjaTrader.Gui.Design.SerializableColor.FromString(value); }
            }

            /// <summary>
            /// </summary>
            [Description("Width for Stop Dots.")]
            [Category("Plots")]
            [Gui.Design.DisplayNameAttribute("Width Stop Dots")]
```

```
public int Plot0Width
{
      get { return plot0Width; }
      set { plot0Width = Math.Max(1, value); }
}

/// <summary>
/// </summary>
[Description("PlotStyle for Stop Dots.")]
[Category("Plots")]
[Gui.Design.DisplayNameAttribute("Plot Style Stop Dots")]
public PlotStyle Plot0Style
{
      get { return plot0Style; }
      set { plot0Style = value; }
}

/// <summary>
/// </summary>
[Description("DashStyle for Stop Dots.")]
[Category("Plots")]
[Gui.Design.DisplayNameAttribute("Dash Style Stop Dots")]
public DashStyle Dash0Style
{
      get { return dash0Style; }
      set { dash0Style = value; }
}

/// <summary>
/// </summary>
[Description("Width for Stop Line.")]
[Category("Plots")]
[Gui.Design.DisplayNameAttribute("Width Stop Line")]
public int Plot1Width
{
      get { return plot1Width; }
      set { plot1Width = Math.Max(1, value); }
}

/// <summary>
/// </summary>
[Description("PlotStyle for Stop Line.")]
[Category("Plots")]
[Gui.Design.DisplayNameAttribute("Plot Style Stop Line")]
public PlotStyle Plot1Style
{
      get { return plot1Style; }
      set { plot1Style = value; }
}

/// <summary>
/// </summary>
[Description("DashStyle for Stop Line.")]
[Category("Plots")]
[Gui.Design.DisplayNameAttribute("Dash Style Stop Line")]
public DashStyle Dash1Style
{
      get { return dash1Style; }
```

```
                set { dash1Style = value; }
            }
            #endregion
    }
}

#region NinjaScript generated code. Neither change nor remove.
// This namespace holds all indicators and is required. Do not change it.
namespace NinjaTrader.Indicator
{
    public partial class Indicator : IndicatorBase
    {
        private anaSuperTrendM1[] cacheanaSuperTrendM1 = null;

        private static anaSuperTrendM1 checkanaSuperTrendM1 = new
anaSuperTrendM1();

        /// <summary>
        /// Modified SuperTrend Indicator based on a Moving Median
        /// </summary>
        /// <returns></returns>
        public anaSuperTrendM1 anaSuperTrendM1(double multiplier, int periodATR,
int periodMedian)
        {
            return anaSuperTrendM1(Input, multiplier, periodATR, periodMedian);
        }

        /// <summary>
        /// Modified SuperTrend Indicator based on a Moving Median
        /// </summary>
        /// <returns></returns>
        public anaSuperTrendM1 anaSuperTrendM1(Data.IDataSeries input, double
multiplier, int periodATR, int periodMedian)
        {
            if (cacheanaSuperTrendM1 != null)
                for (int idx = 0; idx < cacheanaSuperTrendM1.Length; idx++)
                    if (Math.Abs(cacheanaSuperTrendM1[idx].Multiplier - multiplier)
<= double.Epsilon && cacheanaSuperTrendM1[idx].PeriodATR == periodATR &&
cacheanaSuperTrendM1[idx].PeriodMedian == periodMedian &&
cacheanaSuperTrendM1[idx].EqualsInput(input))
                        return cacheanaSuperTrendM1[idx];

            lock (checkanaSuperTrendM1)
            {
                checkanaSuperTrendM1.Multiplier = multiplier;
                multiplier = checkanaSuperTrendM1.Multiplier;
                checkanaSuperTrendM1.PeriodATR = periodATR;
                periodATR = checkanaSuperTrendM1.PeriodATR;
                checkanaSuperTrendM1.PeriodMedian = periodMedian;
                periodMedian = checkanaSuperTrendM1.PeriodMedian;

                if (cacheanaSuperTrendM1 != null)
                    for (int idx = 0; idx < cacheanaSuperTrendM1.Length; idx++)
                        if (Math.Abs(cacheanaSuperTrendM1[idx].Multiplier -
multiplier) <= double.Epsilon && cacheanaSuperTrendM1[idx].PeriodATR == periodATR
&& cacheanaSuperTrendM1[idx].PeriodMedian == periodMedian &&
cacheanaSuperTrendM1[idx].EqualsInput(input))
                            return cacheanaSuperTrendM1[idx];
```

```
                anaSuperTrendM1 indicator = new anaSuperTrendM1();
                indicator.BarsRequired = BarsRequired;
                indicator.CalculateOnBarClose = CalculateOnBarClose;
#if NT7
                indicator.ForceMaximumBarsLookBack256 =
ForceMaximumBarsLookBack256;
                indicator.MaximumBarsLookBack = MaximumBarsLookBack;
#endif
                indicator.Input = input;
                indicator.Multiplier = multiplier;
                indicator.PeriodATR = periodATR;
                indicator.PeriodMedian = periodMedian;
                Indicators.Add(indicator);
                indicator.SetUp();

                anaSuperTrendM1[] tmp = new anaSuperTrendM1[cacheanaSuperTrendM1 ==
null ? 1 : cacheanaSuperTrendM1.Length + 1];
                if (cacheanaSuperTrendM1 != null)
                    cacheanaSuperTrendM1.CopyTo(tmp, 0);
                tmp[tmp.Length - 1] = indicator;
                cacheanaSuperTrendM1 = tmp;
                return indicator;
            }
        }
    }
}

// This namespace holds all market analyzer column definitions and is required. Do
not change it.
namespace NinjaTrader.MarketAnalyzer
{
    public partial class Column : ColumnBase
    {
        /// <summary>
        /// Modified SuperTrend Indicator based on a Moving Median
        /// </summary>
        /// <returns></returns>
        [Gui.Design.WizardCondition("Indicator")]
        public Indicator.anaSuperTrendM1 anaSuperTrendM1(double multiplier, int
periodATR, int periodMedian)
        {
            return _indicator.anaSuperTrendM1(Input, multiplier, periodATR,
periodMedian);
        }

        /// <summary>
        /// Modified SuperTrend Indicator based on a Moving Median
        /// </summary>
        /// <returns></returns>
        public Indicator.anaSuperTrendM1 anaSuperTrendM1(Data.IDataSeries input,
double multiplier, int periodATR, int periodMedian)
        {
            return _indicator.anaSuperTrendM1(input, multiplier, periodATR,
periodMedian);
        }
    }
}
```

```
// This namespace holds all strategies and is required. Do not change it.
namespace NinjaTrader.Strategy
{
    public partial class Strategy : StrategyBase
    {
        /// <summary>
        /// Modified SuperTrend Indicator based on a Moving Median
        /// </summary>
        /// <returns></returns>
        [Gui.Design.WizardCondition("Indicator")]
        public Indicator.anaSuperTrendM1 anaSuperTrendM1(double multiplier, int
periodATR, int periodMedian)
        {
            return _indicator.anaSuperTrendM1(Input, multiplier, periodATR,
periodMedian);
        }

        /// <summary>
        /// Modified SuperTrend Indicator based on a Moving Median
        /// </summary>
        /// <returns></returns>
        public Indicator.anaSuperTrendM1 anaSuperTrendM1(Data.IDataSeries input,
double multiplier, int periodATR, int periodMedian)
        {
            if (InInitialize && input == null)
                throw new ArgumentException("You only can access an indicator with
the default input/bar series from within the 'Initialize()' method");

            return _indicator.anaSuperTrendM1(input, multiplier, periodATR,
periodMedian);
        }
    }
}
#endregion
```