|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) > [Reference Samples](https://ninjatrader.com/es/support/helpGuides/nt8/reference_samples.htm) > [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator2.htm) >  **Multi-Colored Plots** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/manipulating_string_objects.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/indicator2.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/removing_and_custom_formatting.htm) |

With multi-colored plots it becomes easy to pick out changes in value of your indicator from a quick glance.

**Key concepts in this example**

•Adding plots for each color used

•Plotting a SMA line with three different colors depending on the rising, falling, or neutral nature of the line

**Important related documentation**

•[AddPlot()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm)

•[IsFalling()](https://ninjatrader.com/es/support/helpGuides/nt8/falling.htm)

•[IsRising()](https://ninjatrader.com/es/support/helpGuides/nt8/rising.htm)

•[PlotBrushes](https://ninjatrader.com/es/support/helpGuides/nt8/plotbrushes.htm)

**Import instructions**

1.Download the file contained in this Help Guide topic to your PC desktop

2.From the Control Center window, select the menu Tools > Import > NinjaScript

3.Select the downloaded file

[SampleMultiColoredPlot\_NT8.zip](https://ninjatrader.com/support/helpGuides/nt8/samples/SampleMultiColoredPlot_NT8.zip)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) >  **AddPlot()** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/lines.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/areplotsconfigurable.htm) |

**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>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) object is created held in the [Values](https://ninjatrader.com/es/support/helpGuides/nt8/value.htm) collection.

|  |
| --- |
| **Note:**  Plots are **ONLY** visible from the UI property grid when AddPlot() is called from **State.SetDefaults**. If your indicator or strategy dynamically adds plots during **State.Configure**, you will **NOT** have an opportunity to select the plot or to set the plot configuration via the UI.  Alternatively, you may use custom public [Brush](https://ninjatrader.com/es/support/helpGuides/nt8/brushes.htm), [Stroke](https://ninjatrader.com/es/support/helpGuides/nt8/stroke_class.htm), or **PlotStyle** properties which are accessible in **State.SetDefaults** and pass those values to AddPlot() during**State.Configure**.  Calling AddPlot() in this manner should be reserved for special cases.  Please see the examples below. |

**Methods and Properties**

|  |  |
| --- | --- |
| [ArePlotsConfigurable](https://ninjatrader.com/es/support/helpGuides/nt8/areplotsconfigurable.htm) | Determines if the plot(s) used in an indicator are configurable within the indicator dialog window. |
| [Displacement](https://ninjatrader.com/es/support/helpGuides/nt8/displacement.htm) | An offset value that shifts the visually displayed value of an indicator. |
| [PlotBrushes](https://ninjatrader.com/es/support/helpGuides/nt8/plotbrushes.htm) | Holds an array of color series objects holding historical bar colors. |
| [Plots](https://ninjatrader.com/es/support/helpGuides/nt8/plots.htm) | 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()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method during **State.SetDefaults** or **State.Configure** |

**Parameters**

|  |  |
| --- | --- |
| brush | A Brush object used to construct the plot |
| name | A string representing the name of the plot |
| plotStyle | A PlotStyle object used to construct the style of the plot    Possible values:   PlotStyle.Bar PlotStyle.Block PlotStyle.Cross PlotStyle.Dot PlotStyle.Hash PlotStyle.HLine PlotStyle.Line  PlotStyle.PriceBox PlotStyle.Square PlotStyle.TriangleDown PlotStyle.TriangleLeft PlotStyle.TriangleRight PlotStyle.TriangleUp |
| stroke | A Stroke object used to construct the plot |

|  |
| --- |
| **Tips:**  1.We suggest using the NinjaScript wizard to generate your plots.  2.[Plot](https://ninjatrader.com/es/support/helpGuides/nt8/plots.htm) 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](https://ninjatrader.com/es/support/helpGuides/nt8/value.htm) collection.  4.If you script calls AddPlot() multiple times, then multiple Values series are added per the "three value series" example below  5.For [MultiSeries scripts](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm), plots are synched to the primary series of the NinjaScript object. |

**Examples**

| ns | **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");   } } |

| ns | **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" } |

| ns | **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; } |

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) > [AddPlot()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) >  **ArePlotsConfigurable** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/displacement.htm) |

**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**

| ns | |
| --- | --- |
| protected override void OnStateChange() {     if (State == State.SetDefaults)     {         AddPlot(Brushes.Orange, "SMA");         ArePlotsConfigurable = false; // Plots are not configurable in the indicator dialog     } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) > [AddPlot()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) >  **Displacement** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/areplotsconfigurable.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/plotbrushes.htm) |

**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**

| ns | |
| --- | --- |
| 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");     } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) > [AddPlot()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) >  **PlotBrushes** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/displacement.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/plots.htm) |

**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()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) 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**

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if(State == State.SetDefaults)   {     Name = "Example Indicator";           // Add two plots       AddPlot(Brushes.Blue, "Upper");       AddPlot(Brushes.Orange, "Lower");     } }   protected override void OnBarUpdate() {     // Sets values to our two plots     Upper[0] = SMA(High, 20)[0];     Lower[0] = SMA(Low, 20)[0];       // Color the Upper plot based on plot value conditions     if (IsRising(Upper))         PlotBrushes[0][0] = Brushes.Blue;     else if (IsFalling(Upper))         PlotBrushes[0][0] = Brushes.Red;     else         PlotBrushes[0][0] = Brushes.Yellow;       // Color the Lower plot based on plot value conditions     if (IsRising(Lower))         PlotBrushes[1][0] = Brushes.Blue;     else if (IsFalling(Lower))         PlotBrushes[1][0] = Brushes.Red;     else         PlotBrushes[1][0] = Brushes.Yellow; }   public Series<double> Upper {   get { return Values[0]; } }   public Series<double> Lower {   get { return Values[1]; } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) > [AddPlot()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) >  **Plots** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/plotbrushes.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/barsrequiredtoplot.htm) |

**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](https://ninjatrader.com/es/support/helpGuides/nt8/plotbrushes.htm) for information on changing only specific segments of a plot instead. |

**Example**

| ns | |
| --- | --- |
| 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;   } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [Analytical](https://ninjatrader.com/es/support/helpGuides/nt8/market_data.htm) >  **IsFalling()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/highestbar.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/market_data.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/rising.htm) |

**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**

| ns | |
| --- | --- |
| protected override void OnBarUpdate() {   // If the 20 period SMA is falling (in downtrend) go short   if (IsFalling(SMA(20)))       EnterShort();               } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [Analytical](https://ninjatrader.com/es/support/helpGuides/nt8/market_data.htm) >  **IsRising()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/falling.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/market_data.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/least_recent_occurence_lro.htm) |

**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**

| ns |
| --- |
| protected override void OnBarUpdate() {   // If the 20 period SMA is rising (in uptrend) go long   if (IsRising(SMA(20)))       EnterLong(); } |

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) > [AddPlot()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) >  **PlotBrushes** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/displacement.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/plots.htm) |

**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()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) 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**

| ns |
| --- |
| protected override void OnStateChange() {   if(State == State.SetDefaults)   {     Name = "Example Indicator";           // Add two plots       AddPlot(Brushes.Blue, "Upper");       AddPlot(Brushes.Orange, "Lower");     } }   protected override void OnBarUpdate() {     // Sets values to our two plots     Upper[0] = SMA(High, 20)[0];     Lower[0] = SMA(Low, 20)[0];       // Color the Upper plot based on plot value conditions     if (IsRising(Upper))         PlotBrushes[0][0] = Brushes.Blue;     else if (IsFalling(Upper))         PlotBrushes[0][0] = Brushes.Red;     else         PlotBrushes[0][0] = Brushes.Yellow;       // Color the Lower plot based on plot value conditions     if (IsRising(Lower))         PlotBrushes[1][0] = Brushes.Blue;     else if (IsFalling(Lower))         PlotBrushes[1][0] = Brushes.Red;     else         PlotBrushes[1][0] = Brushes.Yellow; }   public Series<double> Upper {   get { return Values[0]; } }   public Series<double> Lower {   get { return Values[1]; } } |

//

// Copyright (C) 2015, NinjaTrader LLC <www.ninjatrader.com>.

// NinjaTrader reserves the right to modify or overwrite this NinjaScript component with each release.

//

#region Using declarations

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Xml.Serialization;

using NinjaTrader.Cbi;

using NinjaTrader.Gui;

using NinjaTrader.Gui.Chart;

using NinjaTrader.Gui.SuperDom;

using NinjaTrader.Data;

using NinjaTrader.NinjaScript;

using NinjaTrader.Core.FloatingPoint;

using NinjaTrader.NinjaScript.DrawingTools;

#endregion

// This namespace holds indicators in this folder and is required. Do not change it.

namespace NinjaTrader.NinjaScript.Indicators

{

/// <summary>

/// The SMA (Simple Moving Average) is an indicator that shows the average value of a security's price over a period of time.

/// </summary>

public class SMA : Indicator

{

private double priorSum;

private double sum;

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = NinjaTrader.Custom.Resource.NinjaScriptIndicatorDescriptionSMA;

Name = NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameSMA;

IsOverlay = true;

IsSuspendedWhileInactive = true;

Period = 14;

AddPlot(Brushes.Orange, "SMA");

}

else if (State == State.Configure)

{

priorSum = 0;

sum = 0;

}

}

protected override void OnBarUpdate()

{

if (IsFirstTickOfBar)

priorSum = sum;

sum = priorSum + Input[0] - (CurrentBar >= Period ? Input[Period] : 0);

Value[0] = sum / (CurrentBar < Period ? CurrentBar + 1 : Period);

}

#region Properties

[Range(1, int.MaxValue), NinjaScriptProperty]

[Display(ResourceType = typeof(Custom.Resource), Name = "Period", GroupName = "NinjaScriptParameters", Order = 0)]

public int Period

{ get; set; }

#endregion

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

public partial class Indicator : NinjaTrader.Gui.NinjaScript.IndicatorRenderBase

{

private SMA[] cacheSMA;

public SMA SMA(int period)

{

return SMA(Input, period);

}

public SMA SMA(ISeries<double> input, int period)

{

if (cacheSMA != null)

for (int idx = 0; idx < cacheSMA.Length; idx++)

if (cacheSMA[idx] != null && cacheSMA[idx].Period == period && cacheSMA[idx].EqualsInput(input))

return cacheSMA[idx];

return CacheIndicator<SMA>(new SMA(){ Period = period }, input, ref cacheSMA);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.SMA SMA(int period)

{

return indicator.SMA(Input, period);

}

public Indicators.SMA SMA(ISeries<double> input , int period)

{

return indicator.SMA(input, period);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

public partial class Strategy : NinjaTrader.Gui.NinjaScript.StrategyRenderBase

{

public Indicators.SMA SMA(int period)

{

return indicator.SMA(Input, period);

}

public Indicators.SMA SMA(ISeries<double> input , int period)

{

return indicator.SMA(input, period);

}

}

}

#endregion//

// Copyright (C) 2015, NinjaTrader LLC <www.ninjatrader.com>

// NinjaTrader reserves the right to modify or overwrite this NinjaScript component with each release.

//

#region Using declarations

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Xml.Serialization;

using NinjaTrader.Cbi;

using NinjaTrader.Gui;

using NinjaTrader.Gui.Chart;

using NinjaTrader.Gui.SuperDom;

using NinjaTrader.Data;

using NinjaTrader.NinjaScript;

using NinjaTrader.Core.FloatingPoint;

using NinjaTrader.NinjaScript.DrawingTools;

#endregion

// This namespace holds all indicators and is required. Do not change it.

namespace NinjaTrader.NinjaScript.Indicators

{

public class SampleMultiColoredPlot : Indicator

{

private int period; // Default setting for the SMA's period

private Series<double> middle; // This DataSeries stores the average of a upper and lower SMA

protected override void OnStateChange()

{

if(State == State.SetDefaults)

{

AddPlot(Brushes.SkyBlue, "Upper");

AddPlot(Brushes.SkyBlue, "Lower");

// Increase the default widths of the plots

Plots[0].Width = 2;

Plots[1].Width = 2;

Calculate = Calculate.OnBarClose;

IsOverlay = true;

Name = "Sample multi-colored Plot";

Period = 14;

}

else if(State == State.Configure)

{

// Initialize the DataSeries.

middle = new Series<double>(this);

}

}

protected override void OnBarUpdate()

{

// Check that we have enough bars on our chart before processing

if(CurrentBar < Period)

return;

// Set the plots and the DataSeries.

Values[0][0] = SMA(High, Period)[0];

Values[1][0] = SMA(Low, Period)[0];

middle[0] = (SMA(Low, Period)[0] + SMA(High, Period)[0]) / 2;

// If the average of the two plots is rising, change the plot colors.

if (IsRising(middle))

{

// The indexers for PlotBrushes are PlotBrushes[plot index][bars back], so the below code would set

// the first plot (Upper) to black and the second plot (Lower) to green.

PlotBrushes[0][0] = Brushes.Black;

PlotBrushes[1][0] = Brushes.LimeGreen;

}

// If the average is falling, change the plot colors.

else if (IsFalling(middle))

{

PlotBrushes[0][0] = Brushes.Red;

PlotBrushes[1][0] = Brushes.Black;

}

// If the average remains the same, set both plots to the same color.

else

{

PlotBrushes[0][0] = Brushes.Blue;

PlotBrushes[1][0] = Brushes.Blue;

}

}

#region Properties

[Range(1, int.MaxValue), NinjaScriptProperty]

[Display(ResourceType = typeof(Custom.Resource), Name = "Period", GroupName = "NinjaScriptParameters", Order = 0)]

public int Period

{ get; set; }

#endregion

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

public partial class Indicator : NinjaTrader.Gui.NinjaScript.IndicatorRenderBase

{

private SampleMultiColoredPlot[] cacheSampleMultiColoredPlot;

public SampleMultiColoredPlot SampleMultiColoredPlot(int period)

{

return SampleMultiColoredPlot(Input, period);

}

public SampleMultiColoredPlot SampleMultiColoredPlot(ISeries<double> input, int period)

{

if (cacheSampleMultiColoredPlot != null)

for (int idx = 0; idx < cacheSampleMultiColoredPlot.Length; idx++)

if (cacheSampleMultiColoredPlot[idx] != null && cacheSampleMultiColoredPlot[idx].Period == period && cacheSampleMultiColoredPlot[idx].EqualsInput(input))

return cacheSampleMultiColoredPlot[idx];

return CacheIndicator<SampleMultiColoredPlot>(new SampleMultiColoredPlot(){ Period = period }, input, ref cacheSampleMultiColoredPlot);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.SampleMultiColoredPlot SampleMultiColoredPlot(int period)

{

return indicator.SampleMultiColoredPlot(Input, period);

}

public Indicators.SampleMultiColoredPlot SampleMultiColoredPlot(ISeries<double> input , int period)

{

return indicator.SampleMultiColoredPlot(input, period);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

public partial class Strategy : NinjaTrader.Gui.NinjaScript.StrategyRenderBase

{

public Indicators.SampleMultiColoredPlot SampleMultiColoredPlot(int period)

{

return indicator.SampleMultiColoredPlot(Input, period);

}

public Indicators.SampleMultiColoredPlot SampleMultiColoredPlot(ISeries<double> input , int period)

{

return indicator.SampleMultiColoredPlot(input, period);

}

}

}

#endregion

|  |  |  |
| --- | --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) > [Reference Samples](https://ninjatrader.com/es/support/helpGuides/nt8/reference_samples.htm) >  **Indicator** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/reference_samples.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/reference_samples.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/calculating_the_highest_high_o.htm) |
| **Indicator Overview**  ›[Calculating the highest high or lowest low for a specified time range](https://ninjatrader.com/es/support/helpGuides/nt8/calculating_the_highest_high_o.htm)  ›[Changing fonts for draw objects](https://ninjatrader.com/es/support/helpGuides/nt8/changing_fonts_for_draw_object.htm)  ›[Coloring a region](https://ninjatrader.com/es/support/helpGuides/nt8/coloring_a_region.htm)  ›[Creating a user-defined parameter type (enum)](https://ninjatrader.com/es/support/helpGuides/nt8/creating_a_user-defined_parame.htm)  ›[Creating your own Level II data book (Accessing market depth)](https://ninjatrader.com/es/support/helpGuides/nt8/creating_your_own_level_ii_dat.htm)  ›[Draw Objects](https://ninjatrader.com/es/support/helpGuides/nt8/draw_objects.htm)  ›[Ensuring indicator plots are valid before programmatically accessing them](https://ninjatrader.com/es/support/helpGuides/nt8/ensuring_indicator_plots_are_v.htm)  ›[Exposing indicator values that are not plots](https://ninjatrader.com/es/support/helpGuides/nt8/exposing_indicator_values_that.htm)  ›[Getting indicator values from a specified time](https://ninjatrader.com/es/support/helpGuides/nt8/getting_indicator_values_from_.htm)  ›[Manipulating DateTime objects](https://ninjatrader.com/es/support/helpGuides/nt8/manipulating_datetime_objects.htm)  ›[Manipulating string objects](https://ninjatrader.com/es/support/helpGuides/nt8/manipulating_string_objects.htm)  ›[Multi-Colored Plots](https://ninjatrader.com/es/support/helpGuides/nt8/multi-colored_plots.htm)  ›[Removing and Custom Formatting an Indicator’s Chart Label](https://ninjatrader.com/es/support/helpGuides/nt8/removing_and_custom_formatting.htm)  ›[Using a secondary series as an input series for an indicator](https://ninjatrader.com/es/support/helpGuides/nt8/using_a_secondary_series_as_an.htm)  ›[Using a Series or DataSeries object to store calculations](https://ninjatrader.com/es/support/helpGuides/nt8/using_a_series_or_dataseries_o.htm)  ›[Using a TypeConverter to Customize Property Grid Behavior](https://ninjatrader.com/es/support/helpGuides/nt8/using_a_typeconverter_to_custo.htm)  ›[Using custom events to output the current Level II data book](https://ninjatrader.com/es/support/helpGuides/nt8/using_custom_events_to_output_.htm)  ›[Using StreamReader to read from a text file](https://ninjatrader.com/es/support/helpGuides/nt8/using_streamreader_to_read_fro.htm)  ›[Using StreamWriter to write to a text file](https://ninjatrader.com/es/support/helpGuides/nt8/using_streamwriter_to_write_to.htm)  ›[Using System.IO File properties to write to and read from a text file](https://ninjatrader.com/es/support/helpGuides/nt8/using_system_io_file_propertie.htm)  ›[Using Try-Catch Blocks](https://ninjatrader.com/es/support/helpGuides/nt8/using_try-catch_blocks.htm) | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [Bars](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) >  **GetBar()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/getask.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/getbid.htm) |

**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**

| ns | |
| --- | --- |
| // 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()); } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [System Indicator Methods](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm) >  **Maximum (MAX)** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/maenvelopes.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/mcclellan_oscillator.htm) |

**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 ([?](https://ninjatrader.com/es/support/helpGuides/nt8/valid_input_data_for_indicator.htm)) |
| period | Number of bars used in the calculation |

**Example**

| ns |
| --- |
| // Prints the highest high value over the last 20 periods double value = MAX(High, 20)[0]; Print("The current MAX value is " + value.ToString());    // Note the above call with a barsAgo of 0 includes the current MAX of the input high series in the value. If we want to check for example for a break of this value, storing the last bar's MAX would be needed. double value = MAX(High, 20)[1];         if (High[0] > value)   Draw.ArrowUp(this, CurrentBar.ToString(), true, 0, Low[0] - TickSize, Brushes.Blue); |

**Source Code**

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [System Indicator Methods](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm) >  **Minimum (MIN)** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/mcclellan_oscillator.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/momentum.htm) |

**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 ([?](https://ninjatrader.com/es/support/helpGuides/nt8/valid_input_data_for_indicator.htm)) |
| period | Number of bars used in the calculation |

**Example**

| ns |
| --- |
| // Prints the lowest low value over the last 20 periods double value = MIN(Low, 20)[0]; Print("The current MIN value is " + value.ToString());    // Note the above call with a barsAgo of 0 includes the current MIN of the input low series in the value. If we want to check for example for a break of this value, storing the last bar's MIN would be needed. double value = MIN(Low, 20)[1];         if (Low[0] < value)   Draw.ArrowDown(this, CurrentBar.ToString(), true, 0, High[0] + TickSize, Brushes.Red); |

**Source Code**

You can view this indicator method source code by selecting the menu **New > NinjaScript Editor > Indicators** within the NinjaTrader Control Center window.

//

// Copyright (C) 2015, NinjaTrader LLC <www.ninjatrader.com>.

// NinjaTrader reserves the right to modify or overwrite this NinjaScript component with each release.

//

#region Using declarations

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Xml.Serialization;

using NinjaTrader.Cbi;

using NinjaTrader.Gui;

using NinjaTrader.Gui.Chart;

using NinjaTrader.Gui.SuperDom;

using NinjaTrader.Data;

using NinjaTrader.NinjaScript;

using NinjaTrader.Core.FloatingPoint;

using NinjaTrader.NinjaScript.DrawingTools;

#endregion

// This namespace holds indicators in this folder and is required. Do not change it.

namespace NinjaTrader.NinjaScript.Indicators

{

/// <summary>

/// The Maximum shows the maximum of the last n bars.

/// </summary>

public class MAX : Indicator

{

private int lastBar;

private double lastMax;

private double runningMax;

private int runningBar;

private int thisBar;

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = NinjaTrader.Custom.Resource.NinjaScriptIndicatorDescriptionMAX;

Name = NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameMAX;

IsOverlay = true;

IsSuspendedWhileInactive = true;

Period = 14;

AddPlot(Brushes.Green, "MAX");

}

else if (State == State.Configure)

{

lastBar = 0;

lastMax = 0;

runningMax = 0;

runningBar = 0;

thisBar = 0;

}

}

protected override void OnBarUpdate()

{

if (CurrentBar == 0)

{

runningMax = Input[0];

lastMax = Input[0];

runningBar = 0;

lastBar = 0;

thisBar = 0;

Value[0] = Input[0];

return;

}

if (CurrentBar - runningBar >= Period)

{

runningMax = double.MinValue;

for (int barsBack = Math.Min(CurrentBar, Period - 1); barsBack > 0; barsBack--)

if (Input[barsBack] >= runningMax)

{

runningMax = Input[barsBack];

runningBar = CurrentBar - barsBack;

}

}

if (thisBar != CurrentBar)

{

lastMax = runningMax;

lastBar = runningBar;

thisBar = CurrentBar;

}

if (Input[0] >= lastMax)

{

runningMax = Input[0];

runningBar = CurrentBar;

}

else

{

runningMax = lastMax;

runningBar = lastBar;

}

Value[0] = runningMax;

}

#region Properties

[Range(1, int.MaxValue), NinjaScriptProperty]

[Display(ResourceType = typeof(Custom.Resource), Name = "Period", GroupName = "NinjaScriptParameters", Order = 0)]

public int Period

{ get; set; }

#endregion

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

public partial class Indicator : NinjaTrader.Gui.NinjaScript.IndicatorRenderBase

{

private MAX[] cacheMAX;

public MAX MAX(int period)

{

return MAX(Input, period);

}

public MAX MAX(ISeries<double> input, int period)

{

if (cacheMAX != null)

for (int idx = 0; idx < cacheMAX.Length; idx++)

if (cacheMAX[idx] != null && cacheMAX[idx].Period == period && cacheMAX[idx].EqualsInput(input))

return cacheMAX[idx];

return CacheIndicator<MAX>(new MAX(){ Period = period }, input, ref cacheMAX);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.MAX MAX(int period)

{

return indicator.MAX(Input, period);

}

public Indicators.MAX MAX(ISeries<double> input , int period)

{

return indicator.MAX(input, period);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

public partial class Strategy : NinjaTrader.Gui.NinjaScript.StrategyRenderBase

{

public Indicators.MAX MAX(int period)

{

return indicator.MAX(Input, period);

}

public Indicators.MAX MAX(ISeries<double> input , int period)

{

return indicator.MAX(input, period);

}

}

}

#endregion//

// Copyright (C) 2015, NinjaTrader LLC <www.ninjatrader.com>.

// NinjaTrader reserves the right to modify or overwrite this NinjaScript component with each release.

//

#region Using declarations

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Xml.Serialization;

using NinjaTrader.Cbi;

using NinjaTrader.Gui;

using NinjaTrader.Gui.Chart;

using NinjaTrader.Gui.SuperDom;

using NinjaTrader.Data;

using NinjaTrader.NinjaScript;

using NinjaTrader.Core.FloatingPoint;

using NinjaTrader.NinjaScript.DrawingTools;

#endregion

// This namespace holds indicators in this folder and is required. Do not change it.

namespace NinjaTrader.NinjaScript.Indicators

{

/// <summary>

/// The Minimum shows the minimum of the last n bars.

/// </summary>

public class MIN : Indicator

{

private int lastBar;

private double lastMin;

private double runningMin;

private int runningBar;

private int thisBar;

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = NinjaTrader.Custom.Resource.NinjaScriptIndicatorDescriptionMIN;

Name = NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameMIN;

IsOverlay = true;

IsSuspendedWhileInactive = true;

Period = 14;

AddPlot(Brushes.Green, "MIN");

}

else if (State == State.Configure)

{

lastBar = 0;

lastMin = 0;

runningMin = 0;

runningBar = 0;

thisBar = 0;

}

}

protected override void OnBarUpdate()

{

if (CurrentBar == 0)

{

runningMin = Input[0];

lastMin = Input[0];

runningBar = 0;

lastBar = 0;

thisBar = 0;

Value[0] = Input[0];

return;

}

if (CurrentBar - runningBar >= Period)

{

runningMin = double.MaxValue;

for (int barsBack = Math.Min(CurrentBar, Period - 1); barsBack > 0; barsBack--)

if (Input[barsBack] <= runningMin)

{

runningMin = Input[barsBack];

runningBar = CurrentBar - barsBack;

}

}

if (thisBar != CurrentBar)

{

lastMin = runningMin;

lastBar = runningBar;

thisBar = CurrentBar;

}

if (Input[0] <= lastMin)

{

runningMin = Input[0];

runningBar = CurrentBar;

}

else

{

runningMin = lastMin;

runningBar = lastBar;

}

Value[0] = runningMin;

}

#region Properties

[Range(1, int.MaxValue), NinjaScriptProperty]

[Display(ResourceType = typeof(Custom.Resource), Name = "Period", GroupName = "NinjaScriptParameters", Order = 0)]

public int Period

{ get; set; }

#endregion

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

public partial class Indicator : NinjaTrader.Gui.NinjaScript.IndicatorRenderBase

{

private MIN[] cacheMIN;

public MIN MIN(int period)

{

return MIN(Input, period);

}

public MIN MIN(ISeries<double> input, int period)

{

if (cacheMIN != null)

for (int idx = 0; idx < cacheMIN.Length; idx++)

if (cacheMIN[idx] != null && cacheMIN[idx].Period == period && cacheMIN[idx].EqualsInput(input))

return cacheMIN[idx];

return CacheIndicator<MIN>(new MIN(){ Period = period }, input, ref cacheMIN);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.MIN MIN(int period)

{

return indicator.MIN(Input, period);

}

public Indicators.MIN MIN(ISeries<double> input , int period)

{

return indicator.MIN(input, period);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

public partial class Strategy : NinjaTrader.Gui.NinjaScript.StrategyRenderBase

{

public Indicators.MIN MIN(int period)

{

return indicator.MIN(Input, period);

}

public Indicators.MIN MIN(ISeries<double> input , int period)

{

return indicator.MIN(input, period);

}

}

}

#endregion//

// Copyright (C) 2015, NinjaTrader LLC <www.ninjatrader.com>

// NinjaTrader reserves the right to modify or overwrite this NinjaScript component with each release.

//

#region Using declarations

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Xml.Serialization;

using NinjaTrader.Cbi;

using NinjaTrader.Gui;

using NinjaTrader.Gui.Chart;

using NinjaTrader.Gui.SuperDom;

using NinjaTrader.Data;

using NinjaTrader.NinjaScript;

using NinjaTrader.Core.FloatingPoint;

using NinjaTrader.NinjaScript.DrawingTools;

#endregion

//This namespace holds Indicators in this folder and is required. Do not change it.

namespace NinjaTrader.NinjaScript.Indicators

{

public class SampleGetHighLowByTimeRange : Indicator

{

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = @"Determines the highest high and lowest low in a specified time range";

Name = "Sample get high low by time range";

Calculate = Calculate.OnBarClose;

IsOverlay = true;

DisplayInDataBox = true;

DrawOnPricePanel = true;

DrawHorizontalGridLines = true;

DrawVerticalGridLines = true;

PaintPriceMarkers = true;

ScaleJustification = NinjaTrader.Gui.Chart.ScaleJustification.Right;

StartHour = 9;

StartMinute = 30;

EndHour = 10;

EndMinute = 15;

AddPlot(Brushes.Green, "HighestHigh");

AddPlot(Brushes.Red, "LowestLow");

}

}

private DateTime startDateTime;

private DateTime endDateTime;

protected override void OnBarUpdate()

{

// Check to make sure the end time is not earlier than the start time

if (EndHour < StartHour)

return;

//Do not calculate the high or low value when the ending time of the desired range is less than the current time of the bar being processed

if (ToTime(EndHour,EndMinute,0) > ToTime(Time[0]))

return;

// If the stored date time date is not the same date as the bar time date, create a new DateTime object

if (startDateTime.Date != Time[0].Date)

{

startDateTime = new DateTime(Time[0].Year, Time[0].Month, Time[0].Day, StartHour, StartMinute, 0);

endDateTime = new DateTime(Time[0].Year, Time[0].Month, Time[0].Day, EndHour, EndMinute, 0);

}

// Calculate the number of bars ago for the start and end bars of the specified time range

int startBarsAgo = Bars.GetBar(startDateTime);

int endBarsAgo = Bars.GetBar(endDateTime);

/\* Now that we have the start and end bars ago values for the specified time range we can calculate the highest high for this range

Note: We add 1 to the period range for MAX and MIN to compensate for the difference between "period" logic and "bars ago" logic.

"Period" logic means exactly how many bars you want to check including the current bar.

"Bars ago" logic means how many bars we are going to go backwards. The current bar is not counted because on that bar we aren't going back any bars so it would be "bars ago = 0" \*/

double highestHigh = MAX(High, endBarsAgo - startBarsAgo + 1)[CurrentBar - endBarsAgo];

// Now that we have the start and end bars ago values for the specified time range we can calculate the lowest low for this range

double lowestLow = MIN(Low, endBarsAgo - startBarsAgo + 1)[CurrentBar - endBarsAgo];

// Set the plot values

HighestHigh[0] = highestHigh;

LowestLow[0] = lowestLow;

}

#region Properties

[Browsable(false)]

[XmlIgnore]

public Series<double> HighestHigh

{

get { return Values[0]; }

}

[Browsable(false)]

[XmlIgnore]

public Series<double> LowestLow

{

get { return Values[1]; }

}

[Range(0,23)]

[NinjaScriptProperty]

[Display(Name="Start hour", Description = "Enter start hour, Military time format 0 - 23", Order=1, GroupName="Parameters")]

public int StartHour

{ get; set; }

[Range(0, 59)]

[NinjaScriptProperty]

[Display(Name="Start minute", Description = "Enter start minute(s) 0 - 59",Order=2, GroupName="Parameters")]

public int StartMinute

{ get; set; }

[Range(0, 23)]

[NinjaScriptProperty]

[Display(Name="End hour", Description = "Enter end hour, Military time format 0 - 23",Order=3, GroupName="Parameters")]

public int EndHour

{ get; set; }

[Range(0, 59)]

[NinjaScriptProperty]

[Display(Name="End minute",Description = " Enter end minute(s) 0 - 59", Order=4, GroupName="Parameters")]

public int EndMinute

{ get; set; }

#endregion

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

public partial class Indicator : NinjaTrader.Gui.NinjaScript.IndicatorRenderBase

{

private SampleGetHighLowByTimeRange[] cacheSampleGetHighLowByTimeRange;

public SampleGetHighLowByTimeRange SampleGetHighLowByTimeRange(int startHour, int startMinute, int endHour, int endMinute)

{

return SampleGetHighLowByTimeRange(Input, startHour, startMinute, endHour, endMinute);

}

public SampleGetHighLowByTimeRange SampleGetHighLowByTimeRange(ISeries<double> input, int startHour, int startMinute, int endHour, int endMinute)

{

if (cacheSampleGetHighLowByTimeRange != null)

for (int idx = 0; idx < cacheSampleGetHighLowByTimeRange.Length; idx++)

if (cacheSampleGetHighLowByTimeRange[idx] != null && cacheSampleGetHighLowByTimeRange[idx].StartHour == startHour && cacheSampleGetHighLowByTimeRange[idx].StartMinute == startMinute && cacheSampleGetHighLowByTimeRange[idx].EndHour == endHour && cacheSampleGetHighLowByTimeRange[idx].EndMinute == endMinute && cacheSampleGetHighLowByTimeRange[idx].EqualsInput(input))

return cacheSampleGetHighLowByTimeRange[idx];

return CacheIndicator<SampleGetHighLowByTimeRange>(new SampleGetHighLowByTimeRange(){ StartHour = startHour, StartMinute = startMinute, EndHour = endHour, EndMinute = endMinute }, input, ref cacheSampleGetHighLowByTimeRange);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.SampleGetHighLowByTimeRange SampleGetHighLowByTimeRange(int startHour, int startMinute, int endHour, int endMinute)

{

return indicator.SampleGetHighLowByTimeRange(Input, startHour, startMinute, endHour, endMinute);

}

public Indicators.SampleGetHighLowByTimeRange SampleGetHighLowByTimeRange(ISeries<double> input , int startHour, int startMinute, int endHour, int endMinute)

{

return indicator.SampleGetHighLowByTimeRange(input, startHour, startMinute, endHour, endMinute);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

public partial class Strategy : NinjaTrader.Gui.NinjaScript.StrategyRenderBase

{

public Indicators.SampleGetHighLowByTimeRange SampleGetHighLowByTimeRange(int startHour, int startMinute, int endHour, int endMinute)

{

return indicator.SampleGetHighLowByTimeRange(Input, startHour, startMinute, endHour, endMinute);

}

public Indicators.SampleGetHighLowByTimeRange SampleGetHighLowByTimeRange(ISeries<double> input , int startHour, int startMinute, int endHour, int endMinute)

{

return indicator.SampleGetHighLowByTimeRange(input, startHour, startMinute, endHour, endMinute);

}

}

}

#endregion

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) > [Reference Samples](https://ninjatrader.com/es/support/helpGuides/nt8/reference_samples.htm) > [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator2.htm) >  **Changing fonts for draw objects** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/calculating_the_highest_high_o.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/indicator2.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/coloring_a_region.htm) |

Drawing text on a chart can be useful for outputting information, but when all information is displayed with the same font and size it could be difficult to quickly see the key information. Since NinjaScript is based on C#, it is possible to use Font objects to create more styles for your text

**Key concepts in this example**

•Drawing text on a chart

•Changing the font size on a chart

**Important related documentation**

•[Text()](https://ninjatrader.com/es/support/helpGuides/nt8/text.htm)

•[SimpleFont()](https://ninjatrader.com/es/support/helpGuides/nt8/simplefont_class.htm)

**Import instructions**

1.Download the file contained in this Help Guide topic to your PC desktop

2.From the Control Center window, select the menu Tools > Import > NinjaScript

3.Select the downloaded file

[SampleChangeFont\_NT8.zip](https://ninjatrader.com/support/helpGuides/nt8/samples/SampleChangeFont_NT8.zip)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [Drawing](https://ninjatrader.com/es/support/helpGuides/nt8/drawing.htm) > [Draw.Text()](https://ninjatrader.com/es/support/helpGuides/nt8/draw_text.htm) >  **Text** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/draw_text.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/draw_text.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/draw_textfixed.htm) |

**Definition**

Represents an interface that exposes information regarding a Text [IDrawingTool](https://ninjatrader.com/es/support/helpGuides/nt8/idrawingtool.htm).

**Methods and Properties**

|  |  |
| --- | --- |
| Anchor | An [IDrawingTool's ChartAnchor](https://ninjatrader.com/es/support/helpGuides/nt8/idrawingtool.htm" \l "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](https://msdn.microsoft.com/en-us/library/system.windows.textalignment(v=vs.110).aspx" \t "_blank)) |
| AreaOpacity | An int value representing the opacity of the area color |
| AreaBrush | A [Brush](http://msdn.microsoft.com/en-us/library/system.windows.media.brush(v=vs.110).aspx) class representing the fill color of the text box |
| Text | A string value representing the text to be drawn |
| TextBrush | A [Brush](http://msdn.microsoft.com/en-us/library/system.windows.media.brush(v=vs.110).aspx) class representing the color of the text |
| Font | A [Font](http://msdn.microsoft.com/en-us/library/system.drawing.font_members(v=vs.90).aspx" \t "_blank) object representing the font for the text |
| OutlineStroke | The [Stroke](https://ninjatrader.com/es/support/helpGuides/nt8/stroke_class.htm) object used to outline the text box |

**Example**

| ns | |
| --- | --- |
| // 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"; | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) >  **SimpleFont** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/istradingdaydefined.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/simplefont_applyto.htm) |

**Definition**

Defines a particular font configuration.

|  |
| --- |
| **Note**: **SimpleFont** objects are used for various [Drawing](https://ninjatrader.com/es/support/helpGuides/nt8/drawing.htm) 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](https://msdn.microsoft.com/en-us/library/system.windows.media.fontfamily(v=vs.110).aspx" \t "_blank) 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](https://msdn.microsoft.com/en-us/library/system.windows.media.typeface%28v=vs.110%29.aspx" \t "_blank) used to represent the variation of the font used |
| [ApplyTo()](https://ninjatrader.com/es/support/helpGuides/nt8/simplefont_applyto.htm) | Applies a custom [SimpleFont](https://ninjatrader.com/es/support/helpGuides/nt8/simplefont_class.htm) object's properties (family, size, and style) to a [Windows Control](https://msdn.microsoft.com/en-us/library/system.windows.controls.control(v=vs.110).aspx" \t "_blank) |
| [ToDirectWriteTextFormat()](https://ninjatrader.com/es/support/helpGuides/nt8/simplefont_todirectwritetextformat.htm) | 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](https://blogs.msdn.microsoft.com/text/2009/12/11/wpf-text-measurement-units/" \t "_blank). |

**Examples**

| ns |
| --- |
| // 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); |

//

// Copyright (C) 2015, NinjaTrader LLC <www.ninjatrader.com>.

// NinjaTrader reserves the right to modify or overwrite this NinjaScript component with each release.

//

#region Using declarations

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

using System.Xml.Serialization;

using NinjaTrader.Cbi;

using NinjaTrader.Gui;

using NinjaTrader.Gui.Chart;

using NinjaTrader.Gui.SuperDom;

using NinjaTrader.Data;

using NinjaTrader.NinjaScript;

using NinjaTrader.Core.FloatingPoint;

using NinjaTrader.NinjaScript.DrawingTools;

#endregion

// This namespace holds indicators in this folder and is required. Do not change it.

namespace NinjaTrader.NinjaScript.Indicators

{

/// <summary>

/// The SMA (Simple Moving Average) is an indicator that shows the average value of a security's price over a period of time.

/// </summary>

public class SMA : Indicator

{

private double priorSum;

private double sum;

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = NinjaTrader.Custom.Resource.NinjaScriptIndicatorDescriptionSMA;

Name = NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameSMA;

IsOverlay = true;

IsSuspendedWhileInactive = true;

Period = 14;

AddPlot(Brushes.Orange, "SMA");

}

else if (State == State.Configure)

{

priorSum = 0;

sum = 0;

}

}

protected override void OnBarUpdate()

{

if (IsFirstTickOfBar)

priorSum = sum;

sum = priorSum + Input[0] - (CurrentBar >= Period ? Input[Period] : 0);

Value[0] = sum / (CurrentBar < Period ? CurrentBar + 1 : Period);

}

#region Properties

[Range(1, int.MaxValue), NinjaScriptProperty]

[Display(ResourceType = typeof(Custom.Resource), Name = "Period", GroupName = "NinjaScriptParameters", Order = 0)]

public int Period

{ get; set; }

#endregion

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

public partial class Indicator : NinjaTrader.Gui.NinjaScript.IndicatorRenderBase

{

private SMA[] cacheSMA;

public SMA SMA(int period)

{

return SMA(Input, period);

}

public SMA SMA(ISeries<double> input, int period)

{

if (cacheSMA != null)

for (int idx = 0; idx < cacheSMA.Length; idx++)

if (cacheSMA[idx] != null && cacheSMA[idx].Period == period && cacheSMA[idx].EqualsInput(input))

return cacheSMA[idx];

return CacheIndicator<SMA>(new SMA(){ Period = period }, input, ref cacheSMA);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.SMA SMA(int period)

{

return indicator.SMA(Input, period);

}

public Indicators.SMA SMA(ISeries<double> input , int period)

{

return indicator.SMA(input, period);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

public partial class Strategy : NinjaTrader.Gui.NinjaScript.StrategyRenderBase

{

public Indicators.SMA SMA(int period)

{

return indicator.SMA(Input, period);

}

public Indicators.SMA SMA(ISeries<double> input , int period)

{

return indicator.SMA(input, period);

}

}

}

#endregion