|  |  |
| --- | --- |
| **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) >  **Using a Series or DataSeries object to store calculations** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_a_secondary_series_as_an.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/indicator2.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_a_typeconverter_to_custo.htm) |

When creating Indicators or Strategies you may find that you need to store values in a way that is similar to the way price data is stored in NinjaTrader.

Series<T> objects are useful for storing various types of values.

Since they are linked to your historical bars object, you can store and link a value to each bar. This allows you the flexibility of accessing the values at any point in the future for further calculations or plotting.

**Key concepts in this example**

•Creating objects that store data

•Storing and retrieving values from these objects

**Important related documentation**

•[Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.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

[SampleCustomSeries\_NT8.zip](https://ninjatrader.com/support/helpGuides/nt8/samples/SampleCustomSeries_NT8.zip)

//

// Copyright (C) 2016, 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.Goldenrod, NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameSMA);

}

else if (State == State.Configure)

{

priorSum = 0;

sum = 0;

}

}

protected override void OnBarUpdate()

{

if (BarsArray[0].BarsType.IsRemoveLastBarSupported)

{

if (CurrentBar == 0)

Value[0] = Input[0];

else

{

double last = Value[1] \* Math.Min(CurrentBar, Period);

if (CurrentBar >= Period)

Value[0] = (last + Input[0] - Input[Period]) / Math.Min(CurrentBar, Period);

else

Value[0] = ((last + Input[0]) / (Math.Min(CurrentBar, Period) + 1));

}

}

else

{

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 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 SampleCustomSeries : Indicator

{

// Defines the Series object

private Series<double> myDoubleSeries;

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

// Adds a plot to our NinjaScript Indicator

AddPlot(new Stroke(Brushes.Orange, 1), PlotStyle.Bar, "Average Range");

Period = 5;

Name = "Sample custom series";

Calculate = Calculate.OnBarClose;

IsOverlay = false;

}

else if (State == State.DataLoaded)

{

// Create a new Series object and assign it to the variable myDoubleSeries declared in the ‘Variables’ region above

myDoubleSeries = new Series<double>(this);

}

}

protected override void OnBarUpdate()

{

/\* To set values to our Series object we use the [] method. Here we are setting the DataSeries

object for the current bar to take on the absolute value of the difference between the current bar's

open and close. \*/

myDoubleSeries[0] = Math.Abs(Close[0] - Open[0]);

/\* Take note that the method for setting the value to be plotted is the same as for setting a value

to a DataSeries object. The difference here is that the custom DataSeries object is not plotted while

this "Average Range" is plotted.

In this case we are plotting the Simple Moving Average of the intermediate calculation step stored in our

Series object. \*/

Value[0] = SMA(myDoubleSeries, Period)[0];

}

#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 SampleCustomSeries[] cacheSampleCustomSeries;

public SampleCustomSeries SampleCustomSeries(int period)

{

return SampleCustomSeries(Input, period);

}

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

{

if (cacheSampleCustomSeries != null)

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

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

return cacheSampleCustomSeries[idx];

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

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.SampleCustomSeries SampleCustomSeries(int period)

{

return indicator.SampleCustomSeries(Input, period);

}

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

{

return indicator.SampleCustomSeries(input, period);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

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

{

public Indicators.SampleCustomSeries SampleCustomSeries(int period)

{

return indicator.SampleCustomSeries(Input, period);

}

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

{

return indicator.SampleCustomSeries(input, period);

}

}

}

#endregion

|  |  |
| --- | --- |
| **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) > [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) > [PriceSeries<double>](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) >  **Values** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/value.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/weighted.htm) |

**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()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) method. In case of a [MultiSeries](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) indicator synched to the primary series.

**Property Value**

A collection of ISeries<double> objects.

**Syntax**

Values[int *index*]

**Examples**

| ns | |
| --- | --- |
| // 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]; } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Performance Metrics](https://ninjatrader.com/es/support/helpGuides/nt8/performance_metrics.htm) >  **Values** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/performanceunit.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/performance_metrics.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/share_service.htm) |

**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**

| ns |
| --- |
| 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**

| ns | |
| --- | --- |
| // 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;  } | |
| **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) >  **Exposing indicator values that are not plots** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/ensuring_indicator_plots_are_v.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/indicator2.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/getting_indicator_values_from_.htm) |

There may be cases where you want to have your indicator calculate non-plotted values that you will want to access when using this indicator inside of another indicator or strategy.

**Key concepts in this example**

•Creating exposed BoolSeries objects

•Storing and retrieving values from BoolSeries objects

**Important related documentation**

•[Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm)

We suggest using an available class that implements the Series interface.

•[Price Series](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm)

•[Time Series](https://ninjatrader.com/es/support/helpGuides/nt8/timeseries.htm)

•[Volume Series](https://ninjatrader.com/es/support/helpGuides/nt8/volumeseries.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

[SampleBoolSeries\_NT8.zip](https://ninjatrader.com/support/helpGuides/nt8/samples/SampleBoolSeries_NT8.zip)

//

// Copyright (C) 2016, 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 MACD (Moving Average Convergence/Divergence) is a trend following momentum indicator

/// that shows the relationship between two moving averages of prices.

/// </summary>

public class MACD : Indicator

{

private Series<double> fastEma;

private Series<double> slowEma;

private double constant1;

private double constant2;

private double constant3;

private double constant4;

private double constant5;

private double constant6;

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = NinjaTrader.Custom.Resource.NinjaScriptIndicatorDescriptionMACD;

Name = NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameMACD;

Fast = 12;

IsSuspendedWhileInactive = true;

Slow = 26;

Smooth = 9;

AddPlot(Brushes.Green, NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameMACD);

AddPlot(Brushes.DarkViolet, NinjaTrader.Custom.Resource.NinjaScriptIndicatorAvg);

AddPlot(new Stroke(Brushes.Navy, 2), PlotStyle.Bar, NinjaTrader.Custom.Resource.NinjaScriptIndicatorDiff);

AddLine(Brushes.DarkGray, 0, NinjaTrader.Custom.Resource.NinjaScriptIndicatorZeroLine);

}

else if (State == State.Configure)

{

constant1 = 2.0 / (1 + Fast);

constant2 = (1 - (2.0 / (1 + Fast)));

constant3 = 2.0 / (1 + Slow);

constant4 = (1 - (2.0 / (1 + Slow)));

constant5 = 2.0 / (1 + Smooth);

constant6 = (1 - (2.0 / (1 + Smooth)));

fastEma = new Series<double>(this);

slowEma = new Series<double>(this);

}

}

protected override void OnBarUpdate()

{

double input0 = Input[0];

if (CurrentBar == 0)

{

fastEma[0] = input0;

slowEma[0] = input0;

Value[0] = 0;

Avg[0] = 0;

Diff[0] = 0;

}

else

{

double fastEma0 = constant1 \* input0 + constant2 \* fastEma[1];

double slowEma0 = constant3 \* input0 + constant4 \* slowEma[1];

double macd = fastEma0 - slowEma0;

double macdAvg = constant5 \* macd + constant6 \* Avg[1];

fastEma[0] = fastEma0;

slowEma[0] = slowEma0;

Value[0] = macd;

Avg[0] = macdAvg;

Diff[0] = macd - macdAvg;

}

}

#region Properties

[Browsable(false)]

[XmlIgnore]

public Series<double> Avg

{

get { return Values[1]; }

}

[Browsable(false)]

[XmlIgnore]

public Series<double> Default

{

get { return Values[0]; }

}

[Browsable(false)]

[XmlIgnore]

public Series<double> Diff

{

get { return Values[2]; }

}

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

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

public int Fast

{ get; set; }

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

[Display(ResourceType = typeof(Custom.Resource), Name = "Slow", GroupName = "NinjaScriptParameters", Order = 1)]

public int Slow

{ get; set; }

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

[Display(ResourceType = typeof(Custom.Resource), Name = "Smooth", GroupName = "NinjaScriptParameters", Order = 2)]

public int Smooth

{ get; set; }

#endregion

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

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

{

private MACD[] cacheMACD;

public MACD MACD(int fast, int slow, int smooth)

{

return MACD(Input, fast, slow, smooth);

}

public MACD MACD(ISeries<double> input, int fast, int slow, int smooth)

{

if (cacheMACD != null)

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

if (cacheMACD[idx] != null && cacheMACD[idx].Fast == fast && cacheMACD[idx].Slow == slow && cacheMACD[idx].Smooth == smooth && cacheMACD[idx].EqualsInput(input))

return cacheMACD[idx];

return CacheIndicator<MACD>(new MACD(){ Fast = fast, Slow = slow, Smooth = smooth }, input, ref cacheMACD);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.MACD MACD(int fast, int slow, int smooth)

{

return indicator.MACD(Input, fast, slow, smooth);

}

public Indicators.MACD MACD(ISeries<double> input , int fast, int slow, int smooth)

{

return indicator.MACD(input, fast, slow, smooth);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

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

{

public Indicators.MACD MACD(int fast, int slow, int smooth)

{

return indicator.MACD(Input, fast, slow, smooth);

}

public Indicators.MACD MACD(ISeries<double> input , int fast, int slow, int smooth)

{

return indicator.MACD(input, fast, slow, smooth);

}

}

}

#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 SampleBoolSeries : Indicator

{

/\* We declare two Series<bool> objects here. We will expose these objects with the use of

public properties later. When we want to expose an object we should always use the associated

ISeries class type with it. This will ensure that the value accessed is kept up-to-date. \*/

private Series<bool> bearIndication;

private Series<bool> bullIndication;

/\* If you happen to have an object that does not have an ISeries class that can be used, you

will need to manually ensure its values are kept up-to-date. This process will be done in the

"Properties" region of the code. \*/

private double exposedVariable;

protected override void OnStateChange()

{

if(State == State.SetDefaults)

{

Name = "Sample bool series";

Calculate = Calculate.OnBarClose;

IsOverlay = true;

}

else if(State == State.Configure)

{

/\* "this" syncs the Series<bool> to the historical bar object of the indicator. It will generate

one bool value for every price bar. \*/

bearIndication = new Series<bool>(this);

bullIndication = new Series<bool>(this);

}

}

protected override void OnBarUpdate()

{

// MACD Crossover: Fast Line cross above Slow Line

if (CrossAbove(MACD(12, 26, 9), MACD(12, 26, 9).Avg, 1))

{

// Paint the current price bar lime to draw our attention to it

BarBrushes[0] = Brushes.Lime;

/\* This crossover condition is considered bullish so we set the "bullIndication" Series<bool> object to true.

We also set the "bearIndication" object to false so it does not take on a null value. \*/

bullIndication[0] = (true);

bearIndication[0] = (false);

}

// MACD Crossover: Fast Line cross below Slow Line

else if (CrossBelow(MACD(12, 26, 9), MACD(12, 26, 9).Avg, 1))

{

// Paint the current price bar magenta to draw our attention to it

BarBrushes[0] = Brushes.Magenta;

/\* This crossover condition is considered bearish so we set the "bearIndication" Series<bool> object to true.

We also set the "bullIndication" object to false so it does not take on a null value. \*/

bullIndication[0] = (false);

bearIndication[0] = (true);

}

// MACD Crossover: No cross

else

{

/\* Since no crosses occured we are not receiving any bullish or bearish signals so we

set our Series<bool> objects both to false. \*/

bullIndication[0] = (false);

bearIndication[0] = (false);

}

// We set our variable to the close value.

exposedVariable = Close[0];

}

// Important code segment in the Properties section. Please expand to view.

#region Properties

// Creating public properties that access our internal Series<bool> allows external access to this indicator's Series<bool>

[Browsable(false)]

[XmlIgnore]

public Series<bool> BearIndication

{

get { return bearIndication; } // Allows our public BearIndication Series<bool> to access and expose our interal bearIndication Series<bool>

}

[Browsable(false)]

[XmlIgnore]

public Series<bool> BullIndication

{

get { return bullIndication; } // Allows our public BullIndication Series<bool> to access and expose our interal bullIndication Series<bool>

}

public double ExposedVariable

{

// We need to call the Update() method to ensure our exposed variable is in up-to-date.

get { Update(); return exposedVariable; }

}

#endregion

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

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

{

private SampleBoolSeries[] cacheSampleBoolSeries;

public SampleBoolSeries SampleBoolSeries()

{

return SampleBoolSeries(Input);

}

public SampleBoolSeries SampleBoolSeries(ISeries<double> input)

{

if (cacheSampleBoolSeries != null)

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

if (cacheSampleBoolSeries[idx] != null && cacheSampleBoolSeries[idx].EqualsInput(input))

return cacheSampleBoolSeries[idx];

return CacheIndicator<SampleBoolSeries>(new SampleBoolSeries(), input, ref cacheSampleBoolSeries);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.SampleBoolSeries SampleBoolSeries()

{

return indicator.SampleBoolSeries(Input);

}

public Indicators.SampleBoolSeries SampleBoolSeries(ISeries<double> input )

{

return indicator.SampleBoolSeries(input);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

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

{

public Indicators.SampleBoolSeries SampleBoolSeries()

{

return indicator.SampleBoolSeries(Input);

}

public Indicators.SampleBoolSeries SampleBoolSeries(ISeries<double> input )

{

return indicator.SampleBoolSeries(input);

}

}

}

#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.Indicators;

using NinjaTrader.NinjaScript.DrawingTools;

#endregion

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

namespace NinjaTrader.NinjaScript.Strategies

{

public class SampleBoolSeriesStrategy : Strategy

{

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = @"Sample strategy demonstrating how to call an exposed BoolSeries object";

Name = "SampleBoolSeriesStrategy";

Calculate = Calculate.OnBarClose;

EntriesPerDirection = 1;

EntryHandling = EntryHandling.AllEntries;

IsExitOnSessionCloseStrategy= true;

ExitOnSessionCloseSeconds = 30;

IsFillLimitOnTouch = false;

MaximumBarsLookBack = MaximumBarsLookBack.TwoHundredFiftySix;

OrderFillResolution = OrderFillResolution.Standard;

Slippage = 0;

StartBehavior = StartBehavior.WaitUntilFlat;

TimeInForce = TimeInForce.Gtc;

TraceOrders = false;

RealtimeErrorHandling = RealtimeErrorHandling.StopCancelClose;

StopTargetHandling = StopTargetHandling.PerEntryExecution;

BarsRequiredToTrade = 20;

}

else if (State == State.Configure)

{

AddChartIndicator(MACD(12,26,9));

AddChartIndicator(SampleBoolSeries());

}

}

protected override void OnBarUpdate()

{

/\* When our indicator gives us a bull signal we enter long. Notice that we are accessing the

public BoolSeries we made in the indicator. \*/

if (SampleBoolSeries().BullIndication[0])

EnterLong();

// When our indicator gives us a bear signal we enter short

if (SampleBoolSeries().BearIndication[0])

EnterShort();

/\* NOTE: This strategy is based on reversals thus there are no explicit exit orders. When you

are long you will be closed and reversed into a short when the bear signal is received. The vice

versa is true if you are short. \*/

/\* Print our exposed variable. Because we manually kept it up-to-date it will print values that

match the bars object. \*/

Print(SampleBoolSeries().ExposedVariable);

}

}

}

|  |  |
| --- | --- |
| **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) > [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) >  **Series<T>** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/reset.htm) |

**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](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm). 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](https://ninjatrader.com/es/support/helpGuides/nt8/maximumbarslookback.htm) property. |

**Parameters**

|  |  |
| --- | --- |
| ninjaScriptBase | The NinjaScript object used to create the Series |
| bars | The [Bars](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) object used to create the Series |
| maximumBarsLookBack | A [MaximumBarsLookBack](https://ninjatrader.com/es/support/helpGuides/nt8/maximumbarslookback.htm) value used for memory performance |

**Methods and Properties**

|  |  |
| --- | --- |
| [GetValueAt()](https://ninjatrader.com/es/support/helpGuides/nt8/getvalueat.htm) | Returns the underlying input value at a specified bar index value. |
| [IsValidDataPoint()](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapoint.htm) | Determines if the specified input is set at a barsAgo value relative to the current bar. |
| [Reset()](https://ninjatrader.com/es/support/helpGuides/nt8/reset.htm) | Resets the internal marker which is used for [IsValidDataPoint()](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapoint.htm) back to false. |
| [Count](https://ninjatrader.com/es/support/helpGuides/nt8/count.htm) | 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()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) method and can be subsequently referenced by the [Value](https://ninjatrader.com/es/support/helpGuides/nt8/value.htm) and/or [Values](https://ninjatrader.com/es/support/helpGuides/nt8/values.htm) 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](https://ninjatrader.com/es/support/helpGuides/nt8/maximumbarslookback.htm) 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()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method, in the State.DataLoaded create a new Series<T> object and assign it to the "myDoubleSeries" variable

| ns |
| --- |
| 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 Series object and assign it to the variable protected override void OnStateChange() {     if (State == State.DataLoaded)     {         // "this" refers to the NinjaScript object itself. This syncs the Series object to historical data bars         // MaximumBarsLookBack determines how many values the Series<double> will have access to         myDoubleSeries = new Series<double>(this, MaximumBarsLookBack.Infinite);     } } |

|  |
| --- |
| **Tip***:*Series<T> objects can be used on supplementary series in a multi-time frame and instrument strategy. Please see our [support forum](http://www.ninjatrader.com/support/forum/showthread.php?t=3572" \t "_blank) 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.

| ns **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()](https://ninjatrader.com/es/support/helpGuides/nt8/triggercustomevent.htm) 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()](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapoint.htm) more information.

|  |
| --- |
| **Warning**:  Calling IsValidDataPoint() will only work a [MaximumBarsLookBackInfinite](https://ninjatrader.com/es/support/helpGuides/nt8/maximumbarslookback.htm) 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).

| ns **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()](https://ninjatrader.com/es/support/helpGuides/nt8/getvalueat.htm) 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()](https://ninjatrader.com/es/support/helpGuides/nt8/getvalueat.htm). |

**Methods that Accept ISeries<T> as Arguments**  
All [indicator methods](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm) accept ISeries<double> objects as arguments. Carrying from the prior examples, let's print out the 10 period simple moving average of range.

| ns **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]);         } | |
| **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) > [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) >  **PriceSeries<double>** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/reset.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/close.htm) |

**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()](https://ninjatrader.com/es/support/helpGuides/nt8/getclose.htm), [Bars.GetOpen()](https://ninjatrader.com/es/support/helpGuides/nt8/getopen.htm), etc. |

**Single ISeries<double>**

|  |  |
| --- | --- |
| [Close](https://ninjatrader.com/es/support/helpGuides/nt8/close.htm) | A collection of historical bar close prices. |
| [High](https://ninjatrader.com/es/support/helpGuides/nt8/high.htm) | A collection of historical bar high prices. |
| [Input](https://ninjatrader.com/es/support/helpGuides/nt8/input.htm) | A collect of the the main historical input values. |
| [Low](https://ninjatrader.com/es/support/helpGuides/nt8/low.htm) | A collection of historical bar low prices. |
| [Median](https://ninjatrader.com/es/support/helpGuides/nt8/median.htm) | A collection of historical bar median prices. |
| [Open](https://ninjatrader.com/es/support/helpGuides/nt8/open.htm) | A collection of historical bar open prices. |
| [Typical](https://ninjatrader.com/es/support/helpGuides/nt8/typical.htm) | A collection of historical bar typical prices. |
| [Value](https://ninjatrader.com/es/support/helpGuides/nt8/value.htm) | A collection of historical references to the first object (Values[0]) in the indicator |
| [Weighted](https://ninjatrader.com/es/support/helpGuides/nt8/weighted.htm) | A collection of historical bar weighted prices. |

**Multi-Time Frame ISeries<double>**

|  |  |
| --- | --- |
| [Closes](https://ninjatrader.com/es/support/helpGuides/nt8/closes.htm) | Holds an array of ISeries<double> objects holding historical bar close prices. |
| [Highs](https://ninjatrader.com/es/support/helpGuides/nt8/highs.htm) | Holds an array of ISeries<double> objects holding historical bar high prices. |
| [Inputs](https://ninjatrader.com/es/support/helpGuides/nt8/inputs.htm) | Holds an array of ISeries<double> objects holding main historical input values |
| [Lows](https://ninjatrader.com/es/support/helpGuides/nt8/lows.htm) | Holds an array of ISeries<double> objects holding historical bar low prices. |
| [Medians](https://ninjatrader.com/es/support/helpGuides/nt8/medians.htm) | Holds an array of ISeries<double>objects holding historical bar median prices. |
| [Opens](https://ninjatrader.com/es/support/helpGuides/nt8/opens.htm) | Holds an array of ISeries<double> objects holding historical bar open prices. |
| [Typicals](https://ninjatrader.com/es/support/helpGuides/nt8/typicals.htm) | Holds an array of ISeries<double> objects holding historical bar typical prices. |
| [Values](https://ninjatrader.com/es/support/helpGuides/nt8/values.htm) | Holds an array of ISeries<double> objects holding hold the indicator's underlying calculated values. |
| [Weighteds](https://ninjatrader.com/es/support/helpGuides/nt8/weighteds.htm) | Holds an array of ISeries<double> objects holding historical bar weighted prices. |

|  |  |
| --- | --- |
| **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) > [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) >  **TimeSeries<DateTime>** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/weighteds.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_time.htm) |

**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()](https://ninjatrader.com/es/support/helpGuides/nt8/gettime.htm), 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 |

|  |  |
| --- | --- |
| **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) > [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) >  **VolumeSeries<double>** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_times.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_volume.htm) |

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

**Single ISeries<double>**

|  |  |
| --- | --- |
| [Volume](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_volume.htm) | A collection of historical bar volume values. |

**Multi-Time Frame ISeries<double>**

|  |  |
| --- | --- |
| [Volumes](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_volumes.htm) | Holds an array of ISeries<**double**> objects holding historical bar times |

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) > [Tips](https://ninjatrader.com/es/support/helpGuides/nt8/tips.htm) >  **Referencing the correct bar** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/parameter_sequencing.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/tips.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/strategy_position_vs__account_.htm) |

When coding an indicator or strategy it is important to be able to access the intended bars for correct calculations. In NinjaScript we are able to access the bars we want through proper use of the bar's indexing.

The bar's indexing is setup in a reverse chronological order. This means "0" refers to the most recent bar, "1" refers to the previous bar, "2" refers to the bar before that one, etc.

For example, if we wanted to subtract the high and low of 10 bars ago from each other we would do this:

| ns |
| --- |
| double value = High[10] - Low[10]; |

Now that we know how the indexing works there are several properties and methods at our disposal that can help us access important keystone bars. The more important ones are [CurrentBar](https://ninjatrader.com/es/support/helpGuides/nt8/currentbar.htm) and [BarsSinceNewTradingDay](https://ninjatrader.com/es/support/helpGuides/nt8/barssincenewtradingday.htm).

**CurrentBar**

CurrentBar returns an int representing the number of bars existing on the chart. This property is most useful when you want to run calculations from the very beginning of the chart.

For example, if you wanted to find the average high value of the first 10 bars on the chart you could do this:

| ns |
| --- |
| double highValue = 0;  int x = CurrentBar;  while (x > CurrentBar - 10)  {     highValue += High[x];     x--;  }  Print("The average high value: " + highValue/10); |

|  |
| --- |
| **Note**: A common mistake in using CurrentBar is using it in the index to access the most recent bar. In this situation, instead of doing something like Close[CurrentBar] you will want to do Close[0]. |

**BarsSinceNewTradingDay**

BarsSinceNewTradingDay is another property that can help you find the first bar of the current trading day. The difference between BarsSinceNewTradingDay and CurrentBar is that BarsSinceNewTradingDay resets its count whenever a new session begins. This means if you use it in an index it will only get you to the beginning of the current session and not any previous sessions.

For example, if you wanted to find the open of the current session you could do this:

| ns |
| --- |
| double openValue = Open[Bars.BarsSinceNewTradingDay]; |

The example used in the discussion about CurrentBar can also be done with Bars.BarsSinceNewTradingDay if you wanted to calculate values based on the current session instead of the start of the chart too.

|  |
| --- |
| **Note**: If you wish to access values older than 256 bars ago you will need to ensure the [MaximumBarsLookBack](https://ninjatrader.com/es/support/helpGuides/nt8/maximumbarslookback.htm)is set to .Infinite. |

**Other Properties and Methods**

There are also a number of other properties and methods that can be useful in helping you locate the correct bars index to reference. Please take a look at these in the help guide:

[BarsSinceEntryExecution()](https://ninjatrader.com/es/support/helpGuides/nt8/barssinceentryexecution.htm)

[BarsSinceExitExecution()](https://ninjatrader.com/es/support/helpGuides/nt8/barssinceexitexecution.htm)

[GetBar()](https://ninjatrader.com/es/support/helpGuides/nt8/getbar.htm)

[GetDayBar()](https://ninjatrader.com/es/support/helpGuides/nt8/getdaybar.htm)

[HighestBar()](https://ninjatrader.com/es/support/helpGuides/nt8/highestbar.htm)

[LowestBar()](https://ninjatrader.com/es/support/helpGuides/nt8/lowestbar.htm)

[LRO()](https://ninjatrader.com/es/support/helpGuides/nt8/least_recent_occurence_lro.htm)

[MRO()](https://ninjatrader.com/es/support/helpGuides/nt8/most_recent_occurence_mro.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) > [Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) >  **BarsSinceEntryExecution()** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/barsrequiredtotrade.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/barssinceexitexecution.htm) |

**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](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm):

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](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) property. |
| entryExecutionsAgo | Number of entry executions ago. Pass in 0 for the number of bars since the last entry execution. |

**Examples**

| ns | |
| --- | --- |
| 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();     } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) >  **BarsSinceExitExecution()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/barssinceentryexecution.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/chartindicators.htm) |

**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](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm):

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](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) property. |
| exitExecutionsAgo | Number of exit executions ago. Pass in 0 for the number of bars since the last exit execution. |

|  |
| --- |
| **Tip**:  Please see [SetStopLoss()](https://ninjatrader.com/es/support/helpGuides/nt8/setstoploss.htm), [SetProfitTarget()](https://ninjatrader.com/es/support/helpGuides/nt8/setprofittarget.htm) or [SetTrailStop()](https://ninjatrader.com/es/support/helpGuides/nt8/settrailstop.htm) for their corresponding signal name |

**Examples**

| ns | |
| --- | --- |
| 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(); } | |
| **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) > [Bars](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) >  **GetDayBar()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/getclose.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/gethigh.htm) |

**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()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm)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()](https://ninjatrader.com/es/support/helpGuides/nt8/current_day_ohl.htm) 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](http://msdn.microsoft.com/en-us/library/system.datetime.aspx" \t "_blank) 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**

| ns | |
| --- | --- |
| 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); } | |
| **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) >  **HighestBar()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/getmedian.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/market_data.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/falling.htm) |

**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**

| ns | |
| --- | --- |
| 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));             } | |
| **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) >  **LowestBar()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/least_recent_occurence_lro.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/market_data.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/most_recent_occurence_mro.htm) |

**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**

| ns | |
| --- | --- |
| 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));             } | |
| **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) >  **Least Recent Occurrence (LRO)** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/rising.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/market_data.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/lowestbar.htm) |

**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](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) 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](http://msdn.microsoft.com/en-us/library/bb397687.aspx) syntax |

**Examples**

| ns |
| --- |
| 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)](https://ninjatrader.com/es/support/helpGuides/nt8/most_recent_occurence_mro.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) > [Analytical](https://ninjatrader.com/es/support/helpGuides/nt8/market_data.htm) >  **Most Recent Occurrence (MRO)** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/lowestbar.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/market_data.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/slope.htm) |

**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](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) 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](http://msdn.microsoft.com/en-us/library/bb397687.aspx) syntax |

**Examples**

| ns |
| --- |
| 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)](https://ninjatrader.com/es/support/helpGuides/nt8/least_recent_occurence_lro.htm)

|  |  |
| --- | --- |
| **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) > [Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/strategy2.htm) >  **Resetting values at the beginning of new trading sessions** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/removing_draw_objects_from_the.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/strategy2.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/rounding_values_to_the_nearest.htm) |

Normally calculated values are carried over between trading sessions, but sometimes you may want to reset these values to begin a trading session fresh. The technique demonstrated in this reference sample can be useful to do things like resetting counters you may be running or clearing bool flags you may have set.

**Key concepts in this example**

•Resetting a variable at the beginning of a new trading session

•Limiting the number of trades a strategy can make per trading session

**Important related documentation**

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

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

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

•[ExitLong()](https://ninjatrader.com/es/support/helpGuides/nt8/exitlong.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

<https://ninjatrader.com/support/helpGuides/nt8/samples/SampleTradeLimiter_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) > [Bars](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) >  **IsFirstBarOfSession** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/getvolume.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/isfirstbarofsessionbyindex.htm) |

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

**Syntax**  
Bars.IsFirstBarOfSession

|  |
| --- |
| **Tip**:  For checking at a specified bar index, please see [IsFirstBarOfSessionByIndex()](https://ninjatrader.com/es/support/helpGuides/nt8/isfirstbarofsessionbyindex.htm) |

**Examples**

| ns | |
| --- | --- |
| 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])); } | |
| **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) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **IsFirstTickOfBar** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isdataseriesrequired.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/isresetonnewtradingdays.htm) |

**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](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm) property is set to **Calculate.OnEachTick** or **Calculate.OnPriceChange**.

|  |
| --- |
| **Warning**: This property should **NOT** be accessed outside of the [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) method. |

|  |
| --- |
| **Note**: If a bar type is set up to [remove the last bar](https://ninjatrader.com/es/support/helpGuides/nt8/removelastbar.htm) 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**

| ns | |
| --- | --- |
| // 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(); } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) > [Order Methods](https://ninjatrader.com/es/support/helpGuides/nt8/order_methods.htm) > [Managed Approach](https://ninjatrader.com/es/support/helpGuides/nt8/managed_approach.htm) >  **EnterLong()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/managed_changeorder.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/managed_approach.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/enterlonglimit.htm) |

**Definition**

Generates a buy market order to enter a long position.

**Method Return Value**

An [Order](https://ninjatrader.com/es/support/helpGuides/nt8/order.htm) read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the [Advanced Order Handling](https://ninjatrader.com/es/support/helpGuides/nt8/advanced_order_handling.htm) 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](https://ninjatrader.com/es/support/helpGuides/nt8/advanced_order_handling.htm) concepts:

EnterLong(int *barsInProgressIndex*, int *quantity*, string *signalName*)

|  |
| --- |
| **Note**: If using a method signature that does not have the parameter quantity, the order quantity will be taken from the quantity value set in the strategy dialog window when running or backtesting a strategy |

**Parameters**

|  |  |
| --- | --- |
| signalName | User defined signal name identifying the order generated. Max 50 characters. |
| quantity | Entry order quantity (if 0 is passed in, will be set to 1, except for stocks 100) |
| barsInProgressIndex | The index of the Bars object the order is to be submitted against. Used to determines what instrument the order is submitted for.      See the [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) property. |

**Examples**

| ns | |
| --- | --- |
| 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"); } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) > [Order Methods](https://ninjatrader.com/es/support/helpGuides/nt8/order_methods.htm) > [Managed Approach](https://ninjatrader.com/es/support/helpGuides/nt8/managed_approach.htm) >  **ExitLong()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/entershortstopmarket.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/managed_approach.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/exitlonglimit.htm) |

**Definition**

Generates a sell market order to exit a long position.

**Method Return Value**

An [Order](https://ninjatrader.com/es/support/helpGuides/nt8/order.htm) read-only object that represents the order. Reserved for experienced programmers, additional information can be found within the [Advanced Order Handling](https://ninjatrader.com/es/support/helpGuides/nt8/advanced_order_handling.htm) 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](https://ninjatrader.com/es/support/helpGuides/nt8/advanced_order_handling.htm) 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](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) property. |

**Examples**

| ns |
| --- |
| 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](https://ninjatrader.com/es/support/helpGuides/nt8/managed_approach.htm)):  •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 |

//

// 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 Average Directional Index measures the strength of a prevailing trend as well as whether movement

/// exists in the market. The ADX is measured on a scale of 0 100. A low ADX value (generally less than 20)

/// can indicate a non-trending market with low volumes whereas a cross above 20 may indicate the start of

/// a trend (either up or down). If the ADX is over 40 and begins to fall, it can indicate the slowdown of a current trend.

/// </summary>

public class ADX : Indicator

{

private Series<double> dmPlus;

private Series<double> dmMinus;

private Series<double> sumDmPlus;

private Series<double> sumDmMinus;

private Series<double> sumTr;

private Series<double> tr;

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = NinjaTrader.Custom.Resource.NinjaScriptIndicatorDescriptionADX;

Name = NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameADX;

IsSuspendedWhileInactive = true;

Period = 14;

AddPlot(Brushes.Green, "ADX");

AddLine(Brushes.DarkViolet, 25, "Lower");

AddLine(Brushes.YellowGreen, 75, "Upper");

}

else if (State == State.Configure)

{

dmPlus = new Series<double>(this);

dmMinus = new Series<double>(this);

sumDmPlus = new Series<double>(this);

sumDmMinus = new Series<double>(this);

sumTr = new Series<double>(this);

tr = new Series<double>(this);

}

}

protected override void OnBarUpdate()

{

double high0 = High[0];

double low0 = Low[0];

if (CurrentBar == 0)

{

tr[0] = high0 - low0;

dmPlus[0] = 0;

dmMinus[0] = 0;

sumTr[0] = tr[0];

sumDmPlus[0] = dmPlus[0];

sumDmMinus[0] = dmMinus[0];

Value[0] = 50;

}

else

{

double high1 = High[1];

double low1 = Low[1];

double close1 = Close[1];

tr[0] = Math.Max(Math.Abs(low0 - close1), Math.Max(high0 - low0, Math.Abs(high0 - close1)));

dmPlus[0] = high0 - high1 > low1 - low0 ? Math.Max(high0 - high1, 0) : 0;

dmMinus[0] = low1 - low0 > high0 - high1 ? Math.Max(low1 - low0, 0) : 0;

if (CurrentBar < Period)

{

sumTr[0] = sumTr[1] + tr[0];

sumDmPlus[0] = sumDmPlus[1] + dmPlus[0];

sumDmMinus[0] = sumDmMinus[1] + dmMinus[0];

}

else

{

double sumTr1 = sumTr[1];

double sumDmPlus1 = sumDmPlus[1];

double sumDmMinus1 = sumDmMinus[1];

sumTr[0] = sumTr1 - sumTr1 / Period + tr[0];

sumDmPlus[0] = sumDmPlus1 - sumDmPlus1 / Period + dmPlus[0];

sumDmMinus[0] = sumDmMinus1 - sumDmMinus1 / Period + dmMinus[0];

}

double sumTr0 = sumTr[0];

double diPlus = 100 \* (sumTr0.ApproxCompare(0) == 0 ? 0 : sumDmPlus[0] / sumTr[0]);

double diMinus = 100 \* (sumTr0.ApproxCompare(0) == 0 ? 0 : sumDmMinus[0] / sumTr[0]);

double diff = Math.Abs(diPlus - diMinus);

double sum = diPlus + diMinus;

Value[0] = sum.ApproxCompare(0) == 0 ? 50 : ((Period - 1) \* Value[1] + 100 \* diff / sum) / 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 ADX[] cacheADX;

public ADX ADX(int period)

{

return ADX(Input, period);

}

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

{

if (cacheADX != null)

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

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

return cacheADX[idx];

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

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.ADX ADX(int period)

{

return indicator.ADX(Input, period);

}

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

{

return indicator.ADX(input, period);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

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

{

public Indicators.ADX ADX(int period)

{

return indicator.ADX(Input, period);

}

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

{

return indicator.ADX(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>

/// Plots the open, high, and low values from the session starting on the current day.

/// </summary>

public class CurrentDayOHL : Indicator

{

private DateTime currentDate = Core.Globals.MinDate;

private double currentOpen = double.MinValue;

private double currentHigh = double.MinValue;

private double currentLow = double.MaxValue;

private Data.SessionIterator sessionIterator;

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = NinjaTrader.Custom.Resource.NinjaScriptIndicatorDescriptionCurrentDayOHL;

Name = NinjaTrader.Custom.Resource.NinjaScriptIndicatorNameCurrentDayOHL;

IsAutoScale = false;

IsOverlay = true;

IsSuspendedWhileInactive = true;

PlotCurrentValue = false;

ShowLow = true;

ShowHigh = true;

ShowOpen = true;

BarsRequiredToPlot = 0;

AddPlot(new Stroke() { Brush = Brushes.Orange, Width = 2, DashStyleHelper = DashStyleHelper.Dash }, PlotStyle.Square, "Current Open");

AddPlot(new Stroke() { Brush = Brushes.Green, Width = 2, DashStyleHelper = DashStyleHelper.Dash }, PlotStyle.Square, "Current High");

AddPlot(new Stroke() { Brush = Brushes.Red, Width = 2, DashStyleHelper = DashStyleHelper.Dash }, PlotStyle.Square, "Current Low");

}

else if (State == State.Configure)

{

currentDate = Core.Globals.MinDate;

currentOpen = double.MinValue;

currentHigh = double.MinValue;

currentLow = double.MaxValue;

sessionIterator = null;

}

}

protected override void OnBarUpdate()

{

if (!Bars.BarsType.IsIntraday)

{

Draw.TextFixed(this, "error msg", Custom.Resource.CurrentDayOHLError, TextPosition.BottomRight);

return;

}

if (sessionIterator == null)

sessionIterator = new Data.SessionIterator(BarsArray[0]);

bool sameDay = true;

if (currentDate != sessionIterator.GetTradingDay(Time[0]) || currentOpen == double.MinValue)

{

currentOpen = Open[0];

currentHigh = High[0];

currentLow = Low[0];

sameDay = false;

}

currentHigh = Math.Max(currentHigh, High[0]);

currentLow = Math.Min(currentLow, Low[0]);

if (ShowOpen)

{

if (!PlotCurrentValue || !sameDay)

CurrentOpen[0] = currentOpen;

else

for (int idx = 0; idx < CurrentBar - 1; idx++)

CurrentOpen[idx] = currentOpen;

}

if (ShowHigh)

{

if (!PlotCurrentValue || currentHigh != High[0])

CurrentHigh[0] = currentHigh;

else

for (int idx = 0; idx < CurrentBar - 1; idx++)

CurrentHigh[idx] = currentHigh;

}

if (ShowLow)

{

if (!PlotCurrentValue || currentLow != Low[0])

CurrentLow[0] = currentLow;

else

for (int idx = 0; idx < CurrentBar - 1; idx++)

CurrentLow[idx] = currentLow;

}

currentDate = sessionIterator.GetTradingDay(Time[0]);

}

#region Properties

[Browsable(false)] // this line prevents the data series from being displayed in the indicator properties dialog, do not remove

[XmlIgnore()] // this line ensures that the indicator can be saved/recovered as part of a chart template, do not remove

public Series<double> CurrentOpen

{

get { return Values[0]; }

}

[Browsable(false)] // this line prevents the data series from being displayed in the indicator properties dialog, do not remove

[XmlIgnore()] // this line ensures that the indicator can be saved/recovered as part of a chart template, do not remove

public Series<double> CurrentHigh

{

get { return Values[1]; }

}

[Browsable(false)] // this line prevents the data series from being displayed in the indicator properties dialog, do not remove

[XmlIgnore()] // this line ensures that the indicator can be saved/recovered as part of a chart template, do not remove

public Series<double> CurrentLow

{

get { return Values[2]; }

}

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

public bool PlotCurrentValue

{ get; set; }

[Display(ResourceType = typeof(Custom.Resource), Name = "Show high", GroupName = "NinjaScriptParameters", Order = 1)]

public bool ShowHigh

{ get; set; }

[Display(ResourceType = typeof(Custom.Resource), Name = "Show low", GroupName = "NinjaScriptParameters", Order = 2)]

public bool ShowLow

{ get; set; }

[Display(ResourceType = typeof(Custom.Resource), Name = "Show open", GroupName = "NinjaScriptParameters", Order = 3)]

public bool ShowOpen

{ get; set; }

#endregion

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

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

{

private CurrentDayOHL[] cacheCurrentDayOHL;

public CurrentDayOHL CurrentDayOHL()

{

return CurrentDayOHL(Input);

}

public CurrentDayOHL CurrentDayOHL(ISeries<double> input)

{

if (cacheCurrentDayOHL != null)

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

if (cacheCurrentDayOHL[idx] != null && cacheCurrentDayOHL[idx].EqualsInput(input))

return cacheCurrentDayOHL[idx];

return CacheIndicator<CurrentDayOHL>(new CurrentDayOHL(), input, ref cacheCurrentDayOHL);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.CurrentDayOHL CurrentDayOHL()

{

return indicator.CurrentDayOHL(Input);

}

public Indicators.CurrentDayOHL CurrentDayOHL(ISeries<double> input )

{

return indicator.CurrentDayOHL(input);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

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

{

public Indicators.CurrentDayOHL CurrentDayOHL()

{

return indicator.CurrentDayOHL(Input);

}

public Indicators.CurrentDayOHL CurrentDayOHL(ISeries<double> input )

{

return indicator.CurrentDayOHL(input);

}

}

}

#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 strategies and is required. Do not change it.

namespace NinjaTrader.NinjaScript.Strategies

{

public class SampleTradeLimiter : Strategy

{

private int aDXPeriod = 14; // Default setting for ADXPeriod

private int tradeCounter = 0; // This variable represents the number of trades taken per day.

private int maxTrades = 5; // This variable sets the maximum number of trades to take per day.

protected override void OnStateChange()

{

if(State == State.SetDefaults)

{

ADXPeriod = 14;

MaxTrades = 5;

Calculate = Calculate.OnBarClose;

Name = "Sample Trade Limiter";

}

else if(State == State.Configure)

{

// Add the current day open, high, low indicator to visually see entry conditions.

AddChartIndicator(CurrentDayOHL());

}

}

protected override void OnBarUpdate()

{

// Make sure there are enough bars.

if (CurrentBar < 1)

return;

// Reset the tradeCounter value at the first tick of the first bar of each session.

if (Bars.IsFirstBarOfSession && IsFirstTickOfBar)

{

Print("resetting tradeCounter");

tradeCounter = 0;

}

// If the amount of trades is less than the permitted value and the position is flat, go on to the next set of conditions.

if (tradeCounter < MaxTrades && Position.MarketPosition == MarketPosition.Flat)

{

/\* If a new low is made, enter short and increase the trade count by 1.

In C#, ++ means increment by one. An equilivent would be tradeCounter = tradeCounter + 1; \*/

if (CurrentDayOHL().CurrentLow[0] < CurrentDayOHL().CurrentLow[1])

{

tradeCounter++;

EnterShort();

}

// If a new high is made, enter long and increase the trade count by 1.

else if (CurrentDayOHL().CurrentHigh[0] > CurrentDayOHL().CurrentHigh[1])

{

tradeCounter++;

EnterLong();

}

}

/\* Exit a position if "the trend has ended" as indicated by ADX.

If the current ADX value is less than the previous ADX value, the trend strength is weakening. \*/

if (ADX(ADXPeriod)[0] < ADX(ADXPeriod)[1] && Position.MarketPosition != MarketPosition.Flat)

{

if (Position.MarketPosition == MarketPosition.Long)

ExitLong();

else if (Position.MarketPosition == MarketPosition.Short)

ExitShort();

}

}

#region Properties

[Display(GroupName="Parameters", Description="Period for the ADX indicator")]

public int ADXPeriod

{

get { return aDXPeriod; }

set { aDXPeriod = Math.Max(1, value); }

}

[Display(GroupName="Parameters", Description="Maximum number of trades to take per day.")]

public int MaxTrades

{

get { return maxTrades; }

set { maxTrades = Math.Max(1, value); }

}

#endregion

}

}