|  |  |
| --- | --- |
| **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>** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/url.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) |

**Definition**

ISeries<T> is an interface that is implemented by all NinjaScript classes that manage historical data as an ISeries<double> (Open, High, Low, Close, etc), used for indicator input, and other object data.  Please see the help guide article on [Working with Price Series](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_price_series.htm) for a basic overview on how to access this information.

**Types of ISeries**

|  |  |
| --- | --- |
| [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) | Represents a generic custom data structure for custom development |
| [PriceSeries](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) | Historical price data structured as an ISeries<double> interface (Close[0], High[0], Low[0], etc) |
| [TimeSeries](https://ninjatrader.com/es/support/helpGuides/nt8/timeseries.htm) | Historical time stamps structured as an ISeries<DateTime> interface (Time[0]) |
| [VolumeSeries](https://ninjatrader.com/es/support/helpGuides/nt8/volumeseries.htm) | Historical volume data structured as an ISeries<double> interface (Volume[0]) |

**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) | Indicates if the specified input is set at a barsAgo value relative to the current bar. |
| [IsValidDataPointAt()](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapointat.htm) | Indicates if the specified input is set at a specified bar index value. |
| [Count](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_count.htm) | Return the number total number of values in the ISeries array |

|  |
| --- |
| **Tips**: (see examples below)  1.By specifying a parameter of type ISeries<double>, you can then pass in an array of closing prices, an indicator, or a user defined data series.  2.When working with ISeries<double> objects in your code you may come across situations where you are not sure if the value being accessed is a valid value or just a "placeholder" value. To check if you are using valid values for your logic calculations that have been explicitly set, please use .IsValidDataPoint(int *barsAgo*)to check. |

**Examples**

| ns | **Using ISeries as a method parameter** |
| --- | --- |
|  | //create custom a method named DoubleTheValue that accepts any object that implements // the ISeries<double> interface as a parameter private double DoubleTheValue(ISeries<double> priceData) {     return priceData[0] \* 2; }   protected override void OnBarUpdate() {   // This custom method is then used twice,   //the first time passing in an array of closing prices     Print(DoubleTheValue(Close));   //and the second time passing in a 20 period simple moving average.     Print(DoubleTheValue(SMA(20))); } |

| ns | **Checking ISeries value before accessing** |
| --- | --- |
|  | protected override void OnBarUpdate() {     // Only set our plot if the input is a valid value     if (Input.IsValidDataPoint(0))         Plot0[0] = Input[0]; } |
| **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) >  **MaximumBarsLookBack** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapointat.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) |

**Definition**

Determines memory performance of custom [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) objects (such as Series<double>, Series<long>, etc.).  When using **MaximumBarsLookBack.TwoHundredFiftySix**, only the last 256 values of the series object will be stored in memory and be accessible for reference. This results in significant memory savings when using multiple series objects. In the rare case should you need older values you can use **MaximumBarsLookBack.Infinite** to allow full access of the series.

|  |
| --- |
| **Notes**:  •ISeries<T> objects that hold bar data (such as Close, High, Volume, Time, etc) always use **MaximumBarsLookBack.Infinite** which ensures all data points are always accessible during the lifetime of your NinjaScript indicator or strategy.  •Series<double> objects that hold indicator [plot values](https://ninjatrader.com/es/support/helpGuides/nt8/values.htm) always use **MaximumBarsLookBack.Infinite** which ensures that charts always display the entire indicator's calculated values. |

**Property Value**

A **MaximumBarsLookBack** enum value. Default value is **MaximumBarsLookBack.TwoHundredFiftySix**

Possible values are:

|  |  |
| --- | --- |
| MaximumBarsLookBack.TwoHundredFiftySix | Only the last 256 values of the series object will be stored in memory and accessible for reference (improves memory performance) |
| MaximumBarsLookBack.Infinite | Allow full access of the series, but you will then not be able to utilize the benefits of memory optimization |

|  |
| --- |
| **Tip**:  A **MaximumBarsLookBack.TwoHundredFiftySix** series works as a circular ring buffer, which will "loop" when the series reaches full capacity.  Specifically, once there are 256 entries in the series, new data added to the series overwrite the oldest data. |

**Syntax**

MaximumBarsLookBack

**Examples**

| ns **Setting all custom series to use the default MaximumBarsLookBack** |
| --- |
| Series<double> myDoubleSeries = null; Series<string> myStringSeries = null;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "Example Indicator";     // Store all series values instead of only the last 256 values     MaximumBarsLookBack = MaximumBarsLookBack.Infinite;   }   else if (State == State.DataLoaded)   {     // The custom Series<t> below are all constructed using only the NinjaScriptBase object (i.e., "this")     // therefore, the Series<T> MaximumBarsLookBack is taken from the NinjaScript's configured MaximumBarsLookBack property     myDoubleSeries = new Series<double>(this);     myStringSeries = new Series<string>(this);   } } |

| ns **Optimizing custom series to use unique MaximumBarsLookBack behavior** | |
| --- | --- |
| Series<double> myDoubleSeries = null; Series<string> myStringSeries = null;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "Example Indicator";   }   else if (State == State.DataLoaded)   {     // The custom Series<t> below are constructed using MaximumBarsLookBack parameter     // therefore, each Series<t> will use their uniquely specified MaximumBarsLookBack properites     myDoubleSeries = new Series<double>(this, MaximumBarsLookBack.Infinite); // stores all values     myStringSeries = new Series<string>(this, MaximumBarsLookBack.TwoHundredFiftySix); // only the last 256 values (better performance)   } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Working with Price Series** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_pixel_coordinates.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/reference_samples.htm) |

**Price Data Overview**

The core objective of developing custom Indicators and Strategies with NinjaScript is to evaluate price data. NinjaScript allows you to reference current and historical price data. There are several categories of price data which include ISeries<T>, Indicator and Custom Historical Series.

**Definitions**

|  |  |
| --- | --- |
| [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) | Standard bar based price types such as closing, opening, high, low prices and volume |
| [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) | Calculated values based on price type values such as a simple moving average |
| Custom Historical [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) | Custom calculated values that you wish to store and associate to each historical bar |

**Referencing Series**

|  |  |  |  |
| --- | --- | --- | --- |
| **ISeries<T>** | **Syntax** | **Editor Shortcut** | **Definition** |
| Close | Close[int *barsAgo*] | "c" + Tab Key | Last traded price of a bar |
| Open | Open[int *barsAgo*] | "o" + Tab Key | Opening price of a bar |
| High | High[int *barsAgo*] | "h" + Tab Key | Highest traded price of a bar |
| Low | Low[int *barsAgo*] | "l" + Tab Key | Lowest traded price of a bar |
| Volume | Volume[int *barsAgo*] | "v" + Tab Key | Number of shares/contracts traded of a bar |
| Input | Input[int *barsAgo*] | "i" + Tab Key | Default price type of a bar |

You will notice that to reference any price data you need to include a value for [int *barsAgo*]. This is a very simple concept; barsAgo represents the number of bars ago to reference and int indicates that barsAgo is an integer value. As an example, we could write a statement to check if the the high price of 1 bar ago is less than the high price of the current bar like this:

 High[1] < High[0];

You could write a statement to calculate the average closing price of the last three bars like this:

 ( Close[2] + Close[1] + Close[0] ) / 3;

As you may have already figured out, referencing the current bar data is accomplished by passing in a value of 0 (zero) to the barsAgo parameter. Basically, we are saying show me the price data of zero bars ago, which means the current bar.

|  |
| --- |
| **Note**:  In most cases, you will access the historical price series using a core event handler such as OnBarUpdate.  For more advance developers, you may find situations where you wish to access historical price series outside of the core event methods, such as your own custom mouse click.  In these advanced scenarios, you may run into situations where the barsAgo pointer is not in sync with the current bar, and may result in errors when trying to obtain this information.  In those cases, please use the Bars.Get...() methods with the absolute bar index (e.g., [Bars.GetClose(](https://ninjatrader.com/es/support/helpGuides/nt8/getclose.htm)), [Bars.GetTime()](https://ninjatrader.com/es/support/helpGuides/nt8/gettime.htm), etc.) |

**Referencing Indicator Data**  
NinjaScript includes a library of built in indicators that you can access. Please see the [Indicator Methods](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm) reference section for clear definitions for how to access each indicator.

All indicator values can be accessed in the following way:

 indicator(parameters)[int barsAgo]

where indicator is the name of the indicator you want to access, parameters is any associated parameters the indicator requires and barsAgo is the number of bars we wish to offset from the current bar.

As an example, we could write a statement to check if the current closing price is greater than the 20 period simple moving average like this:

 Close[0] > SMA(20)[0];

If you wanted to perform the same check but only check against a 20 period simple moving average of high prices you would write it like this:

 Close[0] > SMA(High, 20)[0];

You could write a statement to see if a 14 period CCI indicator is rising like this:

 CCI(14)[0] > CCI(14)[1];

Value of a 10 period CCI 1 bar ago = CCI(10)[1]

Please review the [Indicator Methods](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm) section for proper syntax for accessing different indicator values.

|  |  |
| --- | --- |
| **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) >  **AddDataSeries()** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/addheikenashi.htm) |

**Definition**

Adds a Bars object for developing a multi-series (multi-time frame or multi-instrument) NinjaScript.

**Related Methods and Properties**

|  |  |
| --- | --- |
| [AddHeikenAshi()](https://ninjatrader.com/es/support/helpGuides/nt8/addheikenashi.htm) | This method adds a Heiken Ashi Bars object for multi-series NinjaScript. |
| [AddKagi()](https://ninjatrader.com/es/support/helpGuides/nt8/addkagi.htm) | This method adds a Kagi Bars object for multi-series NinjaScript. |
| [AddLineBreak()](https://ninjatrader.com/es/support/helpGuides/nt8/addlinebreak.htm) | This method adds a Line Break Bars object for multi-series NinjaScript. |
| [AddPointAndFigure()](https://ninjatrader.com/es/support/helpGuides/nt8/addpointandfigure.htm) | This method adds a Point-and-Figure Bars object for multi-series NinjaScript. |
| [AddRenko()](https://ninjatrader.com/es/support/helpGuides/nt8/addrenko.htm) | This method adds a Renko Bars object for multi-series NinjaScript. |
| [AddVolumetric()](https://ninjatrader.com/es/support/helpGuides/nt8/addvolumetric.htm) | This method adds a Order Flow Volumetric Bars object for multi-series NinjaScript. |
| [BarsArray](https://ninjatrader.com/es/support/helpGuides/nt8/barsarray.htm) | An array holding Bars objects that are added via the [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) method. |
| [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) | An index value of the current Bars object that has called the [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) method. |
| [BarsPeriods](https://ninjatrader.com/es/support/helpGuides/nt8/barsperiods.htm) | Holds an array of BarsPeriod objects synchronized to the number of unique Bars objects held within the parent NinjaScript object. |
| [CurrentBars](https://ninjatrader.com/es/support/helpGuides/nt8/currentbars.htm) | Holds an array of int values representing the number of the current bar in a Bars object. |

**Syntax**

The following syntax will add another Bars object for the primary instrument of the script.  
AddDataSeries(BarsPeriod barsPeriod)  
AddDataSeries(BarsPeriodType periodType, int period)

The following syntax allows you to add another Bars object for a different instrument to the script:

AddDataSeries(string instrumentName, BarsPeriodType periodType, int period)  
AddDataSeries(string instrumentName, BarsPeriodType periodType, int period, MarketDataType marketDataType)  
AddDataSeries(string instrumentName, BarsPeriod barsPeriod)  
AddDataSeries(string instrumentName, BarsPeriod barsPeriod, string tradingHoursName)  
AddDataSeries(string instrumentName, BarsPeriod barsPeriod, string tradingHoursName, bool? isResetOnNewTradingDay)  
AddDataSeries(string instrumentName, BarsPeriod barsPeriod, int barsToLoad, string tradingHoursName, bool? isResetOnNewTradingDay)

AddDataSeries(string instrumentName) //only for R15 and higher

|  |
| --- |
| **Warning:**  •This method should **ONLY** be called from the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm)method during **State.Configure**  •Should your script be the host for other scripts that are creating indicators and series dependent resources in **State.DataLoaded**, please make sure that the host is doing the same **AddDataSeries()** calls as those hosted scripts would. For further reference, please also review the 2nd example below and the 'Adding additional Bars Objects to NinjaScript' section in [Multi-Time Frame & Instruments](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm)  •Arguments supplied to **AddDataSeries()** should be hardcoded and **NOT** dependent on run-time variables which cannot be reliably obtained during [State.Configure](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) (e.g., [Instrument](https://ninjatrader.com/es/support/helpGuides/nt8/instrument.htm), [Bars](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm), or user input).  Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided.  Trying to load bars dynamically may result in an error similar to: **Unable to load bars series. Your NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.**  •When adding multiple Data Series of the same instrument and the same Bar Type, the 'barsToLoad' property will only be effective on the first added series. Subsequent series with a different barsToLoad setting will not load a different number of bars then the first series.  •The AddDataSeries(string instrumentName) overload allows loading a different instrument yet using the same BarsPeriod. This could not be supported for [Strategy Analyzer use with the 'Optimize Data Series'](https://ninjatrader.com/es/support/helpGuides/nt8/optimize_a_strategy.htm) option enabled, doing so may result in an error similar to: **Unable to load bars series. Your NinjaScript may be trying to use an additional data series dynamically in an unsupported manner.**  •If your NinjaScript object is using AddDataSeries() allowing to specify a tradingHoursName, please keep in mind that: An indicator / strategy with multiple DataSeries of the same instrument will only process realtime OnBarUpdate() calls when a tick occurs in session of the trading hour template of all added series. Any ticks not processed will be queued and processed as a tick comes in for all subsequent DataSeries.  •When instantiating indicators in a [Multi-Series script](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) in [OnStateChange](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm), the input any hosted indicator is running on should be explicitly stated |

**Parameters**

|  |  |
| --- | --- |
| instrumentName | A string determining instrument name such as "MSFT" |
| barsPeriod | The [BarsPeriod](https://ninjatrader.com/es/support/helpGuides/nt8/barsperiod.htm) object (period type and interval) |
| periodType | The BarsType used for the bars period    Possible values are:    •BarsPeriodType.Tick  •BarsPeriodType.Volume  •BarsPeriodType.Range  •BarsPeriodType.Second  •BarsPeriodType.Minute  •BarsPeriodType.Day  •BarsPeriodType.Week  •BarsPeriodType.Month  •BarsPeriodType.Year |
| period | An int determining the period interval such as "3" for 3 minute bars |
| marketDataType | The MarketDataType used for the bars object (last, bid, ask)    Possible values are:    •MarketDataType.Ask  •MarketDataType.Bid  •MarketDataType.Last    **Note**: Please see the article [here](https://ninjatrader.com/es/support/helpGuides/nt8/using_historical_bid_ask_serie.htm) on using Bid/Ask series. |
| tradingHoursName | A string determining the trading hours template for the instrument |
| isResetOnNewTradingDay | A nullable bool\* determining if the Bars object should [Break at EOD](https://ninjatrader.com/es/support/helpGuides/nt8/break_at_eod.htm)    \*Will accept true, false or null as the input.  If null is used, the data series will use the settings of the primary data series. |
| barsToLoad | An int determining the number of historical bars to load |

|  |
| --- |
| **Tips**:  1. You can optionally add the exchange name as a suffix to the symbol name. This is only advised if the instrument has multiple possible exchanges that it can trade on and it is configured within the Instruments window. For example: AddDataSeries("MSFT Arca", BarsPeriodType.Minute, 5);  2. You can add a custom [BarsType](https://ninjatrader.com/es/support/helpGuides/nt8/bars_type.htm) which is installed on your system by casting the registered enum value for that BarsPeriodType.  For example: AddDataSeries((BarsPeriodType)14, 10);  3. You can specify optional [BarsPeriod](https://ninjatrader.com/es/support/helpGuides/nt8/barsperiod.htm) values (such as [Value2](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness_value.htm)) of a custom BarsType in the BarsPeriod object initializer.  For example: AddDataSeries(new BarsPeriod() { BarsPeriodType = (BarsPeriodType)14, Value = 10, Value2 = 20 });  4. For the instrument name parameter null could be passed in, resulting in the primary data series instrument being used. |

**Examples**

| ns |
| --- |
| protected override void OnStateChange() {     if (State == State.Configure)     {         // Add a 5 minute Bars object - BarsInProgress index = 1         AddDataSeries(BarsPeriodType.Minute, 5);           // Add a 100 tick Bars object for the ES 09-16 contract - BarsInProgress index = 2         AddDataSeries("ES 09-16", BarsPeriodType.Tick, 100);     } }   protected override void OnBarUpdate() {     // Ignore bar update events for the supplementary - Bars object added above     if (BarsInProgress == 1 || BarsInProgress == 2)         return;       // Go long if we have three up bars on all bars objects     if (Close[0] > Open[0] && Closes[1][0] > Opens[1][0] && Closes[2][0] > Opens[2][0])         EnterLong(); } |

| ns | |
| --- | --- |
| protected override void OnStateChange() {     if (State == State.Configure)     {         // Our hosting script needs to have the AddDataSeries call included as well, which the Pivots indicator we call in the 2nd statement below           // also has per default in it's own State.Configure method. This is required since our Pivots indicator below is created in State.DataLoaded           // (which is happening after State.Configure and it depends on the AddDataSeries call to have the bars available to properly calculate in           // daily bars mode.         AddDataSeries(BarsPeriodType.Day, 1);     }      else if (State == State.DataLoaded)     {         //In this state, we pass the 1 day series to the Pivots indicator (as BarsArray[1]) and create its instance         pivots = Pivots(BarsArray[1], PivotRange.Weekly, HLCCalculationMode.DailyBars, 0, 0, 0, 20);     } } | |
| **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 secondary series as an input series for an indicator** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/removing_and_custom_formatting.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_series_or_dataseries_o.htm) |

Adding additional series to a script can be useful. You may also want to use this added data for an indicator's Input Series.

**Key concepts in this example**

•Adding series

•Supplying a series object to an indicator as the input series parameter

•Plotting using data from two different series

**Important related documentation**

•[AddDataSeries()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/adddataseries.htm)

•[AddPlot()](https://ninjatrader.com/support/helpGuides/nt8/en-us/addplot.htm" \t "_blank)

•[IsValidDataPoint()](https://ninjatrader.com/support/helpGuides/nt8/en-us/isvaliddatapoint.htm" \t "_blank)

**Import instructions**

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

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

3.Select the downloaded file

[SampleSecondarySeriesAsInputSeries\_NT8.zip](https://ninjatrader.com/support/helpGuides/nt8/samples/SampleSecondarySeriesAsInputSeries_NT8.zip)

//

// Copyright (C) 2018, 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#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.Gui.Tools;

using NinjaTrader.Data;

using NinjaTrader.NinjaScript;

using NinjaTrader.Core.FloatingPoint;

using NinjaTrader.NinjaScript.DrawingTools;

#endregion

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

namespace NinjaTrader.NinjaScript.Indicators

{

public class SampleSecondarySeriesAsInputSeries : Indicator

{

private SMA sma1;

private SMA sma2;

protected override void OnStateChange()

{

if (State == State.SetDefaults)

{

Description = @"";

Name = "SampleSecondarySeriesAsInputSeries";

Calculate = Calculate.OnBarClose;

IsOverlay = true;

DisplayInDataBox = true;

BarsRequiredToPlot = 14;

AddPlot(Brushes.Blue, "SMA Primary Series");

AddPlot(Brushes.Green, "SMA Secondary Series");

}

else if (State == State.Configure)

{

// Adds a secondary bar object to the indicator

AddDataSeries(BarsPeriodType.Minute, 5);

}

else if (State == State.DataLoaded)

{

// initialize the SMA using the primary series and assign to sma1

sma1 = SMA(BarsArray[0], 14);

// initialize the SMA using the secondary 5 minute series and assign to sma1

sma2 = SMA(BarsArray[1], 14);

}

}

protected override void OnBarUpdate()

{

// ensure both series have at least one bar

if (CurrentBars[0] < 1 || CurrentBars[1] < 1)

return;

// when the 5 minute series is processing set the secondary plot to the sma with the secondary series input

if (BarsInProgress == 1)

SMASecondary[0] = sma2[0];

// when the primary series is processing set the primary plot to the sma with the primary series input

if (BarsInProgress == 0)

{

SMAPrimary[0] = sma1[0];

// if the secondary 5 minute series did not close, set the current bar's value to the previous bar's value to prevent gaps

if (!SMASecondary.IsValidDataPoint(0))

SMASecondary[0] = SMASecondary[1];

}

}

[Browsable(false)]

[XmlIgnore]

public Series<double> SMAPrimary

{

get { return Values[0]; }

}

[Browsable(false)]

[XmlIgnore]

public Series<double> SMASecondary

{

get { return Values[1]; }

}

}

}

#region NinjaScript generated code. Neither change nor remove.

namespace NinjaTrader.NinjaScript.Indicators

{

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

{

private SampleSecondarySeriesAsInputSeries[] cacheSampleSecondarySeriesAsInputSeries;

public SampleSecondarySeriesAsInputSeries SampleSecondarySeriesAsInputSeries()

{

return SampleSecondarySeriesAsInputSeries(Input);

}

public SampleSecondarySeriesAsInputSeries SampleSecondarySeriesAsInputSeries(ISeries<double> input)

{

if (cacheSampleSecondarySeriesAsInputSeries != null)

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

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

return cacheSampleSecondarySeriesAsInputSeries[idx];

return CacheIndicator<SampleSecondarySeriesAsInputSeries>(new SampleSecondarySeriesAsInputSeries(), input, ref cacheSampleSecondarySeriesAsInputSeries);

}

}

}

namespace NinjaTrader.NinjaScript.MarketAnalyzerColumns

{

public partial class MarketAnalyzerColumn : MarketAnalyzerColumnBase

{

public Indicators.SampleSecondarySeriesAsInputSeries SampleSecondarySeriesAsInputSeries()

{

return indicator.SampleSecondarySeriesAsInputSeries(Input);

}

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

{

return indicator.SampleSecondarySeriesAsInputSeries(input);

}

}

}

namespace NinjaTrader.NinjaScript.Strategies

{

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

{

public Indicators.SampleSecondarySeriesAsInputSeries SampleSecondarySeriesAsInputSeries()

{

return indicator.SampleSecondarySeriesAsInputSeries(Input);

}

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

{

return indicator.SampleSecondarySeriesAsInputSeries(input);

}

}

}

#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) >  **GetValueAt()** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_count.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapoint.htm) |

**Definition**

Returns the underlying input value at a specified bar index value.

**Method Return Value**

A double value representing the value at a specified bar.

**Syntax**

GetValueAt(int barIndex)

ISeries<T>.GetValueAt(int barIndex)

|  |
| --- |
| **Tip**:  If called directly from the instance of the NinjaScript object, the value which is returned corresponds to the input series the object is running. (e.g., Close, High, Low, SMA, etc.).  If you're attempting to obtain another indicator value, you will need to pull this from the calculated indicator Value or Plot:    SMA(20).GetValueAt(123); // bar value SMA(20).Values[0].GetValueAt(123); // indicator value  (Input as Indicator).Values[0].GetValueAt(123) *//*passed in indicator value |

**Parameters**

|  |  |
| --- | --- |
| barIndex | An int representing an absolute bar index value |

**Examples**

| ns | |
| --- | --- |
| protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   // make sure there are bars displayed on the chart and the chart control is ready before running   if (Bars == null || chartControl == null)     return;           // loop through all the visable bars on the chart   for (int i = ChartBars.FromIndex - 1; i >= BarsRequiredToPlot; i--)   {     double value = GetValueAt(i);     Print(string.Format("The value at bar {0} is {1}", i, value));         } } | |
| **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) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) >  **Count** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_volumes.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/getvalueat.htm) |

**Definition**

Indicates the number total number of values in the ISeries<T> array.  This value should always be in sync with the [CurrentBars](https://ninjatrader.com/es/support/helpGuides/nt8/currentbars.htm) array for that series.

**Method Return Value**

A int representing the total size of the series

**Syntax**

Count

**Examples**

| ns | |
| --- | --- |
| protected override void OnBarUpdate() {   Print("Input count: " + Input.Count); } | |
| **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) >  **BarsRequiredToTrade** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/getatmstrategyuniqueid.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/barssinceentryexecution.htm) |

**Definition**

The number of historical bars required before the strategy starts processing order methods called in the [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) method. This property is generally set via the UI when starting a strategy.

|  |
| --- |
| **Note**:  In a multi-series strategy this restriction applies only for the primary Bars object.  This means your can run into situations where the primary bars required to trade have been reached, but the additional bars required have not. Should your strategy logic intertwine calculations across different Bars objects please ensure all Bars objects have met the BarsRequiredToTrade requirement before proceeding. This can be done via checks on the [CurrentBars](https://ninjatrader.com/es/support/helpGuides/nt8/currentbars.htm) array. |

**Property Value**

An int value representing the number of historical bars.  Default value is set to 20.

|  |
| --- |
| **Warning**:  This property should **ONLY** bet set from the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method during **State.SetDefaults** or **State.Configure** |

**Syntax**

BarsRequiredToTrade

|  |
| --- |
| **Tip**:  When working with a multi-series strategy, real-time bar update events for a particular Bars object are only received when that Bars object has satisfied the BarsRequiredToTrade requirement. To ensure this requirement is met, please use the CurrentBars array. |

**Examples**

| ns **Setting the default BarsRequiredToTrade value** |
| --- |
| protected override void OnStateChange() {     if (State == State.Configure)     {         BarsRequiredToTrade = 20;     } } |

| ns | **Checking BarsRequiredToTrade againt a CurrentBars array** |
| --- | --- |
|  | protected override void OnStateChange() {   if (State == State.SetDefaults)   {     BarsRequiredToTrade = 20;   }   else if (State == State.Configure)   {     // add 30 minute series for calcuation logic     AddDataSeries(BarsPeriodType.Minute, 30);   } }   protected override void OnBarUpdate() {   // do not process order logic until bars required to trade is met   // for both primary and 30-minute series have reached their bars required to trade   if (CurrentBars[0] < BarsRequiredToTrade || CurrentBars[1] < BarsRequiredToTrade)     return;     //order logic } |

|  |  |
| --- | --- |
| **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>** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/url.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) |

**Definition**

ISeries<T> is an interface that is implemented by all NinjaScript classes that manage historical data as an ISeries<double> (Open, High, Low, Close, etc), used for indicator input, and other object data.  Please see the help guide article on [Working with Price Series](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_price_series.htm) for a basic overview on how to access this information.

**Types of ISeries**

|  |  |
| --- | --- |
| [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) | Represents a generic custom data structure for custom development |
| [PriceSeries](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) | Historical price data structured as an ISeries<double> interface (Close[0], High[0], Low[0], etc) |
| [TimeSeries](https://ninjatrader.com/es/support/helpGuides/nt8/timeseries.htm) | Historical time stamps structured as an ISeries<DateTime> interface (Time[0]) |
| [VolumeSeries](https://ninjatrader.com/es/support/helpGuides/nt8/volumeseries.htm) | Historical volume data structured as an ISeries<double> interface (Volume[0]) |

**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) | Indicates if the specified input is set at a barsAgo value relative to the current bar. |
| [IsValidDataPointAt()](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapointat.htm) | Indicates if the specified input is set at a specified bar index value. |
| [Count](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_count.htm) | Return the number total number of values in the ISeries array |

|  |
| --- |
| **Tips**: (see examples below)  1.By specifying a parameter of type ISeries<double>, you can then pass in an array of closing prices, an indicator, or a user defined data series.  2.When working with ISeries<double> objects in your code you may come across situations where you are not sure if the value being accessed is a valid value or just a "placeholder" value. To check if you are using valid values for your logic calculations that have been explicitly set, please use .IsValidDataPoint(int *barsAgo*)to check. |

**Examples**

| ns | **Using ISeries as a method parameter** |
| --- | --- |
|  | //create custom a method named DoubleTheValue that accepts any object that implements // the ISeries<double> interface as a parameter private double DoubleTheValue(ISeries<double> priceData) {     return priceData[0] \* 2; }   protected override void OnBarUpdate() {   // This custom method is then used twice,   //the first time passing in an array of closing prices     Print(DoubleTheValue(Close));   //and the second time passing in a 20 period simple moving average.     Print(DoubleTheValue(SMA(20))); } |

| ns | **Checking ISeries value before accessing** |
| --- | --- |
|  | protected override void OnBarUpdate() {     // Only set our plot if the input is a valid value     if (Input.IsValidDataPoint(0))         Plot0[0] = Input[0]; } |
| **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) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) >  **Count** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/iseries_volumes.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/getvalueat.htm) |

**Definition**

Indicates the number total number of values in the ISeries<T> array.  This value should always be in sync with the [CurrentBars](https://ninjatrader.com/es/support/helpGuides/nt8/currentbars.htm) array for that series.

**Method Return Value**

A int representing the total size of the series

**Syntax**

Count

**Examples**

| ns | |
| --- | --- |
| protected override void OnBarUpdate() {   Print("Input count: " + Input.Count); } | |
| **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) > [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) >  **MaximumBarsLookBack** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapointat.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) |

**Definition**

Determines memory performance of custom [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) objects (such as Series<double>, Series<long>, etc.).  When using **MaximumBarsLookBack.TwoHundredFiftySix**, only the last 256 values of the series object will be stored in memory and be accessible for reference. This results in significant memory savings when using multiple series objects. In the rare case should you need older values you can use **MaximumBarsLookBack.Infinite** to allow full access of the series.

|  |
| --- |
| **Notes**:  •ISeries<T> objects that hold bar data (such as Close, High, Volume, Time, etc) always use **MaximumBarsLookBack.Infinite** which ensures all data points are always accessible during the lifetime of your NinjaScript indicator or strategy.  •Series<double> objects that hold indicator [plot values](https://ninjatrader.com/es/support/helpGuides/nt8/values.htm) always use **MaximumBarsLookBack.Infinite** which ensures that charts always display the entire indicator's calculated values. |

**Property Value**

A **MaximumBarsLookBack** enum value. Default value is **MaximumBarsLookBack.TwoHundredFiftySix**

Possible values are:

|  |  |
| --- | --- |
| MaximumBarsLookBack.TwoHundredFiftySix | Only the last 256 values of the series object will be stored in memory and accessible for reference (improves memory performance) |
| MaximumBarsLookBack.Infinite | Allow full access of the series, but you will then not be able to utilize the benefits of memory optimization |

|  |
| --- |
| **Tip**:  A **MaximumBarsLookBack.TwoHundredFiftySix** series works as a circular ring buffer, which will "loop" when the series reaches full capacity.  Specifically, once there are 256 entries in the series, new data added to the series overwrite the oldest data. |

**Syntax**

MaximumBarsLookBack

**Examples**

| ns **Setting all custom series to use the default MaximumBarsLookBack** |
| --- |
| Series<double> myDoubleSeries = null; Series<string> myStringSeries = null;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "Example Indicator";     // Store all series values instead of only the last 256 values     MaximumBarsLookBack = MaximumBarsLookBack.Infinite;   }   else if (State == State.DataLoaded)   {     // The custom Series<t> below are all constructed using only the NinjaScriptBase object (i.e., "this")     // therefore, the Series<T> MaximumBarsLookBack is taken from the NinjaScript's configured MaximumBarsLookBack property     myDoubleSeries = new Series<double>(this);     myStringSeries = new Series<string>(this);   } } |

| ns **Optimizing custom series to use unique MaximumBarsLookBack behavior** | |
| --- | --- |
| Series<double> myDoubleSeries = null; Series<string> myStringSeries = null;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "Example Indicator";   }   else if (State == State.DataLoaded)   {     // The custom Series<t> below are constructed using MaximumBarsLookBack parameter     // therefore, each Series<t> will use their uniquely specified MaximumBarsLookBack properites     myDoubleSeries = new Series<double>(this, MaximumBarsLookBack.Infinite); // stores all values     myStringSeries = new Series<string>(this, MaximumBarsLookBack.TwoHundredFiftySix); // only the last 256 values (better performance)   } } | |
| **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) >  **IsValidDataPoint()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/getvalueat.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapointat.htm) |

**Definition**

Indicates if the specified input is set at a barsAgo value relative to the current bar.  Please also see the [Reset()](https://ninjatrader.com/es/support/helpGuides/nt8/reset.htm) method for more information.

|  |
| --- |
| **Notes**:  •If called directly from the instance of the NinjaScript object, the value returned corresponds to the Input Series (e.g., Close, High, Low, SMA, etc.)  •When checking a [Bar](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) or [PriceSeries](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm), IsValidDataPoint() returns **true** as long as the barAgo value falls between 0 and the total count for that series.  These are special series which always contain a value set at every slot index for multi-series scripting purposes (e.g., comparing two price series with various session templates, or one series has more ticks than the other)  •For a [Value](https://ninjatrader.com/es/support/helpGuides/nt8/value.htm) series or custom [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm), IsValidPlot() returns **true** or **false** depending on if you have set a value at that index location |

**Method Return Value**

A bool value, when **true** indicates that specified data point is set; otherwise **false**.

**Syntax**

IsValidDataPoint(int barsAgo)

ISeries<T>.IsValidDataPoint(int barsAgo)

|  |
| --- |
| **Warning**:  Calling IsValidDataPoint() will only work a MaximumBarsLookBackInfinite series.  Attempting to check IsValidDataPoint() MaximumBarsLookBack256 series throw an error.  Please check the Log tab of the Control Center. In addition since this method references BarsAgo data, and therefore cannot be used during [OnRender (see note 5)](https://ninjatrader.com/es/support/helpGuides/nt8/onrender.htm).- instead please use the [IsValidDataPointAt](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapointat.htm) during OnRender. |

**Parameters**

|  |  |
| --- | --- |
| barsAgo | An int representing from the current bar the number of historical bars the method will check. |

**Examples**

| ns | |
| --- | --- |
| protected override void OnBarUpdate() {   // only set plot value if hosted indicator is not reset   if(SMA(20).IsValidDataPoint(0))     MyPlot[0] = SMA(20)[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) >  **IsValidDataPointAt()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapoint.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/maximumbarslookback.htm) |

**Definition**

Indicates if the specified input is set at a specified bar index value.  Please also see the [Reset()](https://ninjatrader.com/es/support/helpGuides/nt8/reset.htm) method for more information.

|  |
| --- |
| **Notes**:  •If called directly from the instance of the NinjaScript object, the value returned corresponds to the Inputs Series (e.g., Close, High, Low, SMA, etc.)  •When checking a [Bar](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) or [PriceSeries](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm), IsValidDataPoint() returns **true** as long as the barIndex value falls between 0 and the total count for that series.  These are special series which always contain a value set at every slot index for multi-series scripting purposes (e.g., comparing two price series with various session templates, or one series has more ticks than the other)  •For a [Value](https://ninjatrader.com/es/support/helpGuides/nt8/value.htm) series or custom [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm), IsValidPlot() returns **true** or **false** depending on if you have set a value at that index location |

**Method Return Value**

A bool value, when **true** indicates that specified data point is set; otherwise **false**.

|  |
| --- |
| **Warning**:  Calling IsValidDataPointAt() will only work a MaximumBarsLookBackInfinite series.  Attempting to check IsValidDataPointAt() MaximumBarsLookBack256 series throw an error.  Please check the Log tab of the Control Center |

**Syntax**

IsValidDataPointAt(int barIndex)

ISeries<T>.IsValidDataPointAt(int barIndex)

**Parameters**

|  |  |
| --- | --- |
| barIndex | An int representing an absolute bar index value |

**Examples**

| ns |
| --- |
| protected override void OnBarUpdate() {   // only set plot value if hosted indicator is not reset   if(SMA(20).IsValidDataPointAt(CurrentBar))     MyPlot[0] = SMA(20)[0];     } |