|  |  |
| --- | --- |
| **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) >  **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) >  **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) > [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]);         } |

//

// 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) >  **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) > [Tips](https://ninjatrader.com/es/support/helpGuides/nt8/tips.htm) >  **Using [] brackets** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/user_definable_color_inputs.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/tips.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) |

In C#, square brackets represent a way to access values stored within an collection. NinjaScript comes with quite a few collections that we call ISeries objects which could be accessed with square brackets. [ISeries objects](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) are linked to the underlying bars series in that they hold the same number of values as the number of bars on a chart. For example, to get the close price one bar ago, you would use Close[1] since the value of 1 within the square brackets represents the number of bars ago whose value you wish to reference. As another example, to get the high three bars ago, you would use High[3].

| ns |
| --- |
| double close1 = Close[1]; // gets the close price one bar ago  double high3 = High[3]; // gets the high of three bars ago  double low = Low; // results in compile error. Low is an array, and can't be accessed directly. It should be Low[n Bars ago]. |

Many of NinjaTrader's indicators store their values in Series as well, generally in a Plot. Plots are essentially a Series<double> object and to retrieve values from them you need to specify which value you want to access. In most cases, you'd like the current value, so you could use SMA(14)[0], not just SMA(14). SMA(14) is the Indicator its self or Series, and you can't access its values by calling it directly. Using SMA(14)[0] retrieves the part of the Series you're interested in--the most current value.

| ns |
| --- |
| double SMA\_current = SMA(14)[0]; // gets the current value of the SMA  double SMA\_1 = SMA(14)[1]; // gets the SMA value one bar ago  double SMA\_value = SMA(14); // results in compile error. SMA(14) is a Series and the variable SMA\_value of type double can't hold a Series. |

Most of the time, you need an index value (number in the square brackets), but there are also cases when you need to use the ISeries instead. CrossAbove() and CrossBelow() are two key examples. If you look at the reference page for CrossAbove(), the two method signatures (overloads) look like this:

| ns |
| --- |
| CrossAbove(ISeries<double> series1, ISeries<double> series2, int lookBackPeriod)  CrossAbove(ISeries<double> series1, double value, int lookBackPeriod) |

This means the first variable must always be a ISeries<double> object, and the second variable can be either another ISeries<double> or a double value (100, 70.25, etc). To specify a ISeries<double> object, you can just leave off the square brackets. For example if(CrossAbove(SMA(14), SMA(28), 1)) checks if the 14 period SMA has crossed above the 28 period SMA within the last bar. if(CrossAbove(SMA(14)[0], SMA(28)[0], 1)) would give you a compile error because it expects a ISeries<double> as input, not a double value (which is returned when an index is present).

| ns | |
| --- | --- |
| if (CrossAbove(SMA(14), SMA(28), 1)) // works fine  if (CrossAbove(SMA(14), 1000, 1)) // works fine, this uses a double for the second argument. See the above overload.  if (CrossAbove(SMA(14)[0], SMA(28)[0], 1)) // compile error: SMA(14)[0] is a double, not a ISeries<double>  if (CrossAbove(SMA(14), SMA(28)[0], 1)) // would work fine with a ISeries<double> as first argument and a double as the second argument | |
| **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>** | | [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>** | [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 |