|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **NinjaScript Lifecycle** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_3rd_party_indicators.htm) |

NinjaTrader uses a [State](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) change system to represent various life cycles of your NinjaScript object.  For more basic indicators and strategies, simply understanding each **State** described on the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) page is sufficient.  However, for more advanced development projects, it is critical to understand how NinjaTrader calls these states for various instances throughout the lifetime of the entire application.

**When NinjaTrader instantiates a NinjaScript object**

There are two categories of instances instantiated by NinjaTrader:

•"UI" instances representing its default properties on various user interfaces

•The "configured" instance executing your custom instructions

In both categories, [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) is called at least twice:  once to **State.SetDefaults** acquiring various default property values, and then again to **State.Terminated** handling internal references cleanup.

|  |
| --- |
| **Note**:  It is important to understand that previous major versions of NinjaTrader were not so diligent in running termination logic for UI instances and the current major NinjaTrader 8 version has been changed to help properly address related issues. |

To elaborate on that process, imagine the sequence of user events required to start an indicator on a chart:

1.User right clicks on a Chart and select "**Indicator**"

2.User adds an Indicator from the **Available** list

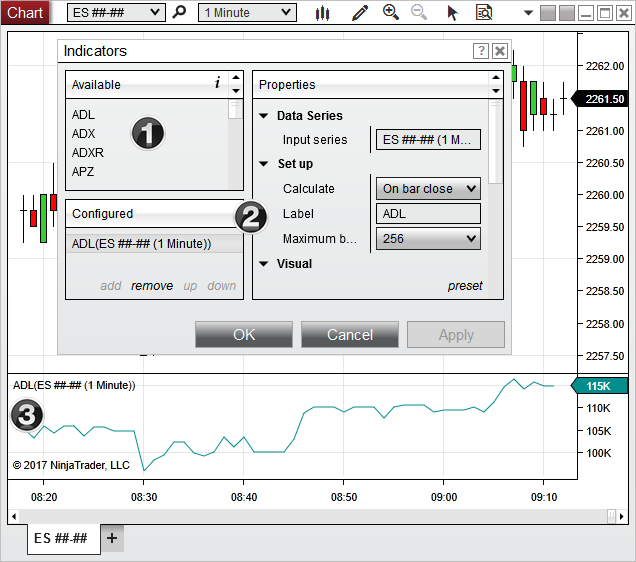
3.User configures desired **Properties** and presses "**Apply**" or "**OK**"

During this sequence, there are actually 3 instances of the same indicator created by NinjaTrader:

1.The instance displaying the **Name** property to the list of "**Available**" indicators (**Note**: this process involves creating an instance of *all* indicators in order to build the complete list)

2.The instance displaying the individual **Name** and its default **Properties**

3.The instance configured and executing on the chart



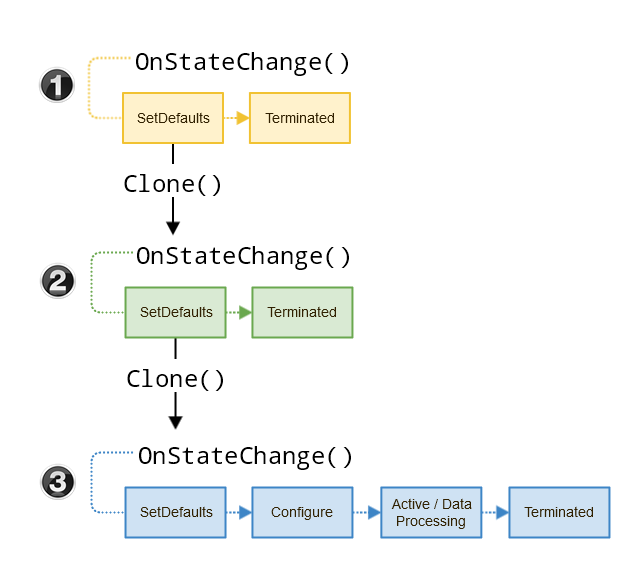
To visualize how each instance goes through its **States**, please consider the logic and flow chart below:

1.In order to display the indicator name in the list of **"Available"** indicators, the NinjaTrader core must find the **Name** of each installed indicator defined in their **SetDefaults**.  This occurs simultaneously for *every indicator installed on the system* in order to build the full list of available indicators.

2.The selected indicator is then [cloned](https://ninjatrader.com/es/support/helpGuides/nt8/clone.htm) and **SetDefaults** is called again in order to display the default properties to the "**Properties**" grid.  This only occurs for the individual indicator.

3.After the user has set their desired property settings and press **OK** or **Apply**, the indicator is once again cloned and runs through its full state management.  This only occurs for the indicator configured to execute on the chart.

|  |
| --- |
| **Warning**:  Since NinjaTrader is multi-threaded, it is possible the **OnStateChange()** logic will be operating on a different thread than your indicator instances.  Due to this fact, if logic in your **OnStateChange()** method is thread sensitivity (e.g., dependent on UI threads vs Instrument threads) please make sure to read the section on [multi-threading considerations](https://ninjatrader.com/es/support/helpGuides/nt8/multi-threading.htm) and check for thread access in your **OnStateChange()** logic |



It is the 3rd "configured" instance you are concerned with developing, but you should also be aware of the "UI" instances which are triggered at various stages of NinjaTrader.

|  |
| --- |
| **Notes**:  1.The example above is written for an indicator, but the same concept of state management applies to every NinjaScript object type  2.The UI instances do not reach **State.Terminated** until the user closes out of the UI feature displaying the object  3.Since [AddOns](https://ninjatrader.com/es/support/helpGuides/nt8/addon_development_overview.htm) run in the background and are not dependent on UI elements, they will run through their **SetDefaults**/**Terminated** states after each NinjaScript compile and startup/shutdown of NinjaTrader.  4.The configured instance will also be cloned back to UI instances during various user actions (e.g, re-opening an indicator dialog to reconfigure settings, or user copying & pasting the indicator to a new panel or chart).  Therefore you should not assume that objects (such as ChartControl) will not be accessible in the UI instances.  5.In some extreme scenarios, you may need to execute custom logic before or after an object is cloned.  Overriding the default behavior can be done via the virtual [Clone()](https://ninjatrader.com/es/support/helpGuides/nt8/clone.htm) method |

**What does this mean for me?**

Since **OnStateChange()** can be called at various times throughout NinjaTrader, you should be diligent in handling each state and managing resources only when it is appropriate that your NinjaScript object was actually configured:

•**State.SetDefaults** should be kept as lean as possible to prevent logic from processing superfluously and causing problems unrelated to the configured instance.  Only properties which need to be displayed on the UI should be set in this state.

•Resources should only be set up once an object has reached **State.Configure** or **State.DataLoaded** (see [best practices](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript_best_practices.htm) for more information)

•**State.Terminated** logic should be specific in when it resets a value or destroys a resource.  Since the running instance can be cloned back to a UI instance, checking that a mutable property exists before accessing sometimes is not enough.  You may need to consider adding a flag to help decide when a resource needs to be reset or destroyed.

**Example**

Let’s say your object was an indicator looking to add a custom toolbar element to the chart, and when the indicator is removed from the chart, you would want to make sure your toolbar elements are also properly removed.  In the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) handler you change could add the custom toolbar once the **State** has reached **State.Historical**, and remove the toolbar once the State has reached **State.Terminated**.

To ensure that the remove logic only runs in instances that were actually configured, you can see we in the example below we also track that the toolbar needs a reset in **State.Terminated** state via a custom bool variable.  In other words, the proper reset request comes from our configured instance and would be ignored if the **State.Terminated** is called from outside our object (i.e., a UI instance). This will prepare our object to properly handle both situations in which **State.Terminated** could be called in the NinjaTrader state management system.

| ns |
| --- |
| // custom flag to help time termination logic private bool toolBarNeedsReset = false;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "State lifetime indicator";   }   else if (State == State.Historical)   {     // before indicator starts historical processing     // add a custom tool bar using a custom method     AddToolBarButton(); // this is a pseudo-method for example purposes     toolBarNeedsReset = true; // use a flag to track this logic was executed   }     else if (State == State.Terminated)   {     // here we intend to remove the custom tool bar when the indicator shuts down     if (toolBarNeedsReset) // flag is only true after actually added         RemoveToolBarButton();   } } |

**Cloning NinjaScript**

Clone is the operation of iterating over all public browsable properties on a NinjaScript object and duplicating the values over to a freshly generated instance. For the majority of NinjaScript with standard properties the clone process is transparent to you and you do not need to be concerned the the clone process. For those of you that want more control or will be utilizing complex properties then knowledge about clone is essential. Cloning is performed in 2 primary use cases:

1.Configuring an instance in an object dialog and then cloning the configured data to an actual NinjaScript instance applied for example to a Chart. (Configuration then Run)

2.When triggering 'Reload NinjaScript' or "Reload All Historical Data'

NinjaScript objects have a base clone method implemented which will iterating over all browsable properties and copy by value to the next instance. The rules follow the 'clone' rules described in the clone documentation located [here](https://ninjatrader.com/es/support/helpGuides/nt8/clone.htm) and described above. The default behavior will work in almost all cases except for when you have some complex custom property which needs specific clone behavior. In which case we allow the ability to override Clone() and specify your own behavior.

|  |
| --- |
| **Note**: If you plan to utilize complex class properties on NinjaScript, you can specify your own clone method. However when NinjaScript is compiled in NinjaTrader a new DLL holding the compiled IL code is 'hot-loaded' into NinjaTrader. As a user or developer would try to reload NinjaScript or configure an existing NinjaScript object, any complex class will not resolve since the class will be residing in two different assemblies. This problem cannot be solved with custom clone method and workarounds for this are setting Browsable(false) attribute on that property so it is not cloned or putting the property it its own dedicated assembly. |

**Saving NinjaScript Properties to the Workspace via XML Serialization**

XML Serialization comes into play when you have a set of properties and want those properties to persist the user saved workspace (or any templates that are user created).

By default basic types such as int, string, bool will all serialize without issue, if you have a complex property you want its setting maintained on restore then you need to create a string serialized representation of that class. The technique is shown in this example post [here](https://ninjatrader.com/es/support/helpGuides/nt8/user_definable_color_inputs.htm) where we show how to serialize a color brush.

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

**Definition**

An event driven method which is called whenever the script enters a new [State](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm). The **OnStateChange()** method can be used to configure script properties, create one-time behavior when going from historical to real-time, as well as manage clean up resources on termination.

|  |
| --- |
| **Notes**:  •Viewing any UI element which lists NinjaScript classes (such as the Indicators or Strategies window, a chart's Chart Style dropdown menu, etc.) will initialize all classes of that Type when it is opened, which causes each script to enter **State.SetDefaults**, even if it is not actively configured or running in any window. It is important to keep this in mind when adding logic within **State.SetDefaults** in **OnStateChange()**, as this logic will be processed each time the script is initialized. For example, opening the Indicators window will trigger **State.SetDefaults** for all indicators in the system, and closing the Indicators window will trigger**State.Terminated** for all Indicators. In addition, disconnecting or connecting to a data provider can cause State transitions for any currently active scripts. Further discussion of this aspect of the state change model can be found via [*Understanding the lifecycle of your NinjaScript objects*](https://ninjatrader.com/es/support/helpGuides/nt8/understanding_the_lifecycle_of.htm).  •When an indicator is configured on a chart while a Compile is taking place in the NinjaScript Editor, it can appear that the script passes through **State.Terminated**. However, this is the result of a copy of the script being initialized at compile-time, NOT the result of the indicator on the chart being disabled and re-initialized. |

**Related Methods and Properties**

|  |  |
| --- | --- |
| [SetState()](https://ninjatrader.com/es/support/helpGuides/nt8/setstate.htm) | Method is used for changing the State of any running NinjaScript object. |
| [State](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) | Represents the current progression of the object as it advances from setup, processing data, to termination. |

**Method Return Value**

This method does not return a value.

**Syntax**  
See example below. The NinjaScript wizards automatically generate the method syntax for you.

Possible states are:

|  |  |  |
| --- | --- | --- |
| **State Name** | **This state is called when** | **This state is where you should** |
| State.SetDefaults | **SetDefaults** is always called when displaying objects in a UI list such as the Indicators dialogue window since temporary objects are created for the purpose of UI display | •Keep as lean as possible  •Set default values (pushed to UI) |
| State.Configure | **Configure** is called after a user adds an object to the applied list of objects and presses the OK or Apply button.  This state is called only once for the life of the object. | •Add additional data series via [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm)  •Declare custom resources |
| State.Active | **Active** is called once after the object is configured and is ready to process data (DrawingTools could see multiple calls as internally an object for hit testing is cloned) | •Used for objects such as [Share Service](https://ninjatrader.com/es/support/helpGuides/nt8/share_service.htm) which do not process price series data  •Indicate the object is ready to being processing information |
| State.DataLoaded | **DataLoaded** is called only once after all data series have been loaded. | •Use for logic that needs to access data related objects like Bars, Instruments, BarsPeriod, TradingHours or instantiating indicators  •Notify that all data series have been loaded  •Initialize any class level variables (including custom [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) objects) |
| State.Historical | **Historical**is called once the object begins to process historical data. This state is called once when running an object in real-time. This object is called multiple times when running a backtest optimization and the property [IsInstantiatedOnEachOptimizationIteration](https://ninjatrader.com/es/support/helpGuides/nt8/isinstantiatedoneachoptimizationiteration.htm) is true (default behavior) | •Notify that the object is processing historical data |
| State.Transition | **Transition** is called once as the object has finished processing historical data but before it starts to process realtime data. | •Notify that the indicator or strategy is is transitioning to realtime data  •Prepare realtime related resources |
| State.Realtime | **Realtime** is called once when the object begins to process realtime data. | •Notify that the indicator or strategy is processing realtime data  •Execute realtime related logic |
| State.Terminated | **Terminated** is called once when the object terminates. | •Notify the object is shutting down  •Use to clean up/dispose of resources |

**Active States vs Data Processing States**

After **State.Configure,**each type of NinjaScript type has its own state management system which can be classified under two categories:

•**Active state:**  State.Active

•**Data Processing states:**State.DataLoaded, State.Historical, State.Transition, State.Realtime

The table below lists each NinjaScript type and it's designed state management system:

|  |  |
| --- | --- |
| **NinjaScript Type** | **State Management System** |
| AddOns\* | Active state |
| BarTypes | Active state |
| ChartStyles | Active state |
| DrawingTools | Active state |
| Indicators | Data Processing states |
| ImportTypes | Active state |
| Market Analyzer Columns | Data Processing states |
| OptimizationFitnesses | Active state |
| Optimizers | Active state |
| PerformanceMetrics | Active state |
| ShareServices | Active state |
| Strategies | Data Processing states |
| SuperDOM Columns | Active state |

|  |
| --- |
| **Tips:**  •Resources created in **State.Configure** and not disposed of immediately will be kept and utilized if the NinjaScript object resides in grids (e.g. Strategy tab on Control Center), even if it is not enabled. Try to create resources in**State.Historical** or **State.DataLoaded** instead, if possible.  •**State.Historical** is called multiple times when running a backtest [optimization](https://ninjatrader.com/es/support/helpGuides/nt8/optimize_a_strategy.htm) on a strategy and the property "[IsInstantiatedOnEachOptimizationIteration](https://ninjatrader.com/es/support/helpGuides/nt8/isinstantiatedoneachoptimizationiteration.htm)" is **true** (default behavior).  • Scripts that require [Calculate](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm) to be set by the developer must set this property in **State.Historical** in order to ensure that if this script is a child (hosted) that the parent.Calculate property which is adopted by the child is overridden again.  •When instantiating indicators in a [Multi-Series script](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) in OnStateChange, the input any hosted indicator is running on should be explicitly stated (since a specific [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) is not guaranteed) |

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if (State == State.SetDefaults)   {     // Calculate once at the end of every single bar     Calculate = Calculate.OnBarClose;         // Add two plots     AddPlot(Brushes.Blue, "Upper"));     AddPlot(Brushes.Orange, "Lower"));   }     else if (State == State.Configure)   {     // Adds a 5-minute Bars object to the strategy and is automatically assigned     // a Bars object index of 1 since the primary data the strategy is run against     // set by the UI takes the index of 0.             AddDataSeries("AAPL", BarsPeriodType.Minute, 5);       } } | |
| **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) >  **OnStateChange()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/marketdeptheventargs.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/setstate.htm) |

**Definition**

An event driven method which is called whenever the script enters a new [State](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm). The **OnStateChange()** method can be used to configure script properties, create one-time behavior when going from historical to real-time, as well as manage clean up resources on termination.

|  |
| --- |
| **Notes**:  •Viewing any UI element which lists NinjaScript classes (such as the Indicators or Strategies window, a chart's Chart Style dropdown menu, etc.) will initialize all classes of that Type when it is opened, which causes each script to enter **State.SetDefaults**, even if it is not actively configured or running in any window. It is important to keep this in mind when adding logic within **State.SetDefaults** in **OnStateChange()**, as this logic will be processed each time the script is initialized. For example, opening the Indicators window will trigger **State.SetDefaults** for all indicators in the system, and closing the Indicators window will trigger**State.Terminated** for all Indicators. In addition, disconnecting or connecting to a data provider can cause State transitions for any currently active scripts. Further discussion of this aspect of the state change model can be found via [*Understanding the lifecycle of your NinjaScript objects*](https://ninjatrader.com/es/support/helpGuides/nt8/understanding_the_lifecycle_of.htm).  •When an indicator is configured on a chart while a Compile is taking place in the NinjaScript Editor, it can appear that the script passes through **State.Terminated**. However, this is the result of a copy of the script being initialized at compile-time, NOT the result of the indicator on the chart being disabled and re-initialized. |

**Related Methods and Properties**

|  |  |
| --- | --- |
| [SetState()](https://ninjatrader.com/es/support/helpGuides/nt8/setstate.htm) | Method is used for changing the State of any running NinjaScript object. |
| [State](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) | Represents the current progression of the object as it advances from setup, processing data, to termination. |

**Method Return Value**

This method does not return a value.

**Syntax**  
See example below. The NinjaScript wizards automatically generate the method syntax for you.

Possible states are:

|  |  |  |
| --- | --- | --- |
| **State Name** | **This state is called when** | **This state is where you should** |
| State.SetDefaults | **SetDefaults** is always called when displaying objects in a UI list such as the Indicators dialogue window since temporary objects are created for the purpose of UI display | •Keep as lean as possible  •Set default values (pushed to UI) |
| State.Configure | **Configure** is called after a user adds an object to the applied list of objects and presses the OK or Apply button.  This state is called only once for the life of the object. | •Add additional data series via [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm)  •Declare custom resources |
| State.Active | **Active** is called once after the object is configured and is ready to process data (DrawingTools could see multiple calls as internally an object for hit testing is cloned) | •Used for objects such as [Share Service](https://ninjatrader.com/es/support/helpGuides/nt8/share_service.htm) which do not process price series data  •Indicate the object is ready to being processing information |
| State.DataLoaded | **DataLoaded** is called only once after all data series have been loaded. | •Use for logic that needs to access data related objects like Bars, Instruments, BarsPeriod, TradingHours or instantiating indicators  •Notify that all data series have been loaded  •Initialize any class level variables (including custom [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) objects) |
| State.Historical | **Historical**is called once the object begins to process historical data. This state is called once when running an object in real-time. This object is called multiple times when running a backtest optimization and the property [IsInstantiatedOnEachOptimizationIteration](https://ninjatrader.com/es/support/helpGuides/nt8/isinstantiatedoneachoptimizationiteration.htm) is true (default behavior) | •Notify that the object is processing historical data |
| State.Transition | **Transition** is called once as the object has finished processing historical data but before it starts to process realtime data. | •Notify that the indicator or strategy is is transitioning to realtime data  •Prepare realtime related resources |
| State.Realtime | **Realtime** is called once when the object begins to process realtime data. | •Notify that the indicator or strategy is processing realtime data  •Execute realtime related logic |
| State.Terminated | **Terminated** is called once when the object terminates. | •Notify the object is shutting down  •Use to clean up/dispose of resources |

**Active States vs Data Processing States**

After **State.Configure,**each type of NinjaScript type has its own state management system which can be classified under two categories:

•**Active state:**  State.Active

•**Data Processing states:**State.DataLoaded, State.Historical, State.Transition, State.Realtime

The table below lists each NinjaScript type and it's designed state management system:

|  |  |
| --- | --- |
| **NinjaScript Type** | **State Management System** |
| AddOns\* | Active state |
| BarTypes | Active state |
| ChartStyles | Active state |
| DrawingTools | Active state |
| Indicators | Data Processing states |
| ImportTypes | Active state |
| Market Analyzer Columns | Data Processing states |
| OptimizationFitnesses | Active state |
| Optimizers | Active state |
| PerformanceMetrics | Active state |
| ShareServices | Active state |
| Strategies | Data Processing states |
| SuperDOM Columns | Active state |

|  |
| --- |
| **Tips:**  •Resources created in **State.Configure** and not disposed of immediately will be kept and utilized if the NinjaScript object resides in grids (e.g. Strategy tab on Control Center), even if it is not enabled. Try to create resources in**State.Historical** or **State.DataLoaded** instead, if possible.  •**State.Historical** is called multiple times when running a backtest [optimization](https://ninjatrader.com/es/support/helpGuides/nt8/optimize_a_strategy.htm) on a strategy and the property "[IsInstantiatedOnEachOptimizationIteration](https://ninjatrader.com/es/support/helpGuides/nt8/isinstantiatedoneachoptimizationiteration.htm)" is **true** (default behavior).  • Scripts that require [Calculate](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm) to be set by the developer must set this property in **State.Historical** in order to ensure that if this script is a child (hosted) that the parent.Calculate property which is adopted by the child is overridden again.  •When instantiating indicators in a [Multi-Series script](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) in OnStateChange, the input any hosted indicator is running on should be explicitly stated (since a specific [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) is not guaranteed) |

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if (State == State.SetDefaults)   {     // Calculate once at the end of every single bar     Calculate = Calculate.OnBarClose;         // Add two plots     AddPlot(Brushes.Blue, "Upper"));     AddPlot(Brushes.Orange, "Lower"));   }     else if (State == State.Configure)   {     // Adds a 5-minute Bars object to the strategy and is automatically assigned     // a Bars object index of 1 since the primary data the strategy is run against     // set by the UI takes the index of 0.             AddDataSeries("AAPL", BarsPeriodType.Minute, 5);       } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **Calculate** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/barsperiod.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/count.htm) |

**Definition**

Determines how often [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) is called for each bar. **OnBarClose** means once at the close of the bar. **OnEachTick** means on every single tick. **OnPriceChange** means once for each price change. If there were two ticks in a row with the same price, the second tick would not trigger OnBarUpdate(). This can improve performance if calculations are only needed when new values are possible.

|  |
| --- |
| **Notes**:  1.On a historical data set, only the OHLCVT of the bar is known and not each tick that made up the bar.  As a result, [State.Historical](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) data processes **OnBarUpdate()** only on the close of each historical bar even if this property is set to **OnEachTick** or **OnPriceChange**.  You can use [TickReplay](https://ninjatrader.com/es/support/helpGuides/nt8/tick_replay.htm) or a [Multi-time frame script](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) to obtain intrabar data.  2.When set to Calculate **OnPriceChange**, the **OnBarUpdate()** method is **ONLY** called when the price has changed which does not necessarily occur the end of the close of the bar |

**Property Value**

An enum value determining the how frequently OnBarUpdate() will be called.  Default value is set to Calculate.OnBarClose

|  |
| --- |
| **Warning**:  If your script relies on volume updates **OnPriceChange** should **NOT** be used since it can potentially miss volume updates if they occur at the same price |

**Syntax**

Calculate.OnBarClose

Calculate.OnEachTick

Calculate.OnPriceChange

|  |
| --- |
| **Tips**  1.Calculating indicators or systems for each incoming tick can be CPU intensive. Only calculate indicators on each incoming tick if you have a requirement to calculate it intra-bar.  2.For an example of how to separate some logic to be Calculate = Calculate.OnBarClose and other logic to be .OnEachTick please see this [reference sample](http://www.ninjatrader.com/support/forum/showthread.php?t=19387).  3.Embedded scripts within a calling parent script should not use a different Calculate property since it is already utilizing the Calculate property of the parent script (i.e. the strategy your indicator is called from).  4. Typically this property would be set in **State.SetDefaults**, however scripts that require Calculate to be set by the developer must set this property in **State.Historical** in its [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) in order to ensure that if this script is a child (hosted) that the parent.Calculate property which is adopted by the child is overridden again. |

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {     if (State == State.SetDefaults)     {         // Calculate on the close of each bar         Calculate = Calculate.OnBarClose;     } } | |
| **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) > [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) >  **State** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/setstate.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/sessioniterator.htm) |

**Definition**

Represents the current progression of the object as it advances from setup, processing data, to termination.  These states can be used for setting up or declaring various resources and properties.

|  |
| --- |
| **Note**:  More detailed explanation of various states along with examples can be found in the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method section of this help guide.  You can also attempt to set a new **State** using the [SetState()](https://ninjatrader.com/es/support/helpGuides/nt8/setstate.htm) method. |

**Property Value**

An enum value representing the current state of the object.  Possible values are:

|  |  |
| --- | --- |
| SetDefaults | Default values are set (pushed to UI). |
| Configure | User the presses the OK or Apply button. |
| Active | Object is configured and is ready to receive instructions |
| DataLoaded | All data series have been loaded |
| Historical | Begins to process historical data |
| Transition | Finished processing historical data |
| Realtime | Begins to process realtime data. |
| Terminated | Begins to shut down |

**Syntax**

State

**Examples**

| ns **Understanding the sequence of States** |
| --- |
| protected override void OnStateChange() {           Print(DateTime.Now + ": Current State is State."+State); } |

|  |
| --- |
| ns **Using State to only process real-time data** |
| protected override void OnBarUpdate() {   // only process real-time OnBarUpdate events   if (State == State.Historical)     return;         //rest of logic           } |

|  |  |
| --- | --- |
| **Navigation:**  [Operaciones](https://ninjatrader.com/es/support/helpGuides/nt8/operations.htm) > [Gráficos](https://ninjatrader.com/es/support/helpGuides/nt8/charts.htm) >  **Tick Replay** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/order_flow_market_depth_map.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/charts.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/power_volume_indicators.htm) |

**¿Qué es la repetición de tick?**

**La repetición** de marca es una propiedad que se puede habilitar opcionalmente en los indicadores y estrategias de NinjaScript que asegurarán que los datos del mercado (oferta / demanda / última) que se introdujeron en la construcción de una barra se carguen en la secuencia exacta de eventos de datos del mercado.  Esto garantiza que sus indicadores y estrategias se calculen históricamente tic por tic exactamente como lo hubieran sido si el indicador / estrategia se ejecutara en vivo durante un período. **La repetición de marca** puede habilitarse para los indicadores utilizados en **Gráficos** , **Analizadores de mercado** y **Estrategias** .

|  |
| --- |
| **Advertencia:**Es importante tener en cuenta que esta propiedad implica que se utilizan más recursos de PC para calcular sus indicadores y estrategias y, como resultado, tendrá un impacto en el rendimiento. La configuración de **repetición de marca** solo debe reservarse para indicadores y estrategias que realmente se beneficiarían de los recursos adicionales dedicados a llegar a estos cálculos.    Por ejemplo, un indicador Pivot simple que solo usa los niveles de precios diarios actuales y anteriores no vería ninguna ventaja al usar la **repetición de ticks** .  Por el contrario, un indicador de perfil de volumen que se basa en la secuencia exacta de operaciones para calcular varios niveles se beneficiaría enormemente con el uso de la **repetición de ticks** . |

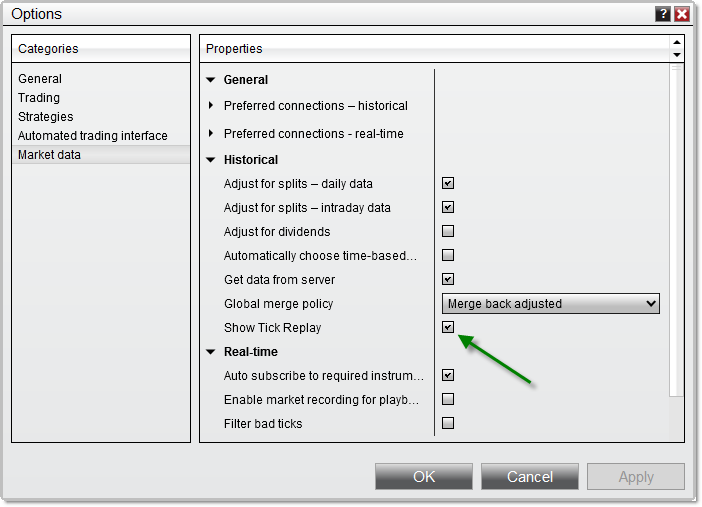
|  |
| --- |
| **Nota**: Tick Replay no está destinado a funcionar en las pruebas de estrategia de NinjaScript, y no proporcionará los mismos resultados que ejecutar una estrategia en datos en vivo con Tick Replay habilitado. Para una mayor resolución de llenado de pedidos en backtests de estrategia, puede usar la Resolución de relleno alta en el Analizador de estrategia. |

Los indicadores y las estrategias solo podrán aprovechar la **repetición** de ticks si se han programado explícitamente para calcular estos eventos de datos de mercado.  Si usted es un programador y le gustaría aprender cómo utilizar Tick Replay con las secuencias de comandos personalizados, consulte la [repetición de garrapatas utilizando](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) la sección de nuestra guía de ayuda NinjaScript.

**Configuración de repetición de marca**

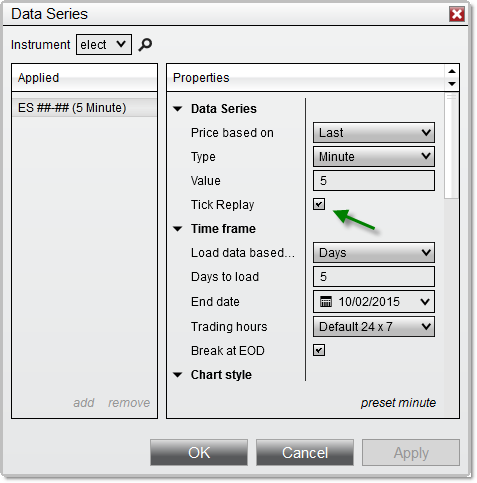
Por defecto, la repetición de marca no estará habilitada.  Para exponer esta propiedad para sus indicadores y estrategias, primero deberá activar la opción de repetición de marca global:

•Vaya al menú **Centro de control** > **Herramientas** > **Opciones** y, en la categoría **Datos de mercado** , marque " **Mostrar reproducción de marca"**



Una vez que se haya habilitado la opción " **Mostrar reproducción de tick** " en la categoría **Datos de mercado** del menú [Opciones](https://ninjatrader.com/es/support/helpGuides/nt8/options.htm), encontrará una opción de " **Reproducción de tick** " que puede seleccionar al configurar sus indicadores o estrategias, o al ejecutar una estrategia en la **Estrategia analizador** .

|  |
| --- |
| **Nota**:  Los [tipos de barra](https://ninjatrader.com/es/support/helpGuides/nt8/bar_types.htm) del sistema "Salto de línea" y "Renko" no se pueden usar con **Tick Replay** y, como resultado, la opción **Tick Replay** se desactivará cuando se configure con esos tipos de barra.  Puede haber otros tipos de barras de terceros que también pueden deshabilitar **Tick Replay** por diseño.  Si es desarrollador, consulte la propiedad [IsRemoveLastBarSupported](https://ninjatrader.com/es/support/helpGuides/nt8/isremovelastbarsupported.htm) para obtener más información. |



|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Multi-Time Frame & Instruments** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/multi-threading.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/understanding_the_lifecycle_of.htm) |

**Multi-Series Scripting Overview**

NinjaScript supports multiple time frames and instruments in a single script. This is possible because you can add additional Bars objects to indicators or strategies, in addition to the primary Bars object to which they are applied. A Bars object represents all of the bars of data on a chart. For example, if you had a MSFT 1 minute chart with 200 bars on it, the 200 bars represent one Bars object. In addition to adding Bars objects for reference or for use with indicator methods, you can execute trades across all the different instruments in a script. There is extreme flexibility in the NinjaScript model that NinjaTrader uses for multiple-bars scripts, so it is very important that you understand how it all works before you incorporate additional Bars objects in a script. An important fact to understand is that NinjaScript is truly event driven; every Bars object in a script will call the [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) method. The significance of this will become evident throughout this page.

|  |
| --- |
| **Note**:  If using [OnMarketData()](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same).  The market data subscription behavior occurs both in real-time and during [TickReplay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) historical |

It is also important that you understand the following method and properties:

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

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

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

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

|  |
| --- |
| **Note**:  As we move through this section, the term "Primary Bars" will be used and for the purpose of clarification, this will always refer to the first Bars object loaded into a script. For example, if you apply a script on MSFT 1 minute chart, the primary Bars would be MSFT 1 minute data set.    **This section is written in sequential fashion. Example code is re-used and built upon from sub section to sub section.** |

tog_minus        [Working With Multi-Time Frame Objects](javascript:HMToggle('toggle','WorkingWithMultiTimeFrameObjects','WorkingWithMultiTimeFrameObjects_ICON'))

tog_minus        [Adding Additional Bars Objects to NinjaScript](javascript:HMToggle('toggle','AddingAdditionalBarsObjectToninjascript','AddingAdditionalBarsObjectToninjascript_ICON'))

tog_minus        [Creating Series<T> Objects](javascript:HMToggle('toggle','CreatingSeriesObjects','CreatingSeriesObjects_ICON'))

tog_minus        [How Bars Data is Referenced](javascript:HMToggle('toggle','HowBarsDataIsReferenced','HowBarsDataIsReferenced_ICON'))

tog_minus        [Using Bars Objects as Input to Indicator Methods](javascript:HMToggle('toggle','UsingBarsObjectsAsInputToIndicatorMethods','UsingBarsObjectsAsInputToIndicatorMethods_ICON'))

tog_minus        [True Event Driven OnBarUpdate() Method](javascript:HMToggle('toggle','TrueEventDrivenOnbarupdateMethod','TrueEventDrivenOnbarupdateMethod_ICON'))

tog_minus        [Accessing the Price Data in a Multi-Bars NinjaScript](javascript:HMToggle('toggle','AccessingThePriceDataInAMultibarsninjascript','AccessingThePriceDataInAMultibarsninjascript_ICON'))

tog_minus        [Entering, Exiting and Retrieving Position Information](javascript:HMToggle('toggle','EnteringExitingAndRetrievingPositionInformation','EnteringExitingAndRetrievingPositionInformation_ICON'))

|  |  |
| --- | --- |
| **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()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) >  **AddRenko()** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/addpointandfigure.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/addvolumetric.htm) |

**Definition**

Similar to the [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) method for adding Bars objects, this method adds a Renko Bars object for multi-series NinjaScript.

|  |
| --- |
| **Notes**:  1.When running NinjaScript, you will be able to choose the first instrument and bar interval to run on. This first Bars object will carry a [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) index of 0.  2. In a multi-time frame and multi-instrument NinjaScript, supplementary Bars objects are added via this method in State.Configure state of the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method and given an incremented BarsInProgress index value. See additional information on running [multi-bars scripts](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm).  3.The [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) property can be used to filter updates between different bars series  4.If using [OnMarketData()](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same).  The market data subscription behavior occurs both in real-time and during [TickReplay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) historical  5.For adding regular Bars types please use [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm)  6.A**Tick Replay** indicator or strategy **CANNOT**use a **MarketDataType.Ask** or **MarketDataType.Bid**series.  Please see [Developing for Tick Replay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) for more information. |

**Syntax**

AddRenko(string instrumentName, int brickSize, Data.MarketDataType marketDataType)  
AddRenko(string instrumentName, int brickSize, Data.MarketDataType marketDataType, string tradingHoursName)  
AddRenko(string instrumentName, int brickSize, Data.MarketDataType marketDataType, string tradingHoursName, bool?isResetOnNewTradingDay)

|  |
| --- |
| **Warnings:**  •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 **AddRenko()** calls as those hosted scripts would. For further reference, please also review 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 **AddRenko()** 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.** |

**Parameters**

|  |  |
| --- | --- |
| instrumentName | A string determining instrument name such as "MSFT" |
| brickSize | An int determining the size (in ticks) of each bar |
| 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. |

|  |
| --- |
| **Tip**: 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: AddRenko("MSFT Arca", 2, MarketDataType.Last) |

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {     if (State == State.Configure)     {         // Add a 1 minute Renko Bars object for the ES 03-18 contract - BarsInProgress index = 1         AddRenko("ES 03-18", 2, MarketDataType.Last);     } }   protected override void OnBarUpdate() {     // Ignore the primary Bars object and only process the Renko Bars object     if (BarsInProgress == 1)     {         // Do something;     } } | |
| **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()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) >  **AddVolumetric()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/addrenko.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/barsarray.htm) |

**Definition**

Similar to the [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) method for adding Bars objects, this method adds a [Order Flow](https://ninjatrader.com/es/support/helpGuides/nt8/order_flow_volumetric_bars.htm) Volumetric Bars object for multi-series NinjaScript.

|  |
| --- |
| **Notes**:  1.When running NinjaScript, you will be able to choose the first instrument and bar interval to run on. This first Bars object will carry a [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) index of 0.  2. In a multi-time frame and multi-instrument NinjaScript, supplementary Bars objects are added via this method in State.Configure state of the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method and given an incremented BarsInProgress index value. See additional information on running [multi-bars scripts](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm).  3.The [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) property can be used to filter updates between different bars series  4.If using [OnMarketData()](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same).  The market data subscription behavior occurs both in real-time and during [TickReplay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) historical  5.For adding regular Bars types please use [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm)  6.A**Tick Replay** indicator or strategy **CANNOT**use a **MarketDataType.Ask** or **MarketDataType.Bid**series.  Please see [Developing for Tick Replay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) for more information.  7.To access additional Volumetric data points programmtically in your NinjaScript studies, please see the example [here](https://ninjatrader.com/es/support/helpGuides/nt8/order_flow_volumetric_bars2.htm). |

**Syntax**

AddVolumetric(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, Data.VolumetricDeltaType deltaType, int tickPerLevel)  
AddVolumetric(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, Data.VolumetricDeltaType deltaType, int tickPerLevel, bool? isResetOnNewTradingDay)  
AddVolumetric(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, Data.VolumetricDeltaType deltaType, int tickPerLevel, string tradingHoursName, bool? isResetOnNewTradingDay)

AddVolumetric(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, Data.VolumetricDeltaType deltaType, int tickPerLevel, int sizeFilter, string tradingHoursName, bool? isResetOnNewTradingDay) (R17 and higher only)

|  |
| --- |
| **Warnings:**  •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 **AddVolumetric()** calls as those hosted scripts would. For further reference, please also review 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 **AddVolumetric()** 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.** |

**Parameters**

|  |  |
| --- | --- |
| instrumentName | A string determining instrument name such as "MSFT" |
| baseBarsPeriodType | The underlying BarsType used for the Volumetric bars period.    Possible values are:    •BarsPeriodType.Tick  •BarsPeriodType.Volume  •BarsPeriodType.Range  •BarsPeriodType.Second  •BarsPeriodType.Minute  •BarsPeriodType.Day  •BarsPeriodType.Week  •BarsPeriodType.Month  •BarsPeriodType.Year |
| baseBarsPeriodTypeValue | An int determining the underlying period interval such as "3" for 3 minute bars |
| deltaType | The DeltaType used for the Volumetric bars object delta calculations    Possible values are:    •VolumetricDeltaType.BidAsk  •VolumetricDetlaType.UpDownTick |
| ticksPerLevel | An int setting the aggregation of price levels for the Volumetric bar, pass in a 1 to analyze each price level individually |
| sizeFilter | An int setting the trade size allowed to count in the delta calculations |
| 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. |

|  |
| --- |
| **Tip**: 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: AddVolumetric("MSFT Arca", BarsPeriodType.Minute, 1, VolumetricDeltaType.BidAsk, 1); |

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if (State == State.SetDefaults)   {       Name = "Examples Indicator";               }   else if (State == State.Configure)   {       // Add a 1 minute Order Flow Volumetric Bars object for the ES 03-18 contract - BarsInProgress index = 1       AddVolumetric("ES 03-18", BarsPeriodType.Minute, 1, VolumetricDeltaType.BidAsk, 1);   } }   protected override void OnBarUpdate() {     // Ignore the primary Bars object and only process the Order Flow Volumetric object     if (BarsInProgress == 1)     {         // Do something;     } } | |
| **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()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) >  **AddLineBreak()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/addkagi.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/addpointandfigure.htm) |

**Definition**

Similar to the [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) method for adding Bars objects, this method adds a Line Break Bars object for multi-series NinjaScript.

|  |
| --- |
| **Notes**:  1.When running NinjaScript, you will be able to choose the first instrument and bar interval to run on. This first Bars object will carry a [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) index of 0.  2. In a multi-time frame and multi-instrument NinjaScript, supplementary Bars objects are added via this method in State.Configure state of the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method and given an incremented BarsInProgress index value. See additional information on running [multi-bars scripts](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm).  3.The [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) property can be used to filter updates between different bars series  4.If using [OnMarketData()](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same).  The market data subscription behavior occurs both in real-time and during [TickReplay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) historical  5.For adding regular Bars types please use [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm)  6.A**Tick Replay** indicator or strategy **CANNOT**use a **MarketDataType.Ask** or **MarketDataType.Bid**series.  Please see [Developing for Tick Replay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) for more information. |

**Syntax**

AddLineBreak(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, int lineBreakCount, Data.MarketDataType marketDataType)  
AddLineBreak(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, int lineBreakCount, Data.MarketDataType marketDataType, string tradingHoursName)  
AddLineBreak(string instrumentName, Data.BarsPeriodType baseBarsPeriodType, int baseBarsPeriodTypeValue, int lineBreakCount, Data.MarketDataType marketDataType, string tradingHoursName, bool? isResetOnNewTradingDay)

|  |
| --- |
| **Warnings:**  •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 **AddLineBreak()** calls as those hosted scripts would. For further reference, please also review 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 **AddLineBreak()** 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.** |

**Parameters**

|  |  |
| --- | --- |
| instrumentName | A string determining instrument name such as "MSFT" |
| baseBarsPeriodType | The underlying BarsType used for the LineBreak bars period    Possible values are:  BarsPeriodType.Day BarsPeriodType.Minute BarsPeriodType.Second BarsPeriodType.Tick BarsPeriodType.Volume |
| baseBarsPeriodTypeValue | An int determining the underlying period interval such as "3" for 3 minute bars |
| lineBreakCount | An int determining the number of bars back used to calculate a line break |
| 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. |

|  |
| --- |
| **Tip**: 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: AddLineBreak("MSFT Arca", PeriodType.Minute, 1, 3, MarketDataType.Last) |

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {         if (State == State.SetDefaults)   {     Name = "Examples Indicator";               }     if (State == State.Configure)   {     // Add a 1 minute Line Break Bars object for the ES 03-18 - BarsInProgress index = 1     AddLineBreak("ES 03-18", BarsPeriodType.Minute, 1, 3, MarketDataType.Last);   }         }   protected override void OnBarUpdate() {     // Ignore the primary Bars object and only process the Line Break Bars object     if (BarsInProgress == 1)     {         // Do something;     } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Using Historical Bid/Ask Series** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_bitmapimage_objects_with_buttons.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_images_and_geometry_with_custom_icons.htm) |

**Historical Bid/Ask Series Overview**

NinjaTrader has the ability to use historical bid and ask price series in your NinjaScript instead of only being able to use a last price series. The following outlines the intricacies of this capability:

|  |
| --- |
| **Notes**:  •You can have multiple bid/ask/last series in your NinjaScript indicator/strategy. Please use the [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) method to add these series to your script.  •The historical bid/ask series holds *all* bid/ask events sent out by the exchange. This would *not* be equivalent to the bid/ask at a specific time a trade went off.  •When processing your NinjaScript, the historical bid/ask series would have the historical portion triggered in the [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) method only. [OnMarketData()](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm) method events for the historical bid/ask series would only be triggered in real-time. |

|  |
| --- |
| **Tips**:  •For using **OnMarketData()** events historically, please see the educational topic on [Developing for Tick Replay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm)  •Changing the price type used for the primary Bars object to which a script is applied can be done in the [Data Series](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_price_data.htm) window from any open chart. |

**Accessing Bid/Ask Series**

When calling AddDataSeries() to add an additional [Bars](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) object to your script, a constructor overload will be available which takes a MarketDataType enumeration as an argument. This will allow you to specify the price series which will be used in that particular object. If you were to pass in MarketDataType.Ask or MarketDataType.Bid, as in the example below, that particular data series will use that price type for all of its [PriceSeries](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) collections, such as [Close](https://ninjatrader.com/es/support/helpGuides/nt8/close.htm), [Open](https://ninjatrader.com/es/support/helpGuides/nt8/open.htm), [High](https://ninjatrader.com/es/support/helpGuides/nt8/high.htm), and [Low](https://ninjatrader.com/es/support/helpGuides/nt8/low.htm).

|  |
| --- |
| **Warning**: A **Tick Replay** indicator or strategy **CANNOT**use a **MarketDataType.Ask** or **MarketDataType.Bid**series.  Please see [Developing for Tick Replay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) for more information. |

**Example**

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if (State == State.Configure)   {       // Add an AAPL data series using the Ask series       AddDataSeries("AAPL", BarsPeriodType.Minute, 30, MarketDataType.Ask);         //Add another AAPL data series using the Bid series, with other settings identical       AddDataSeries("AAPL", BarsPeriodType.Minute, 30, MarketDataType.Bid);   } } | |
| **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()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) >  **CurrentBars** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/barsperiods.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/alert__debugging_and_sharing.htm) |

**Definition**

Holds an array of int values representing the number of the current bar in a Bars object. An int value is added to this array when calling the [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) method. Its purpose is to provide access to the [CurrentBar](https://ninjatrader.com/es/support/helpGuides/nt8/currentbar.htm) of all Bars objects in a multi-instrument or multi-time frame script.

|  |
| --- |
| **Note**:    In [multi series](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) processing, the **CurrentBars** starting value will be -1 until all series have processed the first bar. |

**Property Value**

An array of int values.

|  |
| --- |
| **Warning**: This property should **NOT** be accessed within the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method before the **State** has reached **State.DataLoaded** |

**Syntax**

CurrentBars[int *barSeriesIndex*]

**Examples**

| ns **Indicator**([BarsRequiredToPlot](https://ninjatrader.com/es/support/helpGuides/nt8/barsrequiredtoplot.htm)) |
| --- |
| protected override void OnStateChange() {     if (State == State.Configure)     {         // Adds a 5-minute Bars object to the script. It will automatically be assigned         // a Bars object index of 1 since the primary data the indicator is run against         // set by the UI takes the index of 0.         AddDataSeries("AAPL", BarsPeriodType.Minute, 5);     } }   protected override void OnBarUpdate() {     // Evaluates to make sure we have at least 20 (default value of BarsRequiredToPlot)     // or more bars in both Bars objects before continuing.     if (CurrentBars[0] < BarsRequiredToPlot || CurrentBars[1] < BarsRequiredToPlot)         return;       // Indicator script logic calculation code... } |

| ns **Strategy**([BarsRequiredToTrade](https://ninjatrader.com/es/support/helpGuides/nt8/barsrequiredtotrade.htm)) | |
| --- | --- |
| protected override void OnStateChange() {     if (State == State.Configure)     {         // Adds a 5-minute Bars object to the script. It will automatically be assigned         // a Bars object index of 1 since the primary data the indicator is run against         // set by the UI takes the index of 0.         AddDataSeries("AAPL", BarsPeriodType.Minute, 5);     } }   protected override void OnBarUpdate() {     // Evaluates to make sure we have at least 20 (default value of BarsRequiredToTrade)     // or more bars in both Bars objects before continuing.     if (CurrentBars[0] < BarsRequiredToTrade || CurrentBars[1] < BarsRequiredToTrade)         return;       // Strategy script logic calculation code... } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) >  **BarsRequiredToPlot** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/plots.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/displayindatabox.htm) |

**Definition**

The number of bars on a chart required before the script plots.

|  |
| --- |
| **Note**:  This property is **NOT** the same as a minimum number of bars required to calculate the script values.  OnBarUpdate will always start calculating for the first bar on the chart (CurrentBar 0) |

**Property Value**

An int value that represents the minimum number of bars required.

**Syntax**

BarsRequiredToPlot

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {      if (State == State.SetDefaults)      {           BarsRequiredToPlot = 10; // Do not plot until the 11th bar on the chart          AddPlot(Brushes.Orange, "SMA");      }     } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) >  **OnStateChange()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/marketdeptheventargs.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/setstate.htm) |

**Definition**

An event driven method which is called whenever the script enters a new [State](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm). The **OnStateChange()** method can be used to configure script properties, create one-time behavior when going from historical to real-time, as well as manage clean up resources on termination.

|  |
| --- |
| **Notes**:  •Viewing any UI element which lists NinjaScript classes (such as the Indicators or Strategies window, a chart's Chart Style dropdown menu, etc.) will initialize all classes of that Type when it is opened, which causes each script to enter **State.SetDefaults**, even if it is not actively configured or running in any window. It is important to keep this in mind when adding logic within **State.SetDefaults** in **OnStateChange()**, as this logic will be processed each time the script is initialized. For example, opening the Indicators window will trigger **State.SetDefaults** for all indicators in the system, and closing the Indicators window will trigger**State.Terminated** for all Indicators. In addition, disconnecting or connecting to a data provider can cause State transitions for any currently active scripts. Further discussion of this aspect of the state change model can be found via [*Understanding the lifecycle of your NinjaScript objects*](https://ninjatrader.com/es/support/helpGuides/nt8/understanding_the_lifecycle_of.htm).  •When an indicator is configured on a chart while a Compile is taking place in the NinjaScript Editor, it can appear that the script passes through **State.Terminated**. However, this is the result of a copy of the script being initialized at compile-time, NOT the result of the indicator on the chart being disabled and re-initialized. |

**Related Methods and Properties**

|  |  |
| --- | --- |
| [SetState()](https://ninjatrader.com/es/support/helpGuides/nt8/setstate.htm) | Method is used for changing the State of any running NinjaScript object. |
| [State](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) | Represents the current progression of the object as it advances from setup, processing data, to termination. |

**Method Return Value**

This method does not return a value.

**Syntax**  
See example below. The NinjaScript wizards automatically generate the method syntax for you.

Possible states are:

|  |  |  |
| --- | --- | --- |
| **State Name** | **This state is called when** | **This state is where you should** |
| State.SetDefaults | **SetDefaults** is always called when displaying objects in a UI list such as the Indicators dialogue window since temporary objects are created for the purpose of UI display | •Keep as lean as possible  •Set default values (pushed to UI) |
| State.Configure | **Configure** is called after a user adds an object to the applied list of objects and presses the OK or Apply button.  This state is called only once for the life of the object. | •Add additional data series via [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm)  •Declare custom resources |
| State.Active | **Active** is called once after the object is configured and is ready to process data (DrawingTools could see multiple calls as internally an object for hit testing is cloned) | •Used for objects such as [Share Service](https://ninjatrader.com/es/support/helpGuides/nt8/share_service.htm) which do not process price series data  •Indicate the object is ready to being processing information |
| State.DataLoaded | **DataLoaded** is called only once after all data series have been loaded. | •Use for logic that needs to access data related objects like Bars, Instruments, BarsPeriod, TradingHours or instantiating indicators  •Notify that all data series have been loaded  •Initialize any class level variables (including custom [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) objects) |
| State.Historical | **Historical**is called once the object begins to process historical data. This state is called once when running an object in real-time. This object is called multiple times when running a backtest optimization and the property [IsInstantiatedOnEachOptimizationIteration](https://ninjatrader.com/es/support/helpGuides/nt8/isinstantiatedoneachoptimizationiteration.htm) is true (default behavior) | •Notify that the object is processing historical data |
| State.Transition | **Transition** is called once as the object has finished processing historical data but before it starts to process realtime data. | •Notify that the indicator or strategy is is transitioning to realtime data  •Prepare realtime related resources |
| State.Realtime | **Realtime** is called once when the object begins to process realtime data. | •Notify that the indicator or strategy is processing realtime data  •Execute realtime related logic |
| State.Terminated | **Terminated** is called once when the object terminates. | •Notify the object is shutting down  •Use to clean up/dispose of resources |

**Active States vs Data Processing States**

After **State.Configure,**each type of NinjaScript type has its own state management system which can be classified under two categories:

•**Active state:**  State.Active

•**Data Processing states:**State.DataLoaded, State.Historical, State.Transition, State.Realtime

The table below lists each NinjaScript type and it's designed state management system:

|  |  |
| --- | --- |
| **NinjaScript Type** | **State Management System** |
| AddOns\* | Active state |
| BarTypes | Active state |
| ChartStyles | Active state |
| DrawingTools | Active state |
| Indicators | Data Processing states |
| ImportTypes | Active state |
| Market Analyzer Columns | Data Processing states |
| OptimizationFitnesses | Active state |
| Optimizers | Active state |
| PerformanceMetrics | Active state |
| ShareServices | Active state |
| Strategies | Data Processing states |
| SuperDOM Columns | Active state |

|  |
| --- |
| **Tips:**  •Resources created in **State.Configure** and not disposed of immediately will be kept and utilized if the NinjaScript object resides in grids (e.g. Strategy tab on Control Center), even if it is not enabled. Try to create resources in**State.Historical** or **State.DataLoaded** instead, if possible.  •**State.Historical** is called multiple times when running a backtest [optimization](https://ninjatrader.com/es/support/helpGuides/nt8/optimize_a_strategy.htm) on a strategy and the property "[IsInstantiatedOnEachOptimizationIteration](https://ninjatrader.com/es/support/helpGuides/nt8/isinstantiatedoneachoptimizationiteration.htm)" is **true** (default behavior).  • Scripts that require [Calculate](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm) to be set by the developer must set this property in **State.Historical** in order to ensure that if this script is a child (hosted) that the parent.Calculate property which is adopted by the child is overridden again.  •When instantiating indicators in a [Multi-Series script](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) in OnStateChange, the input any hosted indicator is running on should be explicitly stated (since a specific [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) is not guaranteed) |

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if (State == State.SetDefaults)   {     // Calculate once at the end of every single bar     Calculate = Calculate.OnBarClose;         // Add two plots     AddPlot(Brushes.Blue, "Upper"));     AddPlot(Brushes.Orange, "Lower"));   }     else if (State == State.Configure)   {     // Adds a 5-minute Bars object to the strategy and is automatically assigned     // a Bars object index of 1 since the primary data the strategy is run against     // set by the UI takes the index of 0.             AddDataSeries("AAPL", BarsPeriodType.Minute, 5);       } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **BarsPeriod** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm) |

**Definition**

The primary Bars object time frame (period type and interval).

|  |
| --- |
| **Warning**:  This property should **NOT** be accessed within the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method before the **State** has reached **State.DataLoaded** |

**Property Value**

A [Bars](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) series object representing the time frame of the Bars.

**Syntax**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BarsPeriod.BarsPeriodType | The type of bars used for the period, as well as the enumeration value under which the any of the 14 default NinjaTrader types are registered. Possible values include:     |  |  | | --- | --- | | BarsPeriodType.Tick | 0 | | BarsPeriodType.Volume | 1 | | BarsPeriodType.Range | 2 | | BarsPeriodType.Second | 3 | | BarsPeriodType.Minute | 4 | | BarsPeriodType.Day | 5 | | BarsPeriodType.Week | 6 | | BarsPeriodType.Month | 7 | | BarsPeriodType.Year | 8 | | BarsPeriodType.HeikenAshi | 9 | | BarsPeriodType.Kagi | 10 | | BarsPeriodType.Renko | 11 | | BarsPeriodType.PointAndFigure | 12 | | BarsPeriodType.LineBreak | 13 | | BarsPeriodType.Volumetric | 14 |        |  | | --- | | **Tip**: When creating custom [BarsTypes](https://ninjatrader.com/es/support/helpGuides/nt8/bars_type.htm), it is recommended to pick high, unique enumeration value to avoid conflict from other BarsTypes that may be used by a single installation.    BarsPeriod = new BarsPeriod { BarsPeriodType = (BarsPeriodType)123456, BarsPeriodTypeName = "MyCustomBars", Value = 1 }; | |
| BarsPeriod.BaseBarsPeriodType | Only relevant for [HeikenAshi](https://ninjatrader.com/es/support/helpGuides/nt8/addheikenashi.htm), [Kagi](https://ninjatrader.com/es/support/helpGuides/nt8/addkagi.htm), [LineBreak](https://ninjatrader.com/es/support/helpGuides/nt8/addlinebreak.htm), [PointAndFigure](https://ninjatrader.com/es/support/helpGuides/nt8/addpointandfigure.htm) and [Volumetric](https://ninjatrader.com/es/support/helpGuides/nt8/addvolumetric.htm) Bars objects. Same possible values as BarsPeriod.BarsPeriodType |
| BarsPeriod.BaseBarsPeriodValue | Only relevant for [HeikenAshi](https://ninjatrader.com/es/support/helpGuides/nt8/addheikenashi.htm), [Kagi](https://ninjatrader.com/es/support/helpGuides/nt8/addkagi.htm), [LineBreak](https://ninjatrader.com/es/support/helpGuides/nt8/addlinebreak.htm), [PointAndFigure](https://ninjatrader.com/es/support/helpGuides/nt8/addpointandfigure.htm) and [Volumetric](https://ninjatrader.com/es/support/helpGuides/nt8/addvolumetric.htm) Bars objects. Determines an integer value representing the basePeriodTypeValue parameter |
| BarsPeriod.MarketDataType | The data type used to build the bars.  Possible values:  MarketDataType.Ask MarketDataType.Bid MarketDataType.Last |
| BarsPeriod.PointAndFigurePriceType | Only relevant for PointAndFigure Bars objects. Possible values: PointAndFigurePriceType.Close PointAndFigurePriceType.HighsAndLows |
| BarsPeriod.ReversalType | Only relevant for Kagi Bars objects. Possible values: ReversalType.Percent ReversalType.Tick |
| BarsPeriod.Value | Determines an integer value representing the period parameter.  •When using Kagi Bars objects this represents the "reversal" parameter  •When using LineBreak Bars objects this represents the "lineBreakCount" parameter  •When using PointAndFigure Bars objects this represents the "boxSize" parameter  •When using Renko Bars objects this represents the "brickSize" parameter |
| BarsPeriod.Value2 | Only relevant for PointAndFigure Bars objects. Determines an integer value representing the "reversal" parameter. |

**Examples**

| ns **Checking BarsPeriod values** |
| --- |
| // Calculate only if there is a 100 tick chart or greater protected override void OnBarUpdate() {     if (BarsPeriod.BarsPeriodType == BarsPeriodType.Tick && BarsPeriod.Value >= 100)     {         // Indicator calculation logic here     } } |

| ns **Creating a new BarsPeriod object** | |
| --- | --- |
| protected override void OnStateChange() {     if (State == State.Configure)     {                 // add a 1440 minute apple bars object using the RTH session template           AddDataSeries("AAPL", new BarsPeriod { BarsPeriodType = BarsPeriodType.Minute, Value = 1440 }, "US Equities RTH");                   }       else if (State == State.DataLoaded)     {           // Print out the loaded bars period           Print(Instrument.FullName + " " + BarsPeriod); // MSFT 1 Minute           Print(BarsArray[1].Instrument.FullName + " " + BarsArray[1].BarsPeriod); // AAPL 1440 Minute     } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **Calculate** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/barsperiod.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/count.htm) |

**Definition**

Determines how often [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) is called for each bar. **OnBarClose** means once at the close of the bar. **OnEachTick** means on every single tick. **OnPriceChange** means once for each price change. If there were two ticks in a row with the same price, the second tick would not trigger OnBarUpdate(). This can improve performance if calculations are only needed when new values are possible.

|  |
| --- |
| **Notes**:  1.On a historical data set, only the OHLCVT of the bar is known and not each tick that made up the bar.  As a result, [State.Historical](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) data processes **OnBarUpdate()** only on the close of each historical bar even if this property is set to **OnEachTick** or **OnPriceChange**.  You can use [TickReplay](https://ninjatrader.com/es/support/helpGuides/nt8/tick_replay.htm) or a [Multi-time frame script](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) to obtain intrabar data.  2.When set to Calculate **OnPriceChange**, the **OnBarUpdate()** method is **ONLY** called when the price has changed which does not necessarily occur the end of the close of the bar |

**Property Value**

An enum value determining the how frequently OnBarUpdate() will be called.  Default value is set to Calculate.OnBarClose

|  |
| --- |
| **Warning**:  If your script relies on volume updates **OnPriceChange** should **NOT** be used since it can potentially miss volume updates if they occur at the same price |

**Syntax**

Calculate.OnBarClose

Calculate.OnEachTick

Calculate.OnPriceChange

|  |
| --- |
| **Tips**  1.Calculating indicators or systems for each incoming tick can be CPU intensive. Only calculate indicators on each incoming tick if you have a requirement to calculate it intra-bar.  2.For an example of how to separate some logic to be Calculate = Calculate.OnBarClose and other logic to be .OnEachTick please see this [reference sample](http://www.ninjatrader.com/support/forum/showthread.php?t=19387).  3.Embedded scripts within a calling parent script should not use a different Calculate property since it is already utilizing the Calculate property of the parent script (i.e. the strategy your indicator is called from).  4. Typically this property would be set in **State.SetDefaults**, however scripts that require Calculate to be set by the developer must set this property in **State.Historical** in its [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) in order to ensure that if this script is a child (hosted) that the parent.Calculate property which is adopted by the child is overridden again. |

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {     if (State == State.SetDefaults)     {         // Calculate on the close of each bar         Calculate = Calculate.OnBarClose;     } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **Count** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/currentbar.htm) |

**Definition**

The total number of bars or data points.

**Property Value**

An int value representing the the total number of bars.

**Syntax**

Count

**Examples**

| ns |
| --- |
| //If there are less than 365 bars on the chart, text indicates how many bars are on the chart  if (Count < 365)  {   Draw.TextFixed(this, "tag1", "There are " + Count + " bars on the chart", TextPosition.BottomRight);  } |

|  |  |
| --- | --- |
| **Tip**: [CurrentBar](https://ninjatrader.com/es/support/helpGuides/nt8/currentbar.htm) value is guaranteed to be <= Count - 1. This is because of the NinjaTrader multi-threaded architecture, the Count value can have additional bars as inflight ticks come in to the system. | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **CurrentBar** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/count.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/isdataseriesrequired.htm) |

**Definition**

A number representing the current bar in a Bars object that the OnBarUpdate() method in an indicator or strategy is currently processing. For example, if a chart has 100 bars of data, the very first bar of the chart (left most bar) will be number 0 (zero) and each subsequent bar from left to right is incremented by 1.

|  |
| --- |
| **Note**:    In [multi series](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) processing, the [CurrentBars](https://ninjatrader.com/es/support/helpGuides/nt8/currentbars.htm) starting value will be -1 until all series have processed the first bar. |

**Property Value**

An int value that represents the current bar.

**Syntax**

CurrentBar

**Examples**

| ns | |
| --- | --- |
| // OnBarUpdate method protected override void OnBarUpdate() {     // Evaluates to make sure we have at least 20 or more bars     if (CurrentBar < 20)         return;       // Indicator logic calculation code... } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **IsFirstTickOfBar** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isdataseriesrequired.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/isresetonnewtradingdays.htm) |

**Definition**

Indicates if the incoming tick is the first tick of a new bar. This property is only of value in scripts that run tick by tick which is when the [Calculate](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm) property is set to **Calculate.OnEachTick** or **Calculate.OnPriceChange**.

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

|  |
| --- |
| **Note**: If a bar type is set up to [remove the last bar](https://ninjatrader.com/es/support/helpGuides/nt8/removelastbar.htm) on a chart, **IsFirstTickOfBar** will automatically be set to **True**. |

**Property Value**

This property returns **true** if the incoming tick is the first tick of a new bar; otherwise, **false**.

**Syntax**

IsFirstTickOfBar

**Examples**

| ns | |
| --- | --- |
| // On a tick by tick strategy the only way you know when a bar is closed is when // the IsFirsTickOfBar is true. protected override void OnBarUpdate() {     // Only process entry signals on a bar by bar basis (not tick by tick)     if (IsFirstTickOfBar)     {         if (CCI(20)[1] < -250)               EnterLong();         return;     }       // Process exit signals tick by tick     if (CCI(20)[0] > 250)         ExitLong(); } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **IsResetOnNewTradingDays** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isfirsttickofbar.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/istickreplays.htm) |

**Definition**

Determines if the specified bar series is using Break at EOD

|  |
| --- |
| **Note**:  The property available on the UI will override any values set in code. Please see the help guide topic on using [Break at EOD](https://ninjatrader.com/es/support/helpGuides/nt8/break_at_eod.htm) for more information |

**Property Value**

A bool[]when **true**, indicates the specified [BarsArray](https://ninjatrader.com/es/support/helpGuides/nt8/barsarray.htm) is setup to run Break at EOD; otherwise **false**.  Default value is **false**

**Syntax**

IsResetOnNewTradingDays[int idx]

|  |
| --- |
| **Warning**:  This property should **NOT** be accessed within the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method before the **State** has reached **State.DataLoaded** |

**Examples**

|  |  |
| --- | --- |
| ns | |
| protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "Examples Indicator";   }     else if (State == State.Configure)   {     //Add AAPL 1 minute with RTH trading hours, set to break EOD     AddDataSeries("AAPL", new BarsPeriod() { BarsPeriodType = BarsPeriodType.Minute, Value = 1 }, 50, "US Equities RTH", true);   }   } protected override void OnBarUpdate() {                 //Print out the current bars series name and break EOD setting on start up  //   IsResetOnNewTradingDays[0]  Primary //   IsResetOnNewTradingDays[1]  AAPL  if (CurrentBar == 0)             Print(BarsArray[BarsInProgress].ToChartString() + " " + IsResetOnNewTradingDays[BarsInProgress]);   //Output:   //ES 03-15 (1 Minute) True //AAPL (1 Minute) False         } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **IsTickReplays** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isresetonnewtradingdays.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/update.htm) |

**Definition**

Indicates the specified bar series is using Tick Replay.   Please see the help guide topic on using [Tick Replay](https://ninjatrader.com/es/support/helpGuides/nt8/tick_replay.htm) for general information on this mode.

|  |
| --- |
| **Note**:  For a primary series, the **Tick Replay** option must be configured from the UI before a NinjaScript object can take use of this property.  The setting on the Chart's Data Series menu will always take precedence for an object series which already exists on the user's chart. |

|  |
| --- |
| **Warning**:  This property should **NOT** be accessed within the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method before the **State** has reached **State.DataLoaded** |

**Property Value**

A bool[]when **true**, indicates the specified [BarsArray](https://ninjatrader.com/es/support/helpGuides/nt8/barsarray.htm) is setup to run Tick Replay; otherwise **false**.  Default value is **false**

**Syntax**

IsTickReplays[int idx]

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if(State == State.SetDefaults)   {       Name = "Examples Indicator";   }     else if (State == State.Configure)   {     AddDataSeries("AAPL", BarsPeriodType.Minute, 1);     }   else if (State == Data.Loaded)   {       // IsTickReplays[0] = true;      // Programmatically setting this option here for Primary [0] does not have any effect           // Primary series must be configured from UI                 // It is not possible to combine Tick Replay series and non Tick Replay series in a single chart or script     // The assignment below would not be necessary if the primary series were set to True via the UI     // IsTickReplays[1] = true;   } }   **protected** **override** **void** OnBarUpdate() {                   *//Print out the current bars series name and tick replays setting on start up*   **if** (CurrentBar == 0)               Print(BarsArray[BarsInProgress].ToChartString() + " " + IsTickReplays[BarsInProgress]); } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) >  **Update()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/istickreplays.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/onconnectionstatusupdate.htm) |

**Definition**

Forces the OnBarUpdate() method to be called so that indicator values are updated to the current bar.  If the values are already up to date, the Update() method will not be run.

|  |
| --- |
| **Note**:  This method is only relevant in specific use cases and should only used by advanced programmers |

When indicators are embedded (called) within a NinjaScript strategy, they are optimized to calculate only when they are called upon in a historical backtest. Since the NinjaTrader indicator model is very flexible, it is possible to create public properties on a custom indicator that return values of internal user defined variables. If these properties require that the OnBarUpdate() method is called before returning a value, include a call to this Update() method in the property getter.

**Syntax**

Update()

**Parameters**

|  |  |
| --- | --- |
| idx | The current bar index value to update to |
| bip | The [BarsInProgress](https://ninjatrader.com/es/support/helpGuides/nt8/barsinprogress.htm) to update |

**Examples**

| ns | |
| --- | --- |
| private double tripleValue = 0;   protected override void OnBarUpdate() {   if (CurrentBar < 20)       return;     tripleValue = SMA(20)[0] \* 3;   Value[0] = SMA(20)[0]; }   public double TripleValue {     get     {       //call OnBarUpdate before returning tripleValue       Update();       return tripleValue;     } } | |
| **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()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) >  **BarsInProgress** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/barsarray.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/barsperiods.htm) |

**Definition**

An index value of the current Bars object that has called the [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) method. In a multi-bars script, the OnBarUpdate() method is called for each Bars object of a script. This flexibility allows you to separate trading logic from different bar events.

|  |
| --- |
| **Notes**:  1.In a single Bars script this property will always return an index value of 0 representing the primary Bars and instrument the script is running on.  2.See additional information on running [multi-bars scripts](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm). |

**Property Value**

An int value represents the [Bars](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) object that is calling the OnBarUpdate() method.

**Syntax**

BarsInProgress

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {     if (State == State.Configure)     {         // Add a 5 minute Bars object: BarsInProgress index = 1         AddDataSeries(BarsPeriodType.Minute, 5);     } }   protected override void OnBarUpdate() {     // Check which Bars object is calling the OnBarUpdate() method     if (BarsInProgress == 0)     {         // A value of zero represents the primary Bars which is the ES 09-14         // 1 minute chart.         // Do something within the context of the 1 minute Bars here     }     else if (BarsInProgress == 1)     {         // A value of 1 represents the secondary 5 minute bars added in OnStateChange() State.Configure         // Do something within the context of the 5 minute Bars     } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **AddOn Development Overview** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/developing_add_ons.htm) |

**AddOn Development Basics**

The NinjaScript AddOn framework provides functionality reaching across the NinjaTrader platform while granting access to certain core methods and properties not contained within the NinjaScript namespace. In addition to creating your own independent window or modifying the user interface and functionality of existing NinjaTrader windows (charts, etc.), AddOns can also subscribe to live market data, access account information, and more.

|  |
| --- |
| **Note**: Most of the topics covered on this page and its sub-pages can be seen in a fully functional example of the **AddOn Framework** accessible on this page.  There are two versions deployed when you can access depending on your desired development environment.  The pros and cons of each approach are described in the following section.  In either version, the heavily commented code in the example can supplement the information on these pages to provide deeper insight. |

**NinjaScript Editor Development Environment (NinjaScript Basic)**

The NinjaScript Editor can be used to create and write custom AddOns in C#

**Pros**

•Use the familiar [NinjaScript editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) (if you are uncomfortable with Visual Studio)

•Changes to the AddOn are reflected immediately upon NS Editor Compile and does not require restart

**Cons**

•If you wish to design a custom NTWindow, XAML files cannot be edited in the NinjaScript editor.

•NinjaScript editor lacks support of common development and debugging tools available in an IDE's like Visual Studio

**Below is a NinjaScript Editor compatible zip file (which also contains a XAML file)**

•Download [AddOn Framework NinjaScript Basic](https://ninjatrader.com/support/helpGuides/nt8/samples/Addon_Framework_NinjaScript_Basic.zip) file to your desktop

•From the Control Center window select the menu Tools > Import > NinjaScript

•Select the downloaded file

Once imported, the AddOn can be launched via the New menu in the Control Center

**AddOn Development Environment (Visual Studio Advanced)**

Since AddOns can include multiple classes, unique user interfaces, and various file types (XAML, sounds, etc.), the recommended development environment for AddOns differs from other NinjaScript Types. Following the guidelines below to set up an AddOn development environment can help to streamline the process.

**Pros**

•Use Visual Studio or a comparable IDE to create a solution linking all project files together

•Use your IDE to build a DLL, rather than exporting through NinjaTrader

•This will allow you to bundle XAML and other files into the DLL

•Set a post-build event to place the DLL into the appropriate folder (NinjaTrader 8/bin/Custom)

•Set a Debug Start Action to launch NinjaTrader

**Cons**

•NinjaTrader needs to be restarted in order to re-load the compiled DLL after changes

If you use this setup and build a DLL with your IDE, the IDE will automatically place it where it needs to be and immediately launch the platform for testing any changes.

**Below is a complete Visual Studio project with this setup in place**. Simply unzip the contents of this archive to your desired location, then open the "NinjaTraderAddOnProject.sln" solution in Visual Studio.

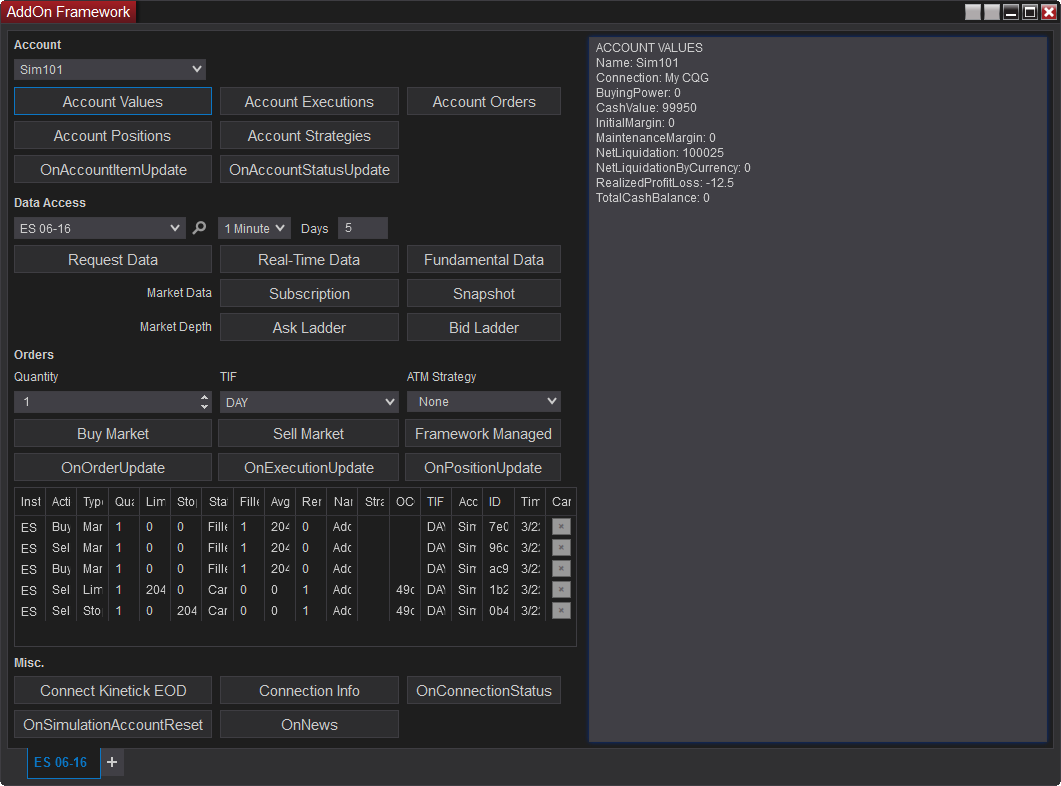
[Download Visual Studio Solution for AddOn Development](https://ninjatrader.com/support/helpGuides/nt8/samples/NinjaTraderAddOnProject.zip)

|  |
| --- |
| **Notes**:  •This Visual Studio solution cannot be imported into NinjaTrader.  It must be opened in Visual Studio.  •The default behavior of the project file uses the following path in the Start Action: C:\Program Files (x86)\NinjaTrader 8\bin64\NinjaTrader.exe. If you have installed NinjaTrader in a different directory, you will need to adjust the file path accordingly. |

**Creating Your Own AddOn Window**

NinjaScript developers can utilize the AddOn framework to create free-standing, independent windows to provide custom functionality. Helper classes are available in the framework to instantiate windows styled the same as pre-built NinjaTrader windows, including familiar functionality such as window linking, the tabbed interface, and the ability to save the window and its state in workspaces. In addition, general WPF user interface elements and XAML can be used to style and modify windows using the .NET framework.

For a detailed walkthrough of creating your own window using NinjaScript helper classes, see the [Creating Your Own AddOn Window](https://ninjatrader.com/es/support/helpGuides/nt8/creating_your_own_addon_window.htm) page.

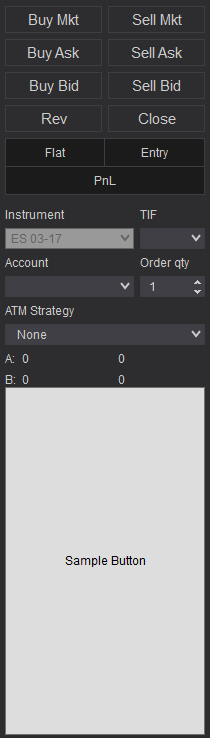


The image above shows a completely new window created by a custom AddOn.

**Other Uses for an AddOn**

An AddOn does not require its own window to function. It can instead be used to accomplish non-UI-driven functionality across the platform, such as monitoring market data or accessing account, position, and order information. AddOns can also be used to add functionality or interface elements to other NinjaTrader windows, such as charts.

For detailed information on other common uses of an AddOn, see the [Other Uses for an AddOn](https://ninjatrader.com/es/support/helpGuides/nt8/other_uses_for_an_addon.htm) page.



In the image above, the custom "Sample button" button has been drawn on a chart window using an AddOn.

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) >  **NinjaScript Editor Components** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/nodoc.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/ns_explorer.htm) |

**Overview**

The NinjaScript Editor is a powerful scripting editor that allows you to create custom indicators, strategies, and any other custom NinjaScript types used to enhance the NinjaTrader platform.  The NinjaScript Editor can be opened by selecting the **New** menu from the NinjaTrader Control Center. Then left mouse click on the menu item **NinjaScript Editor**

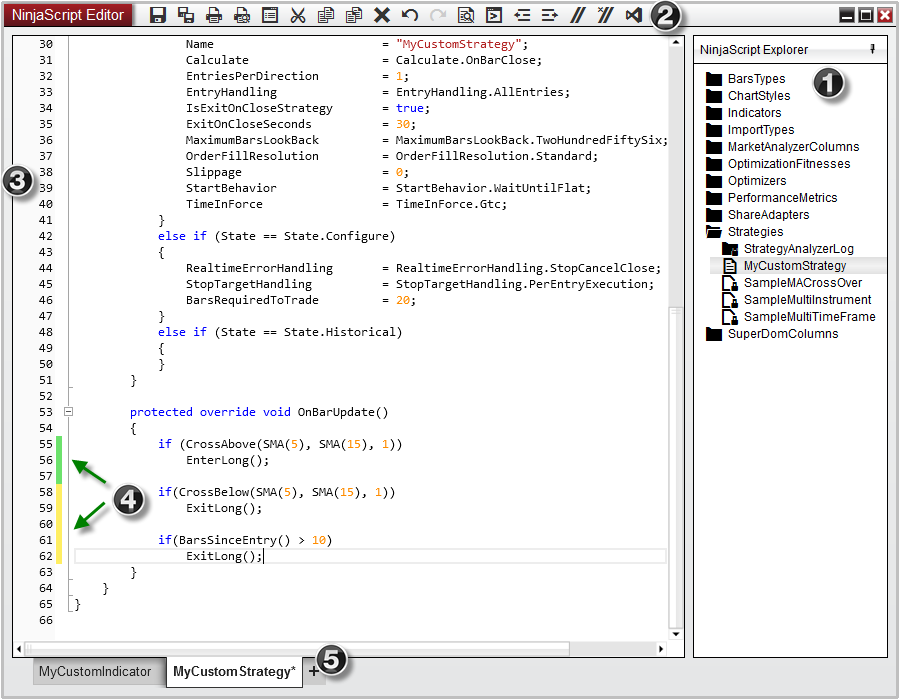
1. NinjaScript Explorer - Displays files, folders, and allows for additional file management

2. Tool bar - Moving your mouse over each icon will display the function of the icon button

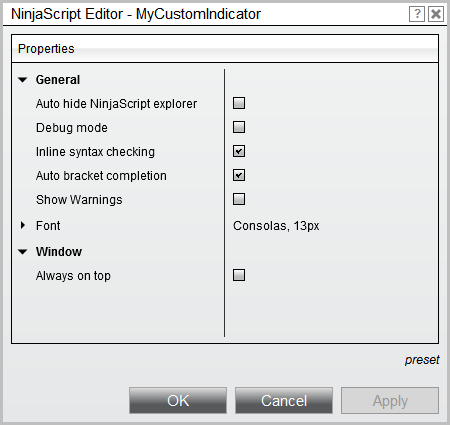
3. Line numbers

4. Line modification marking - Yellow flags indicate unsaved line modifications where green flags indicate saved modifications

5. Tabs for creating new scripts via the [NinjaScript wizard](https://ninjatrader.com/es/support/helpGuides/nt8/ns_wizard.htm) and working on multiple scripts.



**Properties and Definitions**



|  |  |
| --- | --- |
| **General** |  |
| Auto hide NinjaScript explorer | Sets if the NinjaScript explorer should be collapsed by default |
| Debug mode | Sets if a debug dll should be generated on compilation (see [Visual Studio Debugging](https://ninjatrader.com/es/support/helpGuides/nt8/visual_studio_debugging.htm) for more information) |
| Inline syntax checking | Sets if errors and warnings should be detected as code is written (without needing to compile) |
| Auto bracket completion | Sets if opening characters should automatically bed appended closing characters. Works for (parentheses), [brackets], {braces}, <angled brackets> |
| Show Warnings | Sets if code warnings should be show on compilation. |
| Font | Sets the font options |
| **Window** |  |
| Always on top | Sets if the window will be always on top of other windows. |

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) >  **NinjaScript Explorer** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/ns_editor_components.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/ns_wizard.htm) |

The **NinjaScript Explorer** provides a **Folder** view of all the supported NinjaScript categories that can be developed in NinjaTrader.

tog_minus        [Understanding the NinjaScript Explorer display](javascript:HMToggle('toggle','UnderstandingTheNinjascriptExplorerDisplay','UnderstandingTheNinjascriptExplorerDisplay_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Folder Displays**  The **NinjaScript Explorer** will organize each script installed on your system by type of NinjaScript object (Indicator, Strategy, SuperDOM Column, etc).  Each folder will display the following scripts under each category:     |  |  | | --- | --- | | 1. Locked scripts | Pre-built system scripts which come installed with NinjaTrader which can be viewed as read-only and are required for compilation (of course you can safe a custom copy of those to modify) | | 2. Custom scripts | Any script imported, or under development, which can be modified | | 3. Ignored custom scripts | Custom scripts which have been excluded from compilation (see the "*Excluding a script from compilation*" section below for more information) |     NS_Editor_17    **Pinning the NinjaScript Explorer**  1.  By default the NinjaScript Explorer will be "pinned" to the right side of the NinjaScript editor, however it can be collapsed out of view by pressing the pin icon NS_Editor_14 located at the top right of the explorer window.    NS_Editor_12    2. Once the NinjaScript Explorer is collapsed, you can quickly bring it back in view simply by selecting the NinjaTrader Explorer tab located on the right side. Selecting the pin icon NS_Editor_15 again will re-pin the NinjaScript Explorer to the NinjaScript Editor.    NS_Editor_13    **Right Click Menu**  Right clicking on an individual folder or script will give you a number of different menu items to help with the management of your custom scripts.    NS_Editor_16     |  |  | | --- | --- | | New | Opens the [NinjaScript Wizard](https://ninjatrader.com/es/support/helpGuides/nt8/ns_wizard.htm) for the relevant object type. | | Open | Opens the selected script in a new tab in the current NinjaScript Editor window | | Open In New NinjaScript Editor | Opens the selected script(s) in a new NinjaScript Editor window | | Exclude From Compilation | Prevents the selected script(s) from being compiled (see the "*Excluding a script from compilation*" section below for more information) | | Remove | Removes the current file or folder from the system | | New Folder | Creates a new custom folder to organize your scripts | | Rename | Renames the current selected file or folder | |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?ns_explorer.htm#UnderstandingTheNinjascriptExplorerDisplay)

tog_minus        [Managing scripts and folders](javascript:HMToggle('toggle','ManagingScriptsAndFolders','ManagingScriptsAndFolders_ICON'))

tog_minus        [Excluding a script from compilation](javascript:HMToggle('toggle','ExcludingAScriptFromCompilation','ExcludingAScriptFromCompilation_ICON'))

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) >  **Compile Errors** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/code_snippets.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/intelliprompt.htm) |

**When compiling a custom indicator or strategy it is possible and likely that you will generate compile errors.**

•NinjaTrader will compile ALL NinjaScript files NOT only the file you are working on

•A list of compile errors for all files will be displayed in the lower portion of the NinjaScript Editor

•Double click on an error to load the problem file and highlight the problem area

•Click on the error code to bring up Help Documentation on a specific error

•Right click on the error to exclude the problem file from compilation (see the section on [Excluding a script from compilation](https://ninjatrader.com/es/support/helpGuides/nt8/ns_explorer.htm) for more information)

**The image below illustrates a compile error**

1. Section where compile errors are displayed. Errors in the current loaded file are color coded a light color while errors in other files have a darker color code.

2. The file that contains the error

3. A description of the error

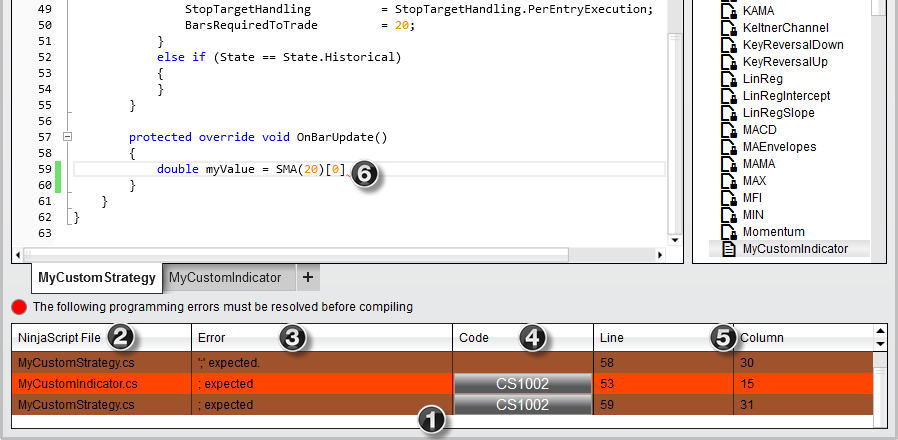
4. A error code link that will open the Help Guide with any relevant error code information

5. Line number and column number of the error

6. Error is underlined with a red wavy line

The error highlighted by icon (6) below shows that the expression is not closed with a semicolon. The expression should be:

double*myValue = SMA(20)[0];*



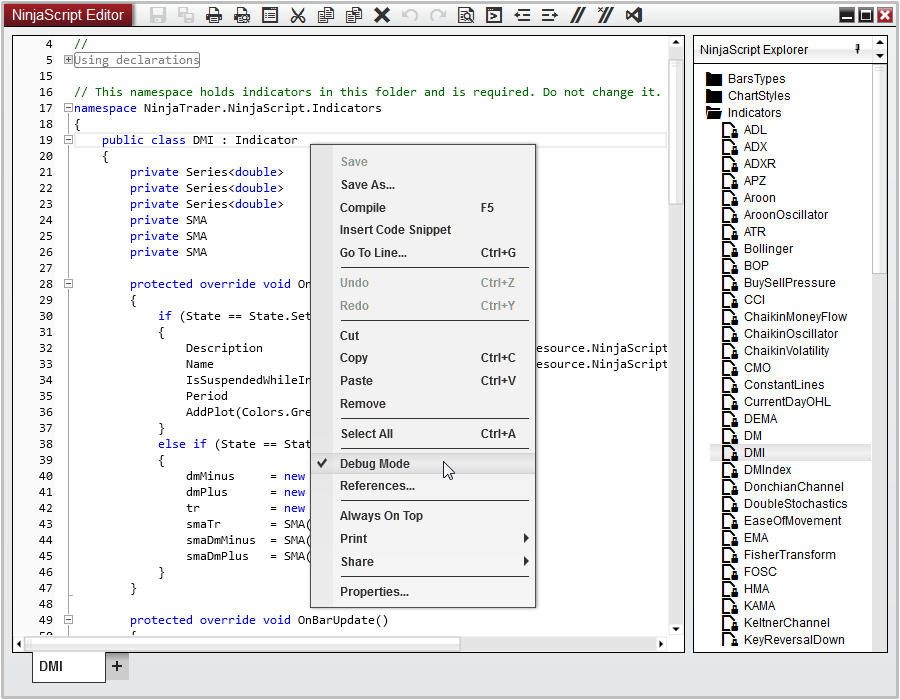
|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) >  **Visual Studio Debugging** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/output.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/editor_keyboard_shortcuts.htm) |

You can debug your NinjaScript objects using Microsoft Visual Studio. NinjaScript objects are compiled into a single DLL, named "NinjaTrader.Custom.dll." When debugging, a special debug DLL is created for temporary use, with the same name as the release version.

|  |
| --- |
| **Notes**:  •Using the debug DLL can incur a runtime performance impact, so it is recommended to disable Visual Studio debugging and re-compile your scripts when finished. This will replace the debug DLL with the release version.  •It will work with Visual Studio 2015, 2017 or 2019 - if multiple versions are installed, it will use the highest one. |

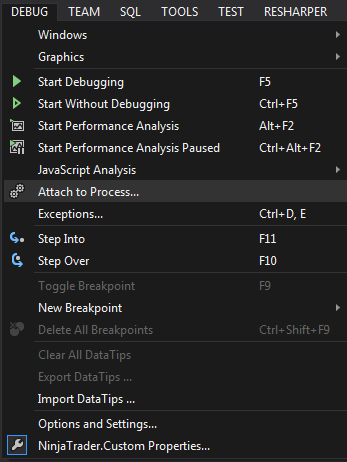
**Using Visual Studio Debugging**

1. In the NinjaScript Editor, enable "Debug Mode" via the right-click menu, as seen in the image below. After this, compile your scripts to create the debug DLL.

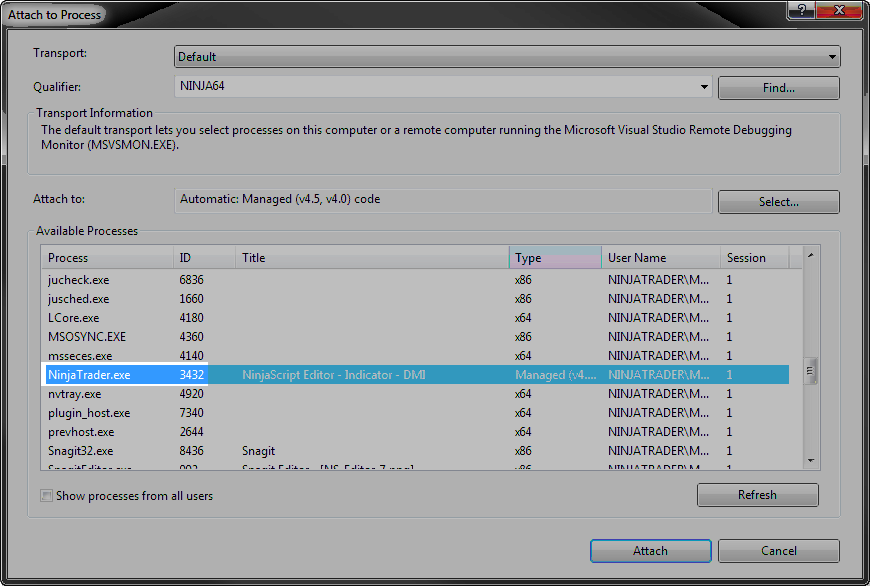


2. From the NinjaScript Editor, click on the Visual Studio icon NS_Editor_6 from the tool bar, which will automatically load the NinjaTrader.Custom project with your installed version of Visual Studio.

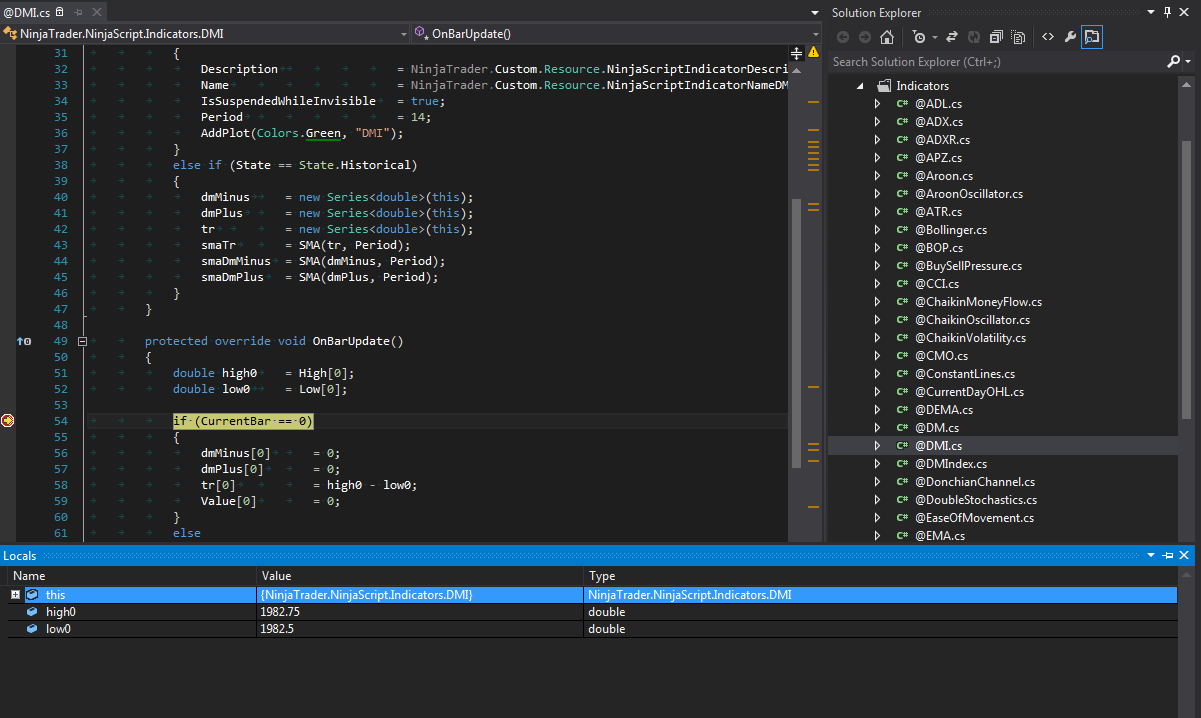
3. In Visual Studio, select Debug, then select **Attach to Process**



4. Select NinjaTrader from the list of processes, then select **Attach.** Be sure the "Attach to" field is set to "Automatic: Managed code" or "Managed code".



4. Open the NinjaScript source file within Microsoft Visual Studio and set your break point(s)



5. Run your NinjaScript object in NinjaTrader and it should stop at your break points and all the debugging tools and information should be available to inspect the current state of the code.

|  |  |
| --- | --- |
| **Tip**:  You can also use Visual Studio as editor for your NinjaScript files - for that open the project as in step 2 above and then use Visual Studio for editing and once done **save the file** (don't run or build the solution then in Visual Studio), preferably with the NinjaScript editor opened still at the same time, so changes would be auto compiled in then. | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS0234** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0201.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0246.htm) |

The following CS0234 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error can occur when an imported DLL (could be a 3rd party indicator) you are referencing no longer exists / has been removed.

To resolve this the DLL must be re-imported.

**To re-import a 3rd party dll:**

1.Open the NinjaScript Editor via New > NinjaScript editor.

2.Right mouse click in the NinjaScript Editor main window and select the menu name "References"

3.In the "References" dialog window press the button "Add"

4.Select the 3rd party DLL

|  |
| --- |
| **Warning**: Please make sure in this step to select only the 'true' DLL file needed for reference, which would not contain any X86 or X64 suffixes in its file-name, otherwise you could run into compile issues later. |

**Error Descriptions**

The type or namespace name '<name>' could not be found (are you missing a using directive or an assembly reference?)

The type or namespace name '<name>' does not exist in the namespace 'NinjaTrader.Indicator' (are you missing an assembly reference?)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS0019** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0006.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0021.htm) |

The following CS0019 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

Strings cannot be compared with relational operators (<, >, <=, >=, ==, !=) to other object types. Strings can only be compared to other strings and only through the use of equality operators (==, !=).

**Error Description #1**  
Operator '==' cannot be applied to operands of type 'string' and 'int'

// Erroneous Sample Code – Cannot compare a string to an integer

if ("string" == 5)

// Resolution Sample Code – Compare a string with another string

if ("string" == intValue.ToString());

**Error Description #2**  
Operator ‘<’ cannot be applied to operands of type ‘string’ and ‘double’

// Erroneous Sample Code - Cannot compare a string to a double

if ("string" >= 1.2)

// Resolution Sample Code - Testing to see if the strings are not the same

if ("string" != "string2")

**Error Description #3**  
Operator ‘>’ cannot be applied to operands of type ‘string’ and ‘string’

// Erroneous Sample Code - Cannot quantitatively compare a string to another string

if ("string" > "string2")

// Resolution Sample Code - Testing to see if both strings are the same

if ("string" == "string2")

**Additional Error Descriptions**  
Operator ‘<’ cannot be applied to operands of type ‘string’ and ‘string’  
Operator ‘<=’ cannot be applied to operands of type ‘string’ and ‘string’  
Operator ‘>=’ cannot be applied to operands of type ‘string’ and ‘string’

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS0021** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0019.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0029.htm) |

The following CS0021 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect.  In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This is a common error when calling indicators methods. It occurs when an indicator is called without its required parameter arguments before accessing an indexed value.

To fix this error you will need to first pass to the indicator method all the necessary parameter arguments. You can do this with '()' after the indicator name. Please note that you will still need to pass an empty parameter argument list even if your indicator requires no arguments.

**Error Description #1**  
Cannot apply indexing with [] to an expression of type 'method group'

**Example #1**

// Erroneous Sample Code - SMA is an indicator and requires parameter arguments

double value = SMA[0];

// Resolution Sample Code - SMA() properly called

double value = SMA(14)[0];

**Example #2**

// Erroneous Sample Code - EMA is an indicator and requires parameter arguments

double maDelta = EMA[0] - EMA[1];

// Resolution Sample Code - SMA() properly called with an overload method (one of several variations)

double maDelta = EMA(High, 14)[0] - EMA(High, 14)[1];

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS0103** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0029.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0200.htm) |

The following CS0103 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

When a variable is used before declaration, the compiler will not know what it is. This error is also commonly invoked by typos.

Please ensure that you have declared your variables prior to using them. If variables are declared or properties already exist, please check for typos.

**Error Description #1**  
The name 'identifier' does not exist in the current context

**Example #1**

// Erroneous Sample Code - 'CurentBar' does not exist since it has been spelled incorrectly (missing an 'r')

if (CurentBar < 10)

// Resolution Sample Code - 'CurrentBar' exists since it is spelled correctly

if (CurrentBar < 10)

**Example #2**

// Erroneous Sample Code - 'newVariable' is not declared

newVariable = 10;

// Resolution Sample Code - 'newVariable' is now declared as an integer

int newVariable = 10;

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS0200** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0103.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0201.htm) |

The following CS0200 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error is most common when you try to assign values to a particular Series<T> index that is read-only. Instead try making your own [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) and assign the value there.

**Error Description**  
Property or indexer 'NinjaTrader.NinjaScript.ISeries<double>.this[int]' cannot be assigned to -- it is read only

**Example #1**

// Erroneous Sample Code - Cannot assign values to something that is read-only

Close[0] = 25;

// Resolution Sample Code - Assigns value to a custom Series<double>

myCustomClose[0] = 25;

**Example #2**

// Erroneous Sample Code - Cannot reassign values to Series<double> indexed value and cannot have an if statement based // on an assignment operator

if (Close[0] = Open[0])

// Resolution Sample Code - Properly compares two Series<double> values

if (Close[0] == Open[0])

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS0234** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0201.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0246.htm) |

The following CS0234 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error can occur when an imported DLL (could be a 3rd party indicator) you are referencing no longer exists / has been removed.

To resolve this the DLL must be re-imported.

**To re-import a 3rd party dll:**

1.Open the NinjaScript Editor via New > NinjaScript editor.

2.Right mouse click in the NinjaScript Editor main window and select the menu name "References"

3.In the "References" dialog window press the button "Add"

4.Select the 3rd party DLL

|  |
| --- |
| **Warning**: Please make sure in this step to select only the 'true' DLL file needed for reference, which would not contain any X86 or X64 suffixes in its file-name, otherwise you could run into compile issues later. |

**Error Descriptions**

The type or namespace name '<name>' could not be found (are you missing a using directive or an assembly reference?)

The type or namespace name '<name>' does not exist in the namespace 'NinjaTrader.Indicator' (are you missing an assembly reference?)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS0234** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0201.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0246.htm) |

The following CS0234 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error can occur when an imported DLL (could be a 3rd party indicator) you are referencing no longer exists / has been removed.

To resolve this the DLL must be re-imported.

**To re-import a 3rd party dll:**

1.Open the NinjaScript Editor via New > NinjaScript editor.

2.Right mouse click in the NinjaScript Editor main window and select the menu name "References"

3.In the "References" dialog window press the button "Add"

4.Select the 3rd party DLL

|  |
| --- |
| **Warning**: Please make sure in this step to select only the 'true' DLL file needed for reference, which would not contain any X86 or X64 suffixes in its file-name, otherwise you could run into compile issues later. |

**Error Descriptions**

The type or namespace name '<name>' could not be found (are you missing a using directive or an assembly reference?)

The type or namespace name '<name>' does not exist in the namespace 'NinjaTrader.Indicator' (are you missing an assembly reference?)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS0428** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0246.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0443.htm) |

The following CS0428 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error can occur when you miscall a method such as indicator methods.

If you are calling an indicator please ensure that you have both the parameters '()' and the indexing value '[]' set. For other methods please ensure you pass all required parameters through the parameters set '()'.

**Error Description #1**  
Cannot convert method group 'SMA' to non-delegate type 'double'. Did you intend to invoke the method?

**Example #1**

// Erroneous Sample Code - SMA() indicator method is improperly called

double myValue = SMA;

// Resolution Sample Code - SMA() indicator method is properly called

double myValue = SMA(5)[0];

**Example #2**

// Erroneous Sample Code - ToString is a method and requires round brackets () to be properly called

string str = Close[5].ToString;

// Resolution Sample Code - ToString() is properly called

string str = Close[5].ToString();

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS0443** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0428.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1002.htm) |

The following CS0443 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error is most commonly invoked when no index value is used inside the indexing brackets.

Please ensure you place a value inside the '[]'.

**Error Description #1**  
Syntax error, value expected

// Erroneous Sample Code - Missing index value

double myValue = SMA(20)[];

// Resolution Sample Code - 'myValue' takes on the current bar's SMA(20) value

double myValue = SMA(20)[0];

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS1002** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs0443.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1061.htm) |

The following CS1002 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error can be invoked when statements are not ended properly.

All statement lines must be closed with a semicolon.

**Error Description #1**  
; expected

// Erroneous Sample Code - Statement is not closed

double myValue = SMA(20)[0]

// Resolution Sample Code - Statement is closed

double myValue = SMA(20)[0];

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS1061** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1002.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1501.htm) |

The following CS1061 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error's code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error can occur when you try to use a method or access an exposed property that does not exist for your particular object.

Please check the methods and exposed property available for your particular object.

**Error Description #1**  
'NinjaTrader.Indicator.CurrentDayOHL' does not contain a definition for 'CurentOpen'

// Erroneous Sample Code - CurrentDayOHL()’s property is 'CurrentOpen' not 'CurentOpen' (typo)

double value = CurrentDayOHL().CurentOpen[0];

// Resolution Sample Code - 'CurrentOpen' property available

double value = CurrentDayOHL().CurrentOpen[0];

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS1501** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1061.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1502.htm) |

The following CS1501 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error can occur when you use use an overload (method parameter signature) that does not exist. This could be because you are passing in 3 arguments when the method only requires 2.

You can cycle through the available overloads with the use of the up and down arrows on the Intelliprompt when you call an indicator method or any other method.

**Error Description #1**  
No overload for method 'SMA' takes '0' arguments

**Example #1**

// Erroneous Sample Code - SMA() does not contain an overload that has 3 arguments

double myValue = SMA(Close, 5, 2)[0];

// Resolution Sample Code - SMA() has an overload consisting of 2 arguments

double myValue = SMA(Close, 5)[0];

**Example #2**

// Erroneous Sample Code - EMA() does not contain an overload that has 0 arguments

double myValue = EMA()[0];

// Resolution Sample Code - EMA() has an overload consisting of 1 argument

double myValue = EMA(5)[0];

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS1502** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1501.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1503.htm) |

The following CS1502 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error can occur when you pass in incorrect parameter object types into a method such as an indicator.

Please check the overload methods for the proper parameter object types and pass in the proper object. You can check the overload methods with NinjaScript editor’s Intelliprompt when you call a method.

**Error Description #1**  
The best overloaded method match for 'NinjaTrader.NinjaScript.StrategyBase.SetStopLoss(CalculationMode, double)' has some invalid arguments

// Erroneous Sample Code - Close is a Series<double> object type and is not a valid value to the SetStopLoss() method

SetStopLoss(CalculationMode.Price, Close);

// Resolution Sample Code - The SetStopLoss() method takes a double value so pass in Close[0]

SetStopLoss(CalculationMode.Price, Close[0]);

**Error Description #2**  
The best overloaded method match for 'NinjaTrader.Indicator.Indicator.SMA(NinjaTrader.NinjaScript.ISeries<double>, int)' has some invalid arguments

// Erroneous Sample Code - Using an integer when the first parameter should be a Series<double>

double myValue = SMA(5, 5);

// Resolution Sample Code - 'myValue' will take the value of the current bar's SMA

double myValue = SMA(Close, 5)[0];

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS1503** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1502.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1513.htm) |

The following CS1503 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

This error can occur when you try to assign a value to a [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) that is not of the correct value type.

Series<double> objects can only contain double values. Series<bool> objects can only contain bool values. Etc.

**Error Description #1**  
Cannot implicity convert type from 'string' to 'double'

// Erroneous Sample Code - Cannot pass in a string to a Series<double>

Value[0] = "Close[0]";

// Resolution Sample Code - Sets Series<double> to the current bar's Close value

Value[0] = Close[0];

**Error Description #2**  
Cannot implicitly convert type 'NinjaTrader.NinjaScript.Indicators.SMA' to 'double'

// Erroneous Sample Code - Cannot pass in a Series<double> object to a Series<double> Set() method

Values[0] = SMA(20);

// Resolution Sample Code - Sets Series<double> to the current bar's SMA(20) value

Values[0] = SMA(20)[0];

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **CS1525** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1513.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/nodoc.htm) |

The following CS1525 error code information is provided within the context of NinjaScript. The examples provided are only a subset of potential problems that this error code may reflect. In any case, the examples below provide a reference of coding flaw possibilities.

**Error Code Explanation**

The compiler detected an invalid character in an expression.

**Error Description #1**  
{ expected

// Erroneous Sample Code - If statement is not opened  
protected override void OnBarUpdate()  
{  
  if(IsFirstTickOfBar)        
}

// Resolution Sample Code - If statement is open and closed

protected override void OnBarUpdate()  
{  
  if (IsFirstTickOfBar)  
  {  
    // do something  
  }  
}

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Editor](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) > [Compile Error Codes](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) >  **NoDoc** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/cs1525.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/compile_error_codes.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/ns_editor_components.htm) |

Unfortunately we do not have NinjaScript context based Help information on this specific error code. You can check the [Microsoft MSDN site section on error codes](http://msdn.microsoft.com/en-us/library/5feh24w0(VS.71).aspx) for futher information.

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Distribution](https://ninjatrader.com/es/support/helpGuides/nt8/distribution.htm) >  **Considerations For Compiled Assemblies** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/distribution.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/distribution.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/import.htm) |

**Using Compiled Assemblies**

Compiled assemblies (DLL's) allow you to bundle your scripts into a format that hides your proprietary code along with any supporting resources. Compiled assemblies provide distinct benefits, especially for commercially distributed code, but there are a few considerations to keep in mind. Typecasting and building resource files (sounds, images, etc.) into your assemblies must be approached differently to ensure cleanly packaged, error-free DLL's.

**Casting Types in a DLL (Using dynamic Types)**

Sometimes, you may need to cast your objects to NinjaScript types, such as when iterating through the DrawObjects collection to obtain a reference to a particular Drawing Object on a chart. When running C# code which has not been compiled into an assembly, typecasting can be done normally, as in the example below:

| ns **Typecasting in code outside of a compiled assembly** |
| --- |
| **protected** **override** **void** OnBarUpdate() {   **foreach**(HorizontalLine line **in** DrawObjects)   {       *// Print the tag of each Horizontal Line on the chart*       Print(String.Format("Horizontal Line {0} found.", line.Tag));   } } |

An obstacle arises with traditional typecasting in a compiled assembly, since the NinjaScript Type you attempt to cast will be present in both your DLL and NinjaTrader's Custom.dll assembly. If you plan to compile your code into a DLL, you will need to use the [dynamic Type](https://msdn.microsoft.com/en-us/library/dd264741.aspx) to avoid this conflict by dynamically assigning the Type at runtime, using the guidelines below:

1.Define a variable of Type 'dynamic'

2.Assign a reference to the needed object to the dynamic variable

3.Access the dynamic variable as if it were of the expected Type

| ns D**ynamic variables as an alternative to typecasting inside of a compiled assembly** |
| --- |
| foreach (dynamic line in DrawObjects.ToList()) {   // Use ToString().Equals() to detect the object's Type   if (line.ToString().Equals("NinjaTrader.NinjaScript.DrawingTools.HorizontalLine"))   {       // Access the object by assuming that it is the Type we expect       Print(String.Format("Horizontal Line {0} detected!", line.Tag));   } } |

The above dynamic approach will work for primitive types. For instantiating more complex types / classes though, such as adding a new [PriceLevel](https://ninjatrader.com/es/support/helpGuides/nt8/pricelevels.htm) programmatically to an existing drawing tool, [Reflection](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/reflection) would need to used.

| ns **Instantiating more complex types such as the PriceLevels class inside of a compiled assembly** |
| --- |
| foreach (dynamic dt in DrawObjects.ToList()) {   if(dt.ToString().Equals("NinjaTrader.NinjaScript.DrawingTools.FibonacciRetracements"))   {     Type type         = dt.PriceLevels.GetType().GetGenericArguments()[0];     Assembly assembly = type.Assembly;     var pl           = assembly.CreateInstance(type.FullName, false, BindingFlags.CreateInstance, null, new object[] { 55.5, Brushes.Red, 2 }, new       System.Globalization.CultureInfo("en-US"), new object[] {});     dt.PriceLevels.GetType().GetMethod("Add").Invoke(dt.PriceLevels, new object[] { pl } );     this.ForceRefresh();   } } |

**Working With the dynamic Type**

Using dynamic variables in the technique above requires careful attention to accessing members appropriately, and thus should be avoided if you do not intend to use or distribute compiled assemblies.

•**No Intelliprompt**: Since the compiler cannot know which Type you assume a dynamic variable to be, no intelliprompt will be displayed to help search through Type members. The same applies to Visual Studio's Intellisense or similar utilities

•**No Compile Errors**: For the same reason, the compiler cannot know if you are using the variable in a way not supported by its expected Type, trying to access members not present in that Type, or other related errors. Thus, any such errors which would be caught by the compiler when typecasting will be missed, and will result in runtime errors instead. If a runtime error were to be triggered, the error may be more difficult to interpret.

oExample: If you tried to access "line.tag" (improper capitalization) in the examples above, you would receive the following errors:

▪Typecasting / Compile Error: *"'NinjaTrader.NinjaScript.DrawingTools.HorizontalLine' does not contain a definition for 'tag' and no extension method accepting a first argument of type 'NinjaTrader.NinjaScript.DrawingTools.HorizontalLine' can be found (are you missing a using directive or an assembly reference?)"*

▪dynamic / Runtime Error: *"Error on calling 'OnBarUpdate' method on bar 0: 'NinjaTrader.NinjaScript.DrawingTools.DrawingTool.tag' is inaccessible due to its protection level"*

**Adding XAML and Other Files Into a DLL**

When [exporting a compiled assembly](https://ninjatrader.com/es/support/helpGuides/nt8/export.htm) through NinjaTrader, no additional resource files can be added. There are two ways around this. The first is to export the DLL from NinjaTrader, then open the exported .zip file, add any additional files, and re-zip the archive, but this will result in your resource files being fully accessible to end users. The second and recommended approach is to use a fully featured IDE such as Visual Studio to build your DLL's.

For more information on how to accomplish this with Visual Studio, see the "AddOn Development Environment" section of the [AddOn Development Overview](https://ninjatrader.com/es/support/helpGuides/nt8/addon_development_overview.htm) page. Although the page focuses on AddOn development, the sample project it provides can be used to develop other NinjaScript Types, as well.

**Exporting custom drawing tools as assembly / DLL**

When planning to distribute your custom drawing tools via assemblies, please understand it's paramount that you implement your own Draw. method to allow the drawing tool getting called programmatically by other NinjaScript objects.

The NinjaTrader default drawing tools would implement this via a partial class, for example you would see -

| ns **Default**NinjaTrader **drawing tool Draw. method handling** |
| --- |
| public static partial class Draw {  } |

However since partial classes could not span across two assemblies, therefore a custom non partial Draw. method for your NinjaScript drawing tool would be needed.

| ns **Custom drawing tool Draw. method handling** |
| --- |
| public static class MyDrawCustom {  } |

**Exports might not be backwards compatible**

NinjaScript exports might not be backwards compatible with previous versions of NinjaTrader.

This is known to happen every time a new type (e.g. Enum) was introduced, since the newly introduced types are not known to prior releases of NinjaTrader

Typically an error message like the following would be seen:

*"Error on calling 'SetState' method: Could not load type 'NinjaTrader.NinjaScript.Indicators.CumulativeDeltaType' from assembly 'NinjaTrader.Vendor, Version=8.0.12.0, Culture=neutral, PublicKeyToken=null'."*

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Developing for Tick Replay** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/c_method_functions_reference.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/historical_order_backfill_logic.htm) |

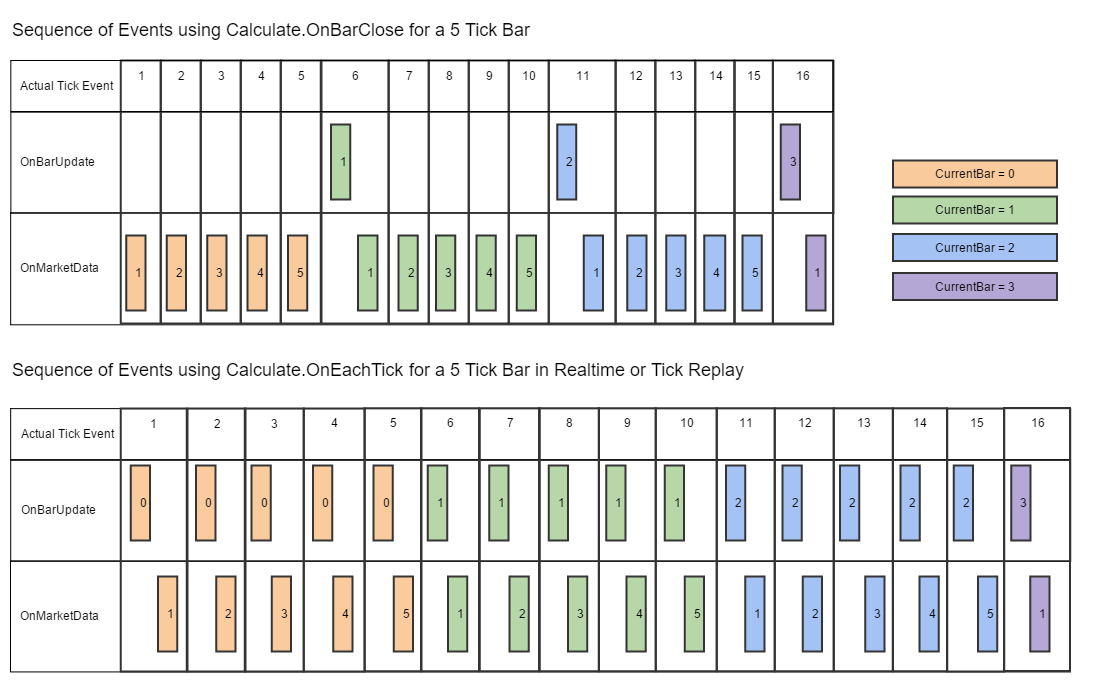
Tick Replay is used to playback 1 tick historical data to build the bars as if they had been build live, this means that tick data will be thrown as Market Data events in historical and subsequently OnMarketData and OnBarUpdate events will be called as if it was live. This provides more granular tick related information and can be helpful if you need to know the most recent last price, last volume, best ask price, or best bid price that occurred on historical data during the bar. An indicator or strategy running Tick Replay needs to have been specifically designed to take advantage of Tick Replay.  In general, this means adding additional logic to the [OnMarketData()](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm) event handler, however, Tick Replay can also be used to call [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) "OnEachTick" or "OnPriceChange" during historical calculations.

**How to Enable Tick Replay**

To enable tick replay it **must** be manually enabled on the primary [Data Series](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_price_data.htm) and the option to allow this mode is hidden by default. The option to allow for Tick Replay is located in **Tools** > **Options** > **Market Data** > "**Show Tick Replay**". The reason why it is hidden by default is that the tick replay engine utilizes 1 tick data to build historical bars. TickReplay can generate thousands of events per bar and may take an excessive amount of time to load.  It is recommended to optimize your indicators that you plan to calculate on such data by only running them in Calculate On Bar Close mode or reducing the amount of data to load to the minimum amount of data required. Since bars are built with tick data you will only be able to build bars back as far as your historical data provider allows download of tick data.

**How the Tick Replay Engine Works**

Tick Replay guarantees an exact sequence of stored events are played back for both the [OnBarUpdate](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) and [OnMarketData](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm) events.  This mode also ensures the **OnMarketData**event is called after every **OnBarUpdate** event used to build the current bar.  Consider the following examples with Tick Replay enabled on a 5-tick input series, each box is when each event occurs during Tick Replay simulation.



As you can see from the table above, the **Calculate** setting will have a varying degree of impact on how your indicator or strategies **OnBarUpdate** event is raised. This process repeats for every historical bar on the chart and would continue as the indicator or strategy transitions to real-time data.

**Accessing the current best bid and ask at the time of a trade**

NinjaTrader stores the best bid price and best ask price as the last trade occurs during the[MarketDataType.Last](https://ninjatrader.com/es/support/helpGuides/nt8/marketdataeventargs.htm) event and provides it per the table below:

|  |  |
| --- | --- |
| marketDataUpdate.Price | The current market data price of the last trade event |
| marketDataUpdate.Ask | The current asking price at the time of the last trade event |
| marketDataUpdate.Bid | The current bidding price at the time of the last trade event |
| marketDataUpdate.Volume | The current market data volume of the last trade event |
| marketDataUpdate.Time | The current time of the last trade event |

An example below shows how to access historical Bid and Ask prices with Tick Replay

| ns **Accessing the current best bid and ask at the time of a trade** |
| --- |
| protected override void OnMarketData(MarketDataEventArgs marketDataUpdate) {   // TickReplay events only occur on the "Last" market data type   if (marketDataUpdate.MarketDataType == MarketDataType.Last)   {     if (marketDataUpdate.Price >= marketDataUpdate.Ask)     {         Print(marketDataUpdate.Volume + " contracts traded at asking price " + marketDataUpdate.Ask);     }       else if (marketDataUpdate.Price <= marketDataUpdate.Bid)     {         Print(marketDataUpdate.Volume + " Contracts Traded at bidding price " + marketDataUpdate.Bid);     }   } } |

**Calling a Tick Replay indicator from another Indicator or Strategy**

A hosting indicator or strategy must be aware of the requirement to run through another indicator's historical Tick Replay data before it reaches [State.Historical](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm). To achieve desired results, you either need to store the reference in **State.DataLoaded** or (for a strategy) you can call [AddChartIndicator().](https://ninjatrader.com/es/support/helpGuides/nt8/addchartindicator.htm)  Either approach ensures that the hosting indicator or strategy is aware of the requirements to process Tick Replay during its **State.Historical** mode and helps to ensure that the hosted indicator calculates as designed up to its current bar using Tick Replay.  Please see the example below.

| ns **Calling a Tick Replay indicator from another Indicator or Strategy** |
| --- |
| TickReplayIndicator myTickReplayIndicator = null;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {       Name   = "TestHost";   }   else if (State == State.DataLoaded)   {       // Store a reference to the Tick Replay indicator before State.Historical       // Doing so ensures the hosted indicator will run through Tick Replay       myTickReplayIndicator = TickReplayIndicator();         // For a strategy, you can just call AddChartIndicator(TickReplayIndicator());       // However this also adds a copy of the indicator to the chart, which may or may not be desired       // For calculation purposes only, storing the reference should all that needs to be required.   } }   protected override void OnBarUpdate() {   // Access the stored reference which calculates through   // historical Tick Replay data and print the value as expected   Print(myTickReplayIndicator[0]); } |

|  |  |
| --- | --- |
| **Notes**:    1.Tick Replay was **NOT** designed to provide accuracy in backtesting concerning order fills and execution and should **NOT** be used to expect the exact sequence of executions as running a strategy on live data. For greater order-fill resolution and accuracy in strategy backtesting, you can use the [High Fill Resolution in the Strategy Analyzer](https://ninjatrader.com/es/support/helpGuides/nt8/understanding_historical_fill_.htm). Furthermore you cannot combine both Tick Replay and High Order Fill resolution.  2.If the data provided has no bid/ask data tied to the last tick data, NinjaTrader substitutes the bid/ask data for consistent user experience purposes (i.e., Bid = Last price, Ask = Bid + 1 tick).   For a list of providers who support tick replay, please see the table from [Understanding the data provided by your connectivity provider](https://ninjatrader.com/es/support/helpGuides/nt8/data_by_provider.htm). Only bid and ask price is made available, bid and ask volume is **NOT** available.  3.Tick Replay **ONLY** replays the Last market data event, and only stores the best inside bid/ask price at the time of the last trade event.  You can think of this as the equivalent of the bid/ask price at the time a trade was reported.  As such, historical bid/ask market data events (i..e, bid/ask volume) **DO** **NOT** work with Tick Replay.  To obtain those values, you need to use a [historical bid/ask series](https://ninjatrader.com/es/support/helpGuides/nt8/using_historical_bid_ask_serie.htm) separately from TickReplay through OnBarUpdate()  4.Tick Replay data is accessed via the [MarketDataEventArgs](https://ninjatrader.com/es/support/helpGuides/nt8/marketdataeventargs.htm) object passed into [OnMarketData()](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm) events, rather than attempting to access it via [GetCurrentAsk()](https://ninjatrader.com/es/support/helpGuides/nt8/getcurrentask.htm) and [GetCurrentBid()](https://ninjatrader.com/es/support/helpGuides/nt8/getcurrentbid.htm), which are methods designed to function on real-time data only  5.Due to the nature of how some unique bars build, Tick Replay is **NOT** available for all bar types.  For example, the default **Renko** and **LineBreak** bars which use [RemoveLastBar()](https://ninjatrader.com/es/support/helpGuides/nt8/removelastbar.htm) are not compatible with Tick Replay.  Other custom bar types which use similar methods encounter the same limitation  6.Tick Replay is forced for all series loaded, and there is **NOT** any method to reduce the number of calculations on a per series basis.  In other words, you cannot mix and match tick replay series with non-tick replay series  7.Tick replay was only **ONLY** designed to work with **MarketDataType.Last**.  A TickReplay indicator or strategy should **NOT** be mixed with a**MarketDataType.Ask** or **MarketDataType.Bid** series | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Multi-Threading Consideration for NinjaScript** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/historical_order_backfill_logic.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) |

**Multi-Threading Overview**

With the introduction of multi-threading in NinjaTrader special considerations should be made when programming your NinjaScript objects. Multi-threading basically allows NinjaTrader to take advantage of multi-core CPUs commonplace in modern computing to do multiple tasks at the same time.  While this has many advantages for multi-tasking, it can cause new types of issues you may have not needed to consider before.  This page was designed to serve as a high-level overview of some of the most common scenarios that can arise due to multi-threading, but should not be considered an exhaustive list.

**Using A Dispatcher**

Depending on your CPU configuration, the NinjaTrader application will usually consist of multiple main UI threads, where various features like Charts or NinjaScript objects run, along with a number of background worker threads where events such as market data updates will be distributed throughout the product.  In principle, an object can only access information related to objects that exist on the same thread.  It is possible (and quite likely), that the thread which a NinjaScript object is running will not be the same thread as the event which is calling the object.  In cases where you need to access objects on the UI from a NinjaScript objects calling event thread, a [dispatcher](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcher(v=vs.110).aspx) can be used.

|  |
| --- |
| **Note**:  As a best practice, you should always make sure to use [Dispatcher.InvokeAsync()](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcher.invokeasync(v=vs.110).aspx)to ensure your action is done asynchronously to any internal NinjaTrader actions.  Calling the synchronous **Dispatcher.Invoke()** method can potentially result in a deadlock scenarios as your script is loaded. |

| ns |
| --- |
| if (State == State.Historical) {   if (ChartControl != null)   {     // add some text to the UserControlCollection through the ChartControls dispatcher     ChartControl.Dispatcher.InvokeAsync(new Action(() => {         UserControlCollection.Add(new System.Windows.Controls.TextBlock {           Text = "\nAdded by the ChartControl Dispatcher."         });     }));   } } |

**Thread Access**

Since market data is distributed across the entire application by a randomly assigned UI thread, there is no guarantee that your object will be running on the same event thread that is calling the object. Therefore it is recommend that you call [Dispatcher.CheckAccess()](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcher.checkaccess(v=vs.110).aspx)in order to test if you truly need to dispatch the requested action.

| ns |
| --- |
| // check if the current object is already on the calling thread if (Dispatcher.CheckAccess()) {   // execute action directly   action(args); } // otherwise run the action from the thread that created the object else {   // dispatch action to calling thread   Dispatcher.InvokeAsync(action, args); } |

**Cross Thread Exceptions**

When accessing objects included on the UI, you may receive the following error if you attempt to access a certain property/method from the wrong thread:

**"Error on calling 'OnBarUpdate' method on bar 0: You are accessing an object which resides on another thread. I.E. creating your own Brush without calling .Freeze(), or trying to access a UI control from the wrong thread without using a Dispatcher"**

This error can be avoided by invoking the **Dispatcher** used on the appropriate UI thread.

**Access Violation Exception**

Should you be using custom resources like text files, static members, etc. it is important to protect your resources from concurrent access. If NinjaTrader tried to use the resource at the same time you would run into errors similar to this one:

**8/20/2010 12:14:29 PM|3|128|Error on calling 'OnBarUpdate' method for strategy 'SampleStrategy/1740b50bfe5d4bd896b0533725622400': The process cannot access the file 'c:\sample.txt' because it is being used by another process.**

| ns |
| --- |
| private object lockObj = new object();   private void WriteFile() {   // lock a generic object to ensure only one thread is accessing the following code block at a time   lock (lockObj)   {       string filePath = @"C:\sample.txt";       using (System.IO.FileStream file = new System.IO.FileStream(filePath, FileMode.Append, FileAccess.Write, FileShare.None)       {         // write something to the file...           // be sure to flush the buffer so everything is written to the file.         file.Flush();           // The "using" block implicitly closes the FileStream object,         // giving other threads access to the file       }   } } |

**Multi-threaded consideration for Order, Execution and Position objects**

These considerations apply to the [OnOrderUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onorderupdate.htm), [OnExecutionUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onexecutionupdate.htm) and [OnPositionUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onpositionupdate.htm) handlers, where both the actual 'core' objects are passed by reference and updating method value parameters are provided. Examplary the [OnOrderUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onorderupdate.htm) is discussed in below.

•[OnOrderUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onorderupdate.htm) method guarantees that you will see each order state change in sequence

•The "order" method parameter represents the core order object updated by NinjaTrader

•The supplementary method parameters provide an updating value representing each order change in sequence.  Think of this as the relevant information on the order at the time the state changed.

•Since the "order"  method parameter represents the current order object state, it is possible for the updating values of that object to be out of sync with the correspond method parameters during a particular order update event.

As an example, the NinjaTrader core may have received "Working" and then "PartFilled" order state change events back from the broker API on thread "B". At some point in time (milliseconds later) the NinjaTrader core will take these events and trigger the OnOrderUpdate() method in the strategy on thread "A". Thus, when the strategy receives the first "Working" state for an order, the **orderState** method parameter will reflect the "Working" state although the actual **order.OrderState** is really in a state of "Part Filled".  You would see that current value truly reflected in the core Order object method parameter or any order objects returned in any of the order methods such as EnterLong(). Of course, the OnOrderUpdate() method parameters will eventually receive the event for "PartFilled" state in the sequence the events were received.

Considering the concept above, if you are unsure if you should be using the core order object value vs the updating method parameter value value, ask your self if you are truly looking for the *most current order state*, or the *sequence of order states*:

•For the most current order state, use the core "order" object property (e.g., order.OrderState, order.LimitPrice, order.StopPrice, etc)

•For the sequence of order states, use the updating method parameter value (e.g., orderState, limitPrice, stopPrice, etc)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Multi-Threading Consideration for NinjaScript** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/historical_order_backfill_logic.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) |

**Multi-Threading Overview**

With the introduction of multi-threading in NinjaTrader special considerations should be made when programming your NinjaScript objects. Multi-threading basically allows NinjaTrader to take advantage of multi-core CPUs commonplace in modern computing to do multiple tasks at the same time.  While this has many advantages for multi-tasking, it can cause new types of issues you may have not needed to consider before.  This page was designed to serve as a high-level overview of some of the most common scenarios that can arise due to multi-threading, but should not be considered an exhaustive list.

**Using A Dispatcher**

Depending on your CPU configuration, the NinjaTrader application will usually consist of multiple main UI threads, where various features like Charts or NinjaScript objects run, along with a number of background worker threads where events such as market data updates will be distributed throughout the product.  In principle, an object can only access information related to objects that exist on the same thread.  It is possible (and quite likely), that the thread which a NinjaScript object is running will not be the same thread as the event which is calling the object.  In cases where you need to access objects on the UI from a NinjaScript objects calling event thread, a [dispatcher](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcher(v=vs.110).aspx) can be used.

|  |
| --- |
| **Note**:  As a best practice, you should always make sure to use [Dispatcher.InvokeAsync()](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcher.invokeasync(v=vs.110).aspx)to ensure your action is done asynchronously to any internal NinjaTrader actions.  Calling the synchronous **Dispatcher.Invoke()** method can potentially result in a deadlock scenarios as your script is loaded. |

| ns |
| --- |
| if (State == State.Historical) {   if (ChartControl != null)   {     // add some text to the UserControlCollection through the ChartControls dispatcher     ChartControl.Dispatcher.InvokeAsync(new Action(() => {         UserControlCollection.Add(new System.Windows.Controls.TextBlock {           Text = "\nAdded by the ChartControl Dispatcher."         });     }));   } } |

**Thread Access**

Since market data is distributed across the entire application by a randomly assigned UI thread, there is no guarantee that your object will be running on the same event thread that is calling the object. Therefore it is recommend that you call [Dispatcher.CheckAccess()](https://msdn.microsoft.com/en-us/library/system.windows.threading.dispatcher.checkaccess(v=vs.110).aspx)in order to test if you truly need to dispatch the requested action.

| ns |
| --- |
| // check if the current object is already on the calling thread if (Dispatcher.CheckAccess()) {   // execute action directly   action(args); } // otherwise run the action from the thread that created the object else {   // dispatch action to calling thread   Dispatcher.InvokeAsync(action, args); } |

**Cross Thread Exceptions**

When accessing objects included on the UI, you may receive the following error if you attempt to access a certain property/method from the wrong thread:

**"Error on calling 'OnBarUpdate' method on bar 0: You are accessing an object which resides on another thread. I.E. creating your own Brush without calling .Freeze(), or trying to access a UI control from the wrong thread without using a Dispatcher"**

This error can be avoided by invoking the **Dispatcher** used on the appropriate UI thread.

**Access Violation Exception**

Should you be using custom resources like text files, static members, etc. it is important to protect your resources from concurrent access. If NinjaTrader tried to use the resource at the same time you would run into errors similar to this one:

**8/20/2010 12:14:29 PM|3|128|Error on calling 'OnBarUpdate' method for strategy 'SampleStrategy/1740b50bfe5d4bd896b0533725622400': The process cannot access the file 'c:\sample.txt' because it is being used by another process.**

| ns |
| --- |
| private object lockObj = new object();   private void WriteFile() {   // lock a generic object to ensure only one thread is accessing the following code block at a time   lock (lockObj)   {       string filePath = @"C:\sample.txt";       using (System.IO.FileStream file = new System.IO.FileStream(filePath, FileMode.Append, FileAccess.Write, FileShare.None)       {         // write something to the file...           // be sure to flush the buffer so everything is written to the file.         file.Flush();           // The "using" block implicitly closes the FileStream object,         // giving other threads access to the file       }   } } |

**Multi-threaded consideration for Order, Execution and Position objects**

These considerations apply to the [OnOrderUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onorderupdate.htm), [OnExecutionUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onexecutionupdate.htm) and [OnPositionUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onpositionupdate.htm) handlers, where both the actual 'core' objects are passed by reference and updating method value parameters are provided. Examplary the [OnOrderUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onorderupdate.htm) is discussed in below.

•[OnOrderUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onorderupdate.htm) method guarantees that you will see each order state change in sequence

•The "order" method parameter represents the core order object updated by NinjaTrader

•The supplementary method parameters provide an updating value representing each order change in sequence.  Think of this as the relevant information on the order at the time the state changed.

•Since the "order"  method parameter represents the current order object state, it is possible for the updating values of that object to be out of sync with the correspond method parameters during a particular order update event.

As an example, the NinjaTrader core may have received "Working" and then "PartFilled" order state change events back from the broker API on thread "B". At some point in time (milliseconds later) the NinjaTrader core will take these events and trigger the OnOrderUpdate() method in the strategy on thread "A". Thus, when the strategy receives the first "Working" state for an order, the **orderState** method parameter will reflect the "Working" state although the actual **order.OrderState** is really in a state of "Part Filled".  You would see that current value truly reflected in the core Order object method parameter or any order objects returned in any of the order methods such as EnterLong(). Of course, the OnOrderUpdate() method parameters will eventually receive the event for "PartFilled" state in the sequence the events were received.

Considering the concept above, if you are unsure if you should be using the core order object value vs the updating method parameter value value, ask your self if you are truly looking for the *most current order state*, or the *sequence of order states*:

•For the most current order state, use the core "order" object property (e.g., order.OrderState, order.LimitPrice, order.StopPrice, etc)

•For the sequence of order states, use the updating method parameter value (e.g., orderState, limitPrice, stopPrice, etc)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Multi-Time Frame & Instruments** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/multi-threading.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/understanding_the_lifecycle_of.htm) |

**Multi-Series Scripting Overview**

NinjaScript supports multiple time frames and instruments in a single script. This is possible because you can add additional Bars objects to indicators or strategies, in addition to the primary Bars object to which they are applied. A Bars object represents all of the bars of data on a chart. For example, if you had a MSFT 1 minute chart with 200 bars on it, the 200 bars represent one Bars object. In addition to adding Bars objects for reference or for use with indicator methods, you can execute trades across all the different instruments in a script. There is extreme flexibility in the NinjaScript model that NinjaTrader uses for multiple-bars scripts, so it is very important that you understand how it all works before you incorporate additional Bars objects in a script. An important fact to understand is that NinjaScript is truly event driven; every Bars object in a script will call the [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) method. The significance of this will become evident throughout this page.

|  |
| --- |
| **Note**:  If using [OnMarketData()](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm), a subscription will be created on all bars series added in your indicator or strategy strategy (even if the instrument is the same).  The market data subscription behavior occurs both in real-time and during [TickReplay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) historical |

It is also important that you understand the following method and properties:

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

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

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

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

|  |
| --- |
| **Note**:  As we move through this section, the term "Primary Bars" will be used and for the purpose of clarification, this will always refer to the first Bars object loaded into a script. For example, if you apply a script on MSFT 1 minute chart, the primary Bars would be MSFT 1 minute data set.    **This section is written in sequential fashion. Example code is re-used and built upon from sub section to sub section.** |

tog_minus        [Working With Multi-Time Frame Objects](javascript:HMToggle('toggle','WorkingWithMultiTimeFrameObjects','WorkingWithMultiTimeFrameObjects_ICON'))

tog_minus        [Adding Additional Bars Objects to NinjaScript](javascript:HMToggle('toggle','AddingAdditionalBarsObjectToninjascript','AddingAdditionalBarsObjectToninjascript_ICON'))

tog_minus        [Creating Series<T> Objects](javascript:HMToggle('toggle','CreatingSeriesObjects','CreatingSeriesObjects_ICON'))

tog_minus        [How Bars Data is Referenced](javascript:HMToggle('toggle','HowBarsDataIsReferenced','HowBarsDataIsReferenced_ICON'))

tog_minus        [Using Bars Objects as Input to Indicator Methods](javascript:HMToggle('toggle','UsingBarsObjectsAsInputToIndicatorMethods','UsingBarsObjectsAsInputToIndicatorMethods_ICON'))

tog_minus        [True Event Driven OnBarUpdate() Method](javascript:HMToggle('toggle','TrueEventDrivenOnbarupdateMethod','TrueEventDrivenOnbarupdateMethod_ICON'))

tog_minus        [Accessing the Price Data in a Multi-Bars NinjaScript](javascript:HMToggle('toggle','AccessingThePriceDataInAMultibarsninjascript','AccessingThePriceDataInAMultibarsninjascript_ICON'))

tog_minus        [Entering, Exiting and Retrieving Position Information](javascript:HMToggle('toggle','EnteringExitingAndRetrievingPositionInformation','EnteringExitingAndRetrievingPositionInformation_ICON'))

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **NinjaScript Lifecycle** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_3rd_party_indicators.htm) |

NinjaTrader uses a [State](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) change system to represent various life cycles of your NinjaScript object.  For more basic indicators and strategies, simply understanding each **State** described on the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) page is sufficient.  However, for more advanced development projects, it is critical to understand how NinjaTrader calls these states for various instances throughout the lifetime of the entire application.

**When NinjaTrader instantiates a NinjaScript object**

There are two categories of instances instantiated by NinjaTrader:

•"UI" instances representing its default properties on various user interfaces

•The "configured" instance executing your custom instructions

In both categories, [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) is called at least twice:  once to **State.SetDefaults** acquiring various default property values, and then again to **State.Terminated** handling internal references cleanup.

|  |
| --- |
| **Note**:  It is important to understand that previous major versions of NinjaTrader were not so diligent in running termination logic for UI instances and the current major NinjaTrader 8 version has been changed to help properly address related issues. |

To elaborate on that process, imagine the sequence of user events required to start an indicator on a chart:

1.User right clicks on a Chart and select "**Indicator**"

2.User adds an Indicator from the **Available** list

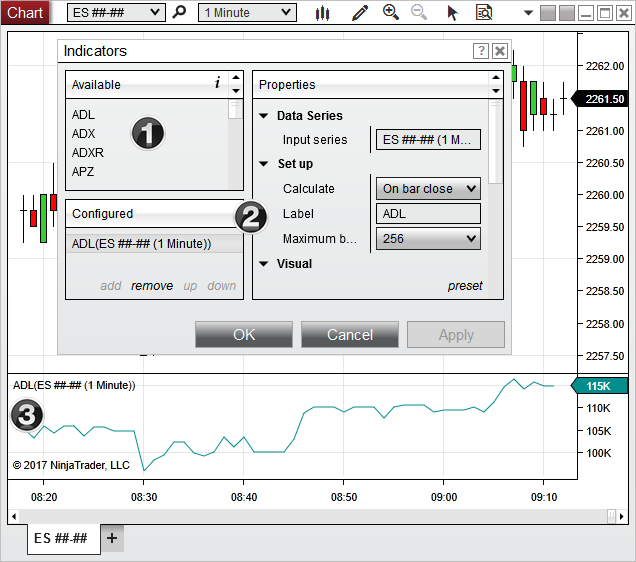
3.User configures desired **Properties** and presses "**Apply**" or "**OK**"

During this sequence, there are actually 3 instances of the same indicator created by NinjaTrader:

1.The instance displaying the **Name** property to the list of "**Available**" indicators (**Note**: this process involves creating an instance of *all* indicators in order to build the complete list)

2.The instance displaying the individual **Name** and its default **Properties**

3.The instance configured and executing on the chart



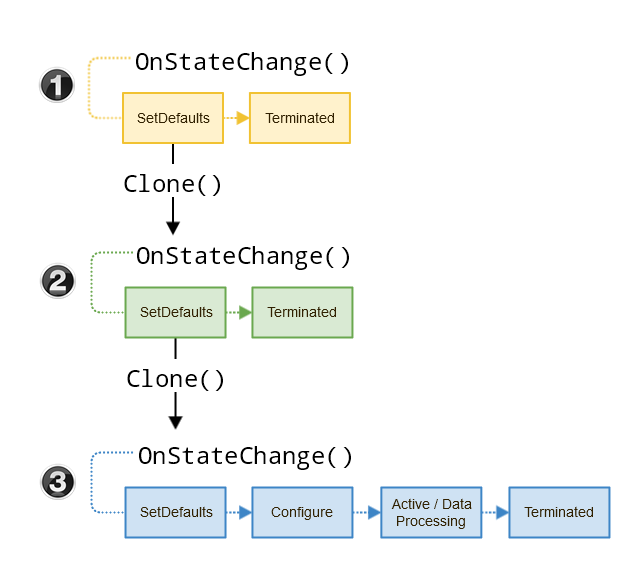
To visualize how each instance goes through its **States**, please consider the logic and flow chart below:

1.In order to display the indicator name in the list of **"Available"** indicators, the NinjaTrader core must find the **Name** of each installed indicator defined in their **SetDefaults**.  This occurs simultaneously for *every indicator installed on the system* in order to build the full list of available indicators.

2.The selected indicator is then [cloned](https://ninjatrader.com/es/support/helpGuides/nt8/clone.htm) and **SetDefaults** is called again in order to display the default properties to the "**Properties**" grid.  This only occurs for the individual indicator.

3.After the user has set their desired property settings and press **OK** or **Apply**, the indicator is once again cloned and runs through its full state management.  This only occurs for the indicator configured to execute on the chart.

|  |
| --- |
| **Warning**:  Since NinjaTrader is multi-threaded, it is possible the **OnStateChange()** logic will be operating on a different thread than your indicator instances.  Due to this fact, if logic in your **OnStateChange()** method is thread sensitivity (e.g., dependent on UI threads vs Instrument threads) please make sure to read the section on [multi-threading considerations](https://ninjatrader.com/es/support/helpGuides/nt8/multi-threading.htm) and check for thread access in your **OnStateChange()** logic |



It is the 3rd "configured" instance you are concerned with developing, but you should also be aware of the "UI" instances which are triggered at various stages of NinjaTrader.

|  |
| --- |
| **Notes**:  1.The example above is written for an indicator, but the same concept of state management applies to every NinjaScript object type  2.The UI instances do not reach **State.Terminated** until the user closes out of the UI feature displaying the object  3.Since [AddOns](https://ninjatrader.com/es/support/helpGuides/nt8/addon_development_overview.htm) run in the background and are not dependent on UI elements, they will run through their **SetDefaults**/**Terminated** states after each NinjaScript compile and startup/shutdown of NinjaTrader.  4.The configured instance will also be cloned back to UI instances during various user actions (e.g, re-opening an indicator dialog to reconfigure settings, or user copying & pasting the indicator to a new panel or chart).  Therefore you should not assume that objects (such as ChartControl) will not be accessible in the UI instances.  5.In some extreme scenarios, you may need to execute custom logic before or after an object is cloned.  Overriding the default behavior can be done via the virtual [Clone()](https://ninjatrader.com/es/support/helpGuides/nt8/clone.htm) method |

**What does this mean for me?**

Since **OnStateChange()** can be called at various times throughout NinjaTrader, you should be diligent in handling each state and managing resources only when it is appropriate that your NinjaScript object was actually configured:

•**State.SetDefaults** should be kept as lean as possible to prevent logic from processing superfluously and causing problems unrelated to the configured instance.  Only properties which need to be displayed on the UI should be set in this state.

•Resources should only be set up once an object has reached **State.Configure** or **State.DataLoaded** (see [best practices](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript_best_practices.htm) for more information)

•**State.Terminated** logic should be specific in when it resets a value or destroys a resource.  Since the running instance can be cloned back to a UI instance, checking that a mutable property exists before accessing sometimes is not enough.  You may need to consider adding a flag to help decide when a resource needs to be reset or destroyed.

**Example**

Let’s say your object was an indicator looking to add a custom toolbar element to the chart, and when the indicator is removed from the chart, you would want to make sure your toolbar elements are also properly removed.  In the [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) handler you change could add the custom toolbar once the **State** has reached **State.Historical**, and remove the toolbar once the State has reached **State.Terminated**.

To ensure that the remove logic only runs in instances that were actually configured, you can see we in the example below we also track that the toolbar needs a reset in **State.Terminated** state via a custom bool variable.  In other words, the proper reset request comes from our configured instance and would be ignored if the **State.Terminated** is called from outside our object (i.e., a UI instance). This will prepare our object to properly handle both situations in which **State.Terminated** could be called in the NinjaTrader state management system.

| ns |
| --- |
| // custom flag to help time termination logic private bool toolBarNeedsReset = false;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "State lifetime indicator";   }   else if (State == State.Historical)   {     // before indicator starts historical processing     // add a custom tool bar using a custom method     AddToolBarButton(); // this is a pseudo-method for example purposes     toolBarNeedsReset = true; // use a flag to track this logic was executed   }     else if (State == State.Terminated)   {     // here we intend to remove the custom tool bar when the indicator shuts down     if (toolBarNeedsReset) // flag is only true after actually added         RemoveToolBarButton();   } } |

**Cloning NinjaScript**

Clone is the operation of iterating over all public browsable properties on a NinjaScript object and duplicating the values over to a freshly generated instance. For the majority of NinjaScript with standard properties the clone process is transparent to you and you do not need to be concerned the the clone process. For those of you that want more control or will be utilizing complex properties then knowledge about clone is essential. Cloning is performed in 2 primary use cases:

1.Configuring an instance in an object dialog and then cloning the configured data to an actual NinjaScript instance applied for example to a Chart. (Configuration then Run)

2.When triggering 'Reload NinjaScript' or "Reload All Historical Data'

NinjaScript objects have a base clone method implemented which will iterating over all browsable properties and copy by value to the next instance. The rules follow the 'clone' rules described in the clone documentation located [here](https://ninjatrader.com/es/support/helpGuides/nt8/clone.htm) and described above. The default behavior will work in almost all cases except for when you have some complex custom property which needs specific clone behavior. In which case we allow the ability to override Clone() and specify your own behavior.

|  |
| --- |
| **Note**: If you plan to utilize complex class properties on NinjaScript, you can specify your own clone method. However when NinjaScript is compiled in NinjaTrader a new DLL holding the compiled IL code is 'hot-loaded' into NinjaTrader. As a user or developer would try to reload NinjaScript or configure an existing NinjaScript object, any complex class will not resolve since the class will be residing in two different assemblies. This problem cannot be solved with custom clone method and workarounds for this are setting Browsable(false) attribute on that property so it is not cloned or putting the property it its own dedicated assembly. |

**Saving NinjaScript Properties to the Workspace via XML Serialization**

XML Serialization comes into play when you have a set of properties and want those properties to persist the user saved workspace (or any templates that are user created).

By default basic types such as int, string, bool will all serialize without issue, if you have a complex property you want its setting maintained on restore then you need to create a string serialized representation of that class. The technique is shown in this example post [here](https://ninjatrader.com/es/support/helpGuides/nt8/user_definable_color_inputs.htm) where we show how to serialize a color brush.

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Using 3rd Party Indicators** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/understanding_the_lifecycle_of.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_atm_strategies.htm) |

**3rd Party Indicators Overview**

You can use 3rd party indicators within your strategies or custom indicators. A 3rd party indicator is an indicator that was not developed by NinjaTrader.

|  |
| --- |
| **Note**:  It is important to understand the functionality provided or **NOT** provided in a 3rd party proprietary indicator. Just because they provide an indicator that displays a bullish or bearish trend on a chart does **NOT** mean that you can access this trend state from their indicator.   It is up to the developer of the indicator to determine what information is accessible. |

3rd party indicators can be provided to you in one of the following ways:

•NinjaScript archive file that can be directly [imported](https://ninjatrader.com/es/support/helpGuides/nt8/import.htm) into NinjaTrader

•A custom installer

•A set of files and instructions for saving them in the correct folders

If you were provided with a NinjaScript archive file that you have successfully imported via the Control Center window "File > Utilities > Import NinjaScript" menu, you can skip over the information below since NinjaTrader automatically configures the indicators ready for use.

If you were provided with a custom installer or a compiled assembly (.DLL) file that you had to manually save in the folder My Documents\NinjaTrader Folder>\bin\Custom then you must follow the instructions below.

**Vendor File**

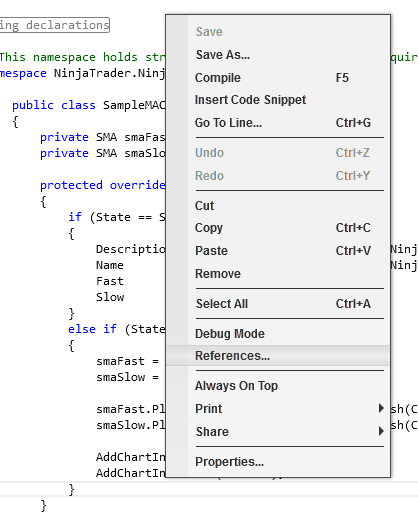
The 3rd party developer should have either installed a "Vendor" file or provided you with one. Its likely in the format "NinjaTrader.VendorName.cs" where VendorName is the name of the 3rd party vendor. This file allows you to conveniently access their indicators.

•If you were provided an installer, you can check with the vendor if this file was included or;

•If they provided you this file, save it to "My Documents\<NinjaTrader Folder>\bin\Custom" and restart NinjaTrader

**Adding a Reference**

1. From within the NinjaScript Editor, right click on your mouse to bring up the context menu and select the sub-menu **References...** as per the image to the right.



2. A **References** window will appear

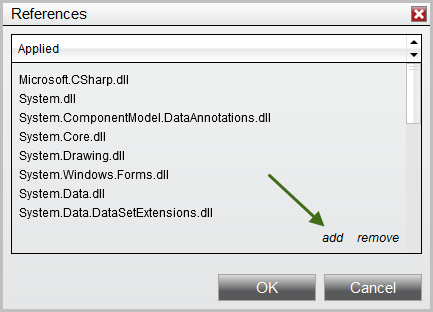
3. Press the **"*add"*** and select the 3rd party vendor DLL file

|  |
| --- |
| **Warning**: Please make sure in this step to select only the 'true' DLL file needed for reference, which would not contain any X86 or X64 suffixes in its file-name, otherwise you could run into compile issues later. |

4. You will see a reference to the 3rd party vendor DLL in the **References** window

5. Press the **OK** button

You will now be able to access the indicator methods provided by the 3rd party vendor



|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Using ATM Strategies** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_3rd_party_indicators.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_bitmapimage_objects_with_buttons.htm) |

You can create an automated strategy that generates a trade signal that executes a NinjaTrader [ATM Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/atm_strategy.htm).

•**ATM Strategies** operate in real-time only and will not execute on historical data thus they can't be backtested

•Executions resulting from an **ATM Strategy** that is created from within a NinjaScript automated strategy will not plot on a chart during real-time operation

•Strategy set up parameters such as [EntriesPerDirection](https://ninjatrader.com/es/support/helpGuides/nt8/entriesperdirection.htm), [EntryHandling](https://ninjatrader.com/es/support/helpGuides/nt8/entryhandling.htm), [IsExitOnSessionCloseStrategy](https://ninjatrader.com/es/support/helpGuides/nt8/isexitonsessionclosestrategy.htm) do not apply when calling the [AtmStrategyCreate()](https://ninjatrader.com/es/support/helpGuides/nt8/atmstrategycreate.htm) method

•Executions from **ATM Strategies** will not have an impact on the hosting NinjaScript strategy position and PnL - the NinjaScript strategy hands off the execution aspects to the ATM, thus no monitoring via the regular NinjaScript strategy methods will take place (also applies to strategy performance tracking)

•**ATM Strategy** stop orders can either be StopMarket or StopLimit orders, depending on which type is defined in the ATM Strategy Template ([Advanced Options)](https://ninjatrader.com/es/support/helpGuides/nt8/advanced_options.htm) you call in the [AtmStrategyCreate()](https://ninjatrader.com/es/support/helpGuides/nt8/atmstrategycreate.htm) method in your NinjaScript strategy. To make the distinction clear which is used, following a naming convention for the template name is highly suggested (i.e. AtmStrategyTemplate\_STPLMT)

•A general sample for calling ATM's is preinstalled with NinjaTrader under the 'SampleATMStrategy' script - for a script showing how to implement reversal type setups, please see [this link](http://www.ninjatrader.com/support/forum/local_links.php?action=jump&catid=8&id=866) to our online resources.

**There is a Clear Line...**

There is a clear line between a NinjaScript Strategy and an **ATM Strategy**. The use model for creating an **ATM Strategy** within a NinjaScript Strategy is when you want to programmatically monitor and generate an entry signal and then manualy manage the resulting open position via an ATM Strategy in one of NinjaTrader's order entry windows.

**!!! IMPORTANT: Manually Closing an ATM Strategy from an Order Entry Window such as the SuperDOM**

It is crucial that when running **ATM Strategies** created by a NinjaScript strategy that you understand how to properly manually close the **ATM Strategy** from any of the order entry windows.

•If the order entry window [ATM Strategy Selection Mode](https://ninjatrader.com/es/support/helpGuides/nt8/atm_strategy_selection_mode.htm) is NOT in "**DisplaySelectedATMStrategyOnly**" click on the "**CLOSE**" button via your middle mouse button (scroll wheel)

•If the order entry window **ATM Strategy Selection Mode** is in "**DisplaySelectedATMStrategyOnly**" you can click on the "**CLOSE**" button with your left mouse button to close the selected active **ATM strategy**

Following the approaches above will internally close the **ATM Strategy**. Not following the approach will close the account/instrument position, terminate all strategies and cancel all orders. The result is that your NinjaScript strategy will be terminated.

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Using BitmapImage objects with Buttons** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_atm_strategies.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_historical_bid_ask_serie.htm) |

**Images as Buttons Overview**

BitmapImage objects can be used to apply an image as a background to a Button object added to a NinjaTrader window.

|  |
| --- |
| **Note**: The following topic covers methods and properties outside of the NinjaScript libraries. Most of the items covered in the example below belong to .NET's System.Windows.Media.Imaging and System.Windows.Controls namespaces. More information on these namespaces can be found at the links below:  •[System.Windows.Controls](https://msdn.microsoft.com/en-us/library/system.windows.controls(v=vs.110).aspx)  •[System.Windows.Media.Imaging](https://msdn.microsoft.com/en-us/library/system.windows.media.imaging(v=vs.110).aspx) |

Using an image as the background for a button can be achieved through a fairly straightforward process using some of the .NET framework's Controls and Imaging methods

There are a few best practices to keep in mind when working with Buttons:

•Dispose of any leftover objects in State.Terminated for efficient memory use

•Use your object's main Dispatcher when adding or removing Buttons to or from your chart, to ensure that the correct thread is used

•Be aware of the proper States in which to initialize objects related to the Button (State.Configure), apply the Button (State.Historical), and dispose of unneeded objects (State.Terminated)

**Adding a Button to a Chart Toolbar Using an Image as the Background**

The example below walks through the process of adding a Button to a chart toolbar specifically, and applying a .jpg image as the Button's background. This example also displays several best practices when working with Buttons, such as proper object disposal and ensuring that the Button is not populated when the indicator is applied in an inactive chart tab.

| ns | |
| --- | --- |
| //Add the following Using statements using System.Windows.Media.Imaging; using System.Windows.Controls;     public class addButton : Indicator {   // Define a Chart object to refer to the chart on which the indicator resides   private Chart chartWindow;     // Define a Button   private System.Windows.Controls.Button myButton = null;     // Instantiate a BitmapImage to hold an image   BitmapImage myBitmapImage = new BitmapImage();     // Instantiate an ImageBrush to apply to the Button   ImageBrush backgroundImage = new ImageBrush();     private bool IsToolBarButtonAdded;     protected override void OnStateChange()   {       if (State == State.Configure)       {           // Assign an image on the filesystem to the BitmapImage.             // This example assumes that a jpg image named "ButtonBackground" resides in the install directory           myBitmapImage.BeginInit();           myBitmapImage.UriSource = new Uri(NinjaTrader.Core.Globals.InstallDir + "ButtonBackground.jpg");           myBitmapImage.EndInit();             // Assign the BitmapImage as the ImageSource of the ImageBrush           backgroundImage.ImageSource = myBitmapImage;       }       else if (State == State.Historical)       {           //Call the custom addButtonToToolbar method in State.Historical to ensure it is only done when applied to a chart             // -- not when loaded in the Indicators window           if (!IsToolBarButtonAdded) AddButtonToToolbar();       }       else if (State == State.Terminated)       {           //Call a custom method to dispose of any leftover objects in State.Terminated           DisposeCleanUp();       }   }     private void AddButtonToToolbar()   {       // Use this.Dispatcher to ensure code is executed on the proper thread       ChartControl.Dispatcher.InvokeAsync((Action)(() =>       {             //Obtain the Chart on which the indicator is configured           chartWindow = Window.GetWindow(this.ChartControl.Parent) as Chart;           if (chartWindow == null)           {               Print("chartWindow == null");               return;           }             // Create a style to apply to the button           Style s = new Style();           s.TargetType = typeof(System.Windows.Controls.Button);           s.Setters.Add(new Setter(System.Windows.Controls.Button.FontSizeProperty, 11.0));           s.Setters.Add(new Setter(System.Windows.Controls.Button.BackgroundProperty, Brushes.Orange));           s.Setters.Add(new Setter(System.Windows.Controls.Button.ForegroundProperty, Brushes.Black));           s.Setters.Add(new Setter(System.Windows.Controls.Button.FontFamilyProperty, new FontFamily("Arial")));           s.Setters.Add(new Setter(System.Windows.Controls.Button.FontWeightProperty, FontWeights.Bold));             // Instantiate the Button           myButton = new System.Windows.Controls.Button();             //Set Button Style                       myButton.Style = s;             // Set the Imagebrush as the Background for the Button           myButton.Background = backgroundImage;             myButton.Content = "Click Here";           myButton.IsEnabled = true;           myButton.HorizontalAlignment = HorizontalAlignment.Left;             // Add the Button to the Chart's Toolbar           chartWindow.MainMenu.Add(myButton);             //Prevent the Button From Displaying when WorkSpace Opens if it is not in an active tab           myButton.Visibility = Visibility.Collapsed;           foreach (TabItem tab in this.chartWindow.MainTabControl.Items)           {               if ((tab.Content as ChartTab).ChartControl == this.ChartControl                     && tab == this.chartWindow.MainTabControl.SelectedItem)               {                   myButton.Visibility = Visibility.Visible;               }           }           IsToolBarButtonAdded = true;       }));   }     private void DisposeCleanUp()   {       //ChartWindow Null Check       if (chartWindow != null)       {           //Dispatcher used to Assure Executed on UI Thread           ChartControl.Dispatcher.InvokeAsync((Action)(() =>           {               //Button Null Check               if (myButton != null)               {                   //Remove Button from Indicator's Chart ToolBar                   chartWindow.MainMenu.Remove(myButton);               }           }));       }   } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Using BitmapImage objects with Buttons** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_atm_strategies.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_historical_bid_ask_serie.htm) |

**Images as Buttons Overview**

BitmapImage objects can be used to apply an image as a background to a Button object added to a NinjaTrader window.

|  |
| --- |
| **Note**: The following topic covers methods and properties outside of the NinjaScript libraries. Most of the items covered in the example below belong to .NET's System.Windows.Media.Imaging and System.Windows.Controls namespaces. More information on these namespaces can be found at the links below:  •[System.Windows.Controls](https://msdn.microsoft.com/en-us/library/system.windows.controls(v=vs.110).aspx)  •[System.Windows.Media.Imaging](https://msdn.microsoft.com/en-us/library/system.windows.media.imaging(v=vs.110).aspx) |

Using an image as the background for a button can be achieved through a fairly straightforward process using some of the .NET framework's Controls and Imaging methods

There are a few best practices to keep in mind when working with Buttons:

•Dispose of any leftover objects in State.Terminated for efficient memory use

•Use your object's main Dispatcher when adding or removing Buttons to or from your chart, to ensure that the correct thread is used

•Be aware of the proper States in which to initialize objects related to the Button (State.Configure), apply the Button (State.Historical), and dispose of unneeded objects (State.Terminated)

**Adding a Button to a Chart Toolbar Using an Image as the Background**

The example below walks through the process of adding a Button to a chart toolbar specifically, and applying a .jpg image as the Button's background. This example also displays several best practices when working with Buttons, such as proper object disposal and ensuring that the Button is not populated when the indicator is applied in an inactive chart tab.

| ns | |
| --- | --- |
| //Add the following Using statements using System.Windows.Media.Imaging; using System.Windows.Controls;     public class addButton : Indicator {   // Define a Chart object to refer to the chart on which the indicator resides   private Chart chartWindow;     // Define a Button   private System.Windows.Controls.Button myButton = null;     // Instantiate a BitmapImage to hold an image   BitmapImage myBitmapImage = new BitmapImage();     // Instantiate an ImageBrush to apply to the Button   ImageBrush backgroundImage = new ImageBrush();     private bool IsToolBarButtonAdded;     protected override void OnStateChange()   {       if (State == State.Configure)       {           // Assign an image on the filesystem to the BitmapImage.             // This example assumes that a jpg image named "ButtonBackground" resides in the install directory           myBitmapImage.BeginInit();           myBitmapImage.UriSource = new Uri(NinjaTrader.Core.Globals.InstallDir + "ButtonBackground.jpg");           myBitmapImage.EndInit();             // Assign the BitmapImage as the ImageSource of the ImageBrush           backgroundImage.ImageSource = myBitmapImage;       }       else if (State == State.Historical)       {           //Call the custom addButtonToToolbar method in State.Historical to ensure it is only done when applied to a chart             // -- not when loaded in the Indicators window           if (!IsToolBarButtonAdded) AddButtonToToolbar();       }       else if (State == State.Terminated)       {           //Call a custom method to dispose of any leftover objects in State.Terminated           DisposeCleanUp();       }   }     private void AddButtonToToolbar()   {       // Use this.Dispatcher to ensure code is executed on the proper thread       ChartControl.Dispatcher.InvokeAsync((Action)(() =>       {             //Obtain the Chart on which the indicator is configured           chartWindow = Window.GetWindow(this.ChartControl.Parent) as Chart;           if (chartWindow == null)           {               Print("chartWindow == null");               return;           }             // Create a style to apply to the button           Style s = new Style();           s.TargetType = typeof(System.Windows.Controls.Button);           s.Setters.Add(new Setter(System.Windows.Controls.Button.FontSizeProperty, 11.0));           s.Setters.Add(new Setter(System.Windows.Controls.Button.BackgroundProperty, Brushes.Orange));           s.Setters.Add(new Setter(System.Windows.Controls.Button.ForegroundProperty, Brushes.Black));           s.Setters.Add(new Setter(System.Windows.Controls.Button.FontFamilyProperty, new FontFamily("Arial")));           s.Setters.Add(new Setter(System.Windows.Controls.Button.FontWeightProperty, FontWeights.Bold));             // Instantiate the Button           myButton = new System.Windows.Controls.Button();             //Set Button Style                       myButton.Style = s;             // Set the Imagebrush as the Background for the Button           myButton.Background = backgroundImage;             myButton.Content = "Click Here";           myButton.IsEnabled = true;           myButton.HorizontalAlignment = HorizontalAlignment.Left;             // Add the Button to the Chart's Toolbar           chartWindow.MainMenu.Add(myButton);             //Prevent the Button From Displaying when WorkSpace Opens if it is not in an active tab           myButton.Visibility = Visibility.Collapsed;           foreach (TabItem tab in this.chartWindow.MainTabControl.Items)           {               if ((tab.Content as ChartTab).ChartControl == this.ChartControl                     && tab == this.chartWindow.MainTabControl.SelectedItem)               {                   myButton.Visibility = Visibility.Visible;               }           }           IsToolBarButtonAdded = true;       }));   }     private void DisposeCleanUp()   {       //ChartWindow Null Check       if (chartWindow != null)       {           //Dispatcher used to Assure Executed on UI Thread           ChartControl.Dispatcher.InvokeAsync((Action)(() =>           {               //Button Null Check               if (myButton != null)               {                   //Remove Button from Indicator's Chart ToolBar                   chartWindow.MainMenu.Remove(myButton);               }           }));       }   } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Using Images and Geometry with Custom Icons** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_historical_bid_ask_serie.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm) |

Custom Icon Overview

When overriding the Icon method in a [Bars Type](https://ninjatrader.com/es/support/helpGuides/nt8/bars_type.htm), [Share Service](https://ninjatrader.com/es/support/helpGuides/nt8/share_service.htm), [Drawing Object](https://ninjatrader.com/es/support/helpGuides/nt8/drawing_tools.htm), or [Chart Style](https://ninjatrader.com/es/support/helpGuides/nt8/chart_style.htm), you can use a variety of inputs to specify what will be displayed on the icon, including UniCode characters (if they exist in the icon pack for the font family used in NinjaTrader), custom Geometry Paths from the System.Windows.Shapes namespace, or image files. Using an image file for a custom icon can allow the flexibility of creating your icon's visuals outside of your code via image editing software. For more information about adding custom Icons, see the "Icon" page under the topics for each of the NinjaScript object types listed above.

tog_minus        [Using an Image as an Icon](javascript:HMToggle('toggle','UsingAnImageAsAnIcon','UsingAnImageAsAnIcon_ICON'))

|  |  |  |  |
| --- | --- | --- | --- |
| **Using an Image as an Icon**  The process for using an image as an icon is fairly straightforward using WPF objects, and is the same for different NinjaScript objects.    1.Instantiate a new BitmapImage object  2.Assign a Uri to the BitmapImage, pointing to an image file  3.Instantiate a Grid of the same dimensions as the icon  4.Instantiate an Image object  5.Assign the BitmapImage as the Image's Source  6.Add the Image to the Grid  7.Return the Grid by overriding the Icon property     |  | | --- | | **Note**: Be careful to instantiate the Grid to be same size as the needed icon. Some icon sizes differ from others. For example, the icon for Share Services is substantially larger than the icon for a Chart Style in the Chart Toolbar. |      | ns | | --- | | // Add the following Using statements using System.Windows.Controls; using System.Windows.Media; using System.Windows.Media.Imaging;   BitmapImage iconBitmapImage = new BitmapImage();   protected override void OnStateChange() {   if (State == State.Configure)   {     // Set the BitmapImage's UriSource to the location of an image file       iconBitmapImage.BeginInit();       iconBitmapImage.UriSource = new Uri(NinjaTrader.Core.Globals.InstallDir + "icon.jpg");       iconBitmapImage.EndInit();   } }   // Override Icon (read-only) to return the custom Grid and Image public override object Icon {   get   {       // Instantiate a Grid on which to place the image       Grid myCanvas = new Grid { Height = 16, Width = 16 };         // Instantiate an Image to place on the Grid       Image image = new Image       {           Height = 16,           Width = 16,           Source = iconBitmapImage       };         // Add the image to the Grid       myCanvas.Children.Add(image);         return myCanvas;   } } | |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?using_images_and_geometry_with_custom_icons.htm#UsingAnImageAsAnIcon)

tog_minus        [Using Geometry on an Icon](javascript:HMToggle('toggle','UsingGeometryOnAnIcon','UsingGeometryOnAnIcon_ICON'))

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Using Historical Bid/Ask Series** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_bitmapimage_objects_with_buttons.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_images_and_geometry_with_custom_icons.htm) |

**Historical Bid/Ask Series Overview**

NinjaTrader has the ability to use historical bid and ask price series in your NinjaScript instead of only being able to use a last price series. The following outlines the intricacies of this capability:

|  |
| --- |
| **Notes**:  •You can have multiple bid/ask/last series in your NinjaScript indicator/strategy. Please use the [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) method to add these series to your script.  •The historical bid/ask series holds *all* bid/ask events sent out by the exchange. This would *not* be equivalent to the bid/ask at a specific time a trade went off.  •When processing your NinjaScript, the historical bid/ask series would have the historical portion triggered in the [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) method only. [OnMarketData()](https://ninjatrader.com/es/support/helpGuides/nt8/onmarketdata.htm) method events for the historical bid/ask series would only be triggered in real-time. |

|  |
| --- |
| **Tips**:  •For using **OnMarketData()** events historically, please see the educational topic on [Developing for Tick Replay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm)  •Changing the price type used for the primary Bars object to which a script is applied can be done in the [Data Series](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_price_data.htm) window from any open chart. |

**Accessing Bid/Ask Series**

When calling AddDataSeries() to add an additional [Bars](https://ninjatrader.com/es/support/helpGuides/nt8/bars.htm) object to your script, a constructor overload will be available which takes a MarketDataType enumeration as an argument. This will allow you to specify the price series which will be used in that particular object. If you were to pass in MarketDataType.Ask or MarketDataType.Bid, as in the example below, that particular data series will use that price type for all of its [PriceSeries](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) collections, such as [Close](https://ninjatrader.com/es/support/helpGuides/nt8/close.htm), [Open](https://ninjatrader.com/es/support/helpGuides/nt8/open.htm), [High](https://ninjatrader.com/es/support/helpGuides/nt8/high.htm), and [Low](https://ninjatrader.com/es/support/helpGuides/nt8/low.htm).

|  |
| --- |
| **Warning**: A **Tick Replay** indicator or strategy **CANNOT**use a **MarketDataType.Ask** or **MarketDataType.Bid**series.  Please see [Developing for Tick Replay](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) for more information. |

**Example**

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if (State == State.Configure)   {       // Add an AAPL data series using the Ask series       AddDataSeries("AAPL", BarsPeriodType.Minute, 30, MarketDataType.Ask);         //Add another AAPL data series using the Bid series, with other settings identical       AddDataSeries("AAPL", BarsPeriodType.Minute, 30, MarketDataType.Bid);   } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Using Images and Geometry with Custom Icons** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_historical_bid_ask_serie.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm) |

Custom Icon Overview

When overriding the Icon method in a [Bars Type](https://ninjatrader.com/es/support/helpGuides/nt8/bars_type.htm), [Share Service](https://ninjatrader.com/es/support/helpGuides/nt8/share_service.htm), [Drawing Object](https://ninjatrader.com/es/support/helpGuides/nt8/drawing_tools.htm), or [Chart Style](https://ninjatrader.com/es/support/helpGuides/nt8/chart_style.htm), you can use a variety of inputs to specify what will be displayed on the icon, including UniCode characters (if they exist in the icon pack for the font family used in NinjaTrader), custom Geometry Paths from the System.Windows.Shapes namespace, or image files. Using an image file for a custom icon can allow the flexibility of creating your icon's visuals outside of your code via image editing software. For more information about adding custom Icons, see the "Icon" page under the topics for each of the NinjaScript object types listed above.

tog_minus        [Using an Image as an Icon](javascript:HMToggle('toggle','UsingAnImageAsAnIcon','UsingAnImageAsAnIcon_ICON'))

tog_minus        [Using Geometry on an Icon](javascript:HMToggle('toggle','UsingGeometryOnAnIcon','UsingGeometryOnAnIcon_ICON'))

|  |  |  |
| --- | --- | --- |
| **Using Geometry on an Icon**  Custom geometry Paths can be used to draw and fill custom shapes, which can then be applied directly to a Canvas returned for use in an Icon. The process for using a Path is similar to that for using an Image:    1.Instantiate a new Path object  2.Instantiate a Grid of the same dimensions as the icon  3.Define the visual properties of the Path  4.Add the Path to the Grid  5.Return the Grid by overriding the Icon property     | ns | | --- | | // Add the following namespace to use Path objects using System.Windows.Shapes;  using System.Windows.Controls;   public override object Icon {   get   {       // Instantiate a Grid on which to place the Path       Grid myCanvas = new Grid { Height = 16, Width = 16 };         // Instantiate a Path object on which to draw geometry       System.Windows.Shapes.Path myPath = new System.Windows.Shapes.Path();         // Define the Path's visual properties       myPath.Fill = Brushes.Red;       myPath.Data = System.Windows.Media.Geometry.Parse("M 0 0 L 5 0 L 5 5 L 10 5 L 10 0 L 15 0 L 15 5 L 10 5 L 10 10 L 5 10 L 5 5 L 0 5 Z");         // Add the Path to the Canvas, then return the Canvas       myCanvas.Children.Add(myPath);       return myCanvas;   } } | |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?using_images_and_geometry_with_custom_icons.htm#UsingGeometryOnAnIcon)

|  |  |
| --- | --- |
| **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 Brushes** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_chart_object_coordinates.htm) |

In order to achieve custom rendering for various chart related objects, a Brush is used to "paint" an area or another chart object.  There are a number of different brushes which are available through the .NET Framework, where the most common type of brush is a [SolidColorBrush](https://msdn.microsoft.com/en-us/library/system.windows.media.solidcolorbrush(v=vs.110).aspx) which is used to paint an area with a single solid color.

|  |
| --- |
| **Notes**:  The following document is written in sequential fashion, starting with the most simple concepts, to the more advance topics.  The majority of the brushes discussed in this document will be referred to as "**WPF" brushes** which exist in the System.Windows.Media namespace, however there are also **"SharpDX" brushes** which exist in the 3rd party SharpDX.Direct2D1 nampspace used for advanced chart rendering.  Advanced brush types should **ONLY** be used by experienced programmers familiar with .NET graphics functionality. |

tog_minus        [Understanding predefined brushes](javascript:HMToggle('toggle','Understandingpredefinedbrushes','Understandingpredefinedbrushes_ICON'))

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Using Predefined Brushes**  For convenience, the .NET Framework supplies a collection of static predefined Brushes, such as Red or Green.  The advantage to using these brushes is that they are readily available, properly named to quickly find a simple color value, and can be reused on-the-fly without having to recreate an instance of the brush at run time, and do not need to be otherwise managed.  There are 256 predefined named brushes which are available in the Brushes class.  You can browse this list in the NinjaScript editor just by typing Brushes. and using Intelliprompt to find the desired named brush of your choice.     |  | | --- | | **Note**:   Since predefined brushes are static, properties of the brush object (such as Color, Opacity, etc.) **CANNOT** be modified.  However, this also means predefined brushes are thread-safe and do **NOT** need to be frozen.  For customizing and freezing a brush, please see the section below on *Creating a Custom Solid Color Brush*. |       Brushes       |  | | --- | | **Tip**:  You can also find a list of these predefined brushes as well as their hexadecimal value on the MSDN article for the [Brushes Class](https://msdn.microsoft.com/en-us/library/system.windows.media.brushes(v=vs.110).aspx) |        | ns |  | | --- | --- | | // set the chart's background color to a predefined "Blue" brush BackBrush = Brushes.Blue;   //draw a line using a predefined "LimeGreen" brush. Draw.Line(this, "tag1", false, 10, 1000, 0, 1001, Brushes.LimeGreen, DashStyleHelper.Dot, 2); | | |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?working_with_brushes.htm#Understandingpredefinedbrushes)

tog_minus        [Understanding custom brushes](javascript:HMToggle('toggle','Understandingcustombrushes','Understandingcustombrushes_ICON'))

tog_minus        [Using brushes defined on the user interface](javascript:HMToggle('toggle','Userdefinedbrushes','Userdefinedbrushes_ICON'))

tog_minus        [Using advanced brush types (SharpDX)](javascript:HMToggle('toggle','AdvancedBrushTypesSharpDX','AdvancedBrushTypesSharpDX_ICON'))

|  |  |
| --- | --- |
| **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 Brushes** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_chart_object_coordinates.htm) |

In order to achieve custom rendering for various chart related objects, a Brush is used to "paint" an area or another chart object.  There are a number of different brushes which are available through the .NET Framework, where the most common type of brush is a [SolidColorBrush](https://msdn.microsoft.com/en-us/library/system.windows.media.solidcolorbrush(v=vs.110).aspx) which is used to paint an area with a single solid color.

|  |
| --- |
| **Notes**:  The following document is written in sequential fashion, starting with the most simple concepts, to the more advance topics.  The majority of the brushes discussed in this document will be referred to as "**WPF" brushes** which exist in the System.Windows.Media namespace, however there are also **"SharpDX" brushes** which exist in the 3rd party SharpDX.Direct2D1 nampspace used for advanced chart rendering.  Advanced brush types should **ONLY** be used by experienced programmers familiar with .NET graphics functionality. |

tog_minus        [Understanding predefined brushes](javascript:HMToggle('toggle','Understandingpredefinedbrushes','Understandingpredefinedbrushes_ICON'))

tog_minus        [Understanding custom brushes](javascript:HMToggle('toggle','Understandingcustombrushes','Understandingcustombrushes_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Creating a Custom Solid Color Brush**  In cases where you would like more specific color than one of the predefined brushes, you can optionally create your own **Brush** object to be used for custom rendering.  In order to achieve this, you will need to initiate your own custom brush object, where you can then specify your color using RGB (red, green, blue) values [Color.FromRgb()](https://msdn.microsoft.com/en-us/library/system.windows.media.color.fromrgb(v=vs.110).aspx).     |  | | --- | | **Notes**:  •Anytime you create a custom brush that will be used by NinjaTrader rendering it must be frozen using the .[Freeze()](https://msdn.microsoft.com/en-us/library/ms557735(v=vs.110).aspx)  method due to the multi-threaded nature of NinjaTrader.  •You may have up to 65535 unique Brush instances, therefore, using static predefined brushes (as in the section above) should be favored.  Alternatively,  in order to use fewer brushes, please try to cache your custom brushes until a new brush would actually need to be created. |        | ns |  | | --- | --- | | // initiate new solid color brush with custom blue color Brush myBrush = new SolidColorBrush(Color.FromRgb(56, 120, 153)); myBrush.Freeze();   Draw.Line(this, "tag1", true, 10, 1000, 0, 1001, myBrush, DashStyleHelper.Dot, 2); | |      |  | | --- | | **Warning**:  If you do not call .[Freeze()](https://msdn.microsoft.com/en-us/library/ms557735(v=vs.110).aspx) on a custom defined brush **WILL**eventually result in threading errors should you try to modify or access that brush after it is defined. |       **Creating a Transparent Solid Color Brush**  You can create a transparent brush using the [Color.FromArgb()](https://msdn.microsoft.com/en-us/library/system.windows.media.color.fromargb(v=vs.110).aspx) where the A parameter defines alpha transparency.     |  | | --- | | **Note**:   Anytime you create a custom brush that will be used by NinjaTrader rendering it must be frozen using the .[Freeze()](https://msdn.microsoft.com/en-us/library/ms557735(v=vs.110).aspx)  method due to the multi-threaded nature of NinjaTrader. |        | ns |  | | --- | --- | | // initiate new solid color brush which has an alpha (transparency) value of 100 MyBrush = new SolidColorBrush(Color.FromArgb(100, 56, 120, 153)); myBrush.Freeze();   Draw.Line(this, "tag1", true, 10, 1000, 0, 1001, myBrush, DashStyleHelper.Dot, 2); | |        |  | | --- | | **Warning**:  If you do not call .[Freeze()](https://msdn.microsoft.com/en-us/library/ms557735(v=vs.110).aspx) on a custom defined brush **WILL**eventually result in threading errors should you try to modify or access that brush after it is defined. | |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?working_with_brushes.htm#Understandingcustombrushes)

tog_minus        [Using brushes defined on the user interface](javascript:HMToggle('toggle','Userdefinedbrushes','Userdefinedbrushes_ICON'))

tog_minus        [Using advanced brush types (SharpDX)](javascript:HMToggle('toggle','AdvancedBrushTypesSharpDX','AdvancedBrushTypesSharpDX_ICON'))

|  |  |
| --- | --- |
| **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 Brushes** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_chart_object_coordinates.htm) |

In order to achieve custom rendering for various chart related objects, a Brush is used to "paint" an area or another chart object.  There are a number of different brushes which are available through the .NET Framework, where the most common type of brush is a [SolidColorBrush](https://msdn.microsoft.com/en-us/library/system.windows.media.solidcolorbrush(v=vs.110).aspx) which is used to paint an area with a single solid color.

|  |
| --- |
| **Notes**:  The following document is written in sequential fashion, starting with the most simple concepts, to the more advance topics.  The majority of the brushes discussed in this document will be referred to as "**WPF" brushes** which exist in the System.Windows.Media namespace, however there are also **"SharpDX" brushes** which exist in the 3rd party SharpDX.Direct2D1 nampspace used for advanced chart rendering.  Advanced brush types should **ONLY** be used by experienced programmers familiar with .NET graphics functionality. |

tog_minus        [Understanding predefined brushes](javascript:HMToggle('toggle','Understandingpredefinedbrushes','Understandingpredefinedbrushes_ICON'))

tog_minus        [Understanding custom brushes](javascript:HMToggle('toggle','Understandingcustombrushes','Understandingcustombrushes_ICON'))

tog_minus        [Using brushes defined on the user interface](javascript:HMToggle('toggle','Userdefinedbrushes','Userdefinedbrushes_ICON'))

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Saving a Brush as a user defined property (Serialization)**  If you would like a brush to become a public UI property, meaning the brush can be set up and defined by a user during configuration, it is important to be able to save the user's brush selection in order to restore that brush either from a workspace or from a template file at a later time.  Saving a custom defined user input is done through a concept of [Serialization](https://msdn.microsoft.com/en-us/library/ms233843.aspx) which writes the object and its value to a .xml file.  This process normally works fine for a simple user defined value type (such as a double or an int) but for more complex types such as Brushes, the object itself cannot be serialized directly to the .xml file and will result in errors upon saving the indicator or strategy to a workspace or template file.  The example below will demonstrate and explain how to properly store a user define brush input which will be correctly serialized.    In order to achieve the desired behavior of saving the user defined brush input, we will add the [XmlIgnore](https://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlignoreattribute(v=vs.110).aspx) property attribute to the public brush resource, which essentially tells the serialization routine to ignore this property.     | ns | | --- | | [XmlIgnore] public Brush MyBrush { get; set; } |       In its place, we create a new public string called "MyBrushSerialize" which will convert the public "MyBrush" to a string type which can then be processed by the serialization routines.  We also add the [Browsable(false)](https://msdn.microsoft.com/en-us/library/system.componentmodel.browsableattribute(v=vs.110).aspx) attribute to this public string to prevent this property from showing up on the UI, which is of no value to the end user.     | ns | | --- | | [Browsable(false)] public string MyBrushSerialize {   get { return Serialize.BrushToString(MyBrush); }   set { MyBrush = Serialize.StringToBrush(value); } } |        |  | | --- | | **Tip**: For a complete example of **User Definable Color Inputs**, please see the reference sample [here](https://ninjatrader.com/es/support/helpGuides/nt8/user_definable_color_inputs.htm). |       **Adding a User Defined Brush to the Color Picker**  You can optionally define a custom brush to be added to the standard color picker by using a [CustomBrush] attribute to a public brush.  The CustomBrush attribute will then add it to the color picker menu for that indicator when you look through the plots, lines, or other brushes from the indicators configured menu and will be listed toward the top of the list (as pictured below)     | ns | | --- | | [CustomBrush] public Brush MyBrush {   get { return new SolidColorBrush(Color.FromRgb(25, 175, 185)); }   set { } } |     custom_brush |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?working_with_brushes.htm#Userdefinedbrushes)

tog_minus        [Using advanced brush types (SharpDX)](javascript:HMToggle('toggle','AdvancedBrushTypesSharpDX','AdvancedBrushTypesSharpDX_ICON'))

|  |  |
| --- | --- |
| **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 Brushes** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_chart_object_coordinates.htm) |

In order to achieve custom rendering for various chart related objects, a Brush is used to "paint" an area or another chart object.  There are a number of different brushes which are available through the .NET Framework, where the most common type of brush is a [SolidColorBrush](https://msdn.microsoft.com/en-us/library/system.windows.media.solidcolorbrush(v=vs.110).aspx) which is used to paint an area with a single solid color.

|  |
| --- |
| **Notes**:  The following document is written in sequential fashion, starting with the most simple concepts, to the more advance topics.  The majority of the brushes discussed in this document will be referred to as "**WPF" brushes** which exist in the System.Windows.Media namespace, however there are also **"SharpDX" brushes** which exist in the 3rd party SharpDX.Direct2D1 nampspace used for advanced chart rendering.  Advanced brush types should **ONLY** be used by experienced programmers familiar with .NET graphics functionality. |

tog_minus        [Understanding predefined brushes](javascript:HMToggle('toggle','Understandingpredefinedbrushes','Understandingpredefinedbrushes_ICON'))

tog_minus        [Understanding custom brushes](javascript:HMToggle('toggle','Understandingcustombrushes','Understandingcustombrushes_ICON'))

tog_minus        [Using brushes defined on the user interface](javascript:HMToggle('toggle','Userdefinedbrushes','Userdefinedbrushes_ICON'))

tog_minus        [Using advanced brush types (SharpDX)](javascript:HMToggle('toggle','AdvancedBrushTypesSharpDX','AdvancedBrushTypesSharpDX_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Understanding SharpDX Brushes**  While the majority of the NinjaTrader platform's UI is **WPF**, under the hood, chart's use a **DirectX API** for faster performance.  To render custom objects to a chart during [OnRender()](https://ninjatrader.com/es/support/helpGuides/nt8/onrender.htm), a particular **SharpDX** **Brush** object must be implemented which reside in the **SharpDX.Direct2D1** namespace.   These brushes can then be passed as arguments to the **SharpDX** [RenderTarget](https://ninjatrader.com/es/support/helpGuides/nt8/rendertarget.htm) methods such [FillRectangle()](https://ninjatrader.com/es/support/helpGuides/nt8/fillrectangle.htm), [DrawLine()](https://ninjatrader.com/es/support/helpGuides/nt8/drawline2.htm), etc.  While **SharpDX Brushes** behave much the same as previously discussed **WPF** **Brushes**, there are a few special considerations you must take as detailed in the following sections.     |  | | --- | | **Note**:  The **SharpDX Brushes** used in [RenderTarget](https://ninjatrader.com/es/support/helpGuides/nt8/rendertarget.htm) methods should **NOT** be confused with the **WPF Brushes** used with [DrawingTool Draw](https://ninjatrader.com/es/support/helpGuides/nt8/drawing.htm) methods. |       **Creating a SharpDX Brush**  A [SharpDX Brush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_brush.htm) must be created either in **OnRender()** or **RenderTargetChanged()**.  If you have custom brushes which may be changed on various conditions such as in OnBarUpdate() or by a user during OnStateChange(), or you are pre-computing a custom brush for performance optimization, you will need to ensure the actual SharpDX instance is updated in OnRender() or RenderTargetChange().     |  | | --- | | **Warning**:  Each DirectX render target requires its own brushes. You **MUST** create brushes directly in **OnRender()** or using **OnRenderTargetChanged()**.  If you do not you will receive an error at runtime similar to:   ***"A direct X error has occured while rendering the chart: HRESULT: [0x88990015], Module: [SharpDX.Direct2D1], ApiCode: [D2DERR\_WRONG\_RESOURCE\_DOMAIN/WrongResourceDomain], Message: The resource was realized on the wrong render target. : Each DirectX render target requires its own brushes. You must create brushes directly in OnRender() or using OnRenderTargetChanged().***    Please see [OnRenderTargetChanged()](https://ninjatrader.com/es/support/helpGuides/nt8/onrendertargetchanged.htm) for examples of a brush that needs to be recalculated, or [OnRender()](https://ninjatrader.com/es/support/helpGuides/nt8/onrender.htm) for an example of recreating a static brush. |        | ns | | --- | | // use predefined "Blue" SharpDX Color SharpDX.Direct2D1.SolidColorBrush solidBlueDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Blue);   // create custom Brush using a "Red" SharpDX Color with "Alpha" (0.100f) transparency/opacity SharpDX.Direct2D1.SolidColorBrush transparentRedDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, new SharpDX.Color4(new SharpDX.Color3(220f, 0f, 0f), 0.100f)); |       **Converting to SharpDX Brush**  For convenience, you can convert a computed WPF Brush to a [SharpDX Brush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_brush.htm) using the [ToDxBrush(](https://ninjatrader.com/es/support/helpGuides/nt8/dxextensions_todxbrush.htm)) extension method.     |  | | --- | | **Warning**:  Converting **ToDxBrush()** can result in performance issues depending on the number of brushes being used. If you experience performance issues with your custom **SharpDX** rendering, you should favor using **SharpDX** brushes directly instead of converting the brush using **ToDxBrush().** |        | ns | | --- | | // convert predefined WPF "Blue" to SharpDX Brush SharpDX.Direct2D1.Brush blueDXBrush = Brushes.Blue.ToDxBrush(RenderTarget);   // convert the computed WPF Brush to SharpDX Brush SharpDX.Direct2D1.Brush customDXBrush = customWPFBrush.ToDxBrush(RenderTarget); |       **Disposing DXBrush**  Since **SharpDX Brushes** reference unmanaged resources, these brushes should always be [disposed](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_disposebase.htm) of after they have been used.     |  | | --- | | **Warning**:  Failing to dispose of a [SharpDX Brush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_brush.htm) and other unmanaged resources can cause the platform to utilize more memory than necessary. |        | ns | | --- | | customDXBrush.Dipose(); |     **Using Complex Brushes**  In addition to the [SolidColorBrush](https://msdn.microsoft.com/en-us/library/system.windows.media.solidcolorbrush(v=vs.110).aspx) object demonstrated on this page, the .NET Framework provides more complex brushes which have more attributes than just filling an area with a solid color.  Information on these special types of brushes can be found on the MSDN website: [LinearGradientBrush](https://msdn.microsoft.com/en-us/library/system.windows.media.lineargradientbrush(v=vs.110).aspx), [RadialGradientBrush](https://msdn.microsoft.com/en-us/library/system.windows.media.radialgradientbrush(v=vs.110).aspx), [ImageBrush](https://msdn.microsoft.com/en-us/library/system.windows.media.imagebrush(v=vs.110).aspx).    These complex types also have an equivalent found in the**SharpDX SDK Reference**: [SharpDX.Direct2D1.LinearGradientBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_lineargradientbrush.htm), [SharpDX.Direct2D1.RadialGradientBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_radialgradientbrush.htm) |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?working_with_brushes.htm#AdvancedBrushTypesSharpDX)

|  |  |
| --- | --- |
| **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) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **C# Method (Functions) Reference** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/other_uses_for_an_addon.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/developing_for__tick_replay.htm) |

**Native Methods**

The Microsoft .NET environment has a rich class library that you can access when developing custom indicators and strategies. There is a plethora of information available online and in print that details class libraries in great depth. Below are quick links to the Microsoft Developers Network for some of the basic classes whose functionality you may harness when developing in NinjaScript.

Complete [list of classes](https://msdn.microsoft.com/en-us/library/d11h6832(v=vs.90).aspx) in the Microsoft .NET environment.

[MSDN (Microsoft Developers Network) C# Language Reference](http://msdn.microsoft.com/en-us/library/ms228593.aspx)

[Keywords](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/index)

[Operators](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/index)

[Arrays](http://msdn.microsoft.com/en-us/library/9b9dty7d)

**System.Math**

Provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions.

Full list of [member](https://msdn.microsoft.com/en-us/library/xaz41263(v=vs.110).aspx) of the System.Math class.

| ns |
| --- |
| // Example of the Max method of the System.Math class  int myInteger = Math.Max(10, 20);  Print("The larger value between 10 and 20 is " + myInteger.ToString()); |

**System.DateTime**

Represents an instant in time, typically expressed as a data and time of day.

Full list of [members](https://msdn.microsoft.com/en-us/library/system.datetime(v=vs.113).aspx) of the Sytem.DateTime structure.

| ns |
| --- |
| // Example of the Now property member of the System.DateTime structure  DateTime startTime = DateTime.Now;  Print("Time elapsed is " + DateTime.Now.Subtract(startTime).TotalMilliseconds.ToString() + " milliseconds."); |

**System.String**

Represents text; that is, a series of unicode characters.

Full list of [members](https://msdn.microsoft.com/en-us/library/system.string(v=vs.113).aspx) of the System.String class.

| ns |
| --- |
| // Example of the ToUpper() method of the System.String class  string myString = "ninjatrader";  Print("The following word is in uppercase " + myString.ToUpper());); |

[Pular para o conteúdo principal](https://learn.microsoft.com/en-us/previous-versions/d11h6832(v=vs.90)?redirectedfrom=MSDN#main)

[**Aprender**](https://learn.microsoft.com/en-us/)

* Descobrir
* Documentação do produto
* Linguagens de desenvolvimento
* Tópicos

[Entrar](https://learn.microsoft.com/en-us/previous-versions/d11h6832(v=vs.90)?redirectedfrom=MSDN)

 Não estamos mais atualizando este conteúdo regularmente. Verifique o [**Microsoft Product Lifecycle**](https://learn.microsoft.com/lifecycle/products) para obter informações sobre como este produto, serviço, tecnologia ou API é suportado.

[**Retornar ao site principal**](https://learn.microsoft.com/)

Dispensar alerta

Principio del formulario

Procurar

Final del formulario

* [**Referência da biblioteca de classes do .NET Framework**](https://learn.microsoft.com/en-us/previous-versions/d11h6832(v=vs.90))

1. [Aprender](https://learn.microsoft.com/en-us/)

1. [Versões anteriores](https://learn.microsoft.com/en-us/previous-versions/)

C#

Salvar

**Referência da biblioteca de classes do .NET Framework**

* Artigo
* 15/09/2008

**Neste artigo**

1. [Espaços de nomes](https://learn.microsoft.com/en-us/previous-versions/d11h6832(v=vs.90)?redirectedfrom=MSDN#namespaces)
2. [Veja também](https://learn.microsoft.com/en-us/previous-versions/d11h6832(v=vs.90)?redirectedfrom=MSDN#see-also)

A biblioteca de classes do .NET Framework é uma biblioteca de classes, interfaces e tipos de valor que estão incluídos no Microsoft .NET Framework. Esta biblioteca fornece acesso à funcionalidade do sistema e é projetada para ser a base sobre a qual os aplicativos, componentes e controles do .NET Framework são criados.

**Espaços de nomes**

A biblioteca de classes do .NET Framework fornece os seguintes namespaces:

* [Microsoft.Aspnet.Snapin](https://msdn.microsoft.com/en-us/library/89ccda1w(v=vs.90))  
  Contém classes necessárias para que o aplicativo de console de gerenciamento ASP.NET interaja com o Microsoft Management Console (MMC).
* [Microsoft.Build.BuildEngine](https://msdn.microsoft.com/en-us/library/ms123531(v=vs.90))  
  Contém as classes que representam o mecanismo MSBuild.
* [Microsoft.Build.Framework](https://msdn.microsoft.com/en-us/library/ms124190(v=vs.90))  
  Contém classes que compõem as tarefas, registradores e eventos do MSBuild.
* [Microsoft.Build.Tasks](https://msdn.microsoft.com/en-us/library/ms124502(v=vs.90))  
  Contém a implementação de todas as tarefas enviadas com o MSBuild.
* [Microsoft.Build.Tasks.Deployment.Bootstrapper](https://msdn.microsoft.com/en-us/library/ms125440(v=vs.90))  
  Contém classes usadas internamente pelo MSBuild.
* [Microsoft.Build.Tasks.Deployment.ManifestUtilities](https://msdn.microsoft.com/en-us/library/ms125618(v=vs.90))  
  Contém classes usadas internamente pelo MSBuild.
* [Microsoft.Build.Utilities](https://msdn.microsoft.com/en-us/library/ms126181(v=vs.90))  
  Fornece classes auxiliares que você pode usar para criar seus próprios registradores e tarefas do MSBuild.
* [Microsoft.Csharp](https://msdn.microsoft.com/en-us/library/w93k91w8(v=vs.90))  
  Contém classes que oferecem suporte à compilação e geração de código usando a linguagem C#.
* [Microsoft.JScript](https://msdn.microsoft.com/en-us/library/ccz24605(v=vs.90))  
  Contém classes que oferecem suporte à compilação e geração de código usando a linguagem JScript.
* [Microsoft.SqlServer.Server](https://msdn.microsoft.com/en-us/library/ms127097(v=vs.90))  
  Contém classes específicas para a integração do componente CLR (Common Language Runtime) do Microsoft .NET Framework no Microsoft SQL Server e no ambiente de execução do processo do mecanismo de banco de dados do SQL Server.
* [Microsoft.VisualBasic](https://msdn.microsoft.com/en-us/library/00zt4b2a(v=vs.90))  
  Contém classes que oferecem suporte à compilação e geração de código usando a linguagem Visual Basic.
* [Microsoft.VisualBasic.ApplicationServices](https://msdn.microsoft.com/en-us/library/ms127483(v=vs.90))  
  Contém tipos que suportam o Modelo de Aplicativo do Visual Basic e fornecem acesso às informações do aplicativo.
* [Microsoft.VisualBasic.CompilerServices](https://msdn.microsoft.com/en-us/library/59xcz346(v=vs.90))  
  Contém tipos somente de uso interno que suportam o compilador Visual Basic.
* [Microsoft.VisualBasic.Devices](https://msdn.microsoft.com/en-us/library/ms127788(v=vs.90))  
  Contém tipos que suportam os objetos My relacionados a dispositivos no Visual Basic.
* [Microsoft.VisualBasic.FileIO](https://msdn.microsoft.com/en-us/library/ms127948(v=vs.90))  
  Contém tipos que suportam o objeto Meu sistema de arquivos no Visual Basic.
* [Microsoft.VisualBasic.Logging](https://msdn.microsoft.com/en-us/library/ms128122(v=vs.90))  
  Contains types that support the My logging objects in Visual Basic and provides a simple log listener that directs logging output to file.
* [Microsoft.VisualBasic.MyServices](https://msdn.microsoft.com/en-us/library/c2se63tx(v=vs.90))  
  Contains types that support My in Visual Basic.
* [Microsoft.VisualBasic.MyServices.Internal](https://msdn.microsoft.com/en-us/library/ms128259(v=vs.90))  
  Contains internal-use only types that support My in Visual Basic.
* [Microsoft.VisualBasic.Vsa](https://msdn.microsoft.com/en-us/library/h8d1wxyh(v=vs.90))
* [Microsoft.VisualC](https://msdn.microsoft.com/en-us/library/713k750c(v=vs.90))
* [Microsoft.Vsa](https://msdn.microsoft.com/en-us/library/16hk56d9(v=vs.90))  
  Contains interfaces that allow you to integrate script for the .NET Framework script engines into applications, and to compile and execute code at run time.
* [Microsoft.Vsa.Vb.CodeDOM](https://msdn.microsoft.com/en-us/library/1zfxefw6(v=vs.90))
* [Microsoft.Win32](https://msdn.microsoft.com/en-us/library/b2hs0tae(v=vs.90))  
  Provides two types of classes: those that handle events raised by the operating system and those that manipulate the system registry.
* [Microsoft.Win32.SafeHandles](https://msdn.microsoft.com/en-us/library/c954fyb1(v=vs.90))  
  Contains classes that are abstract derivations of safe handle classes that provide common functionality supporting file and operating system handles.
* [Microsoft.WindowsCE.Forms](https://msdn.microsoft.com/en-us/library/kbcxyf35(v=vs.90))  
  Contains classes for developing Pocket PC and Smartphone Windows Forms applications using the .NET Compact Framework.
* [Microsoft.WindowsMobile.DirectX](https://msdn.microsoft.com/en-us/library/ms128614(v=vs.90))  
  Contains classes for developing DirectX applications on devices with the .NET Compact Framework. Requires a future release of Windows Mobile to run the applications.
* [Microsoft.WindowsMobile.DirectX.Direct3D](https://msdn.microsoft.com/en-us/library/ms129011(v=vs.90))  
  Contains classes for developing Direct3D applications on devices with the .NET Compact Framework. Requires a future release of Windows Mobile to run the applications.
* [Microsoft\_VsaVb](https://msdn.microsoft.com/en-us/library/yz7k01ka(v=vs.90))
* [System](https://msdn.microsoft.com/en-us/library/yxcx7skw(v=vs.90))  
  Contains fundamental classes and base classes that define commonly used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions. Other classes provide services supporting data type conversion, method parameter manipulation, mathematics, remote and local program invocation, application environment management, and supervision of managed and unmanaged applications.
* [System.CodeDom](https://msdn.microsoft.com/en-us/library/za6cc751(v=vs.90))  
  Contains classes that can be used to represent the elements and structure of a source code document. These elements can be used to model the structure of a source code document that can be output as source code in a supported language using the functionality provided by the [System.CodeDom.Compiler](https://msdn.microsoft.com/en-us/library/z6b99ydt(v=vs.90)) namespace.
* [System.CodeDom.Compiler](https://msdn.microsoft.com/en-us/library/z6b99ydt(v=vs.90))  
  Contains types for managing the generation and compilation of source code in supported programming languages. Code generators can each produce source code in a particular programming language based on the structure of Code Document Object Model (CodeDOM) source code models consisting of elements provided by the [System.CodeDom](https://msdn.microsoft.com/en-us/library/za6cc751(v=vs.90)) namespace.
* [System.Collections](https://msdn.microsoft.com/en-us/library/k166wx47(v=vs.90))  
  Contains interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hashtables and dictionaries.
* [System.Collections.Generic](https://msdn.microsoft.com/en-us/library/0sbxh9x2(v=vs.90))  
  Contains interfaces and classes that define generic collections, which allow users to create strongly typed collections that provide better type safety and performance than non-generic strongly typed collections.
* [System.Collections.ObjectModel](https://msdn.microsoft.com/en-us/library/ms132396(v=vs.90))  
  Contains classes that can be used as collections in the object model of a reusable library. Use these classes when properties or methods return collections.
* [System.Collections.Specialized](https://msdn.microsoft.com/en-us/library/32c13e62(v=vs.90))  
  Contains specialized and strongly typed collections; for example, a linked list dictionary, a bit vector, and collections that contain only strings.
* [System.ComponentModel](https://msdn.microsoft.com/en-us/library/z82ykwhb(v=vs.90))  
  Provides classes that are used to implement the run-time and design-time behavior of components and controls. This namespace includes the base classes and interfaces for implementing attributes and type converters, binding to data sources, and licensing components.
* [System.ComponentModel.Design](https://msdn.microsoft.com/en-us/library/s92caa5w(v=vs.90))  
  Contains classes that developers can use to build custom design-time behavior for components and user interfaces for configuring components at design time. The design time environment provides systems that enable developers to arrange components and configure their properties.
* [System.ComponentModel.Design.Data](https://msdn.microsoft.com/en-us/library/da3ha3c4(v=vs.90))  
  Contains classes for implementing design-time behavior of data-related components.
* [System.ComponentModel.Design.Serialization](https://msdn.microsoft.com/en-us/library/ffw164t5(v=vs.90))  
  Provides types that support customization and control of serialization at design time.
* [System.Configuration](https://msdn.microsoft.com/en-us/library/2a1tyt9s(v=vs.90))  
  Contains the types that provide the programming model for handling configuration data.
* [System.Configuration.Assemblies](https://msdn.microsoft.com/en-us/library/xts0dtk4(v=vs.90))  
  Contains classes that are used to configure an assembly.
* [System.Configuration.Install](https://msdn.microsoft.com/en-us/library/1yece858(v=vs.90))  
  Provides classes that allow you to write custom installers for your own components. The [Installer](https://msdn.microsoft.com/en-us/library/79e7ka7s(v=vs.90)) class is the base class for all custom installers in the .NET Framework.
* [System.Configuration.Provider](https://msdn.microsoft.com/en-us/library/0w5y43xy(v=vs.90))  
  Contains the base classes shared by both server and client applications to support a pluggable model to easily add or remove functionality.
* [System.Data](https://msdn.microsoft.com/en-us/library/ax3wd0k9(v=vs.90))  
  Contains classes that constitute most of the ADO.NET architecture. The ADO.NET architecture enables you to build components that efficiently manage data from multiple data sources. In a disconnected scenario (such as the Internet), ADO.NET provides the tools to request, update, and reconcile data in multiple tier systems. The ADO.NET architecture is also implemented in client applications, such as Windows Forms, or HTML pages created by ASP.NET.
* [System.Data.Common](https://msdn.microsoft.com/en-us/library/9tahwysy(v=vs.90))  
  Contains classes shared by the .NET Framework data providers. A .NET Framework data provider describes a collection of classes used to access a data source, such as a database, in the managed space.
* [System.Data.Design](https://msdn.microsoft.com/en-us/library/25e61h48(v=vs.90))  
  Contains classes that can be used to generate a custom typed-dataset.
* [System.Data.Linq](https://msdn.microsoft.com/en-us/library/bb292763(v=vs.90))  
  Contains classes to access relational data as objects. DataContext and related classes can be used for Reading, Creating, Updating and Deleting objects mapped to a database using mapping specified as attributes in your object model or in a separate external XML file.
* [System.Data.Linq.Mapping](https://msdn.microsoft.com/en-us/library/bb515105(v=vs.90))  
  Provides programmatic access to mapping information used by LINQ to SQL.
* [System.Data.Odbc](https://msdn.microsoft.com/en-us/library/f0tse5zk(v=vs.90))  
  Contains classes that encapsulate the .NET Framework Data Provider for ODBC. The .NET Framework Data Provider for ODBC describes a collection of classes used to access an ODBC data source in the managed space.
* [System.Data.OleDb](https://msdn.microsoft.com/en-us/library/6d9ew87b(v=vs.90))  
  Contains classes that encapsulate the .NET Framework Data Provider for OLE DB. The .NET Framework Data Provider for OLE DB describes a collection of classes used to access an OLE DB data source in the managed space.
* [System.Data.OracleClient](https://msdn.microsoft.com/en-us/library/347d2380(v=vs.90))  
  Contains classes that encapsulate the .NET Framework Data Provider for Oracle. The .NET Framework Data Provider for Oracle describes a collection of classes used to access an Oracle data source in the managed space.
* [System.Data.Sql](https://msdn.microsoft.com/en-us/library/x360htke(v=vs.90))  
  Contains classes that support SQL Server-specific functionality. The API extensions in this class add to the .NET Framework Data Provider for SQL Server ([System.Data.SqlClient](https://msdn.microsoft.com/en-us/library/8t72t3k4(v=vs.90))).
* [System.Data.SqlClient](https://msdn.microsoft.com/en-us/library/8t72t3k4(v=vs.90))  
  Contains classes that encapsulate the .NET Framework Data Provider for SQL Server. The .NET Framework Data Provider for SQL Server describes a collection of classes used to access a SQL Server database in the managed space.
* [System.Data.SqlServerCE](https://msdn.microsoft.com/en-us/library/ec4st0e3(v=vs.90))  
  Describes a collection of classes that can be used to access a database in SQL Server CE from Windows CE-based devices in the managed environment. With this namespace you can create SQL Server CE databases on a device and also establish connections to SQL Server databases that are on a device or on a remote server.
* [System.Data.SqlTypes](https://msdn.microsoft.com/en-us/library/5ft6schb(v=vs.90))  
  Contains classes for native data types within SQL Server. These classes provide a faster alternative to other data types. Using the classes in this namespace helps prevent type conversion errors caused in situations where loss of precision could occur. Because other data types are converted to and from SqlTypes behind the scenes, explicitly creating and using objects within this namespace results in faster code as well.
* [System.Diagnostics](https://msdn.microsoft.com/en-us/library/15t15zda(v=vs.90))  
  Provides classes that allow you to interact with system processes, event logs, and performance counters. This namespace also provides classes that allow you to debug your application and to trace the execution of your code. For more information, see the [Trace](https://msdn.microsoft.com/en-us/library/36hhw2t6(v=vs.90)) and [Debug](https://msdn.microsoft.com/en-us/library/6x31ezs1(v=vs.90)) classes.
* [System.Diagnostics.CodeAnalysis](https://msdn.microsoft.com/en-us/library/ms141071(v=vs.90))  
  Contains classes for interaction with code analysis tools. Code analysis tools are used to analyze code for conformance to coding conventions such as naming or security rules.
* [System.Diagnostics.Design](https://msdn.microsoft.com/en-us/library/t1ye0fs2(v=vs.90))  
  Contains classes that can be used to extend design-time support for application monitoring and instrumentation.
* [System.Diagnostics.SymbolStore](https://msdn.microsoft.com/en-us/library/dac9fh3b(v=vs.90))  
  Provides classes that allow you to read and write debug symbol information, such as source line to Microsoft intermediate language (MSIL) maps. Compilers targeting the .NET Framework can store the debug symbol information into programmer's database (PDB) files. Debuggers and code profiler tools can read the debug symbol information at run time.
* [System.DirectoryServices](https://msdn.microsoft.com/en-us/library/9t2667d1(v=vs.90))  
  Provides easy access to Active Directory from managed code. The namespace contains two component classes, [DirectoryEntry](https://msdn.microsoft.com/en-us/library/z9cddzaa(v=vs.90)) and [DirectorySearcher](https://msdn.microsoft.com/en-us/library/94se97ay(v=vs.90)), which use the Active Directory Services Interfaces (ADSI) technology. ADSI is the set of interfaces that Microsoft provides as a flexible tool for working with a variety of network providers. ADSI gives the administrator the ability to locate and manage resources on a network with relative ease, regardless of the network's size.
* [System.DirectoryServices.ActiveDirectory](https://msdn.microsoft.com/en-us/library/wwzcae1f(v=vs.90))  
  Provides a high level abstraction object model that builds around Microsoft® Active Directory® directory service tasks. The Active Directory® directory service concepts such as forest, domain, site, subnet, partition and schema are part of the object model.
* [System.DirectoryServices.Protocols](https://msdn.microsoft.com/en-us/library/4b2t08s1(v=vs.90))  
  Provides the methods defined in the Lightweight Directory Access Protocol (LDAP) version 3 (V3) and Directory Services Markup Language (DSML) version 2 (V2) standards.
* [System.Drawing](https://msdn.microsoft.com/en-us/library/xs6ftd89(v=vs.90))  
  Provides access to GDI+ basic graphics functionality. More advanced functionality is provided in the [System.Drawing.Drawing2D](https://msdn.microsoft.com/en-us/library/stk4ffd5(v=vs.90)), [System.Drawing.Imaging](https://msdn.microsoft.com/en-us/library/sbashfw3(v=vs.90)), and [System.Drawing.Text](https://msdn.microsoft.com/en-us/library/81w2k207(v=vs.90)) namespaces.
* [System.Drawing.Design](https://msdn.microsoft.com/en-us/library/ks225801(v=vs.90))  
  Contains classes that extend design-time user interface (UI) logic and drawing. You can further extend this design-time functionality to create custom toolbox items, type-specific value editors that can edit and graphically represent values of their supported types, or type converters that can convert values between certain types. This namespace provides the basic frameworks for developing extensions to the design-time UI.
* [System.Drawing.Drawing2D](https://msdn.microsoft.com/en-us/library/stk4ffd5(v=vs.90))  
  Provides advanced 2-dimensional and vector graphics functionality. This namespace includes the gradient brushes, the [Matrix](https://msdn.microsoft.com/en-us/library/w8s1ct0z(v=vs.90)) class (used to define geometric transforms), and the [GraphicsPath](https://msdn.microsoft.com/en-us/library/fbkz3sfb(v=vs.90)) class.
* [System.Drawing.Imaging](https://msdn.microsoft.com/en-us/library/sbashfw3(v=vs.90))  
  Provides advanced GDI+ imaging functionality. Basic graphics functionality is provided by the [System.Drawing](https://msdn.microsoft.com/en-us/library/xs6ftd89(v=vs.90)) namespace.
* [System.Drawing.Printing](https://msdn.microsoft.com/en-us/library/5ekk3hse(v=vs.90))  
  Provides print-related services. Typically, you create a new instance of the [PrintDocument](https://msdn.microsoft.com/en-us/library/68eybb19(v=vs.90)) class, set the properties that describe what to print, and call the [Print](https://msdn.microsoft.com/en-us/library/x29ssh93(v=vs.90)) method to actually print the document.
* [System.Drawing.Text](https://msdn.microsoft.com/en-us/library/81w2k207(v=vs.90))  
  Provides advanced GDI+ typography functionality. Basic graphics functionality is provided by the [System.Drawing](https://msdn.microsoft.com/en-us/library/xs6ftd89(v=vs.90)) namespace. The classes in this namespace allow users to create and use collections of fonts.
* [System.EnterpriseServices](https://msdn.microsoft.com/en-us/library/9b84cbdx(v=vs.90))  
  Provides an important infrastructure for enterprise applications. COM+ provides a services architecture for component programming models deployed in an enterprise environment. This namespace provides .NET Framework objects with access to COM+ services, making the .NET Framework objects more practical for enterprise applications.
* [System.EnterpriseServices.CompensatingResourceManager](https://msdn.microsoft.com/en-us/library/s1kz07a6(v=vs.90))  
  Provides classes that allow you to use a Compensating Resource Manager (CRM) in managed code. A CRM is a service provided by COM+ that enables you to include non-transactional objects in Microsoft Distributed Transaction Coordinator (DTC) transactions. Although CRMs do not provide the capabilities of a full resource manager, they do provide transactional atomicity (all-or-nothing behavior) and durability through the recovery log.
* [System.EnterpriseServices.Internal](https://msdn.microsoft.com/en-us/library/twk4f69d(v=vs.90))  
  Provides infrastructure support for COM+ services. The classes and interfaces in this namespace are specifically intended to support calls into [System.EnterpriseServices](https://msdn.microsoft.com/en-us/library/9b84cbdx(v=vs.90)) from the unmanaged COM+ classes.
* [System.Globalization](https://msdn.microsoft.com/en-us/library/abeh092z(v=vs.90))  
  Contains classes that define culture-related information, including the language, the country/region, the calendars in use, the format patterns for dates, currency, and numbers, and the sort order for strings. These classes are useful for writing globalized (internationalized) applications.
* [System.IO](https://msdn.microsoft.com/en-us/library/29kt2zfk(v=vs.90))  
  Contains types that allow synchronous and asynchronous reading and writing on data streams and files.
* [System.IO.Compression](https://msdn.microsoft.com/en-us/library/3z72378a(v=vs.90))  
  Contains classes that provide basic compression and decompression for streams.
* [System.IO.IsolatedStorage](https://msdn.microsoft.com/en-us/library/x7dzh4ws(v=vs.90))  
  Contains types that allow the creation and use of isolated stores. With these stores, you can read and write data that less trusted code cannot access and help prevent the exposure of sensitive information that can be saved elsewhere on the file system. Data is stored in compartments that are isolated by the current user and by the assembly in which the code exists.
* [System.IO.Ports](https://msdn.microsoft.com/en-us/library/tf8zk72w(v=vs.90))  
  Contains classes that control serial ports, providing a framework for synchronous and event-driven I/O, access to pin and break states, access to serial driver properties, and enumerations for specifying port characteristics.
* [System.Linq](https://msdn.microsoft.com/en-us/library/bb336768(v=vs.90))  
  Provides classes and interfaces that support queries that use Language-Integrated Query (LINQ).
* [System.Linq.Expressions](https://msdn.microsoft.com/en-us/library/bb506649(v=vs.90))  
  Contains classes, interfaces and enumerations that enable language-level code expressions to be represented as objects in the form of expression trees.
* [System.Management](https://msdn.microsoft.com/en-us/library/dwd0y33x(v=vs.90))  
  Provides access to a rich set of management information and management events about the system, devices, and applications instrumented to the Windows Management Instrumentation (WMI) infrastructure.
* [System.Management.Instrumentation](https://msdn.microsoft.com/en-us/library/187fw64w(v=vs.90))  
  Provides the classes necessary for instrumenting applications for management and exposing their management information and events through WMI to potential consumers. Consumers such as Microsoft Application Center or Microsoft Operations Manager can then manage your application easily, and monitoring and configuring of your application is available for administrator scripts or other applications, both managed as well as unmanaged.
* [System.Messaging](https://msdn.microsoft.com/en-us/library/xes6983c(v=vs.90))  
  Provides classes that allow you to connect to, monitor, and administer message queues on the network and send, receive, or peek messages.
* [System.Messaging.Design](https://msdn.microsoft.com/en-us/library/k4cyty78(v=vs.90))  
  Contains classes that can be used to extend design-time support for [System.Messaging](https://msdn.microsoft.com/en-us/library/xes6983c(v=vs.90)) classes.
* [System.Net](https://msdn.microsoft.com/en-us/library/btdf6a7e(v=vs.90))  
  Provides a simple programming interface for many of the protocols used on networks today. The [WebRequest](https://msdn.microsoft.com/en-us/library/5t9y35bd(v=vs.90)) and [WebResponse](https://msdn.microsoft.com/en-us/library/6877ayz6(v=vs.90)) classes form the basis of what are called pluggable protocols, an implementation of network services that enables you to develop applications that use Internet resources without worrying about the specific details of the individual protocols.
* [System.Net.Cache](https://msdn.microsoft.com/en-us/library/ws859082(v=vs.90))  
  Defines the types and enumerations used to define cache policies for resources obtained using the [WebRequest](https://msdn.microsoft.com/en-us/library/5t9y35bd(v=vs.90)) and [HttpWebRequest](https://msdn.microsoft.com/en-us/library/8y7x3zz2(v=vs.90)) classes.
* [System.Net.Configuration](https://msdn.microsoft.com/en-us/library/17bct748(v=vs.90))  
  Contains classes that applications use to programmatically access and update configuration settings for the System.Net namespaces.
* [System.Net.Mail](https://msdn.microsoft.com/en-us/library/dk1fb84h(v=vs.90))  
  Contains classes used to send electronic mail to a Simple Mail Transfer Protocol (SMTP) server for delivery.
* [System.Net.Mime](https://msdn.microsoft.com/en-us/library/dx9a63wd(v=vs.90))  
  Holds types that are used to represent Multipurpose Internet Mail Exchange (MIME) headers. These types are used with the types in the [System.Net.Mail](https://msdn.microsoft.com/en-us/library/dk1fb84h(v=vs.90)) namespace to specify Content-Type, Content-Disposition and Content-transfer-Encoding headers when sending email using the [SmtpClient](https://msdn.microsoft.com/en-us/library/4971yhhc(v=vs.90)) class.
* [System.Net.NetworkInformation](https://msdn.microsoft.com/en-us/library/3ew4thdx(v=vs.90))  
  Provides access to network traffic data, network address information, and notification of address changes for the local computer. The namespace also contains classes that implement the Ping utility. You can use [Ping](https://msdn.microsoft.com/en-us/library/a63bsyf0(v=vs.90)) and related classes to check whether a computer is reachable across the network.
* [System.Net.Sockets](https://msdn.microsoft.com/en-us/library/sb27wehh(v=vs.90))  
  Provides a managed implementation of the Windows Sockets (Winsock) interface for developers who need to help control access to the network.
* [System.Reflection](https://msdn.microsoft.com/en-us/library/136wx94f(v=vs.90))  
  Contains classes and interfaces that provide a managed view of loaded types, methods, and fields, with the ability to dynamically create and invoke types.
* [System.Reflection.Emit](https://msdn.microsoft.com/en-us/library/xd5fw18y(v=vs.90))  
  Contains classes that allow a compiler or tool to emit metadata and Microsoft intermediate language (MSIL) and optionally generate a PE file on disk. The primary clients of these classes are script engines and compilers.
* [System.Resources](https://msdn.microsoft.com/en-us/library/74sfy6d5(v=vs.90))  
  Provides classes and interfaces that allow developers to create, store, and manage various culture-specific resources used in an application.
* [System.Resources.Tools](https://msdn.microsoft.com/en-us/library/90h6wefe(v=vs.90))  
  Contains the [StronglyTypedResourceBuilder](https://msdn.microsoft.com/en-us/library/dkwchk0e(v=vs.90)) class, which provides support for strongly-typed resources. Beginning with the .NET Framework version 2.0, this compile-time feature encapsulates access to resources by creating classes that contain a set of static read-only (get) properties, thus making it easier to consume resources.
* [System.Runtime](https://msdn.microsoft.com/en-us/library/ms145957(v=vs.90))  
  Contains advanced types that support diverse namespaces such as [System](https://msdn.microsoft.com/en-us/library/yxcx7skw(v=vs.90)), the Runtime namespaces, and the Security namespaces.
* [System.Runtime.ConstrainedExecution](https://msdn.microsoft.com/en-us/library/t1d6ykdd(v=vs.90))  
  Defines a set of types that enumerate and define a contract for reliability between the author of some code, and the developers who take a dependency on that code.
* [System.Runtime.Hosting](https://msdn.microsoft.com/en-us/library/ms146086(v=vs.90))  
  Contains advanced types that are used in application activation within application domains.
* [System.Runtime.CompilerServices](https://msdn.microsoft.com/en-us/library/s53w0dze(v=vs.90))  
  Provides functionality for compiler writers using managed code to specify attributes in metadata that affect the run-time behavior of the common language runtime. The classes in this namespace are for compiler writers use only.
* [System.Runtime.InteropServices](https://msdn.microsoft.com/en-us/library/9esea608(v=vs.90))  
  Provides a wide variety of members that support COM interop and platform invoke services. If you are unfamiliar with these services, see [Interoperating with Unmanaged Code](https://msdn.microsoft.com/en-us/library/sd10k43k(v=vs.90)).
* [System.Runtime.InteropServices.ComTypes](https://msdn.microsoft.com/en-us/library/6bch08sd(v=vs.90))  
  Contains methods that are definitions of COM functions for managed code. These functions replace the now-obsolete UCOM\* methods in the [System.Runtime.InteropServices](https://msdn.microsoft.com/en-us/library/9esea608(v=vs.90)) namespace.
* [System.Runtime.InteropServices.CustomMarshalers](https://msdn.microsoft.com/en-us/library/91e60912(v=vs.90))  
  Supports the .NET infrastructure and is not intended to be used directly from your code.
* [System.Runtime.InteropServices.Expando](https://msdn.microsoft.com/en-us/library/cd626saf(v=vs.90))  
  Contains the [IExpando](https://msdn.microsoft.com/en-us/library/8y8e143z(v=vs.90)) interface which allows modification of an object by adding or removing its members.
* [System.Runtime.Remoting](https://msdn.microsoft.com/en-us/library/hsyt0kat(v=vs.90))  
  Provides classes and interfaces that allow developers to create and configure distributed applications.
* [System.Runtime.Remoting.Activation](https://msdn.microsoft.com/en-us/library/stabwfd7(v=vs.90))  
  Provides classes and objects that support server and client activation of remote objects.
* [System.Runtime.Remoting.Channels](https://msdn.microsoft.com/en-us/library/719d0dhf(v=vs.90))  
  Contains classes that support and handle channels and channel sinks, which are used as the transport medium when a client calls a method on a remote object.
* [System.Runtime.Remoting.Channels.Http](https://msdn.microsoft.com/en-us/library/6t1ey4f9(v=vs.90))  
  Contains channels that use the HTTP protocol to transport messages and objects to and from remote locations. By default, the HTTP channels encode objects and method calls in SOAP format for transmission, but other encoding and decoding formatter sinks can be specified in the configuration properties of a channel.
* [System.Runtime.Remoting.Channels.Ipc](https://msdn.microsoft.com/en-us/library/h9ew6576(v=vs.90))  
  Defines a communication channel for remoting that uses the Interprocess Communication (IPC) system of the Windows operating system. Because it does not use network communication, the IPC channel is much faster than the HTTP and TCP channels, but it can only be used for communication between application domains on the same physical computer.
* [System.Runtime.Remoting.Channels.Tcp](https://msdn.microsoft.com/en-us/library/syzak650(v=vs.90))  
  Contains channels that use the TCP protocol to transport messages and objects to and from remote locations. By default, the TCP channels encode objects and method calls in binary format for transmission, but other encoding and decoding formatter sinks can be specified in the configuration properties of a channel.
* [System.Runtime.Remoting.Contexts](https://msdn.microsoft.com/en-us/library/94wcef2b(v=vs.90))  
  Contains objects that define the contexts all objects reside within. A context is an ordered sequence of properties that defines an environment for the objects within it. Contexts are created during the activation process for objects that are configured to require certain automatic services such synchronization, transactions, just-in-time (JIT) activation, security, and so on. Multiple objects can live inside a context.
* [System.Runtime.Remoting.Lifetime](https://msdn.microsoft.com/en-us/library/swd34e51(v=vs.90))  
  Contains classes that manage the lifetime of remote objects. Traditionally, distributed garbage collection uses reference counts and pinging for control over the lifetime of objects. This works well when there are a few clients per service, but doesn't scale well when there are thousands of clients per service. The remoting lifetime service associates a lease with each service, and deletes a service when its lease time expires. The lifetime service can take on the function of a traditional distributed garbage collector, and it also adjusts well when the numbers of clients per server increases.
* [System.Runtime.Remoting.Messaging](https://msdn.microsoft.com/en-us/library/fe7t1zx9(v=vs.90))  
  Contains classes used to create and remote messages. The remoting infrastructure uses messages to communicate with remote objects. Messages are used to transmit remote method calls, to activate remote objects, and to communicate information. A message object carries a set of named properties, including action identifiers, envoy information, and parameters.
* [System.Runtime.Remoting.Metadata](https://msdn.microsoft.com/en-us/library/d23y3b4k(v=vs.90))  
  Contains classes and attributes that can be used to customize generation and processing of SOAP for objects and fields. The classes of this namespace can be used to indicate the SOAPAction, type output, XML element name, and the method XML namespace URI.
* [System.Runtime.Remoting.Metadata.W3cXsd2001](https://msdn.microsoft.com/en-us/library/727a8yzh(v=vs.90))  
  Contains the XML Schema Definition (XSD) defined by the World Wide Web Consortium (W3C) in 2001. The XML Schema Part2: Data types specification from W3C identifies format and behavior of various data types. This namespace contains wrapper classes for the data types that conform to the W3C specification. All date and time types conform to the ISO standards specification.
* [System.Runtime.Remoting.MetadataServices](https://msdn.microsoft.com/en-us/library/e2zw51f1(v=vs.90))  
  Contains the classes used by the Soapsuds.exe command line tool and the user code to convert metadata to and from XML schema for the remoting infrastructure.
* [System.Runtime.Remoting.Proxies](https://msdn.microsoft.com/en-us/library/zw9bxk2k(v=vs.90))  
  Contains classes that control and provide functionality for proxies. A proxy is a local object that is an image of a remote object. Proxies enable clients to access objects across remoting boundaries.
* [System.Runtime.Remoting.Services](https://msdn.microsoft.com/en-us/library/xh1yws2a(v=vs.90))  
  Contains service classes that provide functionality to the .NET Framework.
* [System.Runtime.Serialization](https://msdn.microsoft.com/en-us/library/kd1dc9w5(v=vs.90))  
  Contains classes that can be used for serializing and deserializing objects. Serialization is the process of converting an object or a graph of objects into a linear sequence of bytes for either storage or transmission to another location. Deserialization is the process of taking in stored information and recreating objects from it.
* [System.Runtime.Serialization.Formatters](https://msdn.microsoft.com/en-us/library/5s0z0c1f(v=vs.90))  
  Provides common enumerations, interfaces, and classes that are used by serialization formatters.
* [System.Runtime.Serialization.Formatters.Binary](https://msdn.microsoft.com/en-us/library/fw1y2tet(v=vs.90))  
  Contains the [BinaryFormatter](https://msdn.microsoft.com/en-us/library/y50tb888(v=vs.90)) class, which can be used to serialize and deserialize objects in binary format.
* [System.Runtime.Serialization.Formatters.Soap](https://msdn.microsoft.com/en-us/library/acw44ysk(v=vs.90))  
  Contains the [SoapFormatter](https://msdn.microsoft.com/en-us/library/5ktza7xf(v=vs.90)) class, which can be used to serialize and deserialize objects in the SOAP format.
* [System.Security](https://msdn.microsoft.com/en-us/library/9w6t6kwc(v=vs.90))  
  Provides the underlying structure of the .NET Framework security system, including base classes for permissions.
* [System.Security.AccessControl](https://msdn.microsoft.com/en-us/library/tbsb79h3(v=vs.90))
* [System.Security.Cryptography](https://msdn.microsoft.com/en-us/library/9eat8fht(v=vs.90))  
  Provides cryptographic services, including secure encoding and decoding of data, as well as many other operations, such as hashing, random number generation, and message authentication.
* [System.Security.Cryptography.Pkcs](https://msdn.microsoft.com/en-us/library/6see7k14(v=vs.90))  
  Provides programming elements for Public Key Cryptography Standards (PKCS), including methods for signing data, exchanging keys, requesting certificates, public key encryption and decryption, and other security functions.
* [System.Security.Cryptography.X509Certificates](https://msdn.microsoft.com/en-us/library/ztkw6e67(v=vs.90))  
  Contains the common language runtime implementation of the Authenticode X.509 v.3 certificate. This certificate is signed with a private key that uniquely and positively identifies the holder of the certificate.
* [System.Security.Cryptography.Xml](https://msdn.microsoft.com/en-us/library/215zakfb(v=vs.90))  
  Contains classes to support the creation and validation of XML digital signatures. The classes in this namespace implement the World Wide Web Consortium Recommendation, "XML-Signature Syntax and Processing", described at <https://www.w3.org/TR/xmldsig-core/>.
* [System.Security.Permissions](https://msdn.microsoft.com/en-us/library/24ed02w7(v=vs.90))  
  Defines classes that control access to operations and resources based on policy.
* [System.Security.Policy](https://msdn.microsoft.com/en-us/library/12hak6t3(v=vs.90))  
  Contains code groups, membership conditions, and evidence. These three types of classes are used to create the rules applied by the .NET Framework security policy system. Evidence classes are the input to security policy and membership conditions are the switches; together these create policy statements and determine the granted permission set. Policy levels and code groups are the structure of the policy hierarchy. Code groups are the encapsulation of a rule and are arranged hierarchically in a policy level.
* [System.Security.Principal](https://msdn.microsoft.com/en-us/library/39sc95wz(v=vs.90))  
  Defines a principal object that represents the security context under which code is running.
* [System.ServiceProcess](https://msdn.microsoft.com/en-us/library/3kwsb404(v=vs.90))  
  Provides classes that allow you to implement, install, and control Windows service applications. Services are long-running executables that run without a user interface. Implementing a service involves inheriting from the [ServiceBase](https://msdn.microsoft.com/en-us/library/5xh5cfw0(v=vs.90)) class and defining specific behavior to process when start, stop, pause, and continue commands are passed in, as well as custom behavior and actions to take when the system shuts down.
* [System.Text](https://msdn.microsoft.com/en-us/library/x76y3wky(v=vs.90))  
  Contains classes representing ASCII, Unicode, UTF-7, and UTF-8 character encodings; abstract base classes for converting blocks of characters to and from blocks of bytes; and a helper class that manipulates and formats String objects without creating intermediate instances of String.
* [System.Text.RegularExpressions](https://msdn.microsoft.com/en-us/library/c75he57e(v=vs.90))  
  Contains classes that provide access to the .NET Framework regular expression engine. The namespace provides regular expression functionality that can be used from any platform or language that runs within the Microsoft .NET Framework.
* [System.Threading](https://msdn.microsoft.com/en-us/library/798axes2(v=vs.90))  
  Provides classes and interfaces that enable multithreaded programming. In addition to classes for synchronizing thread activities and access to data ([Mutex](https://msdn.microsoft.com/en-us/library/01985e8f(v=vs.90)), [Monitor](https://msdn.microsoft.com/en-us/library/x090d6tf(v=vs.90)), [Interlocked](https://msdn.microsoft.com/en-us/library/5kczs5b5(v=vs.90)), [AutoResetEvent](https://msdn.microsoft.com/en-us/library/45zkwyy6(v=vs.90)), and so on), this namespace includes a [ThreadPool](https://msdn.microsoft.com/en-us/library/y5htx827(v=vs.90)) class that allows you to use a pool of system-supplied threads, and a [Timer](https://msdn.microsoft.com/en-us/library/saba8ksx(v=vs.90)) class that executes callback methods on thread pool threads.
* [System.Timers](https://msdn.microsoft.com/en-us/library/4ekh5acz(v=vs.90))  
  Provides the [Timer](https://msdn.microsoft.com/en-us/library/0tcs6ww8(v=vs.90)) component, which allows you to raise an event on a specified interval.
* [System.Transactions](https://msdn.microsoft.com/en-us/library/a90c30fy(v=vs.90))  
  Contains classes that allow your code to participate in transactions. The classes support transactions with multiple, distributed participants, multiple phase notifications, and durable enlistments..
* [System.Transactions.Configuration](https://msdn.microsoft.com/en-us/library/y707b4d3(v=vs.90))  
  Contains classes that describe configuration options used by [System.Transactions](https://msdn.microsoft.com/en-us/library/a90c30fy(v=vs.90)) classes
* [System.Web](https://msdn.microsoft.com/en-us/library/w7f1y429(v=vs.90))  
  Supplies classes and interfaces that enable browser-server communication. This namespace includes the [HttpRequest](https://msdn.microsoft.com/en-us/library/h55b6cak(v=vs.90)) class, which provides extensive information about the current HTTP request, the [HttpResponse](https://msdn.microsoft.com/en-us/library/dhy78ycf(v=vs.90)) class, which manages HTTP output to the client, and the [HttpServerUtility](https://msdn.microsoft.com/en-us/library/8409wd29(v=vs.90)) class, which provides access to server-side utilities and processes. [System.Web](https://msdn.microsoft.com/en-us/library/w7f1y429(v=vs.90)) also includes classes for cookie manipulation, file transfer, exception information, and output cache control.
* [System.Web.Caching](https://msdn.microsoft.com/en-us/library/d59zceh7(v=vs.90))  
  Provides classes for caching frequently used data on the server. This includes the [Cache](https://msdn.microsoft.com/en-us/library/8bx0tk8a(v=vs.90)) class, a dictionary that allows you to store arbitrary data objects, such as hash tables and data sets. It also provides expiration functionality for those objects, and methods that allow you to add and remove the objects. You can also add the objects with a dependency upon other files or cache entries, and perform a callback to notify your application when an object is removed from the cache.
* [System.Web.Compilation](https://msdn.microsoft.com/en-us/library/ekdf5z7a(v=vs.90))  
  Contains classes for generating and compiling custom file types within the ASP.NET build environment.
* [System.Web.Configuration](https://msdn.microsoft.com/en-us/library/x1et32w6(v=vs.90))  
  Contains classes that are used to set up ASP.NET configuration.
* [System.Web.Handlers](https://msdn.microsoft.com/en-us/library/4z3b7zw9(v=vs.90))  
  Contains HTTP handler classes that process HTTP requests to a Web server.
* [System.Web.Hosting](https://msdn.microsoft.com/en-us/library/84006ws2(v=vs.90))  
  Provides the functionality for hosting ASP.NET applications from managed applications outside of Microsoft Internet Information Services (IIS).
* [System.Web.Mail](https://msdn.microsoft.com/en-us/library/zx6wtd80(v=vs.90))  
  The classes in this namespace are obsolete; use the classes in the [System.Net.Mail](https://msdn.microsoft.com/en-us/library/dk1fb84h(v=vs.90)) namespace. Contains classes that enable you to construct and send messages using the CDOSYS message component. The mail message is delivered through either the SMTP mail service built into Microsoft Windows 2000 or through an arbitrary SMTP server. The classes in this namespace can be used either from ASP.NET or from any managed application.
* [System.Web.Management](https://msdn.microsoft.com/en-us/library/yc4kf98f(v=vs.90))  
  Contains classes and interfaces for managing and monitoring the health of Web applications.
* [System.Web.Mobile](https://msdn.microsoft.com/en-us/library/8d34s4az(v=vs.90))  
  Contains the core capabilities, including authentication and error-handling, required for building ASP.NET mobile Web applications.
* [System.Web.Profile](https://msdn.microsoft.com/en-us/library/hyhxbhz6(v=vs.90))  
  Contains classes that are used to implement the ASP.NET user profile in Web server applications.
* [System.Web.RegularExpressions](https://msdn.microsoft.com/en-us/library/5awdd3a1(v=vs.90))  
  Provides regular expressions used to parse ASP.NET files. All members of the [System.Web.RegularExpressions](https://msdn.microsoft.com/en-us/library/5awdd3a1(v=vs.90)) namespace are descendants of the [Regex](https://msdn.microsoft.com/en-us/library/6f7hht7k(v=vs.90)) class.
* [System.Web.Security](https://msdn.microsoft.com/en-us/library/kt5ssstk(v=vs.90))  
  Contains classes that are used to implement ASP.NET security in Web server applications.
* [System.Web.Services](https://msdn.microsoft.com/en-us/library/9xe4bs0s(v=vs.90))  
  Consists of the classes that enable you to create XML Web services using ASP.NET and XML Web service clients. XML Web services are applications that provide the ability to exchange messages in a loosely coupled environment using standard protocols such as HTTP, XML, XSD, SOAP, and WSDL. XML Web services enable the building of modular applications within and across companies in heterogeneous environments making them interoperable with a broad variety of implementations, platforms and devices. The SOAP-based XML messages of these applications can have well-defined (structured and typed), or loosely defined parts (using arbitrary XML). The ability of the messages to evolve over time without breaking the protocol is fundamental to the flexibility and robustness of XML Web services as a building block for the future of the Web.
* [System.Web.Services.Configuration](https://msdn.microsoft.com/en-us/library/f9awde80(v=vs.90))  
  Consists of the classes that configure how XML Web services created using ASP.NET run.
* [System.Web.Services.Description](https://msdn.microsoft.com/en-us/library/abffc213(v=vs.90))  
  Consists of the classes that enable you to publicly describe an XML Web service by using the Web Services Description Language (WSDL). Each class in this namespace corresponds to a specific element in the WSDL specification, and the class hierarchy corresponds to the XML structure of a valid WSDL document.
* [System.Web.Services.Discovery](https://msdn.microsoft.com/en-us/library/14eyext9(v=vs.90))  
  Consists of the classes that allow XML Web service clients to locate the available XML Web services on a Web server through a process called XML Web services Discovery.
* [System.Web.Services.Protocols](https://msdn.microsoft.com/en-us/library/4k59awz1(v=vs.90))  
  Consists of the classes that define the protocols used to transmit data across the wire during the communication between XML Web service clients and XML Web services created using ASP.NET.
* [System.Web.SessionState](https://msdn.microsoft.com/en-us/library/z414bbk9(v=vs.90))  
  Supplies classes and interfaces that enable storage of data specific to a single client within a Web application on the server. The session state data is used to give the client the appearance of a persistent connection with the application. State information can be stored within local process memory or, for Web farm configurations, out-of-process using either the ASP.NET State Service or a SQL Server database.
* [System.Web.UI](https://msdn.microsoft.com/en-us/library/wfbwa3bf(v=vs.90))  
  Provides classes and interfaces that allow you to create controls and pages that will appear in your Web applications as user interface on a Web page. This namespace includes the [Control](https://msdn.microsoft.com/en-us/library/983zwx2h(v=vs.90)) class, which provides all controls, whether HTML, Web, or User controls, with a common set of functionality. It also includes the [Page](https://msdn.microsoft.com/en-us/library/dfbt9et1(v=vs.90)) control, which is generated automatically whenever a request is made for a page in your Web application. Also provided are classes which provide the Web Forms Server Controls data binding functionality, the ability to save the view state of a given control or page, as well as parsing functionality for both programmable and literal controls.
* [System.Web.UI.Adapters](https://msdn.microsoft.com/en-us/library/06e98zdt(v=vs.90))  
  Contains the base classes for control adapters and page adapters, which you can use to override lifecycle states of pages and controls to modify their default markup or behavior for new markup standards or for specific browsers.
* [System.Web.UI.Design](https://msdn.microsoft.com/en-us/library/w8tfs2sb(v=vs.90))  
  Contains classes that can be used to extend design-time support for Web Forms and Web server controls.
* [System.Web.UI.Design.MobileControls](https://msdn.microsoft.com/en-us/library/55d23key(v=vs.90))  
  Obsolete. Contains classes that provide design-time support for the classes in the [System.Web.UI.MobileControls](https://msdn.microsoft.com/en-us/library/361h4hy6(v=vs.90)) namespace. The classes in this namespace are obsolete; use the classes in [System.Web.UI.Design.WebControls](https://msdn.microsoft.com/en-us/library/02shef7c(v=vs.90)) instead.
* [System.Web.UI.Design.MobileControls.Converters](https://msdn.microsoft.com/en-us/library/a8wycc8a(v=vs.90))  
  Contains classes that provide design-time support for data type converters in mobile controls.
* [System.Web.UI.Design.WebControls](https://msdn.microsoft.com/en-us/library/02shef7c(v=vs.90))  
  Contains classes that can be used to extend design-time support for Web server controls.
* [System.Web.UI.Design.WebControls.WebParts](https://msdn.microsoft.com/en-us/library/s43zfct0(v=vs.90))  
  Contains classes that provide design-time support for controls derived from classes in the [System.Web.UI.WebControls.WebParts](https://msdn.microsoft.com/en-us/library/a019833t(v=vs.90)) namespace.
* [System.Web.UI.HtmlControls](https://msdn.microsoft.com/en-us/library/tct4wcsd(v=vs.90))  
  Consists of a collection of classes that allow you to create HTML server controls on a Web Forms page. HTML server controls run on the server and map directly to standard HTML tags supported by most browsers. This allows you to programmatically control the HTML elements on a Web Forms page.
* [System.Web.UI.MobileControls](https://msdn.microsoft.com/en-us/library/361h4hy6(v=vs.90))  
  Obsolete. Contains a set of ASP.NET server controls that can intelligently render your application for different mobile devices. The classes in this namespace are obsolete; use the controls in [System.Web.UI.WebControls](https://msdn.microsoft.com/en-us/library/8bhzsw6t(v=vs.90)) instead.
* [System.Web.UI.MobileControls.Adapters](https://msdn.microsoft.com/en-us/library/aaex8fsy(v=vs.90))  
  Contains classes you can use to override lifecycle stages of a mobile control to modify its default HTML, CHTML, or WML markup or behavior for new markup standards or for specific browsers and mobile devices.
* [System.Web.UI.MobileControls.Adapters.XhtmlAdapters](https://msdn.microsoft.com/en-us/library/t98kxe7x(v=vs.90))  
  Contains classes you can use to override lifecycle stages of a mobile control to modify its default XHTML markup or behavior for new markup standards or for specific browsers and mobile devices.
* [System.Web.UI.WebControls](https://msdn.microsoft.com/en-us/library/8bhzsw6t(v=vs.90))  
  Contains classes that allow you to create Web server controls on a Web page. Web server controls run on the server and include form controls such as buttons and text boxes. They also include special purpose controls such as a calendar. Because Web server controls run on the server, you can programmatically control these elements. Web server controls are more abstract than HTML server controls. Their object model does not necessarily reflect HTML syntax.
* [System.Web.UI.WebControls.Adapters](https://msdn.microsoft.com/en-us/library/xs75ydwh(v=vs.90))  
  Contains classes you can use to override lifecycle stages of a Web control to modify a control's default markup or behavior for new markup standards or for specific browsers.
* [System.Web.UI.WebControls.WebParts](https://msdn.microsoft.com/en-us/library/a019833t(v=vs.90))  
  Contains an integrated set of classes and interfaces for creating Web pages whose appearance and behavior can be modified (personalized) by end users. The user-defined settings for each page are saved for future browser sessions.
* [System.Web.Util](https://msdn.microsoft.com/en-us/library/96ck7989(v=vs.90))  
  Contains classes that enable callback methods to be run under the scope of a transaction and that enable work to be posted to separate threads.
* [System.Windows.Forms](https://msdn.microsoft.com/en-us/library/k50ex0x9(v=vs.90))  
  Contains classes for creating Windows-based applications that take full advantage of the rich user interface features available in the Microsoft Windows operating system.
* [System.Windows.Forms.ComponentModel.Com2Interop](https://msdn.microsoft.com/en-us/library/d0hh0852(v=vs.90))  
  Contains helper classes that Visual Studio uses to display property pages while in design mode.
* [System.Windows.Forms.Design](https://msdn.microsoft.com/en-us/library/efdwh4da(v=vs.90))  
  Contains classes that support design-time configuration and behavior for Windows Forms components. These classes consist of: Designer classes that provide support for Windows Forms components, a set of design time services, UITypeEditor classes for configuring certain types of properties, and classes for importing ActiveX controls.
* [System.Windows.Forms.Design.Behavior](https://msdn.microsoft.com/en-us/library/xeskd33s(v=vs.90))  
  Contains classes for creating custom user interface behavior for components at design time.
* [System.Windows.Forms.Layout](https://msdn.microsoft.com/en-us/library/5hbet2e9(v=vs.90))  
  Contains classes that support design-time and run-time layout behaviors.
* [System.Windows.Forms.PropertyGridInternal](https://msdn.microsoft.com/en-us/library/2ky9kd2b(v=vs.90))  
  Provides internal support for the [PropertyGrid](https://msdn.microsoft.com/en-us/library/x495ht92(v=vs.90)) control. The classes in this namespace support the .NET Framework infrastructure and are not intended to be used directly from your code
* [System.Xml](https://msdn.microsoft.com/en-us/library/y3y47afh(v=vs.90))  
  Provides standards-based support for processing XML.
* [System.Xml.Schema](https://msdn.microsoft.com/en-us/library/astfyhd4(v=vs.90))  
  Contém as classes XML que fornecem suporte baseado em padrões para esquemas de linguagem de definição de esquemas XML (XSD).
* [System.Xml.Serialization](https://msdn.microsoft.com/en-us/library/e123c76w(v=vs.90))  
  Contém classes que são usadas para serializar objetos em documentos ou fluxos no formato XML.
* [System.Xml.XPath](https://msdn.microsoft.com/en-us/library/dcxb440a(v=vs.90))  
  Contém o analisador XPath e o mecanismo de avaliação. Ele suporta a Recomendação W3C XML Path Language (XPath) Versão 1.0 ( [www.w3.org/TR/xpath](https://www.w3.org/TR/xpath) ).
* [System.Xml.Xsl](https://msdn.microsoft.com/en-us/library/70k6zz96(v=vs.90))  
  Fornece suporte para transformações Extensible Stylesheet Transformation (XSLT). Ele suporta a Recomendação W3C XSL Transformations (XSLT) Versão 1.0 ( [www.w3.org/TR/xslt](https://www.w3.org/TR/xslt) ).
* [System.Xml.Xsl.Runtime](https://msdn.microsoft.com/en-us/library/ms163510(v=vs.90))  
  Fornece suporte interno para as classes no namespace [System.Xml.Xsl](https://msdn.microsoft.com/en-us/library/70k6zz96(v=vs.90)) . As classes neste namespace suportam a infraestrutura do .NET Framework e não devem ser usadas diretamente do seu código.

**Veja também**

**Conceitos**

[Referência geral do .NET Framework](https://learn.microsoft.com/en-us/previous-versions/x9t6k3aa(v=vs.90))

[Visão geral da biblioteca de classes do .NET Framework](https://msdn.microsoft.com/en-us/library/hfa3fa08(v=vs.90))

[**Inglês (Estados Unidos)**](https://learn.microsoft.com/en-us/locale?target=https%3A%2F%2Flearn.microsoft.com%2Fen-us%2Fprevious-versions%2Fd11h6832(v%3Dvs.90)%3Fredirectedfrom%3DMSDN)

[**Suas escolhas de privacidade**](https://aka.ms/yourcaliforniaprivacychoices)

Tema

* [Versões anteriores](https://learn.microsoft.com/en-us/previous-versions/)
* [Blogue](https://techcommunity.microsoft.com/t5/microsoft-learn-blog/bg-p/MicrosoftLearnBlog)
* [Contribuir](https://learn.microsoft.com/en-us/contribute/)
* [Privacidade](https://go.microsoft.com/fwlink/?LinkId=521839)
* [Termos de Uso](https://learn.microsoft.com/en-us/legal/termsofuse)
* [Marcas registradas](https://www.microsoft.com/legal/intellectualproperty/Trademarks/)
* © Microsoft 2025