|  |  |
| --- | --- |
| **Distribution Procedure** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/best_practices.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/commercial_distribution.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm) |

NinjaTrader makes it easy to distribute complete packages for your clients. Not only can you distribute your indicators and strategies, but you can also seamlessly deploy your own custom assemblies, native DLLs, chart templates, and Market Analyzer templates to your clients.

**Creating the distribution package**

To create a distribution package, please follow the steps shown [here](https://ninjatrader.com/es/support/helpGuides/nt8/export.htm) for creating a Export file containing your NinjaScript indicators and/or strategies.

It is strongly recommended that you export your scripts as an assembly and use SecureTeam's Agile.NET. Only this process will provide you with the highest level of security possible in order to protect your intellectual property. For more information on using SecureTeam's Agile.NET please see the [Protection/DLL Security](https://ninjatrader.com/es/support/helpGuides/nt8/export.htm) section.

After you finish using the Export utility you will find the distribution package as a .zip file located in My Documents\NinjaTrader 8\bin\Custom\ExportNinjaScript. If you only wanted to distribute your NinjaScript files then providing your customers with this .zip and having them go through the [Import](https://ninjatrader.com/es/support/helpGuides/nt8/import.htm) process would install it on their machines. If you wish to add more custom files to your distribution package, please see the sections below.

Critical: It is important to let your customers know that NinjaTrader 8 indicators and strategies are NOT necessarily compatible with NinjaTrader Version 7.

tog_minus        [Adding custom assemblies or native DLLs](javascript:HMToggle('toggle','AddingCustomAssembliesOrNativeDlls','AddingCustomAssembliesOrNativeDlls_ICON'))

|  |  |
| --- | --- |
| 1. Locate your base .zip distribution package  2. Open the .zip  3. Add to the .zip file your assemblies and/or your DLL files to the root directory of the .zip. These files cannot be behind any extra directory structures and must be directly in the root of the .zip    For custom assemblies, you will also need to add to the root of the .zip a .txt file called AdditionalReferences.txt    1.Bring up the Windows Start Menu  2.Go to the Run field and type "notepad" without the quotes and press Enter  3.In Notepad, type the name of your custom assembly and then save the file as a text file with the name "AdditionalReferences".   Ex: If your custom assembly's name was MyCustomAssembly.dll and MyCustomAssembly.cs, in the AdditionalReferences.txt file you would type "MyCustomAssembly" without the quotes.   |  | | --- | | **Note**: If you have multiple custom assemblies to add you can append each of the assembly's names into the same AdditionalReferences.txt file on new lines | |

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

tog_minus        [Adding templates](javascript:HMToggle('toggle','AddingChartTemplates','AddingChartTemplates_ICON'))

|  |
| --- |
| If you are distributing an indicator package, you may also want to distribute a prebuilt Chart Template that your customers can use to quickly bring up preferred settings for your chart setup. The same instructions here would work though for all other templates as well, i.e. MarketAnalyzer, DrawingTools - as long as the relative folder under templates is correctly set per the template category you're working with. The below steps run through the process for Chart templates.    1. Locate your base .zip distribution package  2. Open the .zip  3. Create a new directory called "templates" without the quotes  4. Navigate into the "templates" directory and create another new directory called "Chart"  5. Navigate into the "Chart" directory. Copy the .xml chart templates you wish to distribute from My Documents\NinjaTrader 8\templates\Chart to this directory in the .zip |

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

tog_minus        [Adding workspaces](javascript:HMToggle('toggle','AddingWorkspaces','AddingWorkspaces_ICON'))

|  |
| --- |
| If you are distributing an indicator package, you may also want to distribute a prebuilt Workspace that your customers can use to quickly bring up preferred settings for your workspace. The below steps run through the process for workspaces.    1. Locate your base .zip distribution package  2. Open the .zip  3. Create a new directory called "workspaces" without the quotes  4. Navigate into the "workspaces" directory. Copy the .xml workspace you wish to distribute from My Documents\NinjaTrader 8\workspaces to this directory in the .zip |

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

tog_minus        [Adding custom](javascript:HMToggle('toggle','AddingMarketAnalyzerTemplates','AddingMarketAnalyzerTemplates_ICON')) resource files

|  |
| --- |
| You may run into the need to distribute other custom files such as pictures for buttons for use with your product as well. This can be achieved via the same approach as for the templates, as long as the resources folder is under the parent templates directory.    1. Locate your base .zip distribution package  2. Open the .zip  3. Create a new directory called "templates" without the quotes  4. Navigate into the "templates" directory and create another new directory, for example "MyResources"  5. Navigate to the directory where your files are stored. Copy the resource files you wish to distribute from this directory to your custom directory from step 4 in the .zip |

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

|  |  |
| --- | --- |
| **Note**: When modifying the .zip archives, if your zip utility application has an option for storing or recreating relative paths please be sure to turn this off as it will cause problems when importing the archive to NinjaTrader. | |
| **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'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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'))

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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'))

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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) |
| **NinjaScript Best Practices** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/code_breaking_changes.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/distribution.htm) |

There are some best practices to be aware of when developing NinjaScript classes. The following tables present a non-exhaustive list of considerations to keep in mind when designing and implementing your code.

|  |
| --- |
| **Note**:   NinjaTrader is multi-threaded and event driven. Always assume that any of the methods you implement in NinjaScript could be called from another thread. |

tog_minus        [State management practices](javascript:HMToggle('toggle','StateResourceManagement','StateResourceManagement_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Managing Resources**  The [OnStateChange(](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm)) method is called anytime there has been a change of [State](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) and can be used to help you setup, manage, and destroy several types of resources.  Where these values are setup is highly dependent on the kind of resource you are using.  The section below will cover how to manage various resources throughout different states.    **Setting Default UI Property Grid values**  Reserve **State.SetDefaults** for defaulting any public properties you wish to have exposed on the UI property grid.   You should also use this State for setting default desired NinjaScript property behavior which can be overridden from the property grid (e.g. [Calculate](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm), [IsOverlay](https://ninjatrader.com/es/support/helpGuides/nt8/isoverlay.htm), etc.).  For Plots and Lines you wish to configure, [AddPlot()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm), [AddLine()](https://ninjatrader.com/es/support/helpGuides/nt8/addline.htm) should also have their default values set during this State     |  | | --- | | **Why:**Public values of the NinjaScript object in **SetDefaults** are pushed to the UI property grid for an opportunity to change settings of your object. |      | ns **Best practice** | | --- | | protected override void OnStateChange() {   // these are the values that show up as default on the UI   if (State == State.SetDefaults)   {     Calculate = Calculate.OnPriceChange;     IsOverlay = false;       Period = 50;       AddPlot(Brushes.Blue, "Plot Value");     AddLine(Brushes.Gray, 100, "Threshold");   } } |     For public properties you do **NOT** wish exposed to the UI property grid, set the [Browsable](https://ninjatrader.com/es/support/helpGuides/nt8/browsableattribute.htm) attribute to false:     | ns **Best practice** | | --- | | [Browsable(false)] // prevents from showing up on the UI property grid public int Communicator { get; set; } |     On indicators, properties you wish to set from other objects, set the [NinjaScriptPropertyAttribute](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascriptpropertyattribute.htm):     | ns **Best practice** | | --- | | [NinjaScriptProperty] // can now call MyIndicator(20) from another object public int Period { get; set; } |     The default behavior is to serialize any public properties to a Workspace or Template file when saving.  However, not all objects can be serialized - or you may wish to exclude a property from being saved and restored.  For these scenarios, set the [XMLIgnore](https://ninjatrader.com/es/support/helpGuides/nt8/xmlignoreattribute.htm) attribute to the property:     | ns **Best practice** | | --- | | [XmlIgnore] // removes from serialization     public Brush DownBrush { get; set; } |      |  | | --- | | **Tip:**See the [Working with Brushes](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_brushes.htm) section of the Help Guide for information on properly serializing brushes |     **Calculating run-time object values** Do not attempt to do advanced calculations or try to access object references in **State.SetDefaults**.  This State should be kept as lean as possible, and any calculation logic should be delayed until at least **State.Configure**     |  | | --- | | **Why:**Your object will be called in situations you may not be expecting. You can read more about this subject on [Understanding the life cycle of your NinjaScript objects](https://ninjatrader.com/es/support/helpGuides/nt8/understanding_the_lifecycle_of.htm) |      | ns **Practice to avoid** | | --- | | protected override void OnStateChange() {   if (State == State.SetDefaults)   {       // logic could take longer than desired as the list of indicator names is populated     for (int i = 0; i <= array.length; i ++)         DoWork(i);      // possible null reference exception since TickSize is not set yet     Period = 5 \* TickSize;   } } |      | ns **Best practice** | | --- | | protected override void OnStateChange() {   // Complex operations should be delayed to >= State.Configure   if (State == State.Configure)   {     for (int i = 0; i < = array.length; i ++)         DoWork(i);   }     // information related to market data is not available until at least State.DataLoaded   else if (State == State.DataLoaded)   {     Period = 5 \* TickSize;   } } |     **Setting class level variables**  Do not set variables at the class level unless they are constant.  You should delay setting or resetting variables until the **State** has reached **State.Configure**.  You can use const keyword to differentiate values which do not change from variables which do change.     |  | | --- | | **Why**:  Waiting to set up and define resources until the object has been configured ensures that values not set up and declared prematurely. |      | ns **Best practice** | | --- | | // value is always 5, it can be made constant and declared at the class level private const int multiplier = 5;   // these values can change, may be better to delay setting until State.Configure private int counter; private List<int> myList;  protected override void OnStateChange() {   if (State == State.Configure)   {     counter = 0;     myList = new List<int>();   }   } |     **Resetting class level variables for Strategy Analyzer Optimization**    To take advantage of performance optimizations, developers may need to reset class level variables in the strategy otherwise unexpected results can occur.     |  | | --- | | **Why**:  When optimizing a strategy, instances may or may not be recycled depending on the strategy [IsInstantiatedOnEachOptimizationIteration](https://ninjatrader.com/es/support/helpGuides/nt8/isinstantiatedoneachoptimizationiteration.htm) setting. |      | ns **Best practice** | | --- | | // examples of fields which need to be reset private double myDouble; private bool myBool; private DateTime myDateTime; private Order myOrderObject; private Brush myBrushObject; private Array myIntArray; private List<object> myList; private SMA mySMAIndicator; private Series<double> mySeries;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     // disabled to take advantage of performance gains     // However any strategy state that would be mutable after State.SetDefaults needed to be reset for the next run.     IsInstantiatedOnEachOptimizationIteration = false;   }   else if (State == State.Configure)   {     // Since these values are not dependent on bars, they can be reset as early as State.Configure     myDouble = double.MinValue;     myBool = false;     myDateTime = DateTime.MinValue;     myOrderObject = null;     myBrushObject = null;       if (myIntArray != null)         Array.Clear(myIntArray, 0, myIntArray.Length);     else         myIntArray = new int[20];       if (myList != null)         myList.Clear();     else         myList = new List<object>();   }     else if (State == State.DataLoaded)   {     // Since these values do are dependent on bars, they should only reset during State.DataLoaded     mySMAIndicator = SMA(14);     mySeries = new Series<double>(this);   } } |     **Accessing properties related to market data**  Do not attempt to access objects related to instrument market data until the **State** has reached **State.DataLoaded**     |  | | --- | | **Why**: Waiting to access objects that depend on market data until **DataLoaded** prevents access errors in all scenarios |      | ns **Best practice** | | --- | | protected override void OnStateChange() {   if (State == State.DataLoaded)   {     // these objects and their related members are not available until State.DataLoaded     Print(Bars.Count);     Print(Instrument.FullName);     Print(BarsPeriod.BarsPeriodType);     Print(TradingHours.TimeZone);     Print(Input);   } } |      |  | | --- | | **Note**: All additional data series must be added in **State.Configure**(this includes series that any hosted script potentially needs as well - [more info](http://ninjatrader.com/support/helpGuides/nt8/en-us/adddataseries.htm)). Since objects such as [Instrument](https://ninjatrader.com/es/support/helpGuides/nt8/instrument.htm), [BarsPeriod](https://ninjatrader.com/es/support/helpGuides/nt8/barsperiod.htm), [TradingHours](https://ninjatrader.com/es/support/helpGuides/nt8/tradinghours.htm), etc. are **NOT** guaranteed to be available until **State.DataLoaded**, you cannot reliably use the primary instrument properties as arguments in [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm).  Attempting to add a data series dynamically is **NOT** guaranteed and therefore should be avoided.  In some cases, you may be able to use a [BarsRequest()](https://ninjatrader.com/es/support/helpGuides/nt8/barsrequest.htm) to obtain market data for other instruments and intervals. |     **Setting up resources that rely on market data**  For objects which depend on market data, delay their construction until the **State** has reached **State.DataLoaded**     |  | | --- | | **Why**: Waiting to construct objects that depend on market data until **DataLoaded** ensures that their underlying input contains significant values in all scenarios. |      | ns **Best practice** | | --- | | // these resources depend on bars, wait until State.DataLoaded to instantiated private EMA myEMA; private Series<double> mySeries; private SessionIterator mySessionIterator;   protected override void OnStateChange() {     if (State == State.DataLoaded)   {     myEMA = EMA(20);     mySeries = new Series<double>(this);     mySessionIterator = new SessionIterator(Bars);   } } |     **Accessing element on the UI**  For objects which exist on the UI (e.g., [ChartControl](https://ninjatrader.com/es/support/helpGuides/nt8/chartcontrol.htm), [ChartPanel](https://ninjatrader.com/es/support/helpGuides/nt8/chartpanel.htm), [ChartBars](https://ninjatrader.com/es/support/helpGuides/nt8/chartbars.htm), [NTWindow](https://ninjatrader.com/es/support/helpGuides/nt8/ntwindow.htm), etc.) wait until the State has reached State.Historical.  This practice is correct for both reading properties or should you wish to add custom elements to the existing UI.     |  | | --- | | **Why**:  NinjaTrader UI related objects are not guaranteed to be available until historical data processing has started. |      | ns **Best practice** | | --- | | protected override void OnStateChange() {   // wait until at least State.Historical   if (State == State.Historical)   {     // and double check UI object is not null before accessing     if (ChartControl != null)     {         Print(ChartControl.Properties.ChartBackground);     }   } } |     **Transitioning order references from historical to real-time**  When dealing with strategy based orders which have transitioned from historical to real-time, you will need to ensure that locally stored order references are also updated.     |  | | --- | | **Why**: As the core order object updates, NinjaTrader has no specific way to update your locally stored order references.  You can read more about this subject on the Advanced Order Handling topic: [Transitioning order references from historical to live](https://ninjatrader.com/es/support/helpGuides/nt8/advanced_order_handling.htm) |      | ns **Best practice** | | --- | | protected override void OnStateChange() {   // one time only, as we transition from historical to real-time   if (State == State.Realtime)   {     // convert any old historical order object references     // to the new live order submitted to the real-time account     if (myOrder != null)         myOrder = GetRealtimeOrder(myOrder);   } } |     **Terminating custom resources**  Use a flag to track when resources have been set up properly before attempting to destroy them.     |  | | --- | | **Why**:  Checking that an object has been configured ensures that values not destroyed prematurely. You can read more about this subject on [Understanding the life cycle of your NinjaScript objects](https://ninjatrader.com/es/support/helpGuides/nt8/understanding_the_lifecycle_of.htm) |      | ns **Best practice** | | --- | | protected override void OnStateChange() {   if (State == State.Configure)   {     myObject = new object();     // set a flag to indicator object has been configured     configured = true;   }     else if (State == State.Terminated)   {     // only dispose of object if it has been configured     if (configured)     {         myObject.Dispose();     }   } } | |

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

tog_minus        [Error handling practices](javascript:HMToggle('toggle','Errorhandling','Errorhandling_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Safely accessing reference objects**  Although there are documented **States** where objects are available, the implementation could change.  If you are accessing a reference object, please do so by first checking that the object is not null.     | ns **Best practice** | | --- | | // checking to ensure chart control is available in all situations // will help to ensure this logic below does not generate errors at a later time if(ChartControl != null) {   myBackgroundBrush = ChartControl.Properties.ChartBackground; } |     **Accessing objects which terminate**  To protect against race conditions and access errors, you should temporarily check for reference errors any time you attempt to do something with an object.     |  | | --- | | **Why**: **OnStateChange()** runs asynchronous to other NinjaScript events.  You can run into scenarios where you **State.Terminated** logic is called in the middle of OnBarUpdate(), OnRender() etc. |      | ns **Best practice** | | --- | | protected override void OnStateChange() {   // this logic runs asynchronously to other events   if (State == State.Terminated)   {     myObject = null;   } } protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   if (myObject == null)     return;     // for safety, always check for null references before attempting to access an object   // even if you have once checked for null references earlier run-time   if (myObject != null)     myObject.DoSomething(); } |     **Proving instructions for non-ninjascript properties**  Do not attempt to modify existing UI "Properties" to meet your specific needs.  These features are exposed to allow you to read the environment state and make decisions to alter how your code executes, but should not be relied on to modify settings on behalf of the user.  While these objects from these classes have setters for technical reasons, you should not attempt to amend the values through code.  Instead, you should issue warnings or log errors instructing users to modify settings when required:     |  | | --- | | **Why**:  NinjaTrader makes no guarantee that the requested changes will take effect, and user settings always take precedences.  This includes the user defined [ChartControl.Properties](https://ninjatrader.com/es/support/helpGuides/nt8/chartcontrol_properties.htm), [ChartBars.Properties](https://ninjatrader.com/es/support/helpGuides/nt8/chartbars_properties.htm), and [ChartPanel.Properties](https://ninjatrader.com/es/support/helpGuides/nt8/chartscale_properties.htm). Furthermore, two different user scripts could be installed which also attempt to modify properties you are relying which could introduce conflicts. |      | ns **Best practice** | | --- | | if (State == State.Historical) {   if (ChartControl.Properties.EquidistantBarSpacing == true)   {     Draw.TextFixed(this, "error", "This indicator works best with Equidistant BarSpacing set to false.", TextPosition.BottomRight);   } } |     **Modifying UI elements and multi-threading**  When interacting with UI objects, such as obtaining UI information, or modifying the existing layout, always use the NinjaScript's Dispatcher asynchronously     |  | | --- | | **Critical**:  Improper thread handling from a NinjaScript object is a common cause of application deadlocks.  Please be sure to read more information on [Multi-Threading Consideration for NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/multi-threading.htm) |      | ns **Best practice** | | --- | | // using a Dispatcher will ensure that the corresponding action executes on the associated thread this.Dispatcher.InvokeAsync(() => {   UserControlCollection.Add(new System.Windows.Controls.TextBlock   {     Text = "\nAdded by the ChartControl Dispatcher."   }); }); |       **Properly implementing try/catch blocks**  Unless you are specifically debugging a method, the use of a try-catch block should be scoped to a particular area of logic.  Do **NOT** try to handle all of your execution logic under one giant try-catch block.     |  | | --- | | **Why**:  Larger try-catch blocks can not only be harder to debug, but can introduce performance issues at run-time |      | ns **Practice to avoid** | | --- | | protected override void OnBarUpdate() {   try   {     // encapsulates entire OnBarUpdate logic   }   catch (Exception ex)   {     // attempt to handle all errors in one catch   } } |     **Using WPF brushes**  Try to use a static predefined Brush if possible.  If you need to customize a new brush object, make sure to .Freeze() the brush before using it.     |  | | --- | | **Why**:  The pre-defined brushes are thread safe and do not require any special handling.  Custom defined brushes, on the other hand, are **NOT** thread-safe and must be frozen otherwise cross-thread exceptions can occur. |      | ns **Best practice** | | --- | | // predefined brush BackBrush = Brushes.Blue;   // if you are using a custom brush to e.g., modify the opacity SolidColorBrush opaqueBlue = new SolidColorBrush(Colors.Blue) {Opacity = .25f};   // or just using at custom color not available in pre-defined brushes class SolidColorBrush coolGreen = new SolidColorBrush(Color.FromRgb(30, 255, 128));   // you must freeze these brushes after they are constructed! opaqueBlue.Freeze(); coolGreen.Freeze(); |     **barsAgo indexer vs. absolute bar Index**  As you probably know, you can quickly look up the bar value on the chart by calling a [PriceSeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) barsAgo indexer, e.g., Close[0].  However, the internal indexer and pointers about the barsAgo value are only guaranteed to be correctly synced and updated during a market data event.  As a result, you should favor using the absolute [GetValueAt()](https://ninjatrader.com/es/support/helpGuides/nt8/getvalueat.htm) methods during events which are not driven by price     |  | | --- | | **Why**:  Attempting to call the barsAgo indexer in an event method that is not driven by market data can yield unexpected results. |      | ns **Best practice** | | --- | | // OnRender is not a market data event; barsAgo pointers are not guaranteed to be in sync protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   Print(mySMA.GetValueAt(CurrentBar)); }   // same is true for you custom events private void myCustomClickHandler(object sender, MouseButtonEventArgs e) {   Print(Close.GetValueAt(CurrentBar)); } |      |  | | --- | | **Tip**:  If you have programming requirements which rely on a PriceSeries indexer, you can use the [TriggerCustomEvent()](https://ninjatrader.com/es/support/helpGuides/nt8/triggercustomevent.htm) delegate which will update the internal pointers and indexes before executing the logic you specify. |     **Casting safely**  Avoid type casting and type conversion as much as possible.  Casting from a mixed collection of types is also prone to exceptions especially in situations that may not occur when you originally test your code.     |  | | --- | | **Why**:   The practice to avoid code below could work in some scenarios but would generate errors if other types were added to that collection that you were not anticipating. |      | ns **Practice to avoid** | | --- | | // This would run without errors if there were \_ONLY\_ type HoriztonalLine on the chart // But you risk a likely 'System.InvalidCastException' when other draw types are in that collection foreach (HorizontalLine hLine in DrawObjects) {   } |     If you must cast, do so safely and avoid implicit casts to types which may not be guaranteed to succeeded     | ns **Best practice** | | --- | | // Use the base IDrawingTool type and then cast to the desired type within the for loop foreach (IDrawingTool hLine in DrawObjects) {   // Note:  to prevent further errors, your type casting should be done using the "as" keyword   // Opposed to a direct cast:   // HorizontalLine myLine = (HorizontalLine) hLine;    HorizontalLine myLine = hLine as HorizontalLine;     // This will allow you to ensure the cast actually occurred   if (myLine != null)   {     Print(myLine.StartAnchor.Price);   } } | |

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

tog_minus        [Performance practices](javascript:HMToggle('toggle','Performance','Performance_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Referencing indicator methods**  In general, when calling an Indicator return method, there is some internal caching which occurs by design to help reduce memory conception.     |  | | --- | | **Why**:  While the designed indicator caching improves general memory performance, there is an implied cost of actually looking up the cached indicator |      | ns **Practice to avoid** | | --- | | // each time you call the SMA() return method there is a small performance cost // implied from the time it takes to look up the cached instance if (Close[0] > SMA(20)[0]) {   Print(SMA(20)[0]);   EnterLongLimit(SMA(20)[0]);   Draw.Dot(this, Time[0].ToString(), false, 0, SMA(20)[0], Brushes.DarkGreen); } |      |  | | --- | | **Note**:  Indicator caching **ONLY**occurs when an indicator is recalled with the same **EXACT** parameters and input. (i.e. when a previously called indicator is called a second time with new parameters, a second instance will be created / cached) |     If you are reusing an indicator several times through your code (especially indicators with many parameters), you can take further steps to refine performance by storing a reference to the indicator instance yourself (although it is by no means a requirement, and this suggestion does not need to be followed strictly)     | ns **Best practice** | | --- | | private SMA mySma;   protected override void OnStateChange() {   // when the indicator begins processing   // save an instance of the SMA indicator with the desired input     if (State == State.Historical)   {     mySma = SMA(20);   } }   protected override void OnBarUpdate() {   // use the referenced mySMA throughout the lifetime of the script   if (Close[0] > mySma[0])   {     Print(mySma[0]);     EnterLongLimit(mySma[0]);     Draw.Dot(this, Time[0].ToString(), false, 0, mySma[0], Brushes.DarkGreen);   } } |     **Marking object references for garbage collection**  While it is not always necessary to set objects to null, doing so will mark them for garbage collection sooner and help prevent unnecessary memory resources from being utilized.     |  | | --- | | **Why**:   In general you should be diligent to set stored memory objects to null when you are done using them, especially in situations where a NinjaScript object may be running for an extended period. |      | ns **Best practice** | | --- | | protected override void OnBarUpdate() {   // saving "myDot" creates an additional reference in memory   Dot myDot = Draw.Dot(this, "myDot" + CurrentBar, false, Time[0], Close[0], Brushes.Blue);     if (conditionToRemove)   {     // remove draw object will remove the object from the chart     RemoveDrawObject("myDot");       // but your local object "myDot" is still stored in memory.     // Explicitly setting to null will ensure object is marked for garbage collection     myDot = null;   } } |      |  | | --- | | **Note**:  The example above demonstrates using a draw object, but the practice can be extended to any object you store in memory (e.g., orders, brushes, custom objects, etc) |     **Disposing of custom resources**  Dispose of objects that inherit from IDisposable or put into a Using statement.     |  | | --- | | **Why**:  NinjaTrader is not guaranteed to dispose of objects for you.  To avoid unnecessary memory consumption, always manage your resources by creating a variable and dispose of the object. |      | ns **Best practice** | | --- | | // example of object instantiated which need to be disposed StreamWriter writer = new StreamWriter("some\_file.txt");   // use the object writer.WriteLine("Some text");   // implements IDisposbile, make sure to call .Dispose() when finished writer.Dispose();   // or put in "using" statement which implicitly calls .Dispose() when finished using (StreamWriter writer2 = new StreamWriter("some\_file.txt")) {   writer2.WriteLine("Some text"); } |      |  | | --- | | **Tip**:  This is most commonly applicable when using SharpDX resources for custom rendering.  Please be sure to review the information on [Best Practices for SharpDX Resources](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm#bestpracticesforsharpdxresources) |     **Avoiding duplicate calculations**  Be mindful where and when your potentially complex calculations would be recalculated and thus run the risk of being calculated redundantly. For example, you may have logic which only needs to calculate, e.g., once per instance, once per session, once per bar, etc.     | ns **Best practice** | | --- | | // get GetPreviousTradingDayEnd() is expensive to look up // but value only needs to be looked up once a day -> only calcualte on first bar of session if (Bars.IsFirstBarOfSession) {   TradingHours.GetPreviousTradingDayEnd(Time[0]); } |     The same considerations would apply to variables or function calls that would not change their output value for the currently processed bar on [Calculate.OnEachTick](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm) or [.OnPriceChange](https://ninjatrader.com/es/support/helpGuides/nt8/calculate.htm), thus there would be no need handling them outside of [IsFirstTickOfBar](https://ninjatrader.com/es/support/helpGuides/nt8/isfirsttickofbar.htm)     | ns **Best practice** | | --- | | // dedicated logic to cache the prior sum on each tick of bar // While it is a good practice, this can cause problems for bar types which may remove last bar (see below) if (IsFirstTickOfBar)   priorSum = sum;   sum = priorSum + Input[0] - (CurrentBar >= Period ? Input[Period] : 0); Value[0] = sum / (CurrentBar < Period ? CurrentBar + 1 : Period); |     **Caching values on bars which remove last bar**  Building on the previous example, be careful when caching values on the first tick of bar if using bars types which are [IsRemoveLastBarSupported](https://ninjatrader.com/es/support/helpGuides/nt8/isremovelastbarsupported.htm).  To see how to handle these situations best, take a look at the default SMA indicator which has an additional logic branch which disables caching on those bar types:     | ns **Best practice** | | --- | | // logic below disables first tick of bar caching only on bar types which remove last bar if (BarsArray[0].BarsType.IsRemoveLastBarSupported) {   if (CurrentBar == 0)     Value[0] = Input[0];   else   {     double last = Value[1] \* Math.Min(CurrentBar, Period);       if (CurrentBar >= Period)         Value[0] = (last + Input[0] - Input[Period]) / Math.Min(CurrentBar, Period);     else         Value[0] = ((last + Input[0]) / (Math.Min(CurrentBar, Period) + 1));   } } |     **Precomputing values instead of calculating in OnRender()**  To preserve good performance, always err on the side of caution if you are using OnRender for any calculation logic.     |  | | --- | | **Why**:   OnRender() is called frequently as you interact with the Chart, which can cause calculations to occur much more often than the related market data events and can cause unnecessary spikes in CPU consumption. |      | ns **Practice to avoid** | | --- | | protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {  // continually recalling the same value methods is unnecessary in this situation       double myValue = Bars.GetClose(CurrentBar) + Bars.GetOpen(CurrentBar);     // render myValue } |      | ns **Best practice** | | --- | | private double myValue;  protected override void OnBarUpdate() {   // myValue only needs to update when OnBarUpdate() is called   // and then can be passed to OnRender() for chart rendering purposes   myValue = Close[0] + Open[0]; }   protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   // if needed, you can always check that myValue has actually been set   if (myValue > double.MinValue)   {     // render myValue   } } |     **Restricting OnRender() calculations to visible ChartBars**  Use the [ChartBars.FromIndex](https://ninjatrader.com/es/support/helpGuides/nt8/chartbars_fromindex.htm) and [ChartBars.ToIndex](https://ninjatrader.com/es/support/helpGuides/nt8/chartbars_toindex.htm) to limit calculations to only what is visible on the chart     |  | | --- | | **Why:**Rendering should be reserved for rendering on what is visible on the Chart.  Performing calculations on bar index which are not visible can cause random spikes in CPU consumption. |      | ns **Best practice** | | --- | | protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   // restricting this loop to only the ChartBars.From/ToIndex limits the loop to only what is visible on the chart   for (int barIndex = ChartBars.FromIndex; barIndex <= ChartBars.ToIndex; barIndex++)   {     Print(ChartControl.GetSlotIndexByX(barIndex));   } } |     **Using DrawObjects vs custom graphics in OnRender()**  When using [Draw methods](https://ninjatrader.com/es/support/helpGuides/nt8/drawing.htm), a new instance of the Draw object is created including its custom rendering and calculation logic.  These methods are convenient in many situations, but can quickly introduce performance issues if used too liberally.  In some situations, you may see better performance for rendering via [SharpDX](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx.htm) in [OnRender()](https://ninjatrader.com/es/support/helpGuides/nt8/onrender.htm).     |  | | --- | | **Why**: Each draw object instance will see its own OnRender() called to render values. If you instead implement custom rendering in the your object, you would only see a single OnRender() call for your custom created graphics. |      | ns **Practice to avoid** | | --- | | protected override void OnBarUpdate() {   // this would draw a dot on every bar on the chart   // each instance would need to call its own OnRender() method   // not a very efficient use a draw method   Draw.Dot(this, "everyDot" + CurrentBar, false, 0, Close[0], Brushes.Blue); } |     With just a little extra code (much less than what is in the Draw methods) custom SharpDX rendering greatly reduces CPU and Memory consumption     | ns **Best practice** | | --- | | protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   // achieves the same effect of drawing a dot on every bar   // but only needs to call your object's OnRender()   for (int index = ChartBars.FromIndex; index <= ChartBars.ToIndex; index++)   {     float price = chartScale.GetYByValue(Close.GetValueAt(index));     float bar = chartControl.GetXByBarIndex(ChartBars, index);     float radius = (float) chartControl.BarWidth;       SharpDX.Direct2D1.Ellipse dot = new SharpDX.Direct2D1.Ellipse(new SharpDX.Vector2(bar, price), radius, radius);       using (SharpDX.Direct2D1.SolidColorBrush brush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Blue))     {         RenderTarget.FillEllipse(dot, brush);     }   } } |      |  | | --- | | **Tip**:  One of the advantages of using a Draw.Method is the returned Draw Objects contains metadata which could be used later (such as for obtain the bar index or price value of the dot later on).  If you would use this metadata later on, using a Draw method would be in your best interests.  However, if you are solely looking to render figures on a chart, favoring your custom SharpDX methods can drastically improve performance. |     **Responding to user events**  Do **NOT** use OnRender() for purposes other than rendering.  If you need events to hook into user interactions, consider adding your own event handler.  The example below shows registering the ChartPanel MouseDown event and registering a custom WPF control     |  | | --- | | **Why:**OnRender() may call more or less frequently than you anticipated.  Using your own custom event handlers allows you control and isolate user event logic you are looking to capture |      | ns **Best practice** | | --- | | protected override void OnStateChange() {   if (State == State.Historical)   {     // subscribe to chart panel mouse down event     if (ChartPanel != null) ChartPanel.MouseDown += DoUserClickedChartPanelEvent;       // subscribe to a custom UI element mouse down event     if (myWPFControl != null) myWPFControl.MouseDown += DoCustomWPFControlClickEvent;   }     else if (State == State.Terminated)   {     // remember to unsubscribe when finished     if (ChartPanel != null) ChartPanel.MouseDown -= DoUserClickedChartPanelEvent;     if (myWPFControl != null) myWPFControl.MouseDown -= DoCustomWPFControlClickEvent;   } }   private void DoUserClickedChartPanelEvent(object sender, MouseButtonEventArgs e) {     Print("User clicked on the ChartPanel, executing custom mouse down logic..."); }   private void DoCustomWPFControlClickEvent(object sender, MouseButtonEventArgs e) {     Print("User clicked on my button, executing button logic..."); } |     **Delaying logic for a particular time interval**  Do **NOT** call Thread.Sleep() as it will lock the Instrument thread executing your NinjaScript object.     |  | | --- | | **Why:**Market data events exposed to NinjaScript run on the underlying Instrument thread pool shared by all Instruments. Sleeping the underlying thread of your object will cause the entire Instrument thread to sleep, adversely affecting other features using that same Instrument. |      | ns **Practice to avoid** | | --- | | protected override void OnBarUpdate() {   if (IsFirstTickOfBar && State == State.Realtime)   {     Print("Run some logic before:: " + DateTime.Now);     Thread.Sleep(5000); // sleeping the Instrument thread will have adverse effects on elements outside of your script!     Print("Run some logic after: " + DateTime.Now);   } } |     Instead, try using a Timer object if you need to delay logic execution.     | ns **Best practice** | | --- | | protected override void OnBarUpdate() {   if (IsFirstTickOfBar && State == State.Realtime)   {     // Instead of Thread.Sleep for, create a timer that runs at the desired interval     System.Windows.Forms.Timer timer = new System.Windows.Forms.Timer {Interval = 5000};       // queue the "after" logic to run when the timer elapses     timer.Tick += delegate     {         timer.Stop(); // make sure to stop the timer to only fire ones (if desired)         Print("Run some logic after: " + DateTime.Now);         timer.Dispose(); // make sure to dispose of the timer     };       Print("Run some logic before: " + DateTime.Now);       timer.Start(); // start the timer immediately following the "before" logic   } } | |

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

tog_minus        [Miscellaneous practices](javascript:HMToggle('toggle','MiscellaneousPractices','MiscellaneousPractices_ICON'))

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Floating-point comparison**  Be aware of floating-point precision problems. It can sometimes be more reliable to check within a certain degree of tolerance, such as the [TickSize](https://ninjatrader.com/es/support/helpGuides/nt8/ticksize.htm).     |  | | --- | | **Why**:  You can read more about [Floating-Point Arithmetic](http://ninjatrader.com/support/forum/showthread.php?t=3929) as it applies to NinjaTrader on our support forum |      | ns **Practice to avoid** | | --- | | // depending on how Value[0] was calculated, it could be off by a degree of floating points  // where this logic below would never be true  // e.g., 2050.2499999 vs 2050.50 if (Value[0] == Close[0]) {   // do something } |      | ns**Best practice** | | --- | | // you can avoid these precision issues by rewriting the comparison to evaluate within a certain tolerance. if (Math.Abs(Value[0] - Close[0]) < TickSize) {   // do something }  // You will also see NinjaTrader developed objects use a custom Extension Method // double.ApproxCompare() which Returns an int based on a Epsilon value: if (Close[0].ApproxCompare(Value[0]) == 0) {   // do something } |     **Creating user defined parameter types / enums**  When creating enums for your NinjaScript objects, it is strongly suggested to define those outside the class and in a custom namespace. A reference sample providing all details could be [found here](https://ninjatrader.com/es/support/helpGuides/nt8/creating_a_user-defined_parame.htm).    **Efficiently debugging**  Extremely liberal use of Log() and Print() methods can represent a performance hit on your PC as it takes memory and time to process each one of those method calls. When running custom NinjaScript, especially when using Calculate = Calculate.OnEachTick, please be mindful of how often Log() and Print() methods are processed as it can quickly consume PC resources.    •Log() method should not be used except for critical messages as each log entry makes it to the Control Center log which stays active till the end of the day. Excessive logging can result in huge amounts of memory being allocated just to display all the log messages which would mean less memory for NinjaTrader to do other tasks.  •Print() method can be used more liberally than the Log() method, but can still represent a performance hit if used with extremely high frequency. Consider decreasing the printing from your script if you experience slowdowns when running the script.    **Debug Mode**  The debug mode should only be used if you are actively debugging a script and [attached to a debugger](https://ninjatrader.com/es/support/helpGuides/nt8/visual_studio_debugging.htm).     |  | | --- | | **Why**:  Debug Mode will compile all of the files in the custom project as a "Debug" build, which omits certain optimizations which occur in the C# compilation process.  It is more efficient to use your custom objects in the default "Release" build if you are using your scripts during production. |     **To disable Debug Mode:**  •Right mouse click in any NinjaScript Editor  •Ensure the "Debug Mode" menu item is unchecked  •Press F5 to recompile your scripts  •Your scripts will be re-built using "Release" mode    **Known NinjaScript Wrappers limitations**    •The NinjaScript editor detects code changes in external editors, and will compile on code changes, however code will only be automatically generated by the NinjaScript editor if it's edited within the NinjaScript editor itself (or Visual Studio)  •Wrappers cannot be generated automatically for partial and abstract classes  •Code in the Properties region of the NinjaScript object cannot be commented out with the /\* \*/ style commenting, as it will cause issues with the wrapper generation. Code must be commented out with the // style.  •Subclassing would not allow for wrappers to be generated |
| **8.0.0.12 (Beta) Release Notes** | | [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textlayout.htm) |

**Release Date**

July 11, 2016

**Code Breaking Changes**

**Compile Errors**

•The NinjaScript Strategy "**AccountSize**" concept was removed due to limited functionality.  This change would also impact strategies which were coded to use [SetOrderQuantity.ByAccountSize](https://ninjatrader.com/es/support/helpGuides/nt8/setorderquantity.htm).  Please use your own variables to set quantities by account size. **Tip**:  NinjaTrader 8 can read real-world account values for live trading purposes through the Account class, e.g., Account.Get(AccountItem.CashValue, Currency.UsDollar) provides a method for returning a double value representing the current cash value of the account.

**Implementation changes**

•The concept used to force plot series objects through **ForcePlotsMaximumBarsLookBackInfinite** was disabled and tagged as obsolete.  You should set any indicator/strategy plots to [MaximumBarsLookBack.Infinite](https://ninjatrader.com/es/support/helpGuides/nt8/maximumbarslookback.htm) during its construction.

•It is recommended that any strategies generated with the **'Strategy Builder'** tool via Control Center > New > Strategy Builder be removed and recreated after installing B12. If you had invested significant effort in a **Strategy Builder** generated strategy in B11 which you would like to continue to use, please contact us [platformsupport@ninjatrader.com](mailto:platformsupport@ninjatrader.com)

**Notes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Status** | **Issue #** | **Category** | **Comments** |
| Added | 9127 | NinjaScript | Added XML Comments to reflect Supported NinjaScript and NinjaTrader Core Methods (ongoing) |
| Added | 9897 | NinjaScript Editor | Added Multi-Select to the NS Editor References Win File Dialog |
| Added | 9908 | Rithmic | Added additional Rithmic Adapter Account Items |
| Added | 9933 | Window Linking | SFT-162 - Added more link button colors (up to 11) |
| Changed | 10066 | Account Data | Change Position Display option to "Show gross realized PnL when flat." |
| Changed | 10076 | FXCM | Reverted NT8 FXCM Forex volume to NT7 implementation |
| Added | 9839 | Interactive Brokers | Added TWS File picker to allow users to the specific location of TWS/Gateway executable which can vary in some scenarios. [See the NinjaTrader 8 Interactive Brokers connection guide](http://ninjatrader.com/ConnectionGuides/Interactive-Brokers-Connection-Guide). |
| Changed | 9976 | NinjaScript | Disabled the .ForcePlotsMaximumBarsLookBackInfinite concept |
| Fixed | 10022 | NinjaScript Editor | Delimiter matching was hard to see |
| Fixed | 9946 | Orders | Trace Order reported old price after order change |
| Changed | 9875 | Strategy | Removed of "ByAccountSize" Strategy Order Option |
| Fixed | 9887 | Strategy Builder | Strategy Builder additional data window did not show instrument |
| Fixed | 9996 | UI | Removed UI Formatting of Market Data Volume (real-time and historical bar data) |
| Fixed | 9879 | Account Data | Account Data Window default tab order did not match order of Add Tab Menu Sub Items |
| Fixed | 9992 | Account Data | Excess Position Margin Column width was not saved/restored correctly |
| Fixed | 9441 | Adapter | Kinetick adapter was not properly resolving market data settlement time updates |
| Fixed | 9977 | Alerts | Alerts Log Window "priority" was not working as expected with some localization |
| Fixed | 9876 | Attach Order To Indicator, Chart | IsSuspendedWhileInactive did not always re-suspend after manually attached to orders |
| Fixed | 9906 | Bars | Range Bars were built differently using custom data range |
| Fixed | 9889 | Bars | Renko Bars produced undesired ticks bars at EOD |
| Fixed | 9854 | Bars | Break EOD could cause issues with bar pool series when requested after session EOD |
| Fixed | 9803 | Bars, Chart | Bar spacing issues identified when scrolled into future which resulted in the "Automatically choose time-based series for x-axis (non-equidistant bar spacing)" concept to be removed.  We will revisit this feature in the future. |
| Fixed | 9859 | Bars, NinjaScript | BarsSinceNewTradingDay was not operating as expected |
| Fixed | 9985 | Bars, Playback | Exception in multi-series Playback |
| Fixed | 9856 | Chart | Data Box did not update values as you scroll |
| Fixed | 9843 | Chart | Drag and drop series to tab was not creating new series as expected |
| Fixed | 9934 | Chart | Unable to use Instrument Search from new Data Series window |
| Fixed | 9868 | Chart | Fixed Data Box volume formatting issues for FX |
| Fixed | 9858 | Chart | Gap in Chart and Chart Window at 150% DPI |
| Fixed | 10025 | Chart | Multi-Series chart bar width was incorrect |
| Fixed | 10078 | Chart | Crosshair time label rendering incorrect on left side |
| Fixed | 10058 | Chart | Price market incorrect in condition with days to load and session template |
| Fixed | 9988 | Chart Trader | Changing colors of the action, buy, and sell buttons did not save |
| Fixed | 9884 | Chart, Drawing | Draw objects in future would move when chart timeframe changed |
| Fixed | 9943 | Chart, DrawingTool | Mouse cursor was not reverting to normal when canceling the creation of a draw object by right-clicking |
| Fixed | 9964 | Chart, Indicator | Out of range exception was thrown using Horizontal  Line plot on Multi-Series chart |
| Fixed | 9837 | Chart, NinjaScript, Tick Replay | Tick Replay sometimes caused SynchronizationLockException when loading indicator |
| Fixed | 9831 | Chart, Sessions | Rollover date was missing on Default 24/7 with Break at EOD unchecked |
| Fixed | 9931 | Chart, Strategy Analyzer | Strategy Analyzer Data Box opened from trade review chart doesn't show trade execution |
| Fixed | 9929 | Chart, Strategy Analyzer | Aborting in Strategy Analyzer leaves disabled script on chart |
| Fixed | 10048 | Chart, Strategy Analyzer | Unhandled exception when switching to chart in Strategy Analyzer |
| Fixed | 9864 | Chart, Templates | Drawing tool option selection "Stay in Draw Mode" was not included when saving chart template |
| Fixed | 10069 | Code Wizard, Strategy Builder | Additional data option missed week and year type as choice |
| Fixed | 10012 | Commissions | Some commission values displayed with trailing 0's |
| Fixed | 9890 | Control Center | Restoring live executions were sorted reverse |
| Fixed | 9882 | Control Center, Database | Restoring executions were working differently from NT7 |
| Fixed | 9695 | Core | FXCM Adapter reported duplicate trades in a race condition |
| Fixed | 9894 | CQG | Invalid error occurred when trying to connect to CQG after failed CQG logon |
| Fixed | 9910 | Data Grids | Tri-State Sorting was not working in all grids |
| Fixed | 10075 | Data Grids | Account Tab Filter With Positions Was Not Updating as Account position changes |
| Fixed | 9936 | Database | Performance improvements during first ever application startup using default workspaces |
| Fixed | 10042 | Database, Instruments | Removed instrument in instrument manager would sometimes reappear |
| Fixed | 9932 | Database, Trade Performance | TradesPerformance showing incorrect Bars value for some trades |
| Fixed | 9939 | DrawingTool | Ruler was incorrectly rounding pip sizes |
| Fixed | 9945 | DrawingTool | Stay In Draw mode did not apply to Text tool |
| Fixed | 9970 | DrawingTool | Additional settings displayed when selecting drawing tools in the Configured list. |
| Fixed | 9963 | DrawingTool | Changing drawing tool EndTime into future leaves drawing tool drawn as it was |
| Fixed | 9895 | DrawingTool | Drawing tools when switching contract months do not reappear on the chart. |
| Fixed | 9983 | DrawingTool | Performance improvements on default text drawing tool |
| Fixed | 10035 | DrawingTool | Ruler exhibited localization issue |
| Fixed | 9738 | DrawingTool | Setting DrawObject ChartAnchor.BarsAgo did not change rendered object |
| Fixed | 10009 | DrawingTool | Drawing Tools lost auto scale as chart scrolled into future |
| Fixed | 10052 | DrawingTool | TrendChannel Parallel Line Start Anchor did not snap when using bar and price mode |
| Fixed | 9804 | DrawingTool | Trend Lines second line did not respect Snap Mode. |
| Fixed | 9925 | DrawingTool, NinjaScript | DrawObjects collection was not updating with global draw object until NinjaScript reloaded |
| Fixed | 10038 | DrawingTool, Strategy Analyzer | AddChartIndicator indicator was always rendering draw objects on primary panel in Strategy Analyzer |
| Fixed | 10049 | DrawingTool, Templates | PriceLevels reverted after re-compile in Fibonacci Drawing Tools |
| Fixed | 9962 | DrawingTool, Workspaces | Global lines inconsistently saved across workspaces using" workspace save as" |
| Fixed | 10006 | eSignal | Esignal was not processing real-time index data |
| Fixed | 9935 | FX Board | Unable to load ATM Strategy Template Error from FXBoard |
| Fixed | 9950 | FX Board | FXBoard Unrealized PnL field did not update until unrealized PnL changes |
| Fixed | 10067 | FXCM | FXCM order rejection scenario |
| Fixed | 10008 | Hot Key | Removing custom drawing tool code did not remove assigned hot key |
| Fixed | 9912 | Indicator | Bar timer did not function on Playback connection |
| Fixed | 9953 | Indicator | Indicator was plotting on the second bar on chart despite BarsRequiredToPlot = 0 |
| Fixed | 9965 | Indicator | @VolumeProfile.cs results were skewed towards buy volume using Tick Replay |
| Fixed | 10021 | Indicator, SuperDOM | SuperDOM Indicator Data series input changes using multi-time frame indicators |
| Fixed | 9869 | Interactive Brokers | Unable to receive data from the instrument on the SEHK exchange. |
| Fixed | 9961 | Interactive Brokers | Unexpected behavior with chart "Price Based On" was set to Bid/Ask |
| Fixed | 9893 | Interactive Brokers | Was not processing Index data due to no volume reported |
| Fixed | 9995 | Interactive Brokers | Interactive Brokers MarketDataType.DailyHigh/Low was not coming in |
| Fixed | 9994 | Kinetick | Adapter was processing volume of "1" for indexes instead of "0" |
| Fixed | 10034 | Licensing | VendorLicense did not always show all configured free trials |
| Fixed | 9820 | Licensing, NinjaScript | VendorLicense in AddOn prevented OnWindowCreated from working as expected |
| Fixed | 10040 | Market Replay, Time and Sales | Playback connection T&S Window did not scroll precisely to user actions while using Scroll Wheel |
| Fixed | 9898 | NinjaScript | Multi-series indicator was running into deadlocks |
| Fixed | 9857 | NinjaScript | Drawing objects were still visible to NinjaScript after changing instrument on the chart |
| Fixed | 9795 | NinjaScript | Errors on calling 'EventHandlerBarsUpdate' method from Strategy |
| Fixed | 9937 | NinjaScript | Remove NinjaScript assembly window needed vertical scrollbar |
| Fixed | 9955 | NinjaScript | User configured indicator input series changed after exception |
| Fixed | 9978 | NinjaScript | PlotBrushes were not working as expected with AddChartIndicator() |
| Fixed | 9823 | NinjaScript | Indicator plot did not match when using a custom bar type |
| Fixed | 10041 | NinjaScript | Exception calling Draw.Triangle() from AddChartIndicator() |
| Fixed | 10070 | NinjaScript | Blank string used for Name property resulted in crash |
| Fixed | 9928 | NinjaScript Editor | NinjaScript Editor F3 did not work if Find window was closed |
| Fixed | 9971 | NinjaScript Editor | NinjaScript Editor invalid char allowed as name |
| Fixed | 9865 | NinjaScript, Orders | Stop price checks for strategies with OrderFillResolution High had been executed in the wrong bars series |
| Fixed | 9956 | NinjaScript, Strategy | Strategy disabled by SetState was considered as enabled in Strategies tab when attempting to edit |
| Fixed | 10037 | NinjaScript, Strategy Analyzer | Adding indicator to backtest chart with AddChartIndicators caused indicator plots to disappear |
| Fixed | 9874 | NinjaScript, Strategy Builder | Strategy Wizard did not utilize secondary series for input correctly. **Note**:  This fixed resulted in change which preventing "Plot on chart" from working correctly, which is schedule to be fixed in the next beta release (# 10090) |
| Fixed | 9942 | Options | Changing Auto Close Position settings did not create warning if already connected |
| Fixed | 10051 | Orders | Exception on placing order to non-USD sim account created after connecting to provider |
| Fixed | 9915 | Playback | Unhanded exception on Playback disconnect |
| Fixed | 9907 | Playback | Tick Replay on Playback was frozen when using Go To |
| Fixed | 9947 | Playback | Empty warning pop-up when changing Playback start date |
| Fixed | 9944 | Playback | Playback did not "Go To" available date before current start  date. |
| Fixed | 9968 | Playback | Execution markers were not plotting correctly in Playback Connection |
| Fixed | 9975 | Playback | Minor issues with Playback behavior if no data was available |
| Fixed | 9899 | Playback | Playback data auto-replaying on chart when created after Playback connected |
| Fixed | 9989 | Playback | Strategy exception with multiple range series during Playback |
| Fixed | 9990 | Playback | Did not disconnect all the way when disconnected before loaded |
| Fixed | 9999 | Playback | "Error in real time market data handling" when running Playback |
| Fixed | 9986 | Playback | Gaps on dragging slider on Playback data |
| Fixed | 9979 | Strategy | Strategies grid changes to respect region denomination |
| Fixed | 9987 | Strategy | Unexpected strategy accounts issue on Playback data |
| Fixed | 9870 | Strategy Analyzer | Data Box exception using chart display |
| Fixed | 9855 | Strategy Analyzer | Settings display was not updating include commissions param on subsequent backtests |
| Fixed | 9836 | Strategy Analyzer | Parameter tooltip displayed incorrectly |
| Fixed | 9828 | Strategy Analyzer | Control Center logo was sometimes being displayed in chart |
| Fixed | 9980 | Strategy Analyzer | Taskbar previews were not working |
| Fixed | 9922 | Strategy Analyzer | Analysis displayed incorrectly after running separate backtest on optimization result |
| Fixed | 9973 | Strategy Analyzer | Chart Exception: Cannot call DragMove or Activate before a Window is shown |
| Fixed | 9842 | Strategy Analyzer, Templates | Strategy template saved on chart did not restore correctly in Strategy Analyzer |
| Fixed | 9917 | Strategy Builder | Lookback period defaulted to 0 |
| Fixed | 9916 | Strategy Builder | TimeSpan did not use user-applied settings |
| Fixed | 9891 | Strategy Builder | Various Indicator Syntax changes |
| Fixed | 9888 | Strategy Builder | Did not retain added data series in conditions |
| Fixed | 9885 | Strategy Builder | Unlocking code did not open NinjaScript Editor |
| Fixed | 9883 | Strategy Builder | Strategy opened in NinjaScript Editor sometimes produced unhandled exception |
| Fixed | 9867 | Strategy Builder | Did not prevent input of invalid Int value |
| Fixed | 9861 | Strategy Builder | Duplicate input series displayed in selector |
| Fixed | 9938 | Strategy Builder | Conditions window incorrectly scrolled to the top of the list of options when opening configured condition |
| Fixed | 9984 | Strategy Builder | Used incorrect Set methods OnStateChange() |
| Fixed | 9991 | Strategy Builder | Used incorrect value of 0 for selected order quantity |
| Fixed | 9981 | Strategy Builder | Consistency improvements regarding Condition/action copy/save |
| Fixed | 10011 | Strategy Builder | Did not confirm on removing strategy |
| Fixed | 10015 | Strategy Builder | Could not remove Strategy Builder Strategy when opened from NinjaScript Editor |
| Fixed | 10043 | Strategy Builder | Did not automatically compile strategy on clicking Finish |
| Fixed | 10017 | Strategy Builder | Save As' rejected a new name |
| Fixed | 10050 | Strategy Builder | Print() action was not accepting any arguments |
| Fixed | 10055 | Strategy Builder | File names were cut off in the strategy builder load/save strategy popup |
| Fixed | 10039 | Strategy Builder | Did not perform bars check on barsAgo and multi-series scripts |
| Fixed | 10073 | Strategy Builder | Sometimes lost active button highlight |
| Fixed | 10045 | Strategy Builder | String variable did not escape characters |
| Fixed | 9951 | Strategy Builder | Opening screen referred to "wizard" which caused confusion |
| Fixed | 10077 | Strategy Builder | German region decimal format caused compile errors |
| Fixed | 9871 | Strategy, Workspaces | Switching workspaces do not maintain realized PnL and randomly sorts rows in Strategies tab |
| Fixed | 9924 | TD AMERITRADE | Partial Position  was not displaying while connected to TDA |
| Fixed | 10056 | Trade Performance | Changing start date/end date on trade performance prevented generating report for only current day |
| Fixed | 10057 | Trade Performance | Trade Performance column sorting reset when regenerated |
| Fixed | 10027 | Data Grids | Performance Improvement on data grids (Changes require users to restore custom sorting/filtering manually) |
| Changed | 10063 | Indicator | Removed redundant logic of BuySellVolume and BuySellPressure indicators |
| **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

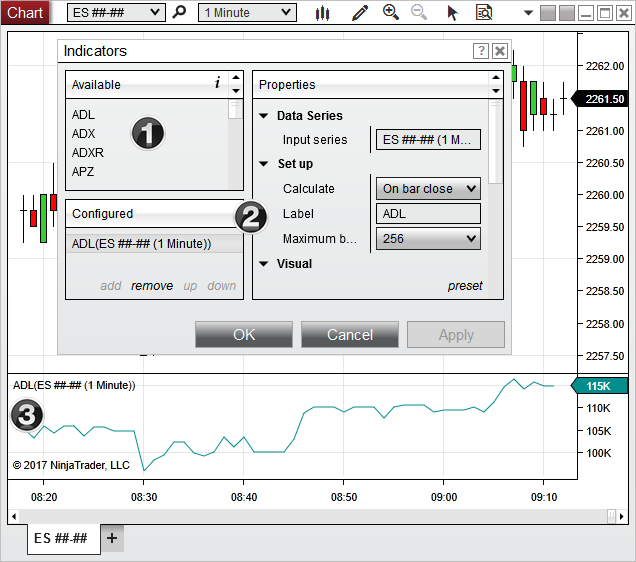
3.User configures desired **Properties** and presses "**Apply**" or "**OK**"

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



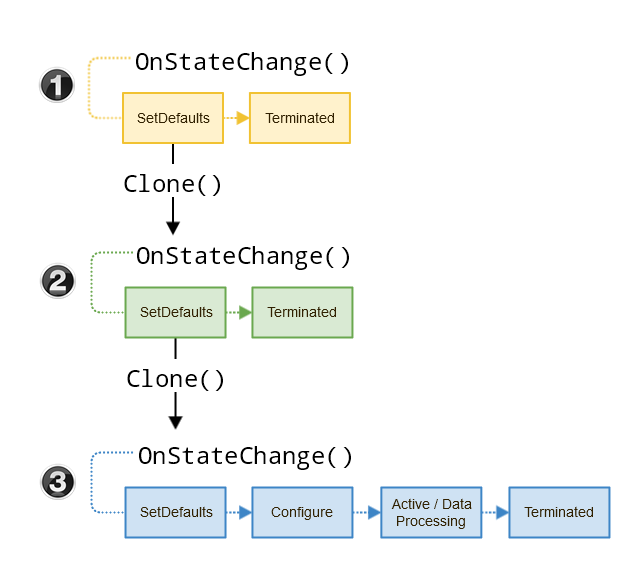
To visualize how each instance goes through its **States**, please consider the logic and flow chart below:

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.

**Definition**

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

**Related Methods and Properties**

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

**Syntax**

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

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

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

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

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

**Parameters**

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

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

**Examples**

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

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