|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) >  **Code Breaking Changes** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript_best_practices.htm) |

The following document is intended as a high level overview of the NinjaScript changes you can expect between NinjaTrader 7 and NinjaTrader 8.  For specific information on a particular method or property, you can refer to the dynamically formatted **Code Breaking table** at the bottom of this page.  We recommend using the **Filter** and **Sorting** features built into the table, as well checking the **Summary** column and expanding the **Details** section of each entry for general information.  Referring to the conveniently linked NinjaTrader 8 and NinjaTrader 7 documentation will provide specific information on syntax, usage, and examples of any new implementation or element names.

|  |
| --- |
| **Note**:  Information on this page focuses on **supported** **(documented)** NinjaTrader methods and properties shared between versions.  NinjaTrader 8 has seen a significant increase in supported NinjaTrader code, however if you were using previously **undocumented** NinjaTrader 7 methods or properties, they will **NOT** be covered in this topic.  You may be able to find more information on previously **undocumented** methods and properties in the NinjaTrader 8 Help Guide, or our support staff will also be happy to personally point you in the right direction. |

|  |
| --- |
| **Critical**:   If your product uses **unsupported (undocumented)** elements we strongly urge you to put your scripts through thorough testing to ensure they still behave as expected.  There is **NO** guarantee that previously **undocumented** method or property behavior has not changed in the new version of NinjaTrader 8. |

For questions or comments, please contact us at platformsupport@ninjatrader.com

tog_minus        [Implementation Changes Overview](javascript:HMToggle('toggle','ImplementationChangesOverview','ImplementationChangesOverview_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialize(), OnStartUp(), OnTermination()**  NinjaTrader 8 has simplified the methods used to set or release various resources during the lifetime of a NinjaTrader object to a single [**OnStateChange()**](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) method. This single method is guaranteed to be called for every change in **State** of the object.  It is from this method you can monitor the progression of the object throughout its lifetime in order to setup various resources, set properties, or take action the moment **State** has changed.  This method also exposes a [**State**](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) variable which can be used in various other methods, such as**OnBarUpdate(),** in order to tell your indicator or strategy to process data depending on the actual **State** of the object.    For example, pushing settings to the UI, or setting initial values for public properties can now be done use **OnStateChange()** when the state has reached**State.SetDefaults**:     | ns | | --- | | protected override void OnStateChange() {   if (State == State.SetDefaults)   {     // set the default properties     Name = "My Indicator";     Fast = 10;     Slow = 25;     IsOverlay = true;     IsAutoScale = true;   } } |       If you have custom resources that need to be setup before the NinjaTrader object is active and processing data, instead of using the**Initialize()** method, you can now set this up once the **OnStateChange()** method has reached **State.Configure** state:     | ns | | --- | | protected override void OnStateChange() {   if (State == State.Configure)   {     // Add a 5 minute Bars object to the strategy     AddDataSeries(Data.BarsPeriodType.Minute, 5);     // setup a custom data series     spread = new Series<double>(this);     // setup a 20-period EMA indicator     ema = EMA(20);     // add indicator to strategy for visual purposes     AddChartIndicator(ema);     } } |       NinjaTrader 7 had no concept to detect when your NinjaTrader object was transitioning from processing Historical data to processing Real-time data.  Now with NinjaTrader 8, the **OnStateChange()** method provides a **State.Transition** state which will notify you when this change is about to occur.  If your NinjaTrader 7 indicators or strategies were using custom methods to try to detect this transition, your custom methods may be refactored under this new state:     | ns | | --- | | protected override void OnStateChange() {   if (State == State.Transition)   {     Print("We're going to real-time data...");     // setup your real-time data resources here   } } |       When your NinjaTrader object is shutting down, and you need clean up any custom device resources, instead of using **OnTermination()**, you should now clean up these resources once the **OnStateChange()** method has reached the **State.Terminated** state:     | ns | | --- | | protected override void OnStateChange() {   if (State == State.Terminated)   {     // release any device resources     if(myTimer != null)         myTimer = null;   } } |     NinjaTrader previously used a **Historical** bool property to notify when an indicator or strategy bar was being processed historically or real-time.  The NinjaTrader 8 **OnStateChange()** approach has now introduced a class level variable **State** where you can check for **State.Historical** or**State.Realtime**in any of the other event methods which will allow you to take action depending on the desired state:     | ns | | --- | | protected override void OnBarUpdate() {   // only process on real-time data   if (State == State.Historical)     return;     else if (State >= State.Realtime)       // rest of logic } |     **Strategies, Orders, and Accounts**  Low level access has been provided to allow more flexibility with the information pertaining to trade data.    •IOrders, IExecution, and IPosition interfaces have all been replaced directly with the corresponding object  •The signatures of the related NinjaScript events have changed to match the NinjaTrader internal Update events  •Methods now return and update with the object instance generated, instead of the previously used interface     |  | | --- | | **Tip**:  Since NinjaTrader 8 now exposes the direct **Order** object, rather than an **IOrder** interface, it is possible to receive **null object reference errors** if you attempt to access an order object before the entry or exit order method has returned.  To prevent these situations, it is recommended to assign your strategies **Order** variables in the **OnOrderUpdate()** method and match them by their **signal name** (order.Name).  Please see the example beginning on line #22 below for demonstration of assigning order objects to private variables. |      | ns | | --- | | Order myOrder = null;   protected override void OnBarUpdate() {           if (Position.MarketPosition == MarketPosition.Flat && myOrder == null)     EnterLongLimit(Low[0], "Entry");     if (myOrder != null)   {     Print(myOrder.OrderState);           if (myOrder.OrderState == OrderState.Cancelled || myOrder.OrderState == OrderState.Filled)         myOrder = null;               } }         protected override void OnOrderUpdate(Cbi.Order order, double limitPrice, double stopPrice,   int quantity, int filled, double averageFillPrice,   Cbi.OrderState orderState, DateTime time, Cbi.ErrorCode error, string comment) {         // compare the order object created via EnterLongLimit by the signal name   if (myOrder == null && order.Name == "Entry")   {     // assign myOrder to matching order update     myOrder = order;           } } |     **Data Series**  Previously there had been type specific Data Series implementations (e.g., IntSeries, TimeSeries, BoolSeries, etc).  Now there just is a template [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) class which could be used generically and even allows support for additional types:     | ns | | --- | | Series<double> mySeries = new Series<double>(this); Series<DateTime> myTimeSeries = new Series<DateTime>(this); |     The **DataSeries.Set()** method used to assign Data Series or Plot values has been removed and values can now be stored using a single assignment operator:     | ns | | --- | | protected override void OnBarUpdate() {   // set public plotting data series close value of current bar   MyPlot[0] = Close[0];   // set custom Series<DateTime> to time value of current bar   myTimeSeries[0] = Time[0];         } |     **Drawing**  The DrawObjects used in NinjaTrader have received a number of changes:    •All DrawObjects have been moved to a separate **NinjaScript.DrawingTools** namespace and are properly known as **DrawingTools**  •Drawing Methods called from indicators or strategies have been moved to a new static partial **Draw** class  •Drawing Methods have all received a signature change which requires you specify the owner (object) which drew the **DrawingTool** object  •Drawing Methods no longer returns an interface but rather an instance of the **DrawingTool** object itself  •Drawing Methods now use the [System.Windows.Media.Brushes](https://msdn.microsoft.com/en-us/library/system.windows.media.brushes%28v=vs.110%29.aspx) class instead of the [System.Drawing.Color](https://msdn.microsoft.com/en-us/library/system.drawing.color(v=vs.110).aspx) structure     |  | | --- | | **Tip**:  DrawingTools are now completely unprotected and you can review their source code from the DrawingTools folder of the NinjaScript Editor's explorer menu |      | ns | | --- | | // example syntax Draw.Line(NinjaScriptBase owner, string tag, int startBarsAgo, double startY, int endBarsAgo, double endY, Brush brush)   // example usage Draw.Line(this, "tag1", true, 10, Low[0], 0, Brushes.Red); |     Casting a member of the **DrawObjects[]**collection must be done safely using the "as" keyword, otherwise you may receive exceptions at run time should another instance of the object (e.g., matching tag) exist from another owner:     | ns | | --- | | NinjaScript.DrawingTools.Line myLine = DrawObjects["tag1"] as DrawingTools.Line; |     **DrawingTools** anchor fields such as "Time" or "Price", etc have been moved to a **ChartAnchor** object owned by the drawing tool, rather than a direct field on the drawing object interface.  Please refer to the NinjaTrader 8 documentation for specific changes for each drawing tool:     | ns | | --- | | double linePrice = myLine.StartAnchor.Price; |     Objects which previously used**System.Drawing.Font** now uses new **NinjaTrader.Gui.Tools.SimpleFont** class:     | ns | | --- | | Gui.Tools.SimpleFont myFont = new Gui.Tools.SimpleFont("Arial", 12); |     Properties and other methods/objects which previously [System.Drawing.Color](https://msdn.microsoft.com/en-us/library/system.drawing.color(v=vs.110).aspx) structure now use the [System.Windows.Media.Brushes](https://msdn.microsoft.com/en-us/library/system.windows.media.brushes%28v=vs.110%29.aspx) class:     | ns | | --- | | BackBrush = Brushes.Blue; |      |  | | --- | | **Note**:  For custom **Brush** objects, it is important to .**Freeze()** the **Brush** due to the multi-threaded architecture of NinjaTrader 8.  Please be sure to review the new information on using [Brushes](https://ninjatrader.com/es/support/helpGuides/nt8/brushes.htm) |     **Namespaces**  The NinjaTrader 7 namespaces**NinjaTrader.Indicator** and **NinjaTrader.Strategy**have been renamed and moved to single **NinjaTrader.NinjaScript** namespace     | ns | | --- | | //This namespace holds indicators in this folder and is required. Do not change it. namespace NinjaTrader.NinjaScript.Indicators {   public class MyCustomIndicator : Indicator   {   } }   //This namespace holds Strategies in this folder and is required. Do not change it. namespace NinjaTrader.NinjaScript.Strategies {   public class MyCustomStrategy : Strategy   {   } } |     **Partial Classes (Porting methods and properties from UserDefinedMethods.cs)**  NinjaTrader 7 used a "UserDefinedMethods" class to define methods to be used across multiple NinjaScript indicators or strategies. In NinjaTrader 8, these pre-built partial classes have been removed to reduce a number of issues which could result from users sharing their UserDefinedMethods.cs files, or overwriting their existing files with copies from a new vendor. Partial classes are now best built manually and saved in the C:\Users\<user>\Documents\NinjaTrader 8\bin\Custom\AddOns folder.     |  | | --- | | **Warning**: If a partial class is saved in one of the folders used for specific NinjaScript objects other than AddOns (e.g., Indicators folder), auto-generated NinjaScript code may be appended to the end of the class by the NinjaScript Editor when compiled, which will cause a compilation error.  Saving these files in the AddOns folder will ensure they are still accessible and will not generate code which may be cause conflicts. |     You can use the template below as a starting point to create your partial class. If your partial class needs to inherit from a parent class, you can append the name of your desired parent class after the " : " to change the inheritance.     |  | | --- | | **Note**: Methods within your partial classes should be using the "public" modifier. |      | ns**Partial Class Example Template** | | --- | | namespace NinjaTrader.NinjaScript.Indicators {   public partial class MyMethods *// : parent class to inherit from*   {       //Sample method which calculates the delta of two prices       public double calculateDelta(double firstPrice, double secondPrice)       {           return Math.Abs(firstPrice - secondPrice);       }         //Sample method which prints Position information       public void printPositionInfo(Position position)       {           Print(String.Format("{0}: {1} {2} at {3}", position.Instrument, position.Quantity, position.MarketPosition, position.AveragePrice));       }           } } |     Below is an example of using one of the methods in this partial class from within an Indicator:     | ns**Partial Class Usage** | | --- | | protected override void OnBarUpdate() {   if (CurrentBar < 1) return;     // Use the static calculateDelta method to calculate the difference between the close of each bar   double delta = MyMethods.calculateDelta(Close[0], Close[1]);     Print(delta); } |      |  | | --- | | **Tip**:  At the time of the Beta implementation, the NinjaScript Editor does **NOT** include a partial class generator wizard, as it does for core NinjaScript Types such as Drawing Tools, Market Analyzer Columns, or Strategies. However, we are currently tracking a suggestion to implement a wizard for partial classes, under ID # **SFT-341**.   Please feel free to contact platformsupport@ninjatrader.com if you would like to add your vote for this enhancement. |     **Prevention of Redundant Data Loading**  In NinjaTrader 7, multiple Data Series could be added within a script, such as an indicator, and that script could then be hosted by another script, such as a strategy. While this is still possible in NinjaTrader 8, there is a new safeguard in place to prevent redundant data loading in both the hosting script and the hosted indicator.    When hosting an indicator which adds Data Series programmatically, the hosting script must include the same calls to the AddDataSeries() method as the hosted script. Without this, an error will result, which reads *"A hosted indicator tried to load additional data. All data must first be loaded by the hosting NinjaScript in its Configure state."* Without this safegaurd in place, it would be possible for unnecessarily large amounts of data to be loaded concurrently, as would be the case in a direct call to an indicator method on each OnBarUpdate(). By adding the calls to AddDataSeries() to the hosting script, you can ensure that the data is loaded when needed. Also, when this is done in the hosting script, all identical calls to AddDataSeries() in the hosted script will be ignored, as the data is already available.    The examples below show this in action:     | ns**Hosted Indicator Loads Additional Data** | | --- | | public class MyCustomIndicator : Indicator {   protected override void OnStateChange()   {     if (State == State.Configure)     {           AddDataSeries("AAPL", BarsPeriodType.Day, 1);           AddDataSeries("EURUSD", BarsPeriodType.Minute, 15);       }   } } |      | ns**Hosting Strategy Mirrors AddDataSeries() calls** | | --- | | public class MyCustomStrategy : Strategy {   // Define a MyCustomIndicator   MyCustomIndicator myIndicator;     protected override void OnStateChange()   {     if (State == State.Configure)     {         // Instantiate the MyCustomIndicator and add it to the chart         myIndicator = MyCustomIndicator();         AddChartIndicator(myIndicator);           // These calls to AddDataSeries() mirror the calls in the hosted indicator         AddDataSeries("AAPL", BarsPeriodType.Day, 1);         AddDataSeries("EURUSD", BarsPeriodType.Minute, 15);     }   } } |     **Bars with 0 Volume**  In previous versions, the NinjaTrader core was designed to replace a tick with a volume of 0 with a volume of 1.  This resulted in all ticks having a volume value of at least 1.  NinjaTrader 8 has removed that design policy and will now allow ticks with a volume of 0 to be processed.  This policy change may require logic changes to any custom bar types, indicators, or strategies which may have previously assumed volume would always be greater than 0.    **Multi-Series default "Trading Hours" templates**  The default behavior in NinjaTrader 8 will ensure that a bars series added to a script using [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm) will use the same "[TradingHours](https://ninjatrader.com/es/support/helpGuides/nt8/tradinghours.htm)" template as the primary series configured by the user. In contrast, the NinjaTrader 7 behavior was highly dependent on a number of variables.  We have updated this behavior to help with consistences and synchronization issues between multiple series; however if you your script relies on two times frames using different trading hours templates, you may consider using one of the new **tradingHours**string overloaded used in [AddDataSeries()](https://ninjatrader.com/es/support/helpGuides/nt8/adddataseries.htm):     | ns | | --- | | protected override void OnStateChange() {   if (State == State.Configure)   {     // adds a 1 minute AAPL bars with a default 24/7 session tempalte.     AddDataSeries("AAPL", new BarsPeriod { BarsPeriodType = BarsPeriodType.Minute, Value = 1 }, "Default 24 x 7");   } } |     **Miscellaneous**  All of the NinjaTrader 7 reference samples posted in our support forum have been updated to demonstrate NinjaTrader 8 functionality.  Please be sure to check the reference sample section to see other undocumented features and concepts which may not have been covered in the help guide:    [Official NinjaScript reference code samples](http://www.ninjatrader.com/support/forum/forumdisplay.php?f=30)    There are several other changes to implementation which are not covered in detail on this overview, please see the code breaking changes table at the bottom of this page which will compare the implementation changes between both versions. |

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

tog_minus        [Signature Changes Overview](javascript:HMToggle('toggle','SignatureChangesOverview','SignatureChangesOverview_ICON'))

|  |  |
| --- | --- |
| **Signature**  A large number of the NinjaTrader methods which were available in NinjaTrader 7 have remained largely the same and should not generate any errors on compilation.  However there are a handful of existing methods signatures which have been updated in NinjaTrader 8 in order to fit within new framework which you would need to be aware of in order to transfer these functions from NinjaTrader 7 to NinjaTrader 8.  In most cases, the fundamental argument type has been restructured, which may result in compile errors depending on the type of object that is being used within the methods signature.     |  | | --- | | **Tip**:  Methods may now have additional signatures which add functionality which was not previously available.  Be sure to check the NinjaTrader 8 documentation which will cover all the available signatures available. | |

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

tog_minus        [Name Changes Overview](javascript:HMToggle('toggle','NameChangesOverview','NameChangesOverview_ICON'))

|  |
| --- |
| **Renamed**  During the NinjaTrader 8 development process, one of our goals to make sure that our core framework matched various coding standards which have been set out in the industry.  As a result of meeting these coding standards, many NinjaTrader methods and properties needed to been renamed.    While the functionality of these methods and properties remains the same, we chose to rename these variables to follow a semantically context specific naming convention which is generally agreed upon to favor readability.  We feel that the renaming of these properties and methods more explicitly describes the intended function to the developer who may be reviewing code.  The largest number of changes is in response to the name convention of bools, where they now follow a more strict verb-adjective or verb-noun structure.    For an example:    •The property **FirstTickOfBar** may have been hard to distinguish precisely what it represented without having to look up documentation.  In NinjaTrader 8, this property has been renamed to **IsFirstTickOfBar**, which now gives this property a more readable identifier name when you read this line of code as "*is the first tick of bar true?*"  •Another example is the case of **BarsSinceEntry()** which was renamed to**BarsSinceEntryExecution()**, which now specifies that this method is looking for an entry *execution*.  •NinjaTrader 7 sometimes had methods or properties which shared names, but references different data or actions.  For example **Add()** could have been used in reference to adding **DataSeries** to a script, adding a **Plot**, or adding a **Line**.  To be more specific, NinjaTrader 8 has renamed these to **AddDataSeries()**,**AddPlot()**, and **AddLine()** respectively.  •There may be cases where the property or method name has changed simply because the type of data it interacted with has changed.  (e.g., **BarColor** vs. **BarBrush**)  •There are other cases where properties may have used unnecessary brevity and was renamed to favor readability (e.g., **AvgPrice** vs **AveragePrice**)    These are just a few examples of the many name changes found in NinjaTrader 8 and some of the rational behind the number of these changes.  For simplicity, you will find a list of all the renamed properties in the table at the bottom of this document by filtering by the "Renamed" keyword. |

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

**Code Breaking Table**

Below you will find a reference table which lists all of the supported NinjaScript changes between NinjaTrader 7 and NinjaTrader 8.

Show 255075100All entries

Filter results:

| **Category** | **Base** | **NT7 Method/Property** | **NT8 Method/Property** | **Summary** |  |
| --- | --- | --- | --- | --- | --- |
| Implementation | Strategy | [GetAccountValue()](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?getaccountvalue.htm) | [Account.Get()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?get.htm) | Access to Account values have been directly exposed | Details |
| Renamed | Strategy | [Add() - Strategy](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?add2.htm) | [AddChartIndicator()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?addchartindicator.htm) | Method renamed to be more specific |  |
| Implementation | General | [Add() - Data](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?add3.htm) | [AddDataSeries()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?adddataseries.htm) | Method was renamed to be more specific, received a number of enhancements. | Details |
| Signature | General | [AddKagi()](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?addkagi.htm) | [AddKagi()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?addkagi.htm) | Received a number of signature changes | Details |
| Renamed | Indicator | [Add() - Line](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?add.htm) | [AddLine()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?addline.htm) | Method renamed to be more specific |  |
| Signature | General | [AddLineBreak()](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?addlinebreak.htm) | [AddLineBreak()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?addlinebreak.htm) | Received a number of signature changes | Details |
| Renamed | Indicator | [Add() - Plot](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?add.htm) | [AddPlot()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?addplot.htm) | Method renamed to be more specific |  |
| Signature | General | [AddPointAndFigure()](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?addpointandfigure.htm) | [AddPointAndFigure()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?addpointandfigure.htm) | Received a number of signature changes | Details |
| Signature | General | [AddRenko()](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?addrenko.htm) | [AddRenko()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?addrenko.htm) | Received a number of signature changes | Details |
| Signature | General | [Alert()](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?alert.htm) | [Alert()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?alert.htm) | Color no longer used, use Brushes instead; soundLocation now requires absolute file path | Details |
| Implementation | Drawing | [IAndrewsPitchfork](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?iandrewspitchfork.htm) | [AndrewsPitchfork](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?andrewspitchfork.htm) | IDrawingObjects have been replaced | Details |
| Implementation | Drawing | [IArc](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?iarc.htm) | [Arc](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?arc.htm) | IDrawingObjects have been replaced | Details |
| Renamed | Indicator | [LinesConfigurable](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?linesconfigurable.htm) | [AreLinesConfigurable](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?arelinesconfigurable.htm) | Property renamed to meet naming conventions |  |
| Renamed | Indicator | [PlotsConfigurable](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?plotsconfigurable.htm) | [ArePlotsConfigurable](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?areplotsconfigurable.htm) | Property renamed to meet naming conventions |  |
| Implementation | Drawing | [IArrowDown](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?iarrowdown.htm) | [ArrowDown](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?arrowdown.htm) | IDrawingObjects have been replaced | Details |
| Implementation | Drawing | [IArrowLine](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?iarrowline.htm) | [ArrowLine](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?arrowline.htm) | IDrawingObjects have been replaced | Details |
| Implementation | Drawing | [IArrowUp](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?iarrowup.htm) | [ArrowUp](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?arrowup.htm) | IDrawingObjects have been replaced | Details |
| Implementation | General | [DataSeries.Set()](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?dataseries_class.htm) | [Assignment Operator (=)](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?seriest.htm) | The .Set() method has been replaced | Details |
| Implementation | Strategy | [AtmStrategyCreate()](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?atmstrategycreate.htm) | [AtmStrategyCreate()](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?atmstrategycreate.htm) | Added a callback signature parameter | Details |
| Renamed | Strategy | [AvgBarsInTrade](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?avgbarsintrade.htm) | [AverageBarsInTrade](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?averagebarsintrade.htm) | Property renamed to favor readability |  |
| Renamed | Strategy | [AvgEtd](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?avgetd.htm) | [AverageEtd](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?averageetd.htm) | Property renamed to favor readability |  |
| Renamed | Strategy | [AvgMae](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?avgmae.htm) | [AverageMae](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?averagemae.htm) | Property renamed to favor readability |  |
| Renamed | Strategy | [AvgMfe](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?avgmfe.htm) | [AverageMfe](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?averagemfe.htm) | Property renamed to favor readability |  |
| Renamed | Strategy | [AvgPrice](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?avgprice.htm) | [AveragePrice](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?position_averageprice.htm) | Property renamed to favor readability |  |
| Renamed | Strategy | [AvgProfit](http://www.ninjatrader.com/support/helpGuides/nt7/index.html?avgprofit.htm) | [AverageProfit](http://www.ninjatrader.com/support/helpGuides/nt8/en-us/index.html?averageprofit.htm) | Property renamed to favor readability |  |

Showing 1 to 25 of 197 entries

Previous12345…8Next

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

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Using SharpDX for Custom Chart Rendering** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_images_and_geometry_with_custom_icons.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_brushes.htm) |

**Understanding the SharpDX .NET Library**

NinjaTrader Chart objects (such as Indicators, Strategies, DrawingTools, ChartStyles) implement an [OnRender()](https://ninjatrader.com/es/support/helpGuides/nt8/onrender.htm) method aimed to render custom lines, shapes, and text to the chart.  To achieve the level of performance required to keep up with market data events, NinjaTrader uses a 3rd-party open-source .NET library named [SharpDX](http://sharpdx.org/).  This 3rd party library provides a C# wrapper for the powerful [Microsoft DirectX API](https://msdn.microsoft.com/en-us/library/windows/desktop/ee663274(v=vs.85).aspx) used for graphics processing and known for its hardware-accelerated performance, including 2D vector and text layout graphics used for **NinjaTrader Chart Rendering**.  The SharpDX/DirectX library is extensive, although NinjaTrader only uses a handful of namespaces and classes, which are documented as a guide in this reference.  In addition to this educational resource, we have also compiled a more focused collection of [SharpDX SDK Reference](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_sdk_reference.htm) resources to help you learn the **SharpDX** concepts used in **NinjaTrader Chart Rendering**.

|  |
| --- |
| **Tips**:  1.There are several pre-installed examples of **OnRender()** and **SharpDX** objects used in the **NinjaTrader.Custom** project.  For starters, please look at the **SampleCustomRender** indicator file  2.Although not entirely identical, the **SharpDX** wrapper is designed to resemble **System.Drawing** namespace; experienced GDI developers will be familiar with concepts discussed in this section.  3.Microsoft provides various [DirectX Programming Guides](https://msdn.microsoft.com/en-us/library/dd372337(v=vs.85).aspx) aimed to educate users with the underlying**C++ DirectX API**.  While **SharpDX (C#)** syntax is different, you may find these guides helpful for understanding **SharpDX** concepts not offered by this guide. |

There are three main **SharpDX** namespaces you need to be familiar with:

|  |  |
| --- | --- |
| [SharpDX](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx.htm) | Contains basic objects used by SharpDX. |
| [SharpDX.Direct2D1](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1.htm) | Contains objects used for rendering for 2D geometry, bitmaps, and text. |
| [SharpDX.DirectWrite](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite.htm) | Contains objects used for text rendering |

The rest of this page will help you navigate the fundamental concepts needed to achieve custom rendering to your charts.

tog_minus        [SharpDX Vectors and Charting Coordinates](javascript:HMToggle('toggle','SharpDXVectorsAndChartingCoordinates','SharpDXVectorsAndChartingCoordinates_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Understanding the SharpDX.Vector2**  SharpDX Draw methods use a [SharpDX.Vector2](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_vector2.htm) object which describes where to render a command relative to the chart panel.  These **Vector2** objects can be thought as a two-dimensional point in the chart panels X and Y axis. Since the chart canvas used to draw on consists of the full panel of the chart, a vector using a value of 0 for both the X and Y coordinates would  be located in the top left corner of the chart:     | ns | | --- | | // creates a vector located at the top left corner of the chart float x = 0; float y = 0; SharpDX.Vector2 myVector2 = new Vector2(x, y); |      |  | | --- | | **Tip**:   You can learn about [Understanding Chart Canvas Coordinates](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_chart_object_coordinates.htm) on another topic |     **Vector2** objects contain **X**and**Y** properties helpful to recalculate new properties based on the initial vector:     | ns | | --- | | float width = endPoint.X - startPoint.X;  float height = endPoint.Y - startPoint.Y; |     Additionally, you can recalculate a new vector from existing vector objects:     | ns | | --- | | SharpDX.Vector2 center = (startPoint + endPoint) / 2; |     It is also helpful to know that **Vector2** objects are similar to the [Windows Point](https://msdn.microsoft.com/en-us/library/system.windows.point(v=vs.110).aspx) structure and these two types can be used interchangeably.  Depending on the mechanism used to obtain user input or other application values, you may receive the coordinates in a **Point**.  For convenience, NinjaTrader provides a [DXExtension.ToVector2()](https://ninjatrader.com/es/support/helpGuides/nt8/dxextensions_tovector2.htm) method used for converting between these two objects if needed:     | ns | | --- | | SharpDX.Vector2 dxVector2 = wpfPoint.ToVector2(); |     **Calculating Chart Coordinates**  If you simply used a vector with static values, your **Vector2** objects would never change, and your drawing would remain fixed on a particular area of the chart (which may be desired).  However, since NinjaTrader charts are dynamic and responded to various market data updates, scroll, resize, and scale operations - you also need a way to recalculate **vectors** to display information dynamically. To assist in this process, NinjaTrader provides some GUI related utilities to help navigate the chart and calculate values for your custom rendering.     | ns | | --- | | // creates a vector located at the top left corner of the chart panel  startPoint = new SharpDX.Vector2(ChartPanel.X, ChartPanel.Y);    // creates a vector located at the bottom right corner of the chart panel  endPoint = new SharpDX.Vector2(ChartPanel.X + ChartPanel.W, ChartPanel.Y + ChartPanel.H); |     Common utilities fall under 4 key components, and you can learn more about their specific functions from the help guide topics linked in the table below:     |  |  | | --- | --- | | [ChartControl](https://ninjatrader.com/es/support/helpGuides/nt8/chartcontrol.htm) | The entire hosting grid of the Chart | | [ChartBars](https://ninjatrader.com/es/support/helpGuides/nt8/chartbars.htm) | The primary bars series configured on the Chart | | [ChartPanel](https://ninjatrader.com/es/support/helpGuides/nt8/chartpanel.htm) | The panel on which the calling script resides | | [ChartScale](https://ninjatrader.com/es/support/helpGuides/nt8/chartscale.htm) | The Y-Axis values of the configured ChartPanel |      |  | | --- | | **Note**:   For full absolute device coordinates always use **ChartPanel** X, Y, W, H values. **ChartScale** and **ChartControl** properties return WPF units, so they can be drastically different depending on DPI of the user's display.  You can learn about [Working with Pixel Coordinates](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_pixel_coordinates.htm) on another topic. | |

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

tog_minus        [SharpDX Brush Resources](javascript:HMToggle('toggle','SharpDXBrushResources','SharpDXBrushResources_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Understanding SharpDX Brush Resources**  To color or "paint" an area of the chart, you must define custom resources which describe how you wish the custom render to appear. **SharpDX** contains special resources modeled after the familiar [WPF Brushes](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_brushes.htm). However, the two objects are different in the way they are constructed and also in how they are managed after they are used.    There are many types of **SharpDX Brush Resources** which all derive from the same base [Direct2D1.Brush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_brush.htm) class.  This base object is not enough to describe how your object should be presented, so in order to use a brush for rendering purposes, you will need to determine exactly what type of brush you wish to use:     |  |  | | --- | --- | | [Direct2D1.SolidColorBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_solidcolorbrush.htm) | Paints an area with a solid color. | | [Direct2D1.RadialGradientBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_radialgradientbrush.htm) | Paints an area with a radial gradient. | | [Direct2D1.LinearGradientBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_lineargradientbrush.htm) | Paints an area with a linear gradient. |     **Describing SolidColorBrush Colors**  The most common and simple brush to use is a [Direct2D1.SolidColorBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_solidcolorbrush.htm)which allows you to paint using a solid color (or with transparency). In the most basic form, **SolidColorBrush** can be constructed using a predefined [SharpDX.Color](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_color.htm)     | ns | | --- | | SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue); |     You can also use a [SharpDX.Color3](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_color3.htm) or [SharpDX.Color4](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_color4.htm) structure as a way to get more customizable colors in your rendering:     | ns | | --- | | // create a 3 component color using rgb values in float notation SharpDX.Color3 dxColor3 = new SharpDX.Color3(1.0f, 0.0f, 0.0f);   // create a 4 component color using rgb + alpha (transparency) in float notation SharpDX.Color4 dxColor4 = new SharpDX.Color4(dxColor3, 0.5f);   // solid color brush uses a Color4 during construction SharpDX.Direct2D1.SolidColorBrush argbColorBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, dxColor4); |     Alternatively, you can set the "transparency" of an existing brush by accessing its [Opacity](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_brush_opacity.htm) property:     | ns | | --- | | customDXBrush.Opacity = .25f; |      |  | | --- | | **Note**:  Unlike their [WPF counterparts](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_brushes.htm), **SharpDX** brushes are thread-safe and do **NOT** need to be frozen. |     **Converting SharpDX Brushes**  **SharpDX Brushes** are **device-dependent resources**, which means they can only be used with the device (i.e., [RenderTarget](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget.htm)) which created them.  In practice, this mean you should **ONLY** create your **SharpDX** brushes during the chart object's [OnRender()](https://ninjatrader.com/es/support/helpGuides/nt8/onrender.htm) or [OnRenderTargetChanged()](https://ninjatrader.com/es/support/helpGuides/nt8/onrendertargetchanged.htm) methods.     |  | | --- | | **Warning**:  Failure to create device-dependent resources during the **OnRender()** or **OnRenderTargetChanged()** can lead to a host of issues including memory and application corruption which can negatively impact the stability of NinjaTrader.  Please be careful your **SharpDX** device-dependent resources are only created and updated during either of these two run-time methods.  Please see the [Best Practices for SharpDX Resources](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm#bestpracticesforsharpdxresources) section on this page for more information. |     Because of this detail, a common problem you may run into is the requirement to share a **SharpDX** device brush resource with a **WPF** application brush.  For example, you may have **WPF** brushes defined in the UI during [OnStateChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onstatechange.htm) or recalculated conditionally during [OnBarUpdate()](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm), but ultimately wish to use also in custom rendering routines.  For convenience, NinjaTrader provide a [DXExtension.ToDxBrush()](https://ninjatrader.com/es/support/helpGuides/nt8/dxextensions_todxbrush.htm) method used for converting these objects if necessary:     | ns | | --- | | areaBrushDx = areaBrush.ToDxBrush(RenderTarget);  smallAreaBrushDx = smallAreaBrush.ToDxBrush(RenderTarget);  textBrushDx = textBrush.ToDxBrush(RenderTarget); |      |  | | --- | | **Note**: If you are using a large number of brushes, and are not tied to WPF resources, you should favor creating the **SharpDX Brush** directly since the ToDxBrush() method can lead to performance issues if called too frequently during a single render pass.  Please see the [Best Practices for SharpDX Resources](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm#bestpracticesforsharpdxresources) section on this page for more information. | |

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

tog_minus        [SharpDX RenderTarget](javascript:HMToggle('toggle','SharpDXRenderTarget','SharpDXRenderTarget_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Understanding the RenderTarget**  A [SharpDX Render Target](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget.htm) is a general purpose object resource used for receiving and executing drawing commands.  When using a NinjaTrader chart object, a pre-constructed Chart [RenderTarget](https://ninjatrader.com/es/support/helpGuides/nt8/rendertarget.htm) object is available for you to use and ready to receive commands.  You can think of the **RenderTarget** as the device context you are using to render to (i.e. the Chart Panel).  While there is nothing special you need to do to setup this resource, it is important to understand some details regarding the **RenderTarget** to learn how it can be used.    The **RenderTarget** is primarily used for executing commands such as drawing shapes or text:     | ns | | --- | | **RenderTarget**.DrawLine(startPoint, endPoint, areaBrushDx) |     It is commonly used for creating various resources such as **Brushes** and other **SharpDX** objects:     | ns | | --- | | SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(**RenderTarget**, SharpDX.Color.DodgerBlue); |     It can also be used to set various properties to describe how the **RenderTarget** should render:     | ns | | --- | | RenderTarget.AntialiasMode   = SharpDX.Direct2D1.AntialiasMode.PerPrimitive; |     **Sequencing RenderTarget commands**  If the sequence in which objects render is essential to your custom rendering, you will need to be mindful of the order in which you call various **RenderTarget** members. For example, we can draw a second line which uses a different [AntialiasMode](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_antialiasmode.htm) and the renders each line in the order the render target received its commands:     | ns | | --- | | RenderTarget.AntialiasMode = SharpDX.Direct2D1.AntialiasMode.Aliased; RenderTarget.DrawLine(startPoint, endPoint, areaBrushDx, 8);  RenderTarget.AntialiasMode = SharpDX.Direct2D1.AntialiasMode.PerPrimitive; RenderTarget.DrawLine(startPoint, endPoint, customDXBrush, 2); |     In the above example, this order of operations would result in the second [RenderTarget.DrawLine()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawline.htm) to be rendered "on top" of the first **RenderTarget.DrawLine().** If you instead called these two methods in reverse order, you would not see the thinner line since it would be covered up by the thicker line.     |  | | --- | | **Note**:  It is important to realize that **RenderTarget** **sequencing** and the [Chart Object ZOrder](https://ninjatrader.com/es/support/helpGuides/nt8/chart_zorder.htm) are two different concepts. The **ZOrder** property controls the overall layer your entire chart object appears relative to other chart objects existing on the same chart. **RenderTarget sequencing** only affects the order objects are rendered relative itself.  Therefore, it is not possible to sequence your chart object's **RenderTarget** to draw on two different **ZOrders** (e.g., one line above chart bars and another line below). |     **Using the RenderTarget with Device Resources**  Throughout the lifetime of a chart, the render target is created and destroyed several times to satisfy various user commands. As a result, any resources that are created need to be recreated and destroyed as that render target is updated.   The NinjaTrader [OnRenderTargetChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onrendertargetchanged.htm) method was designed to help with this process and will be called anytime the **RenderTarget** has changed.  You should use this method if you have objects which are passed around from various other resources.     |  | | --- | | **Warning**:  Failure to create device-dependent resources during the **OnRender()** or **OnRenderTargetChanged()** can lead to a host of issues including memory and application corruption which can negatively impact the stability of NinjaTrader.  Please be careful your **SharpDX** device-dependent resources are only created and updated during either of these two run-time methods.  Please see the [Best Practices for SharpDX Resources](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm#bestpracticesforsharpdxresources) section on this page for more information. | |

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

tog_minus        [SharpDX Lines and Shapes](javascript:HMToggle('toggle','SharpDXLinesAndShapes','SharpDXLinesAndShapes_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RenderTarget Draw Methods**  All drawings consistent of a few basic shapes which can be called through a handful of **RenderTarget** commands.  "Draw..." methods create just the outline of the shape, and "Fill..." will paint the interior of the shape.     |  |  | | --- | --- | | [RenderTarget.DrawEllipse()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawellipse.htm) | Draws the outline of the specified ellipse using the specified stroke style. | | [RenderTarget.DrawGeometry()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawgeometry.htm) | Draws the outline of the specified geometry using the specified stroke style. | | [RenderTarget.DrawLine()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawline.htm) | Draws a line between the specified points. | | [RenderTarget.DrawRectangle()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawrectangle.htm) | Draws the outline of a rectangle that has the specified dimensions and stroke style. | | [RenderTarget.FillEllipse()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_fillellipse.htm) | Paints the interior of the specified ellipse. | | [RenderTarget.FillGeometry()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_fillgeometry.htm) | Paints the interior of the specified geometry. | | [RenderTarget.FillRectangle()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_fillrectangle.htm) | Paints the interior of the specified rectangle. |      |  | | --- | | **Note**: [AntialiasMode.PerPrimitive](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_antialiasmode.htm) allows for graphics to render more sharply, but comes at a performance cost.  It is recommended to set the [RenderTarget.AntialiasMode](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_antialiasmode.htm) back to the default **AntialiasMode.Aliased** after you finish your **RenderTarget**Draw command.   Please see the [Best Practices for SharpDX Resources](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm#bestpracticesforsharpdxresources) section on this page for more information. |     **Line**  The simplest shape is a Line, executed by the [RenderTarget.DrawLine()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawline.htm) command which just takes two [Vector2](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_vector2.htm) objects which describe where to draw the line, and (optionally) the width of the line to draw:     | ns | | --- | | // create two vectors for the line to draw  SharpDX.Vector2 startPoint = new SharpDX.Vector2(ChartPanel.X, ChartPanel.Y); SharpDX.Vector2 endPoint = new SharpDX.Vector2(ChartPanel.X + ChartPanel.W, ChartPanel.Y + ChartPanel.H);  // define the brush used in the line SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue);  // execute the render target draw line with desired values RenderTarget.DrawLine(startPoint, endPoint, customDXBrush, 2);    // always dispose of a brush when finished  customDXBrush.Dispose(); | | render_target_drawline |     **Rectangle**  Using either the [RenderTarget.FillRectangle()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_fillrectangle.htm) or [RenderTarget.DrawRectangle()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawrectangle.htm) requires a [SharpDX.RectangleF](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_rectanglef.htm) structure, constructed using four values to represent the location (x, y) and size (width, height) of the rectangle to draw.     | ns | | --- | | // create two vectors to position the rectangle  SharpDX.Vector2 startPoint = new SharpDX.Vector2(ChartPanel.X, ChartPanel.Y); SharpDX.Vector2 endPoint = new SharpDX.Vector2(ChartPanel.X + ChartPanel.W, ChartPanel.Y + ChartPanel.H);  // calculate the desired width and heigh of the rectangle float width = endPoint.X - startPoint.X; float height = endPoint.Y - startPoint.Y;    // define the brush used in the rectangle  SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue);    // construct the rectangleF struct to describe the with position and size the drawing SharpDX.RectangleF rect = new SharpDX.RectangleF(startPoint.X, startPoint.Y, width, height);  // execute the render target fill rectangle with desired values RenderTarget.FillRectangle(rect, customDXBrush);    // always dispose of a brush when finished  customDXBrush.Dispose(); | | render_target_drawrectangle |     **Ellipse**  Similar to the **Rectangle**, you can draw an **Ellipse** (or circle) using either the [RenderTarget.FillEllipse()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_fillellipse.htm) or [RenderTarget.DrawEllipse()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawellipse.htm) methods using a [SharpDX Direct2D1 Ellipse](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_ellipse.htm) struct.  For this structure, you will need to use a [Vector2](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_vector2.htm) object to determine the **Center** position of the ellipse, a **RadiusX,** and a **RadiusY** which determines the size of the ellipse:     | ns | | --- | | // create two vectors to position the ellipse  SharpDX.Vector2 startPoint = new SharpDX.Vector2(ChartPanel.X, ChartPanel.Y); SharpDX.Vector2 endPoint = new SharpDX.Vector2(ChartPanel.X + ChartPanel.W, ChartPanel.Y + ChartPanel.H);  // calculate the center point of the ellipse from start/end points  SharpDX.Vector2 centerPoint = (startPoint + endPoint) / 2;    // set the radius of the ellipse float radiusX = 50; float radiusY = 50;    // construct the rectangleF struct to describe the position and size the drawing  SharpDX.Direct2D1.Ellipse ellipse = new SharpDX.Direct2D1.Ellipse(centerPoint, radiusX, radiusY);    // define the brush used in the rectangle  SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue);  // execute the render target fill ellipse with desired values RenderTarget.FillEllipse(ellipse, customDXBrush);    // always dispose of a brush when finished  customDXBrush.Dispose(); | | render_target_drawellipse |     **Geometry**  For more complicated shapes, you can use the [RenderTarget.FillGeometry()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_fillgeometry.htm) or [RenderTarget.DrawGeometry()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawgeometry.htm) methods using a [Direct2D1.PathGeometry](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_pathgeometry.htm) object, which is ultimately defined by a [Direct2D1.GeometrySink](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_geometrysink.htm) interface.     |  | | --- | | **Warning**:  Any **SharpDX PathGeometry** object used in your development must be disposed of after they have been used. NinjaTrader is **NOT** guaranteed to dispose of these resources for you!   Please see the [Best Practices for SharpDX Resources](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm#bestpracticesforsharpdxresources) section on this page for more information. |     To describe a **PathGeometry** object's path, use the object's [PathGeometry.Open()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_pathgeometry_open.htm) method to retrieve an **GeometrySink**.  Then, use the **GeometrySink** to populate the geometry with figures and segments.  To create a figure, call the [GeometrySink.BeginFigure()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_geometrysink_beginfigure.htm) method, specify the figure's start point, and then use its Add methods (such as [GeometrySink.AddLine()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_geometrysink_addline.htm)) to add segments.  When you are finished adding segments, call the [GeometrySink.EndFigure()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_geometrysink_endfigure.htm) method. You can repeat this sequence to create additional figures. When you are finished creating figures, call the [GeometrySink.Close()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_geometrysink_close.htm) method.     | ns | | --- | | // create three vectors to position the geometry  SharpDX.Vector2 startPoint = new SharpDX.Vector2(ChartPanel.X, ChartPanel.Y); SharpDX.Vector2 endPoint = new SharpDX.Vector2(ChartPanel.X + ChartPanel.W, ChartPanel.Y + ChartPanel.H); SharpDX.Vector2 centerPoint = (startPoint + endPoint) / 2;   // create the PathGeometry used by the RenderTarget Fill/Draw method SharpDX.Direct2D1.PathGeometry trianglePathGeometry   = new SharpDX.Direct2D1.PathGeometry(Core.Globals.D2DFactory);   // retrieve the GeometrySink used to describe the PathGeometry SharpDX.Direct2D1.GeometrySink geometrySink   = trianglePathGeometry.Open();   // create the points used to define the GeometrySink SharpDX.Vector2 beginPoint = new SharpDX.Vector2(centerPoint.X, startPoint.Y);    // Create a figure using the beginPoint geometrySink.BeginFigure(beginPoint, SharpDX.Direct2D1.FigureBegin.Filled);  // add lines to the figure SharpDX.Vector2 line1 = new SharpDX.Vector2(endPoint.X, centerPoint.Y); geometrySink.AddLine(line1); SharpDX.Vector2 line2 = new SharpDX.Vector2(centerPoint.X, endPoint.Y); geometrySink.AddLine(line2);   // end and close figure when finished geometrySink.EndFigure(SharpDX.Direct2D1.FigureEnd.Closed); geometrySink.Close();  // define the brush used in the geometry  SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue);    // execute the render target fill geometry with desired values RenderTarget.FillGeometry(trianglePathGeometry, customDXBrush);    // always dispose of a PathGeometry when finished  trianglePathGeometry.Dispose();  // always dispose of a brush when finished  customDXBrush.Dispose(); | | render_target_fillgeometry |      |  | | --- | | **Tip**:  For more examples of using **Shapes** for custom rendering, many of the DrawingTools included in the **NinjaTrader.Custom** project use these types of **SharpDX** objects and methods extensively. | |

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

tog_minus        [SharpDX Text Rendering](javascript:HMToggle('toggle','SharpDXTextRendering','SharpDXTextRendering_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Using SharpDX for rendering Text**  Up until this point, we have been using the [SharpDX.Direct2D1](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1.htm) namespace to render shapes.  When dealing with text, there is a separate [SharpDX.DirectWrite](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite.htm) namespace which works along with the **Direct2D1** objects.    There are two principle objects used for text rendering:  A **TextFormat** object which sets the style of the text, and a **TextLayout** object used to construct complex texts with various settings and provides metrics for measuring the shape the formatted text.    Each one of these objects has their own **RenderTarget** methods: [RenderTarget.DrawText()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawtext.htm) for simple **TextFormat** objects and [RenderTarget.DrawTextLayout()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawtextlayout.htm) for more advanced layouts.  Both methods accept a **TextFormat** object; **DrawTextLayout** is more complicated but has better performance since it reuses the same text layout which does not need to be recalculated.     |  | | --- | | **Tip**:  Both the **TextFormat** and **TextLayout** objects require a **DirectWrite** factory during construction.  For convenience, you can simply use the pre-built NinjaTrader[.Core.Globals.DirectWriteFactory](https://ninjatrader.com/es/support/helpGuides/nt8/directwritefactory.htm) property. |     **Formatting Text**  The **TextFormat** object determines the font size, style and family, among other properties.     |  | | --- | | **Warning**:  Any **SharpDX TextFormat** object used in your development must be disposed of after they have been used. NinjaTrader is **NOT** guaranteed to dispose of these resources for you!  Please see the [Best Practices for SharpDX Resources](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm#bestpracticesforsharpdxresources) section on this page for more information. |      | ns | | --- | | SharpDX.DirectWrite.TextFormat textFormat = new SharpDX.DirectWrite.TextFormat(Core.Globals.DirectWriteFactory, "Arial", 12); |     Once the text formatting has been described, you can use this object to immediately start rendering text in the DrawText() method.  This approach also requires a [SharpDX.RectangleF](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_rectanglef.htm) to help determine the size and position the text renders on the chart.     | ns | | --- | | // define the point for the text to render  SharpDX.Vector2 startPoint = new SharpDX.Vector2(ChartPanel.X, ChartPanel.Y);  // construct the text format with desired font family and size SharpDX.DirectWrite.TextFormat textFormat = new SharpDX.DirectWrite.TextFormat(Core.Globals.DirectWriteFactory, "Arial", 36);  // construct the rectangleF struct to describe the position and size the text SharpDX.RectangleF rectangleF = new SharpDX.RectangleF(startPoint.X, startPoint.Y, ChartPanel.W, ChartPanel.H);    // define the brush used for the text  SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue);  // execute the render target text command with desired values RenderTarget.DrawText("I am some text", textFormat, rectangleF, customDXBrush);  // always dispose of textFormat when finished textFormat.Dispose();  // always dipose of brush when finished customDXBrush.Dispose(); | | render_target_drawtext |     **Converting Text**  One common approach to text formatting is to use the same formats as existing chart objects.  This provides familiar text format matching other objects which exist on the chart.  To accomplish this, you can simply use the **ChartControl** NinjaTrader[.Gui.SimpleFont](https://ninjatrader.com/es/support/helpGuides/nt8/simplefont_class.htm) object and convert to **SharpDX** using the [ToDirectWriteTextFormat()](https://ninjatrader.com/es/support/helpGuides/nt8/simplefont_todirectwritetextformat.htm) method.     | ns | | --- | | SharpDX.DirectWrite.TextFormat textFormat = ChartControl.Properties.LabelFont.ToDirectWriteTextFormat(); |     **Text Layouts**  The **TextLayout** object works in combination with the **TextFormat** object by extending its functionality and providing an interface more powerful than a simple Rectangle, enabling you to position, measure, or clip the text to a surrounding shape.    When constructing the **TextLayout** object, you will pass in the exact text as a string you wish to render, along with the desired **TextFormat**.  This gives you the ability to measure the text string after it has been formatted.  During construction, you also have an opportunity to specify the maximum height and width of the **TextLayout**.  For example, we can set the text layout to bound to height and width chart panel:     | ns | | --- | | SharpDX.DirectWrite.TextLayout textLayout = new SharpDX.DirectWrite.TextLayout(Core.Globals.DirectWriteFactory, "I am also some text", textFormat, ChartPanel.W, ChartPanel.H); |     After the text has its format and layout,  you can use the [RenderTarget.DrawTextLayout()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawtextlayout.htm) method to specify the exact location as a [Vector2](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_vector2.htm), as well as the [Brush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_brush.htm) used to draw the text.   | ns | | --- | | RenderTarget.DrawTextLayout(startPoint, textLayout, customDXBrush); |     **Measuring Text Layouts**  Working with an existing **TextLayout** object, you can use its [TextLayout.Metrics](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textlayout_metrics.htm) object to retrieve metadata related to the size of the formatted text.   This is helpful if you are unsure of the size of the text before it is rendered.  For example, you may wish to draw a rectangle around the formatted text calculated width and height.  Using the approach below, the rectangle will dynamically resize to fit the text values used:     | ns | | --- | | // define the point for the text to render  SharpDX.Vector2 startPoint = new SharpDX.Vector2(ChartPanel.X + 20, ChartPanel.Y + 20);  // construct the text format with desired font family and size SharpDX.DirectWrite.TextFormat textFormat = new SharpDX.DirectWrite.TextFormat(Core.Globals.DirectWriteFactory, "Arial", 36);  // construct the text layout with desired text, text format, max width and height SharpDX.DirectWrite.TextLayout textLayout = new SharpDX.DirectWrite.TextLayout(Core.Globals.DirectWriteFactory, "I am also some text", textFormat, ChartPanel.W, ChartPanel.H);  // create a rectangle which will automatically resize to the width/height of the textLayout SharpDX.RectangleF rectangleF = new SharpDX.RectangleF(startPoint.X, startPoint.Y, textLayout.Metrics.Width, textLayout.Metrics.Height);    // define the brush used for the text and rectangle SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue);   // execute the render target draw rectangle with desired values RenderTarget.DrawRectangle(rectangleF, customDXBrush);    // execute the render target text layout command with desired values RenderTarget.DrawTextLayout(startPoint, textLayout, customDXBrush);  // always dispose of textLayout, textFormat, or brush when finished textLayout.Dispose(); textFormat.Dispose(); customDXBrush.Dispose(); | | render_target_drawtextlayout |      |  | | --- | | **Note**:  The **TextLayout.Metrics** height and width properties return the text pixel height, including the line spacing of the font.  Due to the nature of most font families, there will be an amount of line spacing above and below the text.  You can use the [TextLayout.GetLineMetrics()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textlayout_getlinemetrics.htm) method to help calculate the distance from the top of the text line to its baseline. | |

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

tog_minus        [SharpDX Stroke Style](javascript:HMToggle('toggle','SharpDXStrokeStyle','SharpDXStrokeStyle_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Using the StrokeStyle Object**  When rendering **SharpDX** Lines and Shapes, you can optionally configure a [SharpDX.Direct2D1.StrokeStyle](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_strokestyle.htm) allowing you to utilize several pre-made [dash styles](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_strokestyle_dashstyle.htm), or even create a custom dash pattern.     |  | | --- | | **Note**:  Unlike other **SharpDX** objects such as **brushes**, the **StrokeStyle** is a device-independent resource.  This means you only need to create the object once throughout the lifetime of the script.  However, the **StrokeStyle** needs to be disposed of when the script is terminated.  The **Creating a Custom DashStyle** example below shows how to use a stroke style from the beginning to end of the lifetime of your script.   Please see the [Best Practices for SharpDX Resources](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm#bestpracticesforsharpdxresources) section on this page for more information. |     For convenience, **SharpDX** provides the [StrokeStyleProperties](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_strokestyleproperties.htm) struct for creating new a **StrokeStyle:**     | ns | | --- | | // create a stroke style property using a pre-configured "DashDot" dash style SharpDX.Direct2D1.StrokeStyleProperties dxStrokeStyleProperties = new SharpDX.Direct2D1.StrokeStyleProperties {   DashStyle = SharpDX.Direct2D1.DashStyle.DashDot, }; |     Once you have your desired stroke style properties, you can create a new stroke style object.     |  | | --- | | **Warning**:  Any **SharpDX StrokeStyle** object used in your development must be disposed of after they have been used. NinjaTrader is **NOT** guaranteed to dispose of these resources for you!   Please see the [Best Practices for SharpDX Resources](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm#bestpracticesforsharpdxresources) section on this page for more information. |      | ns | | --- | | SharpDX.Direct2D1.StrokeStyle dxStrokeStyle = new SharpDX.Direct2D1.StrokeStyle(NinjaTrader.Core.Globals.D2DFactory, dxStrokeStyleProperties); |      |  | | --- | | **Tip**:  The **SharpDX.Direct2D1.StrokeStyle** require a **Direct2D1** factory during construction.  For convenience, you can simply use the pre-built NinjaTrader[.Core.Globals.D2DFactory](https://ninjatrader.com/es/support/helpGuides/nt8/d2dfactory.htm) property. |     And then use that object with the RenderTarget.DrawLine() method:   | ns | | --- | | RenderTarget.DrawLine(startPoint, endPoint, dxBrush, width, dxStrokeStyle); |     **Creating a Custom DashStyle**  By setting the [StrokeStyle.DashStyle](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_strokestyle_dashstyle.htm) property to "**Custom**", you can further refine the appearance of a **SharpDX** rendered line or shape by describing the length and space between the lines. Creating a custom **DashStyle** is not only useful for using **RenderTarget methods**, but also can be used for customizing the appearance of standard [NinjaScript Plots](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm).    The code example creates a single **StrokeStyle** object using custom dash style properties.  The example then uses those the custom stroke style object with user defined dashes for overriding the default NinjaTrader plot appearances, and using the same stroke style in a [RenderTarget.DrawLine()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawline.htm) command.     | ns | | --- | | // a SharpDX.Direct2D1.StrokeStyle is device independent // it only needs to be setup once throughout the lifetime of your script private SharpDX.Direct2D1.StrokeStyle dxStrokeStyle;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "Custom StrokeStyle";       AddPlot(Brushes.Blue, "Custom StrokeStyle");   }   else if (State == State.Configure)   {     // create a custom stroke style when configured     SharpDX.Direct2D1.StrokeStyleProperties dxStrokeStyleProperties = new SharpDX.Direct2D1.StrokeStyleProperties     {         // set the dash style to "Custom" define the dash pattern         DashStyle = SharpDX.Direct2D1.DashStyle.Custom,           // set further custom/optional StrokeStyle appearances         DashCap = CapStyle.Round,         EndCap   = CapStyle.Flat,         StartCap = CapStyle.Square,         LineJoin = LineJoin.Miter,           // offset in the dash sequence         DashOffset = 10.0f,     };       // define the an array of floating-point values     float[] dashes = { 1.0f, 2.0f, 2.0f, 3.0f, 2.0f, 2.0f };       // create the stroke style using the custom properties and dash array     dxStrokeStyle = new SharpDX.Direct2D1.StrokeStyle(NinjaTrader.Core.Globals.D2DFactory,             dxStrokeStyleProperties, dashes);   }   else if (State == State.Terminated)   {     // make sure to dispose of stroke style when finished     if (dxStrokeStyle != null)     {         if (!dxStrokeStyle.IsDisposed)           dxStrokeStyle.Dispose();     }   } }   protected override void OnBarUpdate() {   Value[0] = Close[0]; }   protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   // override the appearance of the default plot style   Plots[0].StrokeStyle = dxStrokeStyle;   base.OnRender(chartControl, chartScale);     // use the custom dash style in a RenderTarget.DrawLine() commands   using ( SharpDX.Direct2D1.SolidColorBrush dxBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Blue))   {     RenderTarget.DrawLine(new SharpDX.Vector2(ChartPanel.X, ChartPanel.Y), new SharpDX.Vector2(ChartPanel.X + ChartPanel.W, ChartPanel.Y + ChartPanel.H), dxBrush, 2, dxStrokeStyle);   } } | | SharpDX_StrokeStyle | |

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

tog_minus        [Best Practices for SharpDX Resources](javascript:HMToggle('toggle','BestPracticesforSharpDXResources','BestPracticesforSharpDXResources_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Understanding Device-dependent vs Device-independent resources**  Direct2D has several types of resources which may be mapped to the different hardware devices:    •**Device-independent** resources are on the CPU  •**Device-dependent** resources are on the GPU    When **device-dependent** resources are created, system resources are dedicated to that object.  Resources which are **device-dependent** are associated with a particular **RenderTarget** device and are only available on that device.  Therefore, objects which were created using a **RenderTarget** can only be used by that device.  As the **RenderTarget** updates, objects which were previously created will no longer be compatible and can lead to errors.  You can use the NinjaTrader [OnRenderTargetChange()](https://ninjatrader.com/es/support/helpGuides/nt8/onrendertargetchanged.htm)method to detect when the render target has updated and gives you an opportunity to recreate resources.    **Device-dependent resources**  The following objects are associated with a specific **RenderTarget**.  They must be created and dispose of any time the **RenderTarget** is updated:    •[Brush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_brush.htm)  •[GeometrySink](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_geometrysink.htm)  •[GradientStopCollection](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_gradientstopcollection.htm)  •[LinearGradientBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_lineargradientbrush.htm)  •[RadialGradientBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_radialgradientbrush.htm)  •[SolidColorBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_solidcolorbrush.htm)    **Device-independent resources**  The following objects are **NOT** associated with a specific device.  They can be created once and last for the lifetime of your script, or until they need to be modified:    •[PathGeometry](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_pathgeometry.htm)  •[StrokeStyle](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_strokestyle.htm)  •[TextFormat](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textformat.htm)  •[TextLayout](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textlayout.htm)     |  | | --- | | **Note**:  For more technical information on device resources, please see the [MSDN Direct2D Resources Overview](https://msdn.microsoft.com/en-us/library/dd756757(v=vs.85).aspx) |     **SharpDX DisposeBase**  Although most C# objects stored in memory are handled by the operating system, there are a few **SharpDX** resources which are not managed.  It is important to take care of these resources during the lifetime of your script as there is no guarantee that NinjaTrader will be able to dispose of these unmanaged references for you.    The following commonly used objects implement from the [SharpDX.DisposeBase](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_disposebase.htm) and should be disposed any time they are created:    •[Brush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_brush.htm)  •[GeometrySink](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_geometrysink.htm)  •[GradientStopCollection](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_gradientstopcollection.htm)  •[LinearGradientBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_lineargradientbrush.htm)  •[PathGeometry](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_pathgeometry.htm)  •[RadialGradientBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_radialgradientbrush.htm)  •[SolidColorBrush](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_solidcolorbrush.htm)  •[StrokeStyle](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_strokestyle.htm)  •[TextFormat](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textformat.htm)  •[TextLayout](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textlayout.htm)     |  | | --- | | **Warning**:  The list above is **NOT** exhaustive and there are other less common **SharpDX** objects that could implement **DisposeBase**. Failure to clean up these resources **WILL** result in NinjaTrader using more memory than necessary and may expose potential "memory leaks" coming from your script.  If you experience unusual amounts of memory being utilized over time, an unmanaged **SharpDX** resource is often times the culprit. |     Since there is no guarantee that NinjaTrader will release objects from memory when your script is terminated, it is best to protect these resources from issues and call [Dispose()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_disposebase_dispose.htm) as soon as possible.  This commonly involves calling **Dispose(**) at the end of [OnRender()](https://ninjatrader.com/es/support/helpGuides/nt8/onrender.htm),or during [OnRenderTargetChanged()](https://ninjatrader.com/es/support/helpGuides/nt8/onrendertargetchanged.htm) when dealing with **device- dependent** resources such as brush. **Device-independent** resources can be created once and then retained for the life of your application.     | ns | | --- | | protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   // 1 - setup your resource   SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue   // 2 - use your resource   RenderTarget.DrawLine(startPoint, endPoint, customDXBrush);         // 3- dispose of your resource   customDXBrush.Dispose() } |      |  | | --- | | **Note**:  If your resource is setup (i.e., uses the "new" keyword) during **OnRender()** or **OnRenderTargetChange()**, calling **.Dispose()** during [State.Terminated](https://ninjatrader.com/es/support/helpGuides/nt8/state.htm) will **ONLY** dispose of the *very last reference in memory* and is **NOT** sufficient to completely manage all instances created during the lifetime of your script.  You should be diligent in calling **Dispose()**throughout the lifetime of the script. |     You can also consider implementing the [using Statement (C# Reference)](https://msdn.microsoft.com/en-us/library/yh598w02.aspx) which will implicitly call **Dispose() for** you when you are done:     | ns | | --- | | // customDXBrush implicitly calls Dispose() after this block executes  using (SharpDX.Direct2D1.SolidColorBrush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue)) {   RenderTarget.DrawLine(startPoint, endPoint, customDXBrush); } |      |  | | --- | | **Critical**:  Attempting to use an object which has already been disposed can lead to memory corruption that NinjaTrader may not be able to recover.  Attempts to use an object in this manner can result in an error similar to: **Error on calling 'OnRender' method on bar 0: Attempted to read or write protected memory. This is often an indication that other memory is corrupt.** |       You can check to see if can object has been disposed of by using the [DisposeBase.IsDiposed](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_disposebase_isdisposed.htm) property:     | ns | | --- | | SharpDX.Direct2D1.Brush customDXBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.DodgerBlue);  // checks the object is not disposed of before using if(!customDXBrush.IsDisposed) {   RenderTarget.DrawLine(startPoint, endPoint, customDXBrush);   customDXBrush.Dispose(); } |     You should also favor managing these resources yourself, which means methods which accept a **SharpDX DisposeBase** object as an argument should be created before they are passed into the method and disposed of after they are used.  For example, the code below should be avoided:     | ns | **Practice to avoid** | | --- | --- | | // do NOT convert an object as it is passed to an argument.  // You may have no chance to Dispose of the object!  // Finalizer is not guaranteed to release of these resources RenderTarget.DrawLine(startPoint, endPoint, Brushes.AliceBlue.ToDxBrush(RenderTarget));    MyCustomMethod(Brushes.AliceBlue.ToDxBrush(RenderTarget)); | |     Instead, you should manage these objects yourself:   | ns **Best practice** | | --- | | // Do create and store this reference yourself so you can control when it is released (Y) SharpDX.Direct2D1.Brush customDXBrush = WPFBrush.ToDxBrush(RenderTarget);    RenderTarget.DrawLine(startPoint, endPoint, customDXBrush));    MyCustomMethod(customDXBrush);    customDXBrush.Dipose() |     **Other Best Practices**    If possible, you should avoid using the [ToDxBrush()](https://ninjatrader.com/es/support/helpGuides/nt8/dxextensions_todxbrush.htm) method if it is not necessary.  It is relatively harmless to use this approach for a few brushes, but can introduce performance issues if used too liberally.     | ns **Practice to avoid** | | --- | | // do NOT convert from WPF brushes unnecessarily  SharpDX.Direct2D1.Brush dxBrush1 = System.Windows.Media.Brushes.Blue.ToDxBrush(RenderTarget); SharpDX.Direct2D1.Brush dxBrush2 = System.Windows.Media.Brushes.Red.ToDxBrush(RenderTarget); SharpDX.Direct2D1.Brush dxBrush3 = System.Windows.Media.Brushes.Green.ToDxBrush(RenderTarget); SharpDX.Direct2D1.Brush dxBrush4 = System.Windows.Media.Brushes.Purple.ToDxBrush(RenderTarget); SharpDX.Direct2D1.Brush dxBrush5 = System.Windows.Media.Brushes.Orange.ToDxBrush(RenderTarget); SharpDX.Direct2D1.Brush dxBrush6 = System.Windows.Media.Brushes.Yellow.ToDxBrush(RenderTarget); |     Instead, you should construct a SharpDX Brush directly if a WPF brush is not ever needed:   | ns **Best practice** | | --- | | // Do create SharpDX Brushes directly if you have a large amount of brushes SharpDX.Direct2D1.Brush dxBrush1 = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Blue); SharpDX.Direct2D1.Brush dxBrush2 = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Red); SharpDX.Direct2D1.Brush dxBrush3 = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Green); SharpDX.Direct2D1.Brush dxBrush4 = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Purple); SharpDX.Direct2D1.Brush dxBrush5 = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Orange); SharpDX.Direct2D1.Brush dxBrush6 = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Yellow); |     Rendering with anti-aliasing disabled can be used to render a higher qualify shapes but comes as a performance impact.  You should make sure to set this render target property back to its default when you are finished with a render routine.     | ns **Best practice** | | --- | | // AntialiasMode.PerPrimitive is more resource intensive  // store the old reference before setting the desired value SharpDX.Direct2D1.AntialiasMode oldAntialiasMode = RenderTarget.AntialiasMode; RenderTarget.AntialiasMode = SharpDX.Direct2D1.AntialiasMode.PerPrimitive;   // execute your render routines   // and then set back to initial AntialiasMode when finished RenderTarget.AntialiasMode = oldAntialiasMode; | |

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

|  |  |
| --- | --- |
| **Navigation:**  »No topics above this level«  **8.0.0.13 (RC1) Release Notes** | [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textlayout.htm) |

**Release Date**

August 31st, 2016

This release marks our first NinjaTrader 8 Release Candidate.  Since the very first NinjaTrader 8 beta version, we have resolved over 4500 bugs thanks to the ongoing efforts of our beta community, and we feel confident we are closer than ever to a production NinjaTrader 8 release.  If you have installed a NinjaTrader 8 beta version and reported feedback to our support team - thank you!

As this is a release candidate, NinjaTrader 8.0.0.13 is still considered a beta product and we will continue to focus on product quality.  Please continue to report any issues you may encounter to our support staff.   We will monitor the status of this release to determine when we will announce and launch the production release of NinjaTrader 8.

|  |
| --- |
| **Attention MB Trading Users:**Due to limited use and low user feedback during the beta period, we have removed the **MB Trading** adapter from NinjaTrader 8.You may continue to use NinjaTrader 7. |

**Code Breaking Changes**

**Compile Errors**

•The [Stroke](https://ninjatrader.com/es/support/helpGuides/nt8/stroke_class.htm) object **.Dispose()** method was removed due to technical redundancy.  To remove memory resources from any stroke objects, simply set the stroke to null.

•Removed property **Bars.IsTimebased** -> please use [Bars.BarsType.IsTimeBased](https://ninjatrader.com/es/support/helpGuides/nt8/barstype_istimebased.htm) instead

•**Account.Accounts** was renamed to [Account.All](https://ninjatrader.com/es/support/helpGuides/nt8/all.htm)

**Implementation changes**

•The common signature "isInclude60" used in various [SessionIterator](https://ninjatrader.com/es/support/helpGuides/nt8/sessioniterator.htm) methods was renamed to "includesEndTimeStamp" to be more specific

•Category display order values of standard NinjaTrader**Property Grid Categories** were updated to be more consistent application wide.  These changes could impact any customization you were doing using the [CategoryOrderAttribute](https://ninjatrader.com/es/support/helpGuides/nt8/categoryorderattribute.htm), however, the documentation was also updated to reflect the implementation more accurately and will allow you to use this attribute reliably.

•To assist with transitioning historical order objects to real-time order references, please use the new [GetRealtimeOrder()](https://ninjatrader.com/es/support/helpGuides/nt8/getrealtimeorder.htm) method.

**Notes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Status** | **Issue #** | **Category** | **Comments** |
| Fixed | 10267 | Account Data | Sorting by Commission in Executions tab caused unhandled exception |
| Fixed | 10126 | Account Data, Database | Database caused accounts to show Realized PnL when not connected |
| Fixed | 10123 | Account Data, Rithmic, | Realized PnL only reflected 1 side of the commission |
| Fixed | 10003 | Alerts | Alert Condition Localization issue |
| Fixed | 10164 | Alerts | Exception on opening alerts log window when existing alerts log entry used custom brush |
| Fixed | 10192 | Alerts, DrawingTool | Alerts stop triggering after editing drawing tool anchor via drawing objects window |
| Fixed | 10131 | ATM Strategies | ATM field did not change when submitting ATM from another window on startup |
| Fixed | 10253 | ATM Strategies | ATM is terminated and ATM order cancelled upon order modification failure |
| Fixed | 10279 | ATM Strategies | Modifying ATM Order modified ATM template |
| Fixed | 10217 | Attach Order To Indicator, NinjaScript | Attaching order to indicator via CTRL key resulted in Unhandled Exception |
| Fixed | 10137 | Bars | Crashes could occur when restoring workspaces |
| Fixed | 10220 | Bars | Dates were not showing correctly in New Zealand on daily bars |
| Fixed | 9940 | Bars | Errors could occur when loading data from TD Ameritrade. |
| Fixed | 10132 | Bars | RequestBarSeries1 ERROR while removing workspace in the middle of bars request |
| Fixed | 10105 | Bars | Toggling the Break at EOD with DoNotMerge on large data sets caused blank charts or less bars |
| Fixed | 10036 | Bars, NinjaScript | BarsRequest MergePolicy did not match UseGlobalSettings when using the same policy |
| Changed | 10136 | BarsType | Custom BarType time variable was equal to bars.LastBarTime on new data point |
| Fixed | 10160 | BarsType | Custom BarsType OnDataPoint sometimes had unexpected bid ask data |
| Fixed | 10086 | BarsType | Point and Figure Charts were not calling OnBarUpdate() for each tick or price change |
| Fixed | 10095 | Chart | Chart were allowed to remove last series incorrectly |
| Fixed | 10280 | Chart | Crosshair position showed old position when toggled via hotkey until mouse move |
| Fixed | 10240 | Chart | Drag and drop primary series in same panel caused indicator to jump to primary panel |
| Fixed | 10284 | Chart | Extraneous plot selection point displayed on multi-level indicator as input plot |
| Fixed | 10218 | Chart | Global crosshair render issue at chart boundaries |
| Fixed | 10178 | Chart | Global crosshair incorrect x axis flag time when locked via context menu |
| Fixed | 10156 | Chart | Global draw object was not removed with NinjaScript unless NinjaScript removed manually |
| Fixed | 10097 | Chart | Save Chart Image file name only included the primary data series |
| Fixed | 10242 | Chart | When primary series deleted on multi series chart, instrument selector not updated to new primary |
| Fixed | 10236 | Chart | Z-Order: reloading the historical data reset the z-order |
| Fixed | 10245 | Chart Trader | Errors using ChartTrader with Playback connection |
| Fixed | 10248 | Chart Trader, Templates | Chart Trader settings were ignored when chart template is loaded on an open chart |
| Fixed | 10237 | Chart, Drawing | Unhandled exceptions when moving series panels with GlobalDrawObjects |
| Fixed | 9806 | Chart, Drawing, NinjaScript | Chart could freeze using NinjaScript Draw method/Removing Drawing Tools |
| Fixed | 10117 | Chart, Indicator | System indicators did not load properly after connecting to live data |
| Fixed | 10183 | Chart, Templates | Template caused "An item of the same key has been added" when using two of the same indicator |
| Fixed | 10254 | Chart, Workspaces | InvalidOperationException on restoring chart with template/preset |
| Fixed | 10002 | Commissions | Forex Commissions were factoring Per-Unit instead of Per Lot Size |
| Fixed | 10046 | Control Center | Connection status indicator sometimes did not update |
| Fixed | 10144 | Control Center | Edit strategy dialog was throwing incorrect error |
| Fixed | 10091 | Control Center | Deleted account connection could be disconnected |
| Fixed | 10023 | Control Center, Playback, Strategy | Playback trades performance realized PnL did not match strategies tab of control center |
| Changed | 10111 | Control Center, Strategy | SystemPerformance object was not updating for control center enabled strategies |
| Fixed | 10158 | Control Center, Strategy | Strategy CurrentBars index repored incorrectly after changing parameter and enabling strategy |
| Fixed | 10275 | Control Center, Workspaces | Control Center was incorrectly restored to primary screen when saved maximized |
| Fixed | 10139 | CQG, Workspaces | Workspace charts did not load after connecting to account with no data |
| Fixed | 9909 | Drawing, Strategy Analyzer | Strategy analyzer was not releasing memory when adding indicator that draws objects |
| Fixed | 10167 | DrawingTool | "Remove all drawing objects" did not remove objects unless the originating tab is selected |
| Fixed | 10184 | DrawingTool | Draw object incorrect resize or incorrect anchor on attempting to move past start bar of chart |
| Fixed | 10113 | DrawingTool | Draw objects in future would move when days to load changed |
| Fixed | 10265 | DrawingTool | Draw.Text autoscale was not working correctly |
| Fixed | 10264 | DrawingTool | Draw.Text no alignment behavior did not match NT7 |
| Fixed | 10099 | DrawingTool | Drawing tool template with attach to all charts was drawing on other charts even if cancelled |
| Fixed | 10179 | DrawingTool | Moving global draw object on multi series chart changed anchors incorrectly |
| Fixed | 10061 | DrawingTool | Ray selection points were not lined up on logarithmic y-axis scale |
| Fixed | 10127 | DrawingTool, Playback | Global draw anchors were not consistent in multi-series playback |
| Fixed | 10224 | DrawingTool, Templates | Unable to apply a template more than once to a drawing object |
| Fixed | 10124 | eSignal | ESignal historical tick data timestamps did not match NT7 |
| Fixed | 10214 | FX Board, Hotlist Analyzer, Market Analyzer | Create instrument list context menu item did not work if no instruments loaded |
| Changed | 10208 | Indicator | Consistency updates to indicator error handling |
| Fixed | 10157 | Indicator | BuySellPressure when called from another indicator could cause errors |
| Fixed | 10274 | Indicator | Indicator label did not saved with preset |
| Fixed | 9748 | Instruments | @GER30  CFD Data does not show decimal values |
| Changed | 10145 | Instruments, Yahoo | Yahoo connection is not updating Splits and Dividends. |
| Changed | 10170 | Interactive Brokers | Removed IB Linked Account Support |
| Fixed | 10212 | Interactive Brokers | IB Paper Trading account would not connect on version 954 TWS |
| Fixed | 9993 | Interactive Brokers | Incorrect instrument update price on some instruments |
| Fixed | 9923 | Kinetick | Connection loss loop could occur in some situations |
| Fixed | 10088 | Licensing | FreeTrial Vendor License caused excess lines in Config.xml |
| Fixed | 10209 | Licensing | From and to dates in grid in Vendor Licensing window region formatting |
| Fixed | 10060 | Licensing | Vendor License Addon did not update license messages when changing between vendors |
| Fixed | 10108 | Market Analyzer | Market Analyzer Indicator settings were not recognized when applying template |
| Fixed | 10201 | Market Analyzer | Unable to remove expired instruments from market analyzer in some scenarios |
| Changed | 10196 | Market Analyzer, Workspaces | Custom Market Analyzer Column could not serialize CurrentText |
| Changed | 10180 | MBTrading | Removed MBT Adapter in NinjaTrader 8 |
| Changed | 10106 | NinjaScript | Incorrect sound played when SetProfitTarget target is reached |
| Changed | 10260 | NinjaScript | Renamed a few properties to meet coding guidelines |
| Fixed | 10071 | NinjaScript | CancelOrder() would not cancel historical working orders in State.Realtime |
| Fixed | 10222 | NinjaScript | Errors could occur after deleting indicator and reloading chart |
| Fixed | 9927 | NinjaScript | Draw.Region displacement was from the left of the chart and not from bar 0 |
| Fixed | 10206 | NinjaScript | Expandable properties were not reseting to defaults |
| Fixed | 10177 | NinjaScript | Indicator Error on calling 'SetState' method with tick replay |
| Fixed | 10112 | NinjaScript | IsSuspendedWhileInactive did not work if suspended prior to data feed connection |
| Fixed | 10221 | NinjaScript | Null Stroke object in AddPlot() cuased platform crash |
| Fixed | 10153 | NinjaScript | Unhandled exception if using PasswordBox in Addon |
| Fixed | 10172 | NinjaScript | Update() on multiseries indicator caused primary series OBU called twice |
| Changed | 8410 | NinjaScript Editor | Added Additional Default Snippets |
| Fixed | 10085 | NinjaScript, Orders | Only first identical State.Historical order moved to State.Realtime |
| Fixed | 10233 | NinjaScript, Strategy | Enabling multiple AdoptAccountPosition strategies would result in incorrect popup message |
| Fixed | 10140 | NinjaScript, UI | Strategy Catagories/properties could get out of sequence in strategy and in strategy analyzer |
| Changed | 10162 | Options, Strategy | ConnectionLossHandling was overwritten between State.SetDefaults and State.Configure |
| Fixed | 10148 | Orders, SuperDOM | Cancel all order icon was not visible for simulated orders on SuperDOM |
| Fixed | 10205 | Playback | Simulation accounts could be added unexpecitly while connected to playback |
| Fixed | 10138 | Playback | Exception was thrown unsubscribing data disconnecting playback |
| Fixed | 10202 | Playback | Errors existed around persisting errors to database |
| Fixed | 10142 | Playback | Playback controller date, time and speed sometimes was not visible |
| Fixed | 10211 | Playback | Playback controller end date did not update until platform restart |
| Fixed | 10149 | Playback | Playback manual trade MAE/MFE/ETD values were incorrect |
| Fixed | 9974 | Playback | Sustained playback could resultng in chart rendering error |
| Fixed | 10258 | Playback | Errors could occur connecting to playback on UTC time zone |
| Fixed | 10200 | Skins, UI | Chart Properties Tab Name drop down was barely visible |
| Added | 10087 | Strategy | Strategy concept needed to help users manage historical to live transitioned orders |
| Fixed | 10107 | Strategy | Limit order did not fill if CancelOrder() used on protective order |
| Fixed | 10064 | Strategy | Realized PnL was not updating for strategy with secondary series on strategies tab |
| Fixed | 10128 | Strategy | Strategy template options are not available when editing from strategy tab |
| Fixed | 10194 | Strategy Analyzer | Backtest sometimes ran twice if display was not set to summary or settings |
| Fixed | 10143 | Strategy Analyzer | Platform crash during optimization when accessing null object after State.Terminated |
| Fixed | 10010 | Strategy Analyzer | Platform could crash when using duplicate to new window |
| Fixed | 10181 | Strategy Analyzer | Trades Display "Strategy Column" was blank after duplicated |
| Fixed | 10210 | Strategy Analyzer | Optimization results did not match displays |
| Fixed | 10204 | Strategy Analyzer | Tab context menu items could become disabled incorrectly |
| Fixed | 10263 | Strategy Analyzer | Strategy template was not saving changes after running a backtest |
| Changed | 10118 | Strategy Builder | Allows Strategy Builder Strategies to be manged from NinjaScript Editor |
| Changed | 10101 | Strategy Builder | Could not export Strategy Builder strategy |
| Changed | 10168 | Strategy Builder | Now freezing custom brushes in Strategy Builder to match best practices |
| Changed | 10100 | Strategy Builder | No historical trades taken due to indicators with barsAgo check |
| Fixed | 10103 | Strategy Builder | Strategy Builder could lose reference to candlestick pattern |
| Fixed | 10090 | Strategy Builder | Plot On Chart was calling AddChartIndicator() in wrong state |
| Fixed | 10225 | Strategy Builder | Drawing action category was using wrong name |
| Fixed | 10272 | Strategy Builder | Exception selecting indicator with NinjaScriptProperty value not set |
| Fixed | 10163 | Strategy Builder | Alert message defaulted to same color for foreground/background |
| Fixed | 10262 | Strategy Builder | Did not set indicator Brush properties |
| Fixed | 10271 | Strategy Builder | Strategies with errors could not be deleted |
| Fixed | 10155 | Strategy Builder | Time series could not be compared with Time input or variable in Strategy Builder conditions |
| Fixed | 10082 | Strategy, Trade Performance | Strategy performance calcs had been off |
| Fixed | 10083 | SuperDOM | Order action name localization issue |
| Fixed | 10246 | SuperDOM | SuperDOM > Columns "i" did not give information on the Columns |
| Fixed | 10187 | SuperDOM, Workspaces | SuperDOM Indicator Days to load property reverts to 2 on restore workspace |
| Fixed | 10195 | TD AMERITRADE | Historical daily bars could hold more decimal places than TickSize |
| Fixed | 10285 | Time and Sales | Columns unexpectedly resize when removing/adding columns |
| Fixed | 10203 | UI | Account Tab -> Typing or selecting in account selector did not send to other linked windows |
| Fixed | 10223 | UI | Control Center could sometimes be out of focus after using menus |
| Fixed | 10238 | UI | When selecting the same instrument in the data series window (in the most recent list), the instrument did not add. |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [Drawing](https://ninjatrader.com/es/support/helpGuides/nt8/drawing.htm) > [Draw.Text()](https://ninjatrader.com/es/support/helpGuides/nt8/draw_text.htm) >  **Text** | | | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/draw_text.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/draw_text.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/draw_textfixed.htm) |

**Definition**

Represents an interface that exposes information regarding a Text [IDrawingTool](https://ninjatrader.com/es/support/helpGuides/nt8/idrawingtool.htm).

**Methods and Properties**

|  |  |
| --- | --- |
| Anchor | An [IDrawingTool's ChartAnchor](https://ninjatrader.com/es/support/helpGuides/nt8/idrawingtool.htm#chartanchor) representing the point of the drawing object |
| YPixelOffset | An int value representing the offset value in pixels from within the text box area |
| Alignment | Possible values are:    TextAlignment.Center,  TextAlignment.Left,  TextAlignment.Right,  TextAlignment.Justify ([reference](https://msdn.microsoft.com/en-us/library/system.windows.textalignment(v=vs.110).aspx)) |
| AreaOpacity | An int value representing the opacity of the area color |
| AreaBrush | A [Brush](http://msdn.microsoft.com/en-us/library/system.windows.media.brush(v=vs.110).aspx) class representing the fill color of the text box |
| Text | A string value representing the text to be drawn |
| TextBrush | A [Brush](http://msdn.microsoft.com/en-us/library/system.windows.media.brush(v=vs.110).aspx) class representing the color of the text |
| Font | A [Font](http://msdn.microsoft.com/en-us/library/system.drawing.font_members(v=vs.90).aspx) object representing the font for the text |
| OutlineStroke | The [Stroke](https://ninjatrader.com/es/support/helpGuides/nt8/stroke_class.htm) object used to outline the text box |

**Example**

| ns | |
| --- | --- |
| // Instantiate a Text object Text myText = Draw.Text(this, "tag1", "Text to draw", 10, High[10] + (5 \* TickSize), Brushes.Black);   // Change the object's DisplayText myText.DisplayText = "New Display Text"; | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [Drawing](https://ninjatrader.com/es/support/helpGuides/nt8/drawing.htm) > [Draw.TextFixed()](https://ninjatrader.com/es/support/helpGuides/nt8/draw_textfixed.htm) >  **TextFixed** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/draw_textfixed.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/draw_textfixed.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/draw_timecycles.htm) |

**Definition**

Represents an interface that exposes information regarding a Text Fixed [IDrawingTool](https://ninjatrader.com/es/support/helpGuides/nt8/idrawingtool.htm).

**Methods and Properties**

|  |  |
| --- | --- |
| Anchor | AAn [IDrawingTool's ChartAnchor](https://ninjatrader.com/es/support/helpGuides/nt8/idrawingtool.htm#chartanchor) representing the point of the drawing object |
| YPixelOffset | An int value representing the offset value in pixels from within the text box area |
| Alignment | Possible values are:    TextAlignment.Center TextAlignment.Far TextAlignment.Near  TextAlignment.Justify  ([reference](https://msdn.microsoft.com/en-us/library/system.windows.textalignment%28v=vs.110%29.aspx)) |
| AreaOpacity | An int value representing the opacity of the area color |
| AreaBrush | A [Brush](http://msdn.microsoft.com/en-us/library/system.windows.media.brush(v=vs.110).aspx) class representing the fill color of the text box |
| DisplayText | A string value representing the text to be drawn |
| TextBrush | A [Brush](http://msdn.microsoft.com/en-us/library/system.windows.media.brush(v=vs.110).aspx) class representing the color of the text |
| Font | A [Font](https://ninjatrader.com/es/support/helpGuides/nt8/simplefont_class.htm) object representing the font for the text |
| OutlineStroke | The [Stroke](https://ninjatrader.com/es/support/helpGuides/nt8/stroke_class.htm) object used to outline the text box |
| TextPosition | Possible values are:    TextPosition.BottomLeft  TextPosition.BottomRight  TextPosition.Center  TextPosition.TopLeft  TextPosition.TopRight |

**Example**

| ns | | |
| --- | --- | --- |
| // Instantiate a TextFixed object TextFixed myTF = Draw.TextFixed(this, "tag1", "Text to draw", TextPosition.TopRight);   // Change the object's TextPosition myTF.TextPosition = TextPosition.Center; | | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) >  **Language Reference** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using__brackets.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/alphabetical_reference.htm) |
| **NinjaScript Language Reference**  ›[Add On](https://ninjatrader.com/es/support/helpGuides/nt8/add_on.htm)  ›[Bars Type](https://ninjatrader.com/es/support/helpGuides/nt8/bars_type.htm)  ›[Chart](https://ninjatrader.com/es/support/helpGuides/nt8/chart.htm)  ›[Chart Style](https://ninjatrader.com/es/support/helpGuides/nt8/chart_style.htm)  ›[Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm)  ›[Drawing](https://ninjatrader.com/es/support/helpGuides/nt8/drawing.htm)  ›[Drawing Tool](https://ninjatrader.com/es/support/helpGuides/nt8/drawing_tools.htm)  ›[Import Type](https://ninjatrader.com/es/support/helpGuides/nt8/import_type.htm)  ›[Indicator](https://ninjatrader.com/es/support/helpGuides/nt8/indicator.htm)  ›[Indicator Methods](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm)  ›[ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm)  ›[Market Analyzer Column](https://ninjatrader.com/es/support/helpGuides/nt8/market_analyzer_column.htm)  ›[Instrument](https://ninjatrader.com/es/support/helpGuides/nt8/instrument.htm)  ›[Optimization Fitness](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm)  ›[Optimizer](https://ninjatrader.com/es/support/helpGuides/nt8/optimizer.htm)  ›[Performance Metrics](https://ninjatrader.com/es/support/helpGuides/nt8/performance_metrics.htm)  ›[Share Service](https://ninjatrader.com/es/support/helpGuides/nt8/share_service.htm)  ›[Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm)  ›[SuperDOM Column](https://ninjatrader.com/es/support/helpGuides/nt8/superdom_column.htm) | | |

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [SharpDX SDK Reference](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_sdk_reference.htm) > [SharpDX.Direct2D1](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1.htm) >  **SharpDX.Direct2D1.RenderTarget** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_radialgradientbrushproperties.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_antialiasmode.htm) |
| **Disclaimer**: The [SharpDX SDK Reference](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_sdk_reference.htm) section was compiled from the official [SharpDX Documentation](http://sharpdx.org/) and was **NOT** authored by NinjaTrader.  The contents of this section are provided as-is and only cover a fraction of what is available from the **SharpDX SDK**.  This page was intended only as a reference guide to help you get started with some of the 2D Graphics concepts used in the **NinjaTrader.Custom** assembly.  Please refer to the official **SharpDX Documentation** for additional members not covered in this reference.  For more seasoned graphic developers, the original**MSDN** [Direct2D1](https://msdn.microsoft.com/en-us/library/windows/desktop/dd370990.aspx) and [DirectWrite](https://msdn.microsoft.com/en-us/library/windows/desktop/dd368038.aspx) unmanaged API documentation can also be helpful for understanding the DirectX/Direct2D run-time environment. *For NinjaScript development purposes, we document only****essential****members in the structure of this page.* | |

**Definition**

Represents an object that can receive drawing commands.

(See also [unmanaged API documentation](http://msdn.microsoft.com/en-us/library/dd371766.aspx))

**Syntax**

class RenderTarget

|  |
| --- |
| **Tips**:  1.For NinjaScript Development purposes, [DrawingTools](https://ninjatrader.com/es/support/helpGuides/nt8/drawingtool.htm), [ChartStyles](https://ninjatrader.com/es/support/helpGuides/nt8/chartstyletype.htm), [Indicators](https://ninjatrader.com/es/support/helpGuides/nt8/indicators.htm), and [Strategies](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) implement the Chart's [RenderTarget](https://ninjatrader.com/es/support/helpGuides/nt8/rendertarget.htm) ready to be used in the **OnRender()** method  2.General information on **Direct2D** **Render Targets** can be found on the [MSDN Direct2D Render Targets Overview](https://msdn.microsoft.com/en-us/library/dd756757(v=vs.85).aspx) |

**Methods and Properties**

|  |  |
| --- | --- |
| [AntialiasMode](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_antialiasmode.htm) | Retrieves or sets the current antialiasing mode for nontext drawing operations. |
| [DrawEllipse()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawellipse.htm) | Draws the outline of the specified ellipse using the specified stroke style. |
| [DrawGeometry()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawgeometry.htm) | Draws the outline of the specified geometry. |
| [DrawLine()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawline.htm) | Draws a line between the specified points. |
| [DrawRectangle()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawrectangle.htm) | Draws the outline of a rectangle that has the specified dimensions. |
| [DrawText()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawtext.htm) | Draws the specified text using the format information provided by an [SharpDX.DirectWrite.TextFormat](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textformat.htm) object. |
| [DrawTextLayout()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_drawtextlayout.htm) | Draws the formatted text described by the specified [SharpDX.DirectWrite.TextLayout](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_directwrite_textlayout.htm) object. |
| [FillEllipse()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_fillellipse.htm) | Paints the interior of the specified ellipse. |
| [FillGeometry()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_fillgeometry.htm) | Paints the interior of the specified geometry. |
| [FillRectangle()](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_fillrectangle.htm) | Paints the interior of the specified rectangle. |
| [IsDisposed](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_disposebase_isdisposed.htm) | Gets a value indicating whether this instance is disposed.  (Inherited from [SharpDX.DisposeBase](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_disposebase.htm).) |
| [Transform](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_direct2d1_rendertarget_transform.htm) | Gets or sets the current transform of the render target. |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [Charts](https://ninjatrader.com/es/support/helpGuides/nt8/chart.htm) >  **Rendering** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/width.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/chart.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/d2dfactory.htm) |

Rendering methods and properties can be useful when carrying out custom drawing tasks for chart objects. Event handlers such as [OnCalculateMinMax()](https://ninjatrader.com/es/support/helpGuides/nt8/oncalculateminmax.htm) and [OnRender()](https://ninjatrader.com/es/support/helpGuides/nt8/onrender.htm) allow you to override behavior at key points in the rendering process.

|  |
| --- |
| **Notse**:  1.Some rendering methods and properties make use of [SharpDX](http://sharpdx.org/) libraries, which provide a managed framework for working with DirectX technology.  Please see the [SharpDX SDK Reference](https://ninjatrader.com/es/support/helpGuides/nt8/sharpdx_sdk_reference.htm) for more information.  2.For a walk through for using the **SharpDX**, please see the educational resource [Using SharpDX for Custom Chart Rendering](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm) |

**Methods and Properties**

|  |  |
| --- | --- |
| [RenderTarget](https://ninjatrader.com/es/support/helpGuides/nt8/rendertarget.htm) | Creates objects and exposes methods used for drawing in the chart area. |
| [ForceRefresh()](https://ninjatrader.com/es/support/helpGuides/nt8/forcerefresh.htm) | Forces OnRender() to be called, which will re-paint the chart |
| [IsInHitTest](https://ninjatrader.com/es/support/helpGuides/nt8/isinhittest.htm) | Qualifies if object drawn in chart object should be selectable in the hit test procedure |
| [IsSelected](https://ninjatrader.com/es/support/helpGuides/nt8/isselected.htm) | Indicates a chart object is currently selected |
| [IsVisibleOnChart()](https://ninjatrader.com/es/support/helpGuides/nt8/isvisibleonchart.htm) | Indicates a chart object is visible on the chart canvas |
| [MaxValue](https://ninjatrader.com/es/support/helpGuides/nt8/maxvalue.htm) | The maximum value used for the automatic scaling of the y axis |
| [MinValue](https://ninjatrader.com/es/support/helpGuides/nt8/minvalue.htm) | The minimum value used for the automatic scaling of the y axis |
| [OnCalculateMinMax()](https://ninjatrader.com/es/support/helpGuides/nt8/oncalculateminmax.htm) | An event driven method which is called while the chart scale is being updated |
| [OnRender()](https://ninjatrader.com/es/support/helpGuides/nt8/onrender.htm) | Used to render custom drawing to a chart from various chart objects |
| [PanelUI](https://ninjatrader.com/es/support/helpGuides/nt8/panelui.htm) | The chart panel that is configured on the chart's UI |
| [ZOrder](https://ninjatrader.com/es/support/helpGuides/nt8/chart_zorder.htm) | A unique identifier used to control the order in which chart objects are drawn on the chart's Z-axis |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) >  **Working with Brushes** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/educational_resources.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_chart_object_coordinates.htm) |

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

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

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

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

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

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

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

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