|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Optimizer](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) >  **OptimizationParameters** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/onoptimize.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/runiteration.htm) |

**Definition**

The optimization parameters selected for the optimization run. (e.g. user parameters or Data Series)

**Property Value**

A bool value.

**Syntax**

Strategies[0].OptimizationParameters

**Examples**

| ns | |
| --- | --- |
| protected override void OnOptimize()  {      // If there are no optimization parameters to optimize, return      if (Strategies[0].OptimizationParameters.Count == 0)          return;        // Do something with the optimization parameter      Parameter parameter = Strategies[0].OptimizationParameters[0];  } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) >  **Optimizer** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/optimization_fitness_value.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/numberofiterations.htm) |

Custom Optimizers can be used to optimize your Strategy through different algorithms. These may allow you to make trade offs like being able to find adequate results quickly as opposed to trying to find the absolute best result but through a time consuming process. The methods and properties covered in this section are unique to custom Optimizer development.

**In this section**

|  |  |
| --- | --- |
| [NumberOfIterations](https://ninjatrader.com/support/helpGuides/nt8/numberofiterations.htm) | Informs the Strategy Analyzer how many iterations of optimizing it needs to do. |
| [OnOptimize()](https://ninjatrader.com/support/helpGuides/nt8/onoptimize.htm) | This method must be overridden in order to optimize a strategy. |
| [OptimizationParameters](https://ninjatrader.com/support/helpGuides/nt8/optimizationparameters.htm) | The optimization parameters selected for the optimization run. |
| [RunIteration()](https://ninjatrader.com/support/helpGuides/nt8/runiteration.htm) | Runs an iteration of backtesting for the optimizer. |
| [SupportsMultiObjectiveOptimization](https://ninjatrader.com/support/helpGuides/nt8/supportsmultiobjectiveoptimiza.htm) | Informs the Strategy Analyzer if this Optimizer can do multi-objective optimizations. |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Optimizer](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) >  **OptimizationParameters** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/onoptimize.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/runiteration.htm) |

**Definition**

The optimization parameters selected for the optimization run. (e.g. user parameters or Data Series)

**Property Value**

A bool value.

**Syntax**

Strategies[0].OptimizationParameters

**Examples**

| ns | |
| --- | --- |
| protected override void OnOptimize()  {      // If there are no optimization parameters to optimize, return      if (Strategies[0].OptimizationParameters.Count == 0)          return;        // Do something with the optimization parameter      Parameter parameter = Strategies[0].OptimizationParameters[0];  } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Optimizer](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) >  **SupportsMultiObjectiveOptimization** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/runiteration.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/performance_metrics.htm) |

**Definition**

Informs the Strategy Analyzer if this Optimizer can do multi-objective optimizations.

**Property Value**

A bool value.

**Syntax**

SupportsMultiObjectiveOptimization

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange()  {      if (State == State.SetDefaults)      {           Name = "MyOptimizer";           SupportsMultiObjectiveOptimization = true;      }  } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Strategy](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm) >  **IsInstantiatedOnEachOptimizationIteration** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/isfilllimitontouch.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/isinstrategyanalyer.htm) |

**Definition**

Determines if the strategy should be re-instantiated (re-created) after each optimization run when using the [Strategy Analyzer Optimizer](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm).

The **default behavior** is to re-instantiate the strategy for each optimization backtest run. However, the process of re-instantiating a strategy requires more time and computer resources to return results, which could impact the amount of time it takes to run an optimization.  When **false**, the strategy is re-used to save time and computer resources.  Under this design, internal properties are reset to default values after each iteration, but it is possible that user-defined properties and other custom resources may carry their state over from the previous iteration into a new backtest run.  To take advantage of performance optimizations, developers may need to reset class level variables in the strategy otherwise unexpected results can occur.

|  |
| --- |
| **Note**:  If you choose to take advantage of the performance benefits during strategy optimization by setting the **IsInstantiatedOnEachOptimizationIteration** property to **false**, any objects you create in your code **MUST** be reset duringthe appropriate**State**within the [OnStateChange()](https://ninjatrader.com/support/helpGuides/nt8/onstatechange.htm)method.  Please see the example below on "*Manually resetting class level variables to take advantage of Strategy Analyzer optimizer performance benefits*". |

**Property Value**

This property returns **true** if the strategy is not recycled; otherwise, **false**. Default set to **true**.

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

**Syntax**

IsInstantiatedOnEachOptimizationIteration

|  |
| --- |
| **Tip**:  The default NinjaTrader indicators and strategies have been optimized to take advantage of performance optimizations as their resources are setup >= **State.Configure**.  Please see the default system indicators and strategies for an idea of how you may improve your strategy and indicator performance, or you may also reference the example code below. |

**Examples**

| ns **Using IsInstantiatedOnEachOptimizationIteration to reset class level variables** |
| --- |
| // A custom trades dictionary is created when strategy is instantiated // since we later set "IsInstantiatedOnEachOptimizationIteration" to true, // we are guaranteed to start with a new object on each optimization run private Dictionary<DateTime, string> myTrades = new Dictionary<DateTime, string>();   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name       = "My Optimization Test 1";     Description = "Demonstrates using IsInstantiatedOnEachOptimizationIteration to reset a class level variable";     Fast       = 10;     Slow       = 25;       // setting to true so our custom trades dictionary is reset on each optimization run (comes with a performance penalty)     // This is the default behavior.     IsInstantiatedOnEachOptimizationIteration = true;   }     else if (State == State.Terminated)   {     // Print the number of trades at the end of the optimization     if (myTrades != null)     {         // if we set "IsInstantiatedOnEachOptimizationIteration" to false (so not using the default of true), the values here would be unexpected         // since the custom trade dictionary was never explicitly reset at the end of each optimization         Print(myTrades.Count);     }   } }   protected override void OnBarUpdate() {   if (CurrentBar < BarsRequiredToTrade)     return;     if (CrossAbove(SMA(Fast), SMA(Slow), 1))   {     EnterLong();     myTrades.Add(Time[0], "long");     }   else if (CrossBelow(SMA(Fast), SMA(Slow), 1))   {     EnterShort();     myTrades.Add(Time[0], "short");   } }   [Range(1, int.MaxValue), NinjaScriptProperty] [Display(Name = "Fast", GroupName = "NinjaScriptStrategyParameters", Order = 0)] public int Fast { get; set; }   [Range(1, int.MaxValue), NinjaScriptProperty] [Display(Name = "Slow", GroupName = "NinjaScriptStrategyParameters", Order = 1)] public int Slow { get; set; } |

| ns **Manually resetting class level variables to take advantage of Strategy Analyzer optimizer performance benefits** | |
| --- | --- |
| // A custom trades dictionary is declared when strategy is first optimized, // but not instantiated until later in State.DataLoaded, private Dictionary<DateTime, string> myTrades;   // examples of other fields which need to be reset private double myDouble; private bool myBool; private DateTime myDateTime; private Order myOrderObject; private Brush myBrushObject; private SMA mySMAIndicator; private Array myIntArray; private List<object> myList; private Series<double> mySeries;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "My Optimization Test 2";     Description = "Demonstrates manually resetting a class level variable without re-instantiating the strategy";     Fast = 10;     Slow = 25;           // in this case, we do not need to re-instantiate the strategy after each optimization     // because we are explicitly resetting the custom trade dictionary in State.DataLoaded     // This design of re-using the strategy instance comes with performance benefits     IsInstantiatedOnEachOptimizationIteration = false;   }     else if (State == State.DataLoaded)   {     // re-create custom trade dictionary on each optimization run     // we are guaranteed to start with a new object on each optimization run     if (myTrades != null)       myTrades.Clear();     else       myTrades = new Dictionary<DateTime, string>();           //Any strategy defaults which are maintained do not need to be reset if they are not mutable as the strategy runs.     //Any strategy state that would be mutable after State.SetDefaults needed to be reset for the next run.     myDouble = double.MinValue;     myBool = false;     myDateTime = DateTime.MinValue;     myOrderObject = null;     myBrushObject = null;     mySMAIndicator = SMA(14);           if (myIntArray != null)         Array.Clear(myIntArray, 0, myIntArray.Length);     else         myIntArray = new int[20];           if (myList != null)         myList.Clear();     else         myList = new List<object>();           mySeries = new Series<double>(this);   } }   protected override void OnBarUpdate() {   if (CurrentBar < BarsRequiredToTrade)     return;     if (CrossAbove(SMA(Fast), SMA(Slow), 1))   {     EnterLong();     myTrades.Add(Time[0], "long");     }   else if (CrossBelow(SMA(Fast), SMA(Slow), 1))   {     EnterShort();     myTrades.Add(Time[0], "short");   } }   [Range(1, int.MaxValue), NinjaScriptProperty] [Display(Name = "Fast", GroupName = "NinjaScriptStrategyParameters", Order = 0)] public int Fast { get; set; }   [Range(1, int.MaxValue), NinjaScriptProperty] [Display(Name = "Slow", GroupName = "NinjaScriptStrategyParameters", Order = 1)] public int Slow { get; set; } | |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) >  **Multi-Objective Optimization** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/walk_forward_optimize_a_strate.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/ai-generate.htm) |

Multi-Objective optimization takes standard optimization a step further by allowing you to choose multiple objectives to test for. When results are returned instead of a singlular list of best results ranked from best to least best instead you will be presented a graph. With multiple objective there is no single best result, instead its up to the trader to choose what is the best tradeoff between two objectives. To run a Multi-objective optimization you will need:

•Access to [historical data](https://ninjatrader.com/support/helpGuides/nt8/data_by_provider.htm)

•Custom NinjaScript \*[strategy](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm)

•A thorough understanding of the Strategy Analyzer's backtesting and optimization capabilities

|  |
| --- |
| **Tip**:  There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore. |

|  |
| --- |
| playVideo |

|  |
| --- |
| **Note**: The [IncludeTradeHistoryInBacktest](https://ninjatrader.com/support/helpGuides/nt8/includetradehistoryinbacktest.htm) property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for optimization. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as [SystemPerformance.AllTrades.Count](https://ninjatrader.com/support/helpGuides/nt8/alltrades.htm) that rely on references to **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state. For more information, see the [Working with Historical Trade Data](https://ninjatrader.com/support/helpGuides/nt8/strategyanalyzer_properties_2.htm) page. |

tog_minus        [How to run a Multi-Objective Optimization](javascript:HMToggle('toggle','HowtorunaMulti-ObjectiveOptimization','HowtorunaMulti-ObjectiveOptimization_ICON'))

|  |  |  |
| --- | --- | --- |
| **Start a Multi-Objective Optimization**  To run a **Multi-Objective Optimization**select the **Backtest type** of "**Multi-Objective Optimization"**in the settings panel of the **Strategy Analyzer**.    StrategyAnalyzer_Optimization_MOORun     |  | | --- | | **Note**: When making the selection additional parameters to configure your optimization will be made visible. |     **Setting the Test Range**  You can set the test range of strategy parameters to be tested by left clicking on the triangle to expand the strategies sub parameters.     |  | | --- | | **Note**: If you don't see the triangle make sure that the **Backtest type** is set to "**Multi-Objective Optimization**". |     StrategyAnalyzer_Optimization_Paramters    **Min**.  - The starting value you want to test **Max**.  - The last value to test **Increment** - The increment value (step value) used to increment the starting value by for each subsequent optimization pass    In the image above, the input "Fast" has a starting (initial) value of 10 and an ending value of 30 with an increment of 1. This means that the first value tested will be 10, then 11, then 12 all the way through 30. The input "Slow" has a starting value of 6, ending value of 16 with an increment of 1. Based on these settings, a total of 200 (20 unique values for "Fast" multiplied by 10 unique values for "Slow") backtest iterations will be processed in order to find the optimal combination of input values based on the best optimization fitness. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?multi-objective_optimization.htm#HowtorunaMulti-ObjectiveOptimization)

tog_minus        [Understanding Multi-Objective  properties](javascript:HMToggle('toggle','UnderstandingMulti-Objectiveproperties','UnderstandingMulti-Objectiveproperties_ICON'))

|  |  |  |  |
| --- | --- | --- | --- |
| **Setting Multiple Optimization Fitness**  Apart from the "**Optimize on**" property described below, the properties are identical to the ones found in the Optimization properties window. Please see the "*Understanding optimization properties*" section of the [Optimize a Strategy](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm) page of the Help Guide for more information.    Multi-Objective Optimization is based on the best optimization fitness you select. If you set the property "**Optimize on**" to "Max. net profit", "Max profit factor", and "Min. draw down" the optimizer will seek the optimal input values based on those three optimization fitness objectives.  There are over 10 different optimization criterion you can select and can be customized via NinjaScript.    StrategyAnalyzer_Optimization_MultiObjectiveProperties     |  |  | | --- | --- | | Optimize on... | Sets the optimization fitness to base the optimization results on, left clicking on the field will open the "Edit Optimization Fitness" window where you can enable what optimization fitnesses you want to be tested and to be available for multi-objective analysis. |      |  | | --- | | **Note**: For running Multi-Objective Optimizations the Default optimizer will be used. |     StrategyAnalyzer_Optimization_OptimizationFitnessDialog |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?multi-objective_optimization.htm#UnderstandingMulti-Objectiveproperties)

tog_minus        [Understanding Multi-Objective results](javascript:HMToggle('toggle','UnderstandingMultiobjectiveResults','UnderstandingMultiobjectiveResults_ICON'))

|  |
| --- |
| **Understanding Multi-Objective Results**  **Multi-objective** results are displayed on a graph instead of a grid. The reason we use a graph is with a **multi-objective** problem there is no one best solution and instead you must compare individual tradeoff between two often competing objectives. Please see the image below to the left with some sample data, each optimization has been performed and the results of each test plotted on the graph. We can narrow down our solution further by only showing results that have the best tradeoff between both objectives  known as a Pareto optimal result. In the graph to the right the line drawn connects the 5 single results that are Pareto optimal forming the Paretor frontier. Any result that falls behind the Pareto frontier is discarded leaving us with 5 best tradeoff solutions between the two objectives.  pareto_frontier_graph  **Using the Multi-Objective Graph**  There are two combo box selections to choose the optimization fitness will be graphed. You will be able to choose any optimization fitness that you have enabled in the optimize on field in the optimization strategies. See the multi-objective optimization properties section above for more information.    StrategyAnalyzer_Optimization_Multi-Objective    Left clicking on one of the dots will select that optimization run and NinjaTrader will run a backtest with these strategy parameters to retrieve the detailed trade data for further analysis. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?multi-objective_optimization.htm#UnderstandingMultiobjectiveResults)

|  |  |
| --- | --- |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) >  **Multi-Objective Optimization** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/walk_forward_optimize_a_strate.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/ai-generate.htm) |

Multi-Objective optimization takes standard optimization a step further by allowing you to choose multiple objectives to test for. When results are returned instead of a singlular list of best results ranked from best to least best instead you will be presented a graph. With multiple objective there is no single best result, instead its up to the trader to choose what is the best tradeoff between two objectives. To run a Multi-objective optimization you will need:

•Access to [historical data](https://ninjatrader.com/support/helpGuides/nt8/data_by_provider.htm)

•Custom NinjaScript \*[strategy](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm)

•A thorough understanding of the Strategy Analyzer's backtesting and optimization capabilities

|  |
| --- |
| **Tip**:  There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore. |

|  |
| --- |
| playVideo |

|  |
| --- |
| **Note**: The [IncludeTradeHistoryInBacktest](https://ninjatrader.com/support/helpGuides/nt8/includetradehistoryinbacktest.htm) property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for optimization. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as [SystemPerformance.AllTrades.Count](https://ninjatrader.com/support/helpGuides/nt8/alltrades.htm) that rely on references to **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state. For more information, see the [Working with Historical Trade Data](https://ninjatrader.com/support/helpGuides/nt8/strategyanalyzer_properties_2.htm) page. |

tog_minus        [How to run a Multi-Objective Optimization](javascript:HMToggle('toggle','HowtorunaMulti-ObjectiveOptimization','HowtorunaMulti-ObjectiveOptimization_ICON'))

|  |  |  |
| --- | --- | --- |
| **Start a Multi-Objective Optimization**  To run a **Multi-Objective Optimization**select the **Backtest type** of "**Multi-Objective Optimization"**in the settings panel of the **Strategy Analyzer**.    StrategyAnalyzer_Optimization_MOORun     |  | | --- | | **Note**: When making the selection additional parameters to configure your optimization will be made visible. |     **Setting the Test Range**  You can set the test range of strategy parameters to be tested by left clicking on the triangle to expand the strategies sub parameters.     |  | | --- | | **Note**: If you don't see the triangle make sure that the **Backtest type** is set to "**Multi-Objective Optimization**". |     StrategyAnalyzer_Optimization_Paramters    **Min**.  - The starting value you want to test **Max**.  - The last value to test **Increment** - The increment value (step value) used to increment the starting value by for each subsequent optimization pass    In the image above, the input "Fast" has a starting (initial) value of 10 and an ending value of 30 with an increment of 1. This means that the first value tested will be 10, then 11, then 12 all the way through 30. The input "Slow" has a starting value of 6, ending value of 16 with an increment of 1. Based on these settings, a total of 200 (20 unique values for "Fast" multiplied by 10 unique values for "Slow") backtest iterations will be processed in order to find the optimal combination of input values based on the best optimization fitness. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?multi-objective_optimization.htm#HowtorunaMulti-ObjectiveOptimization)

tog_minus        [Understanding Multi-Objective  properties](javascript:HMToggle('toggle','UnderstandingMulti-Objectiveproperties','UnderstandingMulti-Objectiveproperties_ICON'))

|  |  |  |  |
| --- | --- | --- | --- |
| **Setting Multiple Optimization Fitness**  Apart from the "**Optimize on**" property described below, the properties are identical to the ones found in the Optimization properties window. Please see the "*Understanding optimization properties*" section of the [Optimize a Strategy](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm) page of the Help Guide for more information.    Multi-Objective Optimization is based on the best optimization fitness you select. If you set the property "**Optimize on**" to "Max. net profit", "Max profit factor", and "Min. draw down" the optimizer will seek the optimal input values based on those three optimization fitness objectives.  There are over 10 different optimization criterion you can select and can be customized via NinjaScript.    StrategyAnalyzer_Optimization_MultiObjectiveProperties     |  |  | | --- | --- | | Optimize on... | Sets the optimization fitness to base the optimization results on, left clicking on the field will open the "Edit Optimization Fitness" window where you can enable what optimization fitnesses you want to be tested and to be available for multi-objective analysis. |      |  | | --- | | **Note**: For running Multi-Objective Optimizations the Default optimizer will be used. |     StrategyAnalyzer_Optimization_OptimizationFitnessDialog |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?multi-objective_optimization.htm#UnderstandingMulti-Objectiveproperties)

tog_minus        [Understanding Multi-Objective results](javascript:HMToggle('toggle','UnderstandingMultiobjectiveResults','UnderstandingMultiobjectiveResults_ICON'))

|  |
| --- |
| **Understanding Multi-Objective Results**  **Multi-objective** results are displayed on a graph instead of a grid. The reason we use a graph is with a **multi-objective** problem there is no one best solution and instead you must compare individual tradeoff between two often competing objectives. Please see the image below to the left with some sample data, each optimization has been performed and the results of each test plotted on the graph. We can narrow down our solution further by only showing results that have the best tradeoff between both objectives  known as a Pareto optimal result. In the graph to the right the line drawn connects the 5 single results that are Pareto optimal forming the Paretor frontier. Any result that falls behind the Pareto frontier is discarded leaving us with 5 best tradeoff solutions between the two objectives.  pareto_frontier_graph  **Using the Multi-Objective Graph**  There are two combo box selections to choose the optimization fitness will be graphed. You will be able to choose any optimization fitness that you have enabled in the optimize on field in the optimization strategies. See the multi-objective optimization properties section above for more information.    StrategyAnalyzer_Optimization_Multi-Objective    Left clicking on one of the dots will select that optimization run and NinjaTrader will run a backtest with these strategy parameters to retrieve the detailed trade data for further analysis. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?multi-objective_optimization.htm#UnderstandingMultiobjectiveResults)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

|  |  |
| --- | --- |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) >  **Optimization** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/backtest_a_strategy.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/genetic_algorithm.htm) |

You can fine tune the input parameters of a strategy through optimization. Optimization is the process of testing a range of values through iterative backtests to determine the optimal input values over the historical test period based on your optimization fitness. To run an optimization you will need:

•Access to [historical data](https://ninjatrader.com/support/helpGuides/nt8/data_by_provider.htm)

•Custom NinjaScript \*[strategy](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm)

•A thorough understanding of the Strategy Analyzer's [backtesting](https://ninjatrader.com/support/helpGuides/nt8/backtest_a_strategy.htm) capabilities

|  |
| --- |
| **Tip**:  There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore. |

|  |
| --- |
| playVideo |

tog_minus        [How to run an Optimization](javascript:HMToggle('toggle','HowToRunAnOptimization','HowToRunAnOptimization_ICON'))

|  |  |
| --- | --- |
| **Running an Optimization**  To run an **Optimization**select the **Backtest type** of "**Optimization"**in the settings panel of the **Strategy Analyzer**.    StrategyAnalyzer_OptimizationRiun     |  | | --- | | **Note**: When making the selection additional parameters to configure your optimization will be made visible. |     **Setting the Test Range**  You can the test range of strategy parameters to be tested by left clicking on the triangle to expand the strategies sub parameters.    Note: If you don't see the triangle make sure that the **Backtest type** is set to "**Optimization**".    StrategyAnalyzer_Optimization_Paramters    **Min**.  - The starting value you want to test **Max**.  - The last value to test **Increment** - The increment value (step value) used to increment the starting value by for each subsequent optimization pass    In the image above, the input "Fast" has a starting (initial) value of 10 and an ending value of 30 with an increment of 1. This means that the first value tested will be 10, then 11, then 12 all the way through 30. The input "Slow" has a starting value of 6, ending value of 16 with an increment of 1. Based on these settings, a total of 200 (20 unique values for "Fast" multiplied by 10 unique values for "Slow") backtest iterations will be processed in order to find the optimal combination of input values based on the best optimization fitness.    **Setting the Optimization Fitness**  Optimization is based on the best optimization fitness you select. If you set the property "Optimize on..." to "Max. net profit", the optimizer will seek the optimal input values that return the maximum profit possible. There are over 10 different optimization criterion you can select and can be customized via NinjaScript. Please see the "*Understanding Optimization properties*" section below for more information. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?optimize_a_strategy.htm#HowToRunAnOptimization)

tog_minus        [Understanding optimization properties](javascript:HMToggle('toggle','UnderstandingOptimizationProperties','UnderstandingOptimizationProperties_ICON'))

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Optimization Properties**  Apart from the optimization specific properties described below, the properties are identical to the ones found in the backtest properties window. Please see the "*Understanding backtest properties*" section of the[Backtest a Strategy](https://ninjatrader.com/support/helpGuides/nt8/backtest_a_strategy.htm) page of the Help Guide for more information.    The following Optimization specific properties are available:    StrategyAnalyzer_Optimization_OptimizationParamaters       |  | | --- | | **Tip**:  You can optionally "**Optimize on**" multiple objectives by using a [Multi-Objective optimization](https://ninjatrader.com/support/helpGuides/nt8/multi-objective_optimization.htm) |      |  |  | | --- | --- | | Keep best # results | Sets the number of best results to display | | Optimize data series | If set to true, the Data Series Value property will be available for optimization (Not supported for Kagi, Point and Figure, Line Break and Heiken Ashi period Types) | | Optimize on... | Sets the optimization fitness to base the optimization results on | | Optimizer | Sets the optimization algorithm that is used. NinjaTrader comes with "Default" and "[Genetic](https://ninjatrader.com/support/helpGuides/nt8/genetic_algorithm.htm)" optimizer algorithms. When the "Genetic" option is selected, the genetic algorithm's optimization properties fields will appear below the Optimizer selection  You can program your own [optimization algorithm](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) using NinjaScript. | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?optimize_a_strategy.htm#UnderstandingOptimizationProperties)

tog_minus        [Understanding optimization results](javascript:HMToggle('toggle','UnderstandingOptimizationResults','UnderstandingOptimizationResults_ICON'))

|  |
| --- |
| **Understanding Optimization Results**  Once the optimization process is complete, you will see a the Optimization Results Grid appear in the Analyzer tab. The results will be grouped per instrument and shows the parameter combination that achieved the highest performance. The "Performance" column is dynamic and will always be the Optimization Fitness that you selected for the "**Optimize**" parameter when you ran the optimization.    StrategyAnalyzer_Optimization_BestResults1    **The Top Optimization Results**  The Optimizer tab will display the top number of results based on the value you set for the "Keep best # results" property in the Optimizer dialog window. The column Parameters displays the optimized input values.    StrategyAnalyzer_Optimization_BestResults    1.The optimal value for the "Fast" input for the demonstration strategy used for this optimization  2.The optimal value for the "Slow" input for the demonstration strategy used for this optimization |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?optimize_a_strategy.htm#UnderstandingOptimizationResults)

tog_minus        [Running a basket test](javascript:HMToggle('toggle','RunningABasketTest','RunningABasketTest_ICON'))

|  |  |  |
| --- | --- | --- |
| **Basket test**  Running an optimization across a list of instruments works very much the same as running a regular basket backtest.  For general information, please refer to the [Basket testing multiple instruments page.](https://ninjatrader.com/support/helpGuides/nt8/basket_test.htm)  However, when running an optimization across multiple instruments, an optional "Aggregated" option will be available.      StrategyAnalyzer_Optimization_Aggregated     |  |  | | --- | --- | | Aggregated | If set to True, NinjaTrader attempts to find the optimal results for the whole basket of instruments. The COMBINED row in the results tab will show an aggregation of results across the basket of instruments. (This parameter is only available when an [Instrument List](https://ninjatrader.com/support/helpGuides/nt8/instrument_lists.htm) is selected for optimization.) | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?optimize_a_strategy.htm#RunningABasketTest)

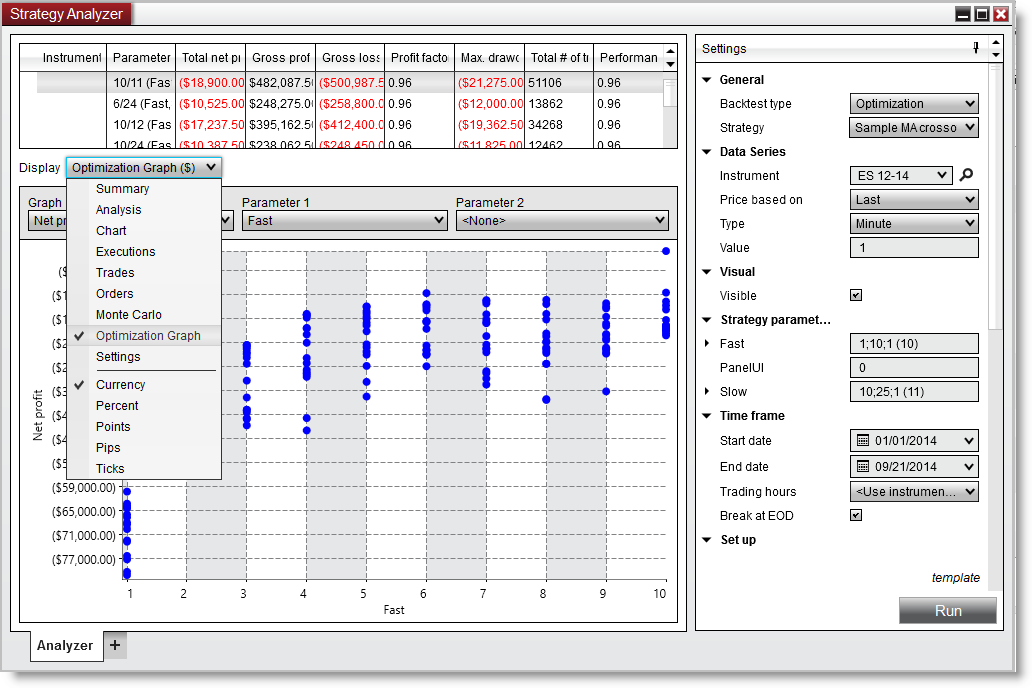
tog_minus        [Understanding factors that affect optimization performance](javascript:HMToggle('toggle','UnderstandingFactorsThatAffectOptimizationPerformance','UnderstandingFactorsThatAffectOptimizationPerformance_ICON'))

|  |
| --- |
| **32 bit vs 64 bit**  When you run an optimization in the**32 bit** version of NinjaTrader to consume less memory we do not store any trade data for each backtest that is run. Therefore if you want to do trade analysis on one of the backtest results returned from an optimization NinjaTrader must re-run the backtest to get the trade data, this adds a small delay when switching between tests. The**64 bit** version of NinjaTrader will take advantage of the extra RAM available to NinjaTrader and will keep the trade results for each kept backtest, allowing you to quickly change between backtest result reports.    **Keep best # results**  If you are finding that you are running low on system memory during your backtests reduce this number of results to keep will make a significant improvement to the memory used by NinjaTrader.    **Running multiple tests at a time**  You will not get more done in a smaller time frame by separating multiple tests out manually and running them at the same time on the same PC. NinjaTrader will efficiently use all CPU cores for any optimization for fastest possible testing.    **CPU Resources**  Please insure that you have as much system resources available to the optimization as possible, this usually means making sure all other applications are closed. Furthermore as as the NinjaTrader optimization engine is optimized to take advantage of as much system resources as possible it is advisible not to trigger an optimization during a time where you would need to be using the PC. For example it is not advised to start an optimization while you are  managing the exit of a trade.    **Historical Trade Data**  The [IncludeTradeHistoryInBacktest](https://ninjatrader.com/support/helpGuides/nt8/includetradehistoryinbacktest.htm) property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for optimization. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as [SystemPerformance.AllTrades.Count](https://ninjatrader.com/support/helpGuides/nt8/alltrades.htm) that rely on references to **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state, but this can result in greater memory usage. For more information, see the [Working with Historical Trade Data](https://ninjatrader.com/support/helpGuides/nt8/strategyanalyzer_properties_2.htm) page.    **Running Efficient Optimizations**  Strategy optimizations are expected to consume a good deal of CPU resources, simply due to the nature of the iterative data processing they perform. Strategies with a relatively large number of parameters for optimization can multiply this impact. When working with strategies with a large number of parameters, avoid using "1" as the increment value for the optimizer, to avoid forcing the optimizer algorithm to run the maximum number of permutations. Changing the increment value to as little as "2" can cut the number of permutations in half, and increasing this value can have progressively less of an impact.    The [Genetic Algorithm](https://ninjatrader.com/support/helpGuides/nt8/genetic_algorithm.htm) can offer an alternative solution to increasing parameter increment values. Rather than running brute-force tests by iterating over all permutations, the Genetic Algorithm intentionally ignores parameter combinations which are likely to produce sub-optimal results.    Especially on larger parameter sets with finer increment values, the upper limits of potential permutations / parameter combinations could be reached for both approaches to optimization - the error message *"The strategy needs at least one parameter to optimize"* would be then an indication to rework the # of parameters or increase the increment values to reach a more meaningful permutation count.    **Using a virtual / cloud server**  If you are using a virtual or cloud server as basis for your setup when running optimization testing in the Strategy Analyzer, please keep in mind that such environments can typically allocate available resources on demand. NinjaTrader will still take advantage of all available threads for it's processing, however those resources available would be determined at the start-up of the NinjaTrader platform. So if your virtual resources would have changed while you were in a working session, then please restart fresh to ensure performance will be optimal. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?optimize_a_strategy.htm#UnderstandingFactorsThatAffectOptimizationPerformance)

|  |  |
| --- | --- |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) >  **Optimization Graphs** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/running_a_monte_carlo_simulati.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/discrepancies_real-time_vs_bac.htm) |

The **Optimization Graph** can only be selected in the **Display** selector only after an optimization has been run. The optimization graph can be displayed in a 2D or 3D graph. A 2D graph is used when only graphing a single parameter. If you graph 2 parameters then a 3D graph is displayed.



tog_minus        [Understanding the 2D optimization graph](javascript:HMToggle('toggle','UnderstandingThe2dOptimizationGraph','UnderstandingThe2dOptimizationGraph_ICON'))

|  |  |
| --- | --- |
| **Understanding the 2D Optimization Graph**  The 2D optimization graph displays each and every test run for the optimization. This allows you to see the entire range of results produced from an optimization run. Allowing you to take a look over the entire solution domain to determine if your top results are stable. Instead of choosing the absolute best parameter set that might be an outlier you may instead desire to choose a parameter that has a gradual build up which may indicate stability in the result set.    The 2D Optimization graph will be displayed when you have only selected a single parameter and is the default graph view.     |  | | --- | | **Note**: Selecting a 2nd Parameter will switch to the 3d graph. |     **Using the 2D Optimization Graph**  Each dot signifies a backtest result, graphed by the X-Axis and the Y-Axis. The X-Axis can be changed by selecting the **Graph** parameter. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?2d__3d_optimization_graphs.htm#UnderstandingThe2dOptimizationGraph)

tog_minus        [Understanding the 3D optimization graph](javascript:HMToggle('toggle','UnderstandingThe3dOptimizationGraph','UnderstandingThe3dOptimizationGraph_ICON'))

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| StrategyAnalyzer_OptimizationGraph3d    **Understanding the 3D Optimization Graph**  The 3D optimization graph expands upon the 2D optimization graph by allowing an additional axis to place an additional parameter. You must have at least 2 parameters being optimized and with the 'Parameter 2' combo box select the secondary parameter. This will trigger the display of the 3D optimization graph. Select 'None' to return to the 2D optimization graph.    **Using the 3D Optimization Graph**  Using the following mouse controls you can interact with the 3D optimization graph.     |  |  | | --- | --- | | Pan | Press the Middle Mouse Button to pan the graph | | Orbit | Pres the Left Mouse Button to rotate / orbit the graph | | Zoom | Use the Scroll Wheel to zoom in / out | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?2d__3d_optimization_graphs.htm#UnderstandingThe3dOptimizationGraph)

|  |  |
| --- | --- |
| **Navigation:**  »No topics above this level«  **Video Library** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/welcome.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/welcome.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/release_notes.htm) |

Within the Help Guide are numerous videos providing a step by step tour through the NinjaTrader Platform. Select your area of interest below to view an expanded list of all available topics within each category.

tog_minus        [Order Entry](javascript:HMToggle('toggle','OrderEntry','OrderEntry_ICON'))

|  |
| --- |
| **Trade Controls Overview**  The Trade Controls Overview video provides a walkthrough of various trade management features which can be accessed in a variety of order-entry windows. Several order-entry windows are shown in the video to show the commonalities in how trade controls operate in NinjaTrader.        **Chart Trader Overview**  Chart Trader allows orders and positions to be entered and managed directly within a chart window. Advanced Trade Management strategies can be employed directly in the Chart Trader window. Orders and positions on multiple instruments can be managed within a single chart window, as well. This video covers the basics of enabling and using Chart Trader, including a visual method of placing Limit and Stop orders.        **SuperDOM Order Submission Overview**  This video covers submitting new orders in the SuperDOM.        **SuperDOM Position Management Overview**  The SuperDOM Position Management Overview video covers scaling in, scaling out, and closing positions directly in the SuperDOM.        **SuperDOM Order Modification Overview**  This video shows the ways in which resting orders can be modified directly in the SuperDOM.        **Attach to Indicator Overview**  The Attach to Indicator feature allows resting orders to be attached to indicator plots, automatically updating an order's price as the indicator value changes. This feature can be used to partially automate entries, exits, stop losses, and profit targets. The Attach to Indicator Overview video provides working examples of using this feature in Chart Trader and the SuperDOM.        **Overview of the Basic Entry and FX Pro Windows**  The Basic Entry and FX Pro windows conveniently group order-entry and market analysis features into compact windows which function in similar ways. This video explores the layout and basic features of both window. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#OrderEntry)

tog_minus        [Advanced Trade Management (ATM) Strategies](javascript:HMToggle('toggle','AdvancedTradeManagementStrategies','AdvancedTradeManagementStrategies_ICON'))

|  |
| --- |
| **Advanced Trade Management Overview**  Advanced Trade Management Strategies, also referred to as ATM Strategies, provide a layer of discretionary automation to manage a position's exit orders without the need to make continual manual modifications. The Advanced Trade Management Overview video introduces and defines ATM Strategies, while demonstrating a simple ATM setup.        **ATM Stop Strategies**  ATM Stop Strategies provide additional functionality for the stop losses placed by an ATM Strategy, including auto-breakeven, auto-trail, and **Simulated Stop** orders.        **ATM Additional Options and Strategy Selection Modes**  Additional options for ATM Strategies include Auto-Reverse and Auto-Chase features, and the ability to specific the order type used for profit targets and stop losses. ATM Strategy Selection Modes determine the behavior of the ATM Strategy Control List after placing an order.        **Advanced Trade Management Examples**  In the final Advanced Trade Management video, several real-world examples are created and saved as templates for later use, showing many core ATM features in use in a live market. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#AdvancedTradeManagementStrategies)

tog_minus        [SuperDOM](javascript:HMToggle('toggle','SuperDOM','SuperDOM_ICON'))

|  |
| --- |
| **SuperDOM Display Overview**  The SuperDOM displays five levels of market depth on a price ladder, and allows for the entry and management of orders and positions, as well as the use of Advanced Trade Management Strategies. The SuperDOM Display Overview video covers the layout and basic functionality of the SuperDOM.        **Static vs. Dynamic Price Ladder**  Two versions of the SuperDOM are available: Static and Dynamic. This video covers the difference between the two.          **Working with Indicators on the SuperDOM**  The Working with Indicators video provides an overview of configuring technical indicators on the SuperDOM. Just like charts, the SuperDOM can display a wide range of price- and volume-based indicators, and allows resting orders to be attached to indicator plots moving in real-time.          **SuperDOM Columns Overview**  The SuperDOM Columns Overview shows how additional columns can be added to a SuperDOM window to display profit and loss, volume, notes, or any information configured in a custom column created via NinjaScript. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#SuperDOM)

tog_minus        [Control Center](javascript:HMToggle('toggle','ControlCenter','ControlCenter_ICON'))

|  |
| --- |
| **Control Center Overview**  The Control Center acts as the primary window in NinjaTrader, providing access to all trading windows, performance reporting, and other features of the platform. This video provides an overview of the Control Center's layout and menus.        **Control Center Tabbed Display and Account Data**  The Control Center's tabbed layout provides quick access to Orders, Positions, Accounts, Strategies, and Executions. The Control Center Tabbed Display and Account Data video covers navigating Control Center tabs, as well as managing and editing connected brokerage accounts. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#ControlCenter)

tog_minus        [Market Analyzer](javascript:HMToggle('toggle','MarketAnalyzer','MarketAnalyzer_ICON'))

|  |
| --- |
| **Market Analyzer Display Overview**  The Market Analyzer is NinjaTrader's answer to the traditional quote sheet, adding a wide range of functionality to extend the features of traditional quote sheets, such as the ability to view indicator values, create alerts, and link to charts and order-entry windows for instant instrument switching. The Market Analyzer Display video covers these features in detail.        **Market Analyzer Columns and Indicators**  The Market Analyzer can be configured with a wide range of pre-built and custom columns and indicators. This video demonstrates applying and configuring these items. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#MarketAnalyzer)

tog_minus        [Alerts](javascript:HMToggle('toggle','Alerts','Alerts_ICON'))

|  |
| --- |
| **Overview of Alerts**  Alerts can be configured on Charts or Market Analyzer windows, allowing you to set custom actions to take when predefined conditions are met in the market, including automatically placing orders or sharing messages via social media. This video covers configuring and testing alerts using the Simulated Data Feed.        **Alerts Examples**  In this video, a few real-world examples of alerts are set up in a Chart and Market Analyzer to show the Alerts feature in action. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#Alerts)

tog_minus        [Charts](javascript:HMToggle('toggle','Charts','Charts_ICON'))

|  |
| --- |
| **Creating Charts Overview**  NinjaTrader charts feature a wide range of advanced features and options, which are covered in several videos. The Creating Charts Overview provides a walkthrough of creating a new chart or duplicating an existing chart.        **Navigating Charts Overview**  The Navigating Charts Overview video picks up where the Creating Charts video left off, showing you how to manage instruments, navigate chart windows, and manipulate the viewable area of charts.        **Working With Indicators on Charts**  Technical indicators plot mathematical derivatives of price action graphical on charts. Over 100 indicators come preloaded with NinjaTrader and can be applied right away. Additionally, custom indicators can be developed via NinjaScript or obtained through third-party vendors for an even greater array of indicator selections.        **Chart Panels and Objects Overview**  Charts can contain numerous objects, including bars series, indicator plots, **Drawing Objects**, and execution plots. The Chart Panels and Objects Overview video shows how to manage, drag and drop, or copy and paste chart objects.        **Chart Drawing Objects**  **Drawing Objects** allow you to mark any area of a chart panel in a variety of ways. Numerous **Drawing Objects** are available for use right away, including several Fibonacci tools, and additional **Drawing Objects** can be created via NinjaScript or obtained from third-party vendors.        **Working with Price Data on Charts**  Charts allow you to view price data in a wide variety of formats, including different Chart Styles, Bar Types, and intervals. This video provides an overview of setting up price data to your liking on a chart. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#Charts)

tog_minus        [Market Data Windows](javascript:HMToggle('toggle','MarketDataWindows','MarketDataWindows_ICON'))

|  |
| --- |
| **FX Board Display**  The FX Board allows forex traders to view a wide range of forex instruments at a glance, using an advanced interface to quick enter, exit, and manage trades on numerous instruments from within a single window. The FX Board Display video covers the layout and primary functions of the FX Board.        **Level II Window Overview**  The Level II window presents a complete view of market depth events for an instrument, displaying all 10 levels of depth, including the price, size, volume, and spread of each order. This video covers the basics of opening, populating, and reading the Level II window.        **Time & Sales Window Overview**  The Time & Sales window can be used to view granular details about all orders being filled at the exchange for a particular instrument. This video provides an overview of the layout and basic operation of the Time & Sales window. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#MarketDataWindows)

tog_minus        [Miscellaneous](javascript:HMToggle('toggle','Miscellaneous','Miscellaneous_ICON'))

|  |
| --- |
| **Window Management Overview**  NinjaTrader windows include common features to increase workflow and workspace efficiency. The Window Management Overview video covers such topics as creating and editing tabs within windows, and duplicating existing content in new windows.        **Hot List Analyzer Overview**  The Hot List Analyzer screens equity instruments based on a variety of criteria with over 30 filters. For example, this window can be used to spot the most active, highest gaining, or highest losing stocks of the day on an individual exchange. This video shows how to set up and populate the Hot List Analyzer.        **Share and Print Overview**  Sharing content such as positions and chart screenshots is an integral feature in NinjaTrader. The Share and Print Overview video covers linking social media accounts to NinjaTrader and sharing content from a variety of windows within the platform.        **Trade Performance Overview**  The Trade Performance window provides robust reporting on the performance of completed trades, including a number of graphs covering popular performance metrics. The Trade Performance Overview video provides a high-level overview of using the Trade Performance window and it's various filters to view meaningful performance reports.        **Playback Connection**  The Playback connection allows you to use Market Replay data, or historical data, to play back market action from previous days. This video provides an overview of downloading Market Replay data, setting up the Playback connection, and playing data back at different speeds.        **Strategy Builder Overview**  The Strategy Builder is used to generate NinjaScript based strategies for automated systems trading. This video provides an overview of creating a strategy. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#Miscellaneous)

tog_minus        [Strategy Backtesting and Optimization](javascript:HMToggle('toggle','StrategyBacktestingAndOptimization','StrategyBacktestingAndOptimization_ICON'))

|  |
| --- |
| **Strategy Analyzer Overview**  NinjaScript strategies can be backtested and optimized to test theoretical strategy performance on historical data within the Strategy Analyzer. The Strategy Analyzer Overview covers the basic layout of the Strategy Analyzer, and individual test types will be covered in greater detail in future videos.        **Backtesting Strategies**  The Backtesting Strategies video walks through the process of configuring, running, and analyzing the results of a standard strategy backtest in the Strategy Analyzer. All configurable backtest properties are covered in this video.        **Optimizing Strategies**  Strategy optimizations allow you to iterate over a pre-defined range of strategy input values to determine the combination of property values which score highest on a chosen performance metric. The Optimizing Strategies video covers the aspects of optimizations which differ from standard backtests.        **Understanding Walk-Forward Optimization**  Walk-Forward Optimization combines the features of optimizations and standard backtests. This Backtest Type performs an optimization over a pre-defined date range, then applies the optimal parameter combinations to a standard backtest over another pre-defined date range.        **Understanding Multi-Objective Optimization**  Multi-Objective Optimization uses Pareto Analysis to find a set of possible input-value combinations which score higher or lower on individual metrics (of which there can be many), but for which there are no obviously superior alternatives on all metrics tested. This video introduces the goals of Multi-Objective Optimization and explains the concept of the Pareto Frontier.        **Understanding the Genetic Algorithm**  The Genetic Algorithm is an optional optimization engine which leverages evolutionary theory to find optimal combinations of strategy input parameters through multi-generational crossover and mutation, focusing on the fittest individuals in each generation. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?video_library.htm#StrategyBacktestingAndOptimization)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/support/helpGuides/nt8/common.htm) > [Attributes](https://ninjatrader.com/support/helpGuides/nt8/attributes.htm) >  **NinjaScriptPropertyAttribute** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/displayattribute.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/attributes.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/rangeattribute.htm) |

**Definition**

Determines if the following declared property should be included in the NinjaScript object's constructor as a parameter.  This is useful if you plan on calling a NinjaScript object from another (e.g., calling a custom indicator from a strategy) or customizing the display parameter data on a grid or from a chart. This also used to make parameters [optimizable](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm) in the Strategy Analyzer.

|  |
| --- |
| **Warning**:  Only types which can be[Xml Serialized](https://ninjatrader.com/support/helpGuides/nt8/xmlignoreattribute.htm) should be marked as a **NinjaScriptAttribute**, otherwise you may run into errors when persisting values in various scenarios (e.g., saving workspace, or running [Strategy Optimizations](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm)).  Should you have a property you wish to use as user defined input, you will need to implement a secondary simple type (such as an int or string) as the value to be serialized as user input. Please see the example below which demonstrates using a simple type as the **NinjaScriptProperty** against types which cannot be serialized |

**Syntax**

[NinjaScriptProperty]

**Parameters**

This object contains no parameters

**Examples**

| ns **Basic usage of NinjaScriptProperty** |
| --- |
| #region Properties         // set NinjaScriptProperty to ensure this property is used when calling from another object [NinjaScriptProperty] public bool MyBool   { get; set; }   // do not set NinjaScriptProperty since this property is not required to call // nor do we wish to display it on the chart label public int MyInt { get; set; }   #endregion |

| ns **Using a simple type as the NinjaScriptProperty against types which cannot be serialized** | |
| --- | --- |
| [XmlIgnore] // cannot serialize type of TimeSpan, use the BeginTimeSpanSerialize object to persist properties       [Browsable(false)] // prevents this property from showing up on the UI public TimeSpan BeginTimeSpan { get; set; }   // users will configure this "string" as the TimeSpan which will be set as a TimeSpan object used in data processing [NinjaScriptProperty] [Display(Name = "Begin TimeSpan", GroupName = "NinjaScriptStrategyParameters", Order = 1)] public string BeginTimeSpanSerialize {   get { return BeginTimeSpan.ToString(); }   set { BeginTimeSpan = TimeSpan.Parse(value); } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Educational Resources](https://ninjatrader.com/support/helpGuides/nt8/educational_resources.htm) > [Developing Strategies](https://ninjatrader.com/support/helpGuides/nt8/developing_strategies.htm) > [Intermediate - RSI with Stop Loss & Profit Target](https://ninjatrader.com/support/helpGuides/nt8/intermediate_-_rsi_with_stop_l.htm) >  **Entering Strategy Logic** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/set_up11.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/intermediate_-_rsi_with_stop_l.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/compiling8.htm) |

**Using the OnStateChange() Method to Configure the Strategy**

The [OnStateChange()](https://ninjatrader.com/support/helpGuides/nt8/onstatechange.htm) method is called once prior to running a strategy and can be used to set properties or call methods in preparation for running a strategy.

Enter the code contained within the OnStateChange() method in the image below into the OnStateChange() method when we are in the State.DataLoaded state in the NinjaScript Editor.

| ns |
| --- |
| protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Description                               = @"RSI with a Stop Loss and Profit Target";     Name                                     = "RSIwithStopAndTarget";     Calculate                                 = Calculate.OnBarClose;     EntriesPerDirection                       = 1;     EntryHandling                             = EntryHandling.AllEntries;     IsExitOnSessionCloseStrategy             = true;     ExitOnSessionCloseSeconds                 = 30;     IsFillLimitOnTouch                       = false;     MaximumBarsLookBack                       = MaximumBarsLookBack.TwoHundredFiftySix;     OrderFillResolution                       = OrderFillResolution.Standard;     Slippage                                 = 0;     StartBehavior                             = StartBehavior.WaitUntilFlat;     TimeInForce                               = TimeInForce.Gtc;     TraceOrders                               = false;     RealtimeErrorHandling                     = RealtimeErrorHandling.StopCancelClose;     StopTargetHandling                       = StopTargetHandling.PerEntryExecution;     BarsRequiredToTrade                       = 20;     // Disable this property for performance gains in Strategy Analyzer optimizations     // See the Help Guide for additional information     IsInstantiatedOnEachOptimizationIteration = true;     RSIPeriod                                 = 14;     RSISmooth                                 = 3;     ProfitTarget                             = 12;     StopLoss                                 = 6;   }   else if (State == State.DataLoaded)   {     AddChartIndicator(RSI(RSIPeriod, RSISmooth));           SetStopLoss(CalculationMode.Ticks, StopLoss);     SetProfitTarget(CalculationMode.Ticks, ProfitTarget);   } } |

For more information on the strategy properties added in State.SetDefaults, please see our complete [Strategy](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm) documentation.

The [AddChartIndicator()](https://ninjatrader.com/support/helpGuides/nt8/addchartindicator.htm) method is called and the RSI() indicator method is passed in which will automatically plot this indicator on a chart when the strategy runs.

The method signature for the RSI() indicator is:

| ns |
| --- |
| RSI(int *period*, int *smooth*); |

It is valid to have used the Add() method in the following manner:

| ns |
| --- |
| AddChartIndicator(RSI(14, 3)); |

However, instead of hard coding the period value to 14 and the smooth value to 3, we substituted the values for the user defined inputs we defined in the wizard. Calling the Add() method in the following manner:

| ns |
| --- |
| AddChartIndicator(RSI(RSIPeriod, RSISmooth)); |

Allows us to change the period and smooth parameters of the embedded RSI indicator in the strategy at run time. This gives us a higher level of flexibility when working with our strategy.

[SetStopLoss()](https://ninjatrader.com/support/helpGuides/nt8/setstoploss.htm) and [SetProfitTarget()](https://ninjatrader.com/support/helpGuides/nt8/setprofittarget.htm) are called with CalculationMode.Ticks. This means that when a position is opened, the strategy will immediately submit a stop and target order with a price that is calculated based on the StopLoss and ProfitTarget parameters passed in offset from the positions average entry price.

**Using the OnBarUpdate() Method for the Core Strategy Logic**

The OnBarUpdate() method is called for each incoming tick or on the close of a bar (user defined) when performing real-time calculations. Therefore, this is the main method called for strategy calculation and we will use this method to enter the script that check for entry and exit conditions.

Enter the code contained within the OnBarUpdate() method in the image below into the OnBarUpdate() method in the NinjaScript Editor:

| ns |
| --- |
| protected override void OnBarUpdate() {   if (CurrentBar < RSIPeriod)     return;     if(CrossAbove(RSI(RSIPeriod, RSISmooth), 20, 1))     EnterLong(); } |

Since our strategy exit logic has already been set up in the OnStateChange() method above, we only need to focus on expressing our entry logic. The strategy entry logic is very straight forward and can be translated to English:

*if we have not seen the number of bars specified by the user defined input "RSIPeriod" then do not go any further*

*if RSI crosses****above****a value of 20 within the last bar, go long*

To accomplish this we used the following methods and properties:

[CurrentBar](https://ninjatrader.com/support/helpGuides/nt8/currentbar.htm) - A value representing the current bar being processed (think of a chart where the left most bar would be equal to one)  
[CrossAbove()](https://ninjatrader.com/support/helpGuides/nt8/crossabove.htm) - Checks for a cross above condition and returns true or false  
[RSI()](https://ninjatrader.com/support/helpGuides/nt8/relative_strength_index_rsi.htm) - Returns the value of the RSI indicator  
[EnterLong()](https://ninjatrader.com/support/helpGuides/nt8/enterlong.htm) - Enters a market order long

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Optimizer](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) >  **OnOptimize()** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/numberofiterations.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/optimizationparameters.htm) |

**Definition**

This method must be overridden in order to optimize a strategy. This method is called once per optimization run (not once per iteration).

**Method Return Value**

This method does not return a value.

**Syntax**  
You must override the method in your Optimizer with the following syntax.

**protected override void OnOptimize()**  
**{**

**}**

**Examples**

| ns | |
| --- | --- |
| protected override void OnOptimize()  {      // If there is no optimization objective, return      if (Strategies[0].OptimizationParameters.Count == 0)          return;        // Optimizer logic  } | |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) >  **Anchored Walk Forward Optimization** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/walk_forward_optimize_a_strate.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/multi-objective_optimization.htm) |

Anchored Walk Forward optimization is similar to the Walk Forward Optimization. However, each optimization is run from the start time then increases the end date.

Walk Forward optimization is the process by which you optimize strategy input parameters on a historical segment of market data, then test the strategy forward in time on data following the optimization segment using the optimized input values. The central idea is that you evaluate strategy performance data on the test data, not the data used in the optimization. This process is then repeated by moving the optimization and test segments forward in time. To run a walk forward optimization you will need:

•Access to [historical data](https://ninjatrader.com/support/helpGuides/nt8/data_by_provider.htm)

•Custom NinjaScript \*[strategy](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm)

•A thorough understanding of the Strategy Analyzer's backtesting and optimization capabilities

|  |
| --- |
| **Tip**:  There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore. |

|  |
| --- |
| **Note**: The [IncludeTradeHistoryInBacktest](https://ninjatrader.com/support/helpGuides/nt8/includetradehistoryinbacktest.htm) property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for optimization. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as [SystemPerformance.AllTrades.Count](https://ninjatrader.com/support/helpGuides/nt8/alltrades.htm) that rely on references to **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state. For more information, see the [Working with Historical Trade Data](https://ninjatrader.com/support/helpGuides/nt8/strategyanalyzer_properties_2.htm) page. |

tog_minus        [How to run an Anchored Walk Forward Optimization](javascript:HMToggle('toggle','HowToRunAnAnchoredWalkForwardOptimization','HowToRunAnAnchoredWalkForwardOptimization_ICON'))

|  |  |
| --- | --- |
| **Start an Anchored Walk Forward Optimization**  To run an **Anchored Walk Forward Optimization**select the **Backtest type** of "**Anchored Walk Forward Optimization"**in the settings panel of the **Strategy Analyzer**.    StrategyAnalyzer_Optimization_AWFORuni     |  | | --- | | **Note**: When making the selection additional parameters to configure your optimization will be made visible. |     **Setting the Test Range**  You can the test range of strategy parameters to be tested by left clicking on the triangle to expand the strategies sub parameters.    Note: If you don't see the triangle make sure that the **Backtest type** is set to "**Anchored Walk Forward** **Optimization**".    StrategyAnalyzer_Optimization_Paramters    **Min**.  - The starting value you want to test **Max**.  - The last value to test **Increment** - The increment value (step value) used to increment the starting value by for each subsequent optimization pass    In the image above, the input "Fast" has a starting (initial) value of 10 and an ending value of 30 with an increment of 1. This means that the first value tested will be 10, then 11, then 12 all the way through 30. The input "Slow" has a starting value of 6, ending value of 16 with an increment of 1. Based on these settings, a total of 200 (20 unique values for "Fast" multiplied by 10 unique values for "Slow") backtest iterations will be processed in order to find the optimal combination of input values based on the best optimization fitness.    **Setting the Optimization Fitness**  Optimization is based on the best optimization fitness you select. If you set the property "Optimize on..." to "Max. net profit", the optimizer will seek the optimal input values that return the maximum profit possible. There are over 10 different optimization criterion you can select and can be customized via NinjaScript. Please see the "*Understanding Walk Forward properties*" section below for more information. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?anchored-walk-forward-optimiza.htm#HowToRunAnAnchoredWalkForwardOptimization)

tog_minus        [Understanding Anchored Walk Forward properties](javascript:HMToggle('toggle','UnderstandingAnchoredWalkForwardProperties','UnderstandingAnchoredWalkForwardProperties_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Anchored Walk Forward Properties**  Apart from the anchored anchored walk forward optimization specific properties described below, the properties are identical to the ones found in the Optimization properties window. Please see the "*Understanding optimization properties*" section of the [Optimize a Strategy](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm) page of the Help Guide for more information.    StrategyAnalyzer_Optimization_WFOParameters     |  | | --- | | **Tip**:  You can optionally "**Optimize on**" multiple objectives by using a [Multi-Objective optimization](https://ninjatrader.com/support/helpGuides/nt8/multi-objective_optimization.htm) |      |  |  | | --- | --- | | Keep best # results | Sets the number of best results to display | | Optimize data series | If set to true, the Data Series Value property will be available for optimization (Not supported for Kagi, PointAndFigure, and Line Break period Types) | | Optimize on... | Sets the optimization fitness to base the optimization results on | | Optimizer | Sets the optimization algorithm that is used. NinjaTrader comes with "Default" and "[Genetic](https://ninjatrader.com/support/helpGuides/nt8/genetic_algorithm.htm)" optimizer algorithms. When the "Genetic" option is selected, the genetic algorithm's optimization properties fields will appear below the Optimizer selection  You can program your own [optimization algorithm](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) using NinjaScript. | | Optimization period (days) | Sets the number of days used for the "in sample" optimization data set | | Test period (days) | Sets the number of days used for the "out of sample" real backtest using the optimized input values generated from the "in sample" period | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?anchored-walk-forward-optimiza.htm#UnderstandingAnchoredWalkForwardProperties)

tog_minus        [Understanding Anchored Walk Forward results](javascript:HMToggle('toggle','UnderstandingAnchoredWalkForwardResults','UnderstandingAnchoredWalkForwardResults_ICON'))

|  |  |
| --- | --- |
| **Understanding Anchored Walk Forward Test Results**  From the Start date to the End date the walk forward optimization will do a standard optimization on the number of days set for parameter "Optimization period (days)". This is known as the "In Sample" test period. After the optimization period NinjaTrader will use the best parameter combination found and test that forward on non-optimized data that has not been seen yet for the number of days set for parameter "Test period (days)". This is known as the "Out of sample" test period. Please see the graph below for a better understanding of how the walk forward results are found.  anchoredwalkforward_graph    The results for each "Test period" are returned and shown in the Optimization Results Grid along with the Start date, End date, and the best combination found by the optimization period.    StrategyAnalyzer_AWFO_BestResults1       |  | | --- | | **Note**: NinjaTrader does save the "Keep best # results" for each Optimization period, if you want to see each individual optimization results you can right click on the anchored walk forward result and select "View Optimization Results". | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?anchored-walk-forward-optimiza.htm#UnderstandingAnchoredWalkForwardResults)

|  |  |
| --- | --- |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) >  **Walk Forward Optimization** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/optimization_fitness_metrics.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/anchored-walk-forward-optimiza.htm) |

Walk Forward optimization is the process by which you optimize strategy input parameters on a historical segment of market data, then test the strategy forward in time on data following the optimization segment using the optimized input values. The central idea is that you evaluate strategy performance data on the test data, not the data used in the optimization. This process is then repeated by moving the optimization and test segments forward in time. To run a walk forward optimization you will need:

•Access to [historical data](https://ninjatrader.com/support/helpGuides/nt8/data_by_provider.htm)

•Custom NinjaScript \*[strategy](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm)

•A thorough understanding of the Strategy Analyzer's backtesting and optimization capabilities

|  |
| --- |
| **Tip**:  There are several pre-defined sample strategies that are installed with NinjaTrader that you can explore. |

|  |
| --- |
| playVideo |

|  |
| --- |
| **Note**: The [IncludeTradeHistoryInBacktest](https://ninjatrader.com/support/helpGuides/nt8/includetradehistoryinbacktest.htm) property is set to **false** by default when a strategy is applied in the **Strategy Analyzer** for optimization. This provides for leaner memory usage, but at the expense of not being able to access **Trade** objects for historical trades. Thus, fields such as [SystemPerformance.AllTrades.Count](https://ninjatrader.com/support/helpGuides/nt8/alltrades.htm) that rely on references to **Trade** objects will not have any such references to work with. If you would like to save these objects for reference in your code, you can set **IncludeTradeHistoryInBacktest** to **true** in the **Configure** state. For more information, see the [Working with Historical Trade Data](https://ninjatrader.com/support/helpGuides/nt8/strategyanalyzer_properties_2.htm) page. |

tog_minus        [How to run a Walk Forward Optimization](javascript:HMToggle('toggle','HowToRunAWalkForwardOptimization','HowToRunAWalkForwardOptimization_ICON'))

|  |  |
| --- | --- |
| **Start a Walk Forward Optimization**  To run a **Walk Forward Optimization**select the **Backtest type** of "**Walk Forward Optimization"**in the settings panel of the **Strategy Analyzer**.    StrategyAnalyzer_Optimization_WFORuni     |  | | --- | | **Note**: When making the selection additional parameters to configure your optimization will be made visible. |     **Setting the Test Range**  You can the test range of strategy parameters to be tested by left clicking on the triangle to expand the strategies sub parameters.    Note: If you don't see the triangle make sure that the **Backtest type** is set to "**Walk Forward** **Optimization**".    StrategyAnalyzer_Optimization_Paramters    **Min**.  - The starting value you want to test **Max**.  - The last value to test **Increment** - The increment value (step value) used to increment the starting value by for each subsequent optimization pass    In the image above, the input "Fast" has a starting (initial) value of 10 and an ending value of 30 with an increment of 1. This means that the first value tested will be 10, then 11, then 12 all the way through 30. The input "Slow" has a starting value of 6, ending value of 16 with an increment of 1. Based on these settings, a total of 200 (20 unique values for "Fast" multiplied by 10 unique values for "Slow") backtest iterations will be processed in order to find the optimal combination of input values based on the best optimization fitness.    **Setting the Optimization Fitness**  Optimization is based on the best optimization fitness you select. If you set the property "Optimize on..." to "Max. net profit", the optimizer will seek the optimal input values that return the maximum profit possible. There are over 10 different optimization criterion you can select and can be customized via NinjaScript. Please see the "*Understanding Walk Forward properties*" section below for more information. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?walk_forward_optimize_a_strate.htm#HowToRunAWalkForwardOptimization)

tog_minus        [Understanding Walk Forward properties](javascript:HMToggle('toggle','UnderstandingWalkForwardProperties','UnderstandingWalkForwardProperties_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Walk Forward Properties**  Apart from the walk forward optimization specific properties described below, the properties are identical to the ones found in the Optimization properties window. Please see the "*Understanding optimization properties*" section of the [Optimize a Strategy](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm) page of the Help Guide for more information.    StrategyAnalyzer_Optimization_WFOParameters     |  | | --- | | **Tip**:  You can optionally "**Optimize on**" multiple objectives by using a [Multi-Objective optimization](https://ninjatrader.com/support/helpGuides/nt8/multi-objective_optimization.htm) |      |  |  | | --- | --- | | Keep best # results | Sets the number of best results to display | | Optimize data series | If set to true, the Data Series Value property will be available for optimization (Not supported for Kagi, PointAndFigure, and Line Break period Types) | | Optimize on... | Sets the optimization fitness to base the optimization results on | | Optimizer | Sets the optimization algorithm that is used. NinjaTrader comes with "Default" and "[Genetic](https://ninjatrader.com/support/helpGuides/nt8/genetic_algorithm.htm)" optimizer algorithms. When the "Genetic" option is selected, the genetic algorithm's optimization properties fields will appear below the Optimizer selection  You can program your own [optimization algorithm](https://ninjatrader.com/support/helpGuides/nt8/optimizer.htm) using NinjaScript. | | Optimization period (days) | Sets the number of days used for the "in sample" optimization data set | | Test period (days) | Sets the number of days used for the "out of sample" real backtest using the optimized input values generated from the "in sample" period | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?walk_forward_optimize_a_strate.htm#UnderstandingWalkForwardProperties)

tog_minus        [Understanding Walk Forward results](javascript:HMToggle('toggle','UnderstandingWalkForwardResults','UnderstandingWalkForwardResults_ICON'))

|  |  |
| --- | --- |
| **Understanding Walk Forward Test Results**  From the Start date to the End date the walk forward optimization will do a standard optimization on the number of days set for parameter "Optimization period (days)". This is known as the "In Sample" test period. After the optimization period NinjaTrader will use the best parameter combination found and test that forward on non-optimized data that has not been seen yet for the number of days set for parameter "Test period (days)". This is known as the "Out of sample" test period. Please see the graph below for a better understanding of how the walk forward results are found.  walkforward_graph    The results for each "Test period" are returned and shown in the Optimization Results Grid along with the Start date, End date, and the best combination found by the optimization period.    StrategyAnalyzer_Optimization_BestResults1       |  | | --- | | **Note**: NinjaTrader does save the "Keep best # results" for each Optimization period, if you want to see each individual optimization results you can right click on the walk forward result and select "View Optimization Results". | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?walk_forward_optimize_a_strate.htm#UnderstandingWalkForwardResults)

|  |  |
| --- | --- |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) > [Optimization](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm) >  **Genetic Algorithm** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/optimization_fitness_metrics.htm) |

Very simply put the **Genetic** **Algorithm** attempts to find the most optimal set of parameters for a strategy. It does this not by brute force testing each individual combination as the default optimization method does, but instead using the concept of evolutionary theory borrowed from biology where only the fittest parents (combined with mutation and crossover) produce children for the next generation. Through testing of multiple generations you should have narrowed down on the most optimal parameters and therefore saving you time from having to test every single parameter combination.

|  |
| --- |
| playVideo |

tog_minus        [Understanding the Genetic Algorithm](javascript:HMToggle('toggle','UnderstandingTheGeneticAlgorithm','UnderstandingTheGeneticAlgorithm_ICON'))

|  |  |
| --- | --- |
| **Overview**  The general idea of how the GA solves an optimization problem is analogous to the concept of how evolution via natural selection adapts a species to the environment. In biology, only the strongest individuals will be able to reproduce and pass on their superior genes to the next generation. Assuming each generation can only pass on the strongest genes, after several iterations we would be left with the optimal attributes for the environment. Through this same mechanism, the GA will test a random preset of your parameters. Through multiple generations of testing, the parameters will zero in on an optimum solution.     |  | | --- | | **Note**: It is important to understand that GA will find approximate optimum solutions. Since it does not test every combination possible there is no guarantee its solutions are absolute optimums. |     **How the GA calculates**  The GA determines its solution through the following steps:    1.Begin with an initial population size consisting of randomly selected individuals (parameter setting combinations)  2.Compute the fitness (Optimize on...) for each individual in the population and assign probabilities to the population based on the fitness results. More fit results have more probability in being selected for breeding of the next generation.  3.Generate a new population for the next generation by selecting individuals from the prior generation to produce offspring via crossover and mutation (see below)  4.Repeat from step 2 till you reach the number of generations in your test    **Crossover and Mutation**  Crossover is the process in generating offspring that are not 100% identical to their parents. It is done by taking half of the parameter settings from parent A and mixing it with the other half from parent B. Crossover allows GA to test different combinations of parameters and hone in on the optimal solution. Crossover alone however will eventually yield identical offsprings in the population through several generations and so through mutation, some random parameter settings will be interjected in a few of the offsprings to allow for an adaptive quality to the algorithm. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?genetic_algorithm.htm#UnderstandingTheGeneticAlgorithm)

tog_minus        [Understanding Genetic Algorithm parameters](javascript:HMToggle('toggle','UnderstandingGeneticAlgorithmParameters','UnderstandingGeneticAlgorithmParameters_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Please see the "[Optimize a Strategy](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm)" article for how to run an optimization.    When you select the Genetic optimizer you will see the following optimization properties after you left click the triangle to the left of "GO Properties" to expand the properties.    StrategyAnalyzer_Optimization_GOParams     |  |  | | --- | --- | | Convergence threshold | Setting this will terminate the Genetic Optimization if there is more than a certain number of duplicate children in a single generation, defined by the Convergence Threshold value. This allows the optimization to terminate if no new work is getting done because it has already converged in on the most optimal solution. Example: In the screenshot above Generation size is set to 25, therefore each generation will contain 25 children, if 20 of these children are duplicates that have already been tested then the  optimization will be terminated. | | Crossover rate (%) | Each new generation is created from a combination of randomly generated offspring and offspring created from combining (crossing over) parent parameters. Crossover Rate determines the percentage of the new generation that is generated from the crossover process. | | Generation size | Sets the number of combinations to test in each generation (children). The higher the size, the more variety of combinations that will be tested in each generation. You want to make sure to set this high enough to test enough parameter combinations to get good coverage of the problem domain but not so high that each possible parameter combination is being tested in a single generation. | | Generations | Sets the number of generations to test. Each generation will hold the number of children set in "Generation Size".  The number of total parameter combinations tested is equal to the Generation Size \* Generations. | | Minimum performance | If this performance value is reached before all generations are evaluated the optimizer will end and present results immediately, where the type of this value is directly tied to your used optimization fitness metric (i.e. Profit Factor). A Value of 0 means no minimum performance is in use. | | Mutation rate (%) | Sets the probability that a crossover offspring will contain some mutated parameters (applies to all parameter types) | | Mutation strength (%) | Sets the maximum offset from crossover values that an offspring marked for mutation can have its parameters changed (applies only to input parameters of type double) | | Reset size (%) | When each new generation is created, all individuals from previous generations are possible parents for the new offsprings. If the top performing x% (stability size %) of children from the newly created generation is the same as the top performing x% of parents, reset all parents and repopulate a new generation randomly while leaving only the top performing y% of parents (reset size %) for future generations. Note: This occurs before convergence threashold is tested. | | Stability size (%) | See "Reset Size %" | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?genetic_algorithm.htm#UnderstandingGeneticAlgorithmParameters)

|  |  |
| --- | --- |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) >  **AI Generate (Experimental)** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/multi-objective_optimization.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/understanding_historical_fill_.htm) |

The AI Generate optimizer is an experimental tool designed to help traders find new strategy approaches. It can combine up to 73 NinjaTrader default indicators, 25 [Candlestick](https://ninjatrader.com/support/helpGuides/nt8/candlestickpattern.htm) patterns, and single series custom indicators.

Internally a [Genetic Algorithm](https://ninjatrader.com/support/helpGuides/nt8/genetic_algorithm.htm) is used to search through the potential entry and exit combinations possible to find the best performing ones according to the [Max Strength optimization criterion](https://ninjatrader.com/support/helpGuides/nt8/optimization_fitness_metrics.htm).

To prevent against potentially over-fitting against historical data, the AI Generate will check its own results after each generation using a [Monte Carlo Simulation](https://ninjatrader.com/support/helpGuides/nt8/monte_carlo_simulation.htm), it finds the 95% confidence interval.

We are excited to bring you this new tool to enhance your NinjaTrader strategy trading and are looking for feedback to further enhance it.

To run an AI Generate optimization you will need:

•Access to [historical data](https://ninjatrader.com/support/helpGuides/nt8/data_by_provider.htm)

•A thorough understanding of the Strategy Analyzer's backtesting and optimization capabilities

tog_minus        [How to run an AI Generate Optimization](javascript:HMToggle('toggle','HowToRunAWalkForwardOptimization','HowToRunAWalkForwardOptimization_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Start a AI Generate Optimization**  To run a **AI Generate Optimization**select the **Backtest type** of "**AI Generate (Experimental)"**in the settings panel of the **Strategy Analyzer**.    iStrategyAnalyzer_AIGenerate1     |  | | --- | | **Note**: When making the selection additional parameters to configure your AI Generate optimization will be made visible. |     **Setting the AI Generate Properties**  You can set the various AI Generate strategy parameters by left clicking on the triangles to expand the sub parameters (Entry conditions and Exit conditions)    StrategyAnalyzer_AIGenerate2     |  |  | | --- | --- | | Indicators | Select up to 73 NinjaTrader default indicators which to include in your AI Generation optimization and/or custom indicator that are single series | | Candle stick pattern | Select up to 25 NinjaTrader default [Candle stick patterns](https://ninjatrader.com/support/helpGuides/nt8/candlestickpattern.htm) which to include in your AI Generation optimization | | Day of week | If checked, the AI Generate optimization will include or exclude certain days of the week as part of the generated entry conditions for the strategies | | Session time | If checked, the AI Generate optimization will include or exclude certain parts of the trading session via time filters as part of the generated entry conditions for the strategies or would include them in its exit conditions to allow for time exits    •for entries between 0 and 60 minutes after session opening, for a duration of max. 120 minutes in 15 minutes steps  •for exits between 0 and 60 minutes before session close, going back max. 120 minutes in 15 minutes steps | | Parabolic stop | If checked, the [SetParablicStop](https://ninjatrader.com/support/helpGuides/nt8/setparabolicstop.htm) from NinjaScript could be used as an exit for the strategies | | Stops / Targets | If checked, would allow for [SetStopLoss](https://ninjatrader.com/support/helpGuides/nt8/setstoploss.htm), [SetTrailStop](https://ninjatrader.com/support/helpGuides/nt8/settrailstop.htm), [SetProfitTarget](https://ninjatrader.com/support/helpGuides/nt8/setprofittarget.htm) from NinjaScript could be used as exit for the strategies | | Session close | If checked, would allow the scripts to exit any open positions by the session end time | | Generations | Sets the number of generations to test. Each generation will hold the number of children set in "Generation Size".  The number of total parameter combinations tested is equal to the Generation Size \* Generations. | | Generation size | Sets the number of combinations to test in each generation (children). The higher the size, the more variety of combinations that will be tested in each generation. You want to make sure to set this high enough to test enough parameter combinations to get good coverage of the problem domain but not so high that each possible parameter combination is being tested in a single generation. | | Threshold of generations | Determines if the optimization process can be aborted if for the property number of consecutive generations the average of the performance values of the 'stable individuals' (the best 1/5 of the population is not touched on next generation = 'stable individuals') results did not improve. This allows for 'infinite' runs which would be terminated if no improvement is found. This logic is disabled if this property is set to 0. | | Keep best # results | Sets the number of best results to display |      |  | | --- | | **Notes**:    1. You can press 'Abort' to abort the AI Generate optimization, however you would have to wait until the 'generation size' iterations have passed to see the best found solutions so far.  2. In its current experimental state, the AI Generate sits on top the existing optimization framework inside NinjaTrader, as part of that you could see Strategy added indicators as well as the name from the last selected strategy (prior to switching over to the AI Generate optimization) still to appear on the Strategy Analyzer charts. |     **Viewing and saving results of the AI Generate optimization**    Pressing the 'View' button in the optimization results section would let you open the individual generated strategy code in the NinjaScript editor. From there you could then review and also save and further customize.    StrategyAnalyzer_AIGenerate3 |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?ai-generate.htm#HowToRunAWalkForwardOptimization)

|  |  |
| --- | --- |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) >  **AI Generate (Experimental)** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/multi-objective_optimization.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/understanding_historical_fill_.htm) |

The AI Generate optimizer is an experimental tool designed to help traders find new strategy approaches. It can combine up to 73 NinjaTrader default indicators, 25 [Candlestick](https://ninjatrader.com/support/helpGuides/nt8/candlestickpattern.htm) patterns, and single series custom indicators.

Internally a [Genetic Algorithm](https://ninjatrader.com/support/helpGuides/nt8/genetic_algorithm.htm) is used to search through the potential entry and exit combinations possible to find the best performing ones according to the [Max Strength optimization criterion](https://ninjatrader.com/support/helpGuides/nt8/optimization_fitness_metrics.htm).

To prevent against potentially over-fitting against historical data, the AI Generate will check its own results after each generation using a [Monte Carlo Simulation](https://ninjatrader.com/support/helpGuides/nt8/monte_carlo_simulation.htm), it finds the 95% confidence interval.

We are excited to bring you this new tool to enhance your NinjaTrader strategy trading and are looking for feedback to further enhance it.

To run an AI Generate optimization you will need:

•Access to [historical data](https://ninjatrader.com/support/helpGuides/nt8/data_by_provider.htm)

•A thorough understanding of the Strategy Analyzer's backtesting and optimization capabilities

tog_minus        [How to run an AI Generate Optimization](javascript:HMToggle('toggle','HowToRunAWalkForwardOptimization','HowToRunAWalkForwardOptimization_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Start a AI Generate Optimization**  To run a **AI Generate Optimization**select the **Backtest type** of "**AI Generate (Experimental)"**in the settings panel of the **Strategy Analyzer**.    iStrategyAnalyzer_AIGenerate1     |  | | --- | | **Note**: When making the selection additional parameters to configure your AI Generate optimization will be made visible. |     **Setting the AI Generate Properties**  You can set the various AI Generate strategy parameters by left clicking on the triangles to expand the sub parameters (Entry conditions and Exit conditions)    StrategyAnalyzer_AIGenerate2     |  |  | | --- | --- | | Indicators | Select up to 73 NinjaTrader default indicators which to include in your AI Generation optimization and/or custom indicator that are single series | | Candle stick pattern | Select up to 25 NinjaTrader default [Candle stick patterns](https://ninjatrader.com/support/helpGuides/nt8/candlestickpattern.htm) which to include in your AI Generation optimization | | Day of week | If checked, the AI Generate optimization will include or exclude certain days of the week as part of the generated entry conditions for the strategies | | Session time | If checked, the AI Generate optimization will include or exclude certain parts of the trading session via time filters as part of the generated entry conditions for the strategies or would include them in its exit conditions to allow for time exits    •for entries between 0 and 60 minutes after session opening, for a duration of max. 120 minutes in 15 minutes steps  •for exits between 0 and 60 minutes before session close, going back max. 120 minutes in 15 minutes steps | | Parabolic stop | If checked, the [SetParablicStop](https://ninjatrader.com/support/helpGuides/nt8/setparabolicstop.htm) from NinjaScript could be used as an exit for the strategies | | Stops / Targets | If checked, would allow for [SetStopLoss](https://ninjatrader.com/support/helpGuides/nt8/setstoploss.htm), [SetTrailStop](https://ninjatrader.com/support/helpGuides/nt8/settrailstop.htm), [SetProfitTarget](https://ninjatrader.com/support/helpGuides/nt8/setprofittarget.htm) from NinjaScript could be used as exit for the strategies | | Session close | If checked, would allow the scripts to exit any open positions by the session end time | | Generations | Sets the number of generations to test. Each generation will hold the number of children set in "Generation Size".  The number of total parameter combinations tested is equal to the Generation Size \* Generations. | | Generation size | Sets the number of combinations to test in each generation (children). The higher the size, the more variety of combinations that will be tested in each generation. You want to make sure to set this high enough to test enough parameter combinations to get good coverage of the problem domain but not so high that each possible parameter combination is being tested in a single generation. | | Threshold of generations | Determines if the optimization process can be aborted if for the property number of consecutive generations the average of the performance values of the 'stable individuals' (the best 1/5 of the population is not touched on next generation = 'stable individuals') results did not improve. This allows for 'infinite' runs which would be terminated if no improvement is found. This logic is disabled if this property is set to 0. | | Keep best # results | Sets the number of best results to display |      |  | | --- | | **Notes**:    1. You can press 'Abort' to abort the AI Generate optimization, however you would have to wait until the 'generation size' iterations have passed to see the best found solutions so far.  2. In its current experimental state, the AI Generate sits on top the existing optimization framework inside NinjaTrader, as part of that you could see Strategy added indicators as well as the name from the last selected strategy (prior to switching over to the AI Generate optimization) still to appear on the Strategy Analyzer charts. |     **Viewing and saving results of the AI Generate optimization**    Pressing the 'View' button in the optimization results section would let you open the individual generated strategy code in the NinjaScript editor. From there you could then review and also save and further customize.    StrategyAnalyzer_AIGenerate3 |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?ai-generate.htm#HowToRunAWalkForwardOptimization)

|  |  |
| --- | --- |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) > [Monte Carlo Simulation](https://ninjatrader.com/support/helpGuides/nt8/monte_carlo_simulation.htm) >  **Running a Monte Carlo Simulation** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/monte_carlo_simulation.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/monte_carlo_simulation.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/2d__3d_optimization_graphs.htm) |

The following page covers how to set up and run NinjaTrader's Monte Carlo Simulation

tog_minus        [Understanding Monte Carlo simulation](javascript:HMToggle('toggle','UnderstandingMonteCarloSimulation','UnderstandingMonteCarloSimulation_ICON'))

|  |
| --- |
| **What is Monte Carlo Simulation?**  Monte Carlo Simulation is a mathematical technique that uses repeated random sampling ("sampling with replacement") to compute a range of possible results with their respective probability. NinjaTrader runs Monte Carlo Simulation by randomly combining the trade results in a defined series of simulations. A graph of the results are plotted with the statistic values or Profit/Loss on the Y - axis and the probability on the X - axis as a percentage.    **Why use Monte Carlo Simulation?**  Although a backtest of a NinjaScript strategy may produce profitable results, those results may have just been due to good luck. In real life, you may have a string of bad trades that can wipe out the account before the good trades appear, therefore it would be helpful to understand the probability of such a string of bad trades. Monte Carlo Simulation will randomize your trade results over and over again in multiple simulations to provide you with a normal distribution of simulation performance. The trader can use this information to see the top or bottom percent of trades (outliers) that will cause the most variability in the strategy as well as the most statistically probable results. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?running_a_monte_carlo_simulati.htm#UnderstandingMonteCarloSimulation)

tog_minus        [How to run a Monte Carlo simulation](javascript:HMToggle('toggle','HowToRunAMonteCarloSimulation','HowToRunAMonteCarloSimulation_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **Monte Carlo Simulation window**  To open the Monte Carlo Simulation window:    1.Run a [Backtest](https://ninjatrader.com/support/helpGuides/nt8/backtest_a_strategy.htm), [Optimization](https://ninjatrader.com/support/helpGuides/nt8/optimize_a_strategy.htm), or [Walk-Forward Optimization](https://ninjatrader.com/support/helpGuides/nt8/walk_forward_optimize_a_strate.htm).  2.Left mouse click on the Trades tab within any of the reports  3.Right mouse click in the data grid and select the item **Monte Carlo Simulation...**    StrategyAnalyzer_MonteCarloDisplay |     **Running a Monte Carlo Simulation**  To run a Monte Carlo Simulation:    1.Open the Monte Carlo Simulation display (see sub-section above for how to open)  2.Set desired simulation parameters and press the Generate button.    StrategyAnalyzer_MonteCarloOptions2    **Monte Carlo Simulation Parameters**  The following parameters are adjustable when running a Monte Carlo Simulation:     |  |  | | --- | --- | | Graph | Sets the statistic to generate the report on | | W/L | Sets the results to show only winners, only loser, or both | | Long/Short | Sets the results to show only long trades, only short trades, or both | | Remove winning outliers (%) | Removes the top % outliers from the results | | Remove losing outliers (%) | Removes the bottom % outliers from the results | | # of simulations | Sets the # of simulations to run | | # of trades per simulation | Sets the # of trades in each simulation (will default to the # of trades in the Trades tab) | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?running_a_monte_carlo_simulati.htm#HowToRunAMonteCarloSimulation)

tog_minus        [Understanding the Monte Carlo Simulation report](javascript:HMToggle('toggle','UnderstandingTheMonteCarloSimulationReport','UnderstandingTheMonteCarloSimulationReport_ICON'))

|  |
| --- |
| **Monte Carlo Simulation Report**  The results of the Monte Carlo Simulation are displayed in a graph below the parameters.    StrategyAnalyzer_MonteCarloGraph    **X-Axis**  The horizontal axis of the Monte Carlo Simulation graph shows the percentage of simulations that have fallen below the Y - axis value. For example, if you run a Monte Carlo Simulation setting the # of Simulations to "100" and using the Cumulative Profit graph, the intersection of the 50% X - value and the associated Y value means that 50 of your simulations will be below that cumulative profit/loss value, and oppositely the remaining 50 simulations will have a greater cumulative profit/loss. This type of report allows you to analyze if the risk/reward ratio between worst and best case scenarios is acceptable or not.    **Y-Axis**  The vertical axis of the Monte Carlo Simulation graph displays the measured unit for the **Graph** item selected such as Profit/Loss, statistical information, or time and changes based on the Graph selection. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?running_a_monte_carlo_simulati.htm#UnderstandingTheMonteCarloSimulationReport)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/support/helpGuides/nt8/common.htm) >  **Clone()** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/timezoneinfo.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/common.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/description.htm) |

**Definition**

Used to override the default NinjaScript Clone() method which is called any time an instance of a NinjaScript object is created.  By default,  the NinjaScript Clone() method will copy all the [Property Info](https://msdn.microsoft.com/en-us/library/system.reflection.propertyinfo%28v=vs.110%29.aspx) and [Browsable Attributes](https://msdn.microsoft.com/en-us/library/system.componentmodel.browsableattribute%28v=vs.110%29.aspx) to the new instance when the object is created (e.g., when an optimization is ran a new instance of the strategy will be created for each iteration).  However it is possible to override this behavior if desired for custom development.  There is no requirement to override the Clone behavior and this method will use the default constructor if not overridden.

|  |
| --- |
| **Note**:  This method is reserved for advanced developers who would like to change the default behavior when a NinjaScript object is created |

**Method Return Value**

A [virtual](https://msdn.microsoft.com/en-us/library/9fkccyh4.aspx) object representing the NinjaScript type.

**Syntax**

public override object Clone()

**Parameters**

This method does not take any parameters

**Examples**

| ns | |
| --- | --- |
| public override object Clone() {   // custom logic to handle before the base clone     return base.Clone();     // custom logic to hand after the base clone } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/support/helpGuides/nt8/common.htm) > [Charts](https://ninjatrader.com/support/helpGuides/nt8/chart.htm) > [Rendering](https://ninjatrader.com/support/helpGuides/nt8/rendering.htm) >  **OnCalculateMinMax()** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/minvalue.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/rendering.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/onrender.htm) |

**Definition**

An event driven method which is called while the chart scale is being updated.  This method is used to determine the highest and lowest value that can be used for the chart scale. It is only called when the chart object is either set to [IsAutoScale](https://ninjatrader.com/support/helpGuides/nt8/isautoscale.htm) while there are multiple charts objects rendered or only a single object would be rendered on the chart.

|  |
| --- |
| **Note**:  The indexer used to look up a [Series<T>](https://ninjatrader.com/support/helpGuides/nt8/seriest.htm) value through barsAgo is **NOT** guaranteed to be in sync when the OnCalculateMinMax() method is called.  You will need to use [GetValueAt()](https://ninjatrader.com/support/helpGuides/nt8/getvalueat.htm) to obtain a historical value at a specified absolute index. |

**Method Return Value**

This method does not return a value.

**Syntax**  
You must override the method in your NinjaScript object with the following syntax:

public override void OnCalculateMinMax()  
{  
   
}

**Method Parameters**

This method does not accept any parameters.

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name       = "Example Indicator";     IsOverlay   = true;       // set this to true to ensure CalculateMinMix() is called     IsAutoScale = true;   } }   public override void OnCalculateMinMax() {   // make sure to always start fresh values to calculate new min/max values   double tmpMin = double.MaxValue;   double tmpMax = double.MinValue;     // For performance optimization, only loop through what is viewable on the chart   for (int index = ChartBars.FromIndex; index <= ChartBars.ToIndex; index++)   {     // since using Close[0] is not guaranteed to be in sync     // retrieve "Close" value at the current viewable range index     double plotValue = Close.GetValueAt(index);       // return min/max of close value     tmpMin = Math.Min(tmpMin, plotValue);     tmpMax = Math.Max(tmpMax, plotValue);   }     // Finally, set the minimum and maximum Y-Axis values to +/- 50 ticks from the primary close value   MinValue = tmpMin - 50 \* TickSize;   MaxValue = tmpMax + 50 \* TickSize; } | |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Order Entry](https://ninjatrader.com/support/helpGuides/nt8/order_entry.htm) >  **Working With Forex** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/fifo_optimization.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/order_entry.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/where_do_your_orders_reside_.htm) |

NinjaTrader supports trading and viewing market data for spot forex pairs, in addition to other supported instrument types. Due to the unique nature of forex markets, there are a number of features throughout the platform tailored specifically to these instruments, and a few considerations to keep in mind when working with forex in NinjaTrader.

tog_minus        [Pips Calculation Mode](javascript:HMToggle('toggle','PipsCalculationMode','PipsCalculationMode_ICON'))

|  |
| --- |
| **Pips vs. Ticks**  The "Pips" Calculation Mode can be used to calculate PnL and performance metrics throughout the platform. This mode allows you to tailor performance reporting specifically to your forex trades. Similar to the "Ticks" mode, "Pips" takes the lowest granularity of price movement for a forex instrument (called a tick in NinjaTrader), then divides it by 10 to arrive at the pip value for the instrument. For example, when viewing a USD/JPY quote of 113.67'5, the "7" would be the pip value, and the "5" would be the tick. Using the Pips Calculation Mode, the number of ticks in profit (the "5" in the example) will be divided by 10 to arrive at the number of pips of profit or loss.    Forex1    **Setting the Pips Calculation Mode**  The Pips calculation mode can be used in realized/unrealized PnL fields in trading windows ([Chart Trader](https://ninjatrader.com/support/helpGuides/nt8/chart_trader.htm), [SuperDOM](https://ninjatrader.com/support/helpGuides/nt8/superdom.htm), [Basic Entry](https://ninjatrader.com/support/helpGuides/nt8/basic_entry.htm), etc.), the [Trade Performance](https://ninjatrader.com/support/helpGuides/nt8/trade_performance.htm) window, and the [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm). In Trading Windows, the calculation mode can be changed by left-clicking within the PnL field, or by opening the window's Properties dialogue. For more information, see the relevant pages for each trading window.    Forex2    In the Trade Performance window and Strategy Analyzer, the calculation mode can be changed via the **Display** dropdown menu, which affects all relevant statistics.    Forex3 |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?working_with_forex.htm#PipsCalculationMode)

tog_minus        [Pips in ATM Strategies](javascript:HMToggle('toggle','PipsInATMStrategies','PipsInATMStrategies_ICON'))

|  |  |
| --- | --- |
| **ATM Strategy Parameters**  The **Parameter Type** field within the ATM Strategy Parameters window can be changed to "Pips" to affect the way that stop loss and profit target prices are set by an ATM strategy. Just like the Pips PnL calculation mode, the Pips parameter type is based on a multiplicative factor of the Ticks parameter type (1 Pip = 10 Ticks). For example, rather than entering 200 ticks for your profit target (200 ticks = 20 pips), you can simply specify 20 pips.    Forex4     |  | | --- | | **Note**: If your forex data provider supports tenth-pip quotes, then you can also use the Ticks parameter type to set ATM orders with a sub-pip granularity. | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?working_with_forex.htm#PipsInATMStrategies)

tog_minus        [Forex Lot Sizes](javascript:HMToggle('toggle','ForexLotSizes','ForexLotSizes_ICON'))

|  |  |
| --- | --- |
| **Setting Your FX Lot Size**  A "Forex Lot Size" property can be set for accounts shown in the Accounts tab of the Control Center. This setting affects the default position size populated in trading windows when a forex instrument is selected. To access this property, first select the Accounts  tab in the Control Center. Next, right click on the account you wish to edit, and select the **Edit Account** menu item. In the window that appears, set the Forex Lot Size property to your desired value. You can enter any amount here, whether or not it corresponds to a standard position size (Lot, Mini-Lot, Micro-Lot). For example, you could enter "102000" to automatically use a position size equal to one standard lot (100,000) plus two micro lots (2,000).    Forex6     |  | | --- | | **Notes**:  •The Forex Lot Size property does not prevent you from entering or selecting different position sizes in trading windows, but only controls what is populated in the Quantity field by default. | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?working_with_forex.htm#ForexLotSizes)

tog_minus        [Forex-Specific Trading Windows](javascript:HMToggle('toggle','ForexSpecificTradingWindows','ForexSpecificTradingWindows_ICON'))

|  |
| --- |
| **FX Pro**  The FX Pro window is laid out similarly to the Basic Entry window, with a few enhancements and modifications tailored specifically to forex instruments. For more information on using this window, see the [FX Pro](https://ninjatrader.com/support/helpGuides/nt8/fx_pro.htm) page.    **FX Board**  The FX Board is a unique forex trading window featuring a grid of two-sided tiles updated in real time, offering market data, spread info, and order management functionality for multiple pairs at once. FX Pro and FX Board windows can be linked together via [Instrument Linking](https://ninjatrader.com/support/helpGuides/nt8/linking_windows.htm). When linked, you can simply click any tile in the FX Board, and the corresponding instrument will be selected in a linked FX Pro window. For more information on using this window, see the [FX Board](https://ninjatrader.com/support/helpGuides/nt8/fx_board.htm) page.    **Other Windows**  Forex instruments can be traded in other windows, as well, and are not limited to the two mentioned above. Forex-specific windows can also be linked to others via Instrument Linking. Other windows, such as Chart Trader or the Market Analyzer, do not include forex-specific features, but are capable of handling FX instruments just like any others. |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?working_with_forex.htm#ForexSpecificTradingWindows)

tog_minus        [How Bars Are Built and Orders Filled](javascript:HMToggle('toggle','HowBarsAreBuiltAndOrdersFilled','HowBarsAreBuiltAndOrdersFilled_ICON'))

|  |  |
| --- | --- |
| **Building Bars with "Last Price" Data Type**  Forex price quotes do not use the concept of "Last Price" the same as other markets; only Bid and Ask quotes are available. Thus, when building bars using the default "Last" price type, the Bid price will be used instead. Using this price type, all bars on a chart will be built using Bid price updates, but you can choose to use the Ask price instead, if you wish. To change the price type used, first open the Data Series window on a chart, then toggle the value in the "Price Based On" field to your desired type.    **Realtime Order Fills vs. Backtesting**  Due to the absence of a last traded price quote in forex, all Buy orders in a live market are filled at the Ask price, and all Sell orders are filled at the Bid. However, when backtesting NinjaScript strategies, all simulated order fills will occur at the Bid price, regardless of whether they were Buy or Sells orders    Forex5    1.Ask Price: All realtime Buy orders are filled at the Ask  2.Bid Price: All realtime Sell orders and all backtest Buys and Sells are filled at the Bid     |  | | --- | | **Note**: In backtesting, a slippage value can be set to recreate the impact of the Bid/Ask spread on trade profit and loss. NinjaScript developers can calculate the spread in strategy logic, then dynamically set the [Slippage](https://ninjatrader.com/support/helpGuides/nt8/slippage.htm) property before entering orders. For non-programmers, an estimated slippage value can be applied to all trades via the Backtest/Optimization Properties section in the [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm). | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?working_with_forex.htm#HowBarsAreBuiltAndOrdersFilled)

tog_minus        [Forex Trading Hours](javascript:HMToggle('toggle','ForexTradingHours','ForexTradingHours_ICON'))

|  |
| --- |
| **Forex Trading Hours Template**  All forex instruments are configured to use the pre-defined "Forex" Trading Hours template, which runs 24 hours per day from 5:00pm EST on Sunday to 5:00pm EST on Friday, with an End-of-Day session break at 5:00pm each day. This covers the full range of forex trading throughout the week, but other Trading Hours templates can be applied to restrict the data on your charts to be in line with any local market timing on which you may wish to focus. For more information, see the [Trading Hours](https://ninjatrader.com/support/helpGuides/nt8/trading_hours.htm) page.    forex7 |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?working_with_forex.htm#ForexTradingHours)

tog_minus        [Calculating Pip Value](javascript:HMToggle('toggle','CalculatingPipValue','CalculatingPipValue_ICON'))

|  |  |
| --- | --- |
| **How to Calculate the Pip Value for a Forex Pair**  Multiplying the pip size of your currency pair by the lot size of your order will provide you the pip value. This will be in the quote/counter currency of the forex pair. The quote/counter currency is the second currency in the pair.    **Example in USD for an USD Quote/Counter Currency**  In the following example we will do this for a 10,000 lot on the EURUSD. The quote/counter currency is USD and the EURUSD's point size is 0.0001.  10,000 x 0.0001 = 1  This indicates that 1 pip would be $1 USD.    CalculatigPip_USD    **Example in USD for a GBP Base Currency**  In the following example we will use the EURGBP. Let's say our account is in USD and we want to convert the pip value to USD. Again we will say we are trading a 10,000 lot size and the pip value for the EURGBP is 0.0001.  10,000 x 0.0001 = 1  This indicates that 1 pip would be £1 GBP. We would then multiply this by what the GBPUSD is trading at.  In this example the GBPUSD is trading at $1.26 (rounded).  1 x 1.26 = 1.26  This indicates that 1 pip would be $1.26 USD.    CalculatigPip_GBP       |  | | --- | | **Note**: If the conversion rate is not available the PnL information will be in the counter/quote currency of the pair. This would create a discrepancy in your Trade Performance. | |

[permalink](https://ninjatrader.com/support/helpGuides/nt8/index.html?working_with_forex.htm#CalculatingPipValue)

|  |  |
| --- | --- |
| **Navigation:**  [Operations](https://ninjatrader.com/support/helpGuides/nt8/operations.htm) > [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) >  **Strategy Parameter Templates** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/discrepancies_real-time_vs_bac.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/strategyanalyzer_properties.htm) |

NinjaTrader allows a convenient way to save strategy parameters to easily transition to a live running strategy.

**Saving a Template**

Using the 'template' button on the bottom of the settings button shows 'Save' and 'Load'. Selecting **'Save'** allows you to save the selected settings for this strategy. If you have performed an optimization the selected optimization result set will be saved. This is signified by the "(" + ")" number directly to the right of the strategy parameter control.

If you save as 'Default' the template will be automatically loaded as you load the strategy.

**Loading a Template**

Using the 'template' button on the bottom of the settings button shows 'Save' and 'Load'. Selecting **'Load'** opens the loading dialog box where any templates specific to this strategy can be loaded. This allows you to have multiple configurations customized per instrument.

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Strategy](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm) >  **AddChartIndicator()** | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/strategy_account.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/addperformancemetric.htm) |

**Definition**

Adds an indicator to the strategy only for the purpose of displaying it on a chart.

|  |
| --- |
| **Notes**:  •Only the Plot properties of an indicator added by AddChartIndicator() will be accessible in the Indicators dialogue on charts. Other properties must be set in code.  •To add Bars objects to your strategy for calculation purposes see the [AddDataSeries()](https://ninjatrader.com/support/helpGuides/nt8/adddataseries.htm) method.  •An indicator being added via AddChartIndicator() cannot use any additional data series hosted by the calling strategy, but can only use the strategy's primary data series. If you wish to use a different data series for the indicator's input, you can add the series in the indicator itself and explicitly reference it in the indicator code (please make sure though the hosting strategy has the same [AddDataSeries()](https://ninjatrader.com/support/helpGuides/nt8/adddataseries.htm) call included as well)  o If a secondary or null Bars series is specified by the calling strategy (not the indicator itself), the strategy's primary series will be substituted instead.  •Dynamically using [DrawOnPricePanel](https://ninjatrader.com/support/helpGuides/nt8/drawonpricepanel.htm) in an indicator outside of State.SetDefaults may show issues when working with that indicator through a hosting strategy via [AddChartIndicator()](https://ninjatrader.com/support/helpGuides/nt8/addchartindicator.htm). |

**Method Return Value**

This method does not return a value.

**Syntax**  
AddChartIndicator(IndicatorBase indicator)

|  |
| --- |
| **Warning**:  This method should **ONLY** be called from the [OnStateChange()](https://ninjatrader.com/support/helpGuides/nt8/onstatechange.htm) method during **State.DataLoaded** |

**Parameters**

|  |  |
| --- | --- |
| indicator | An indicator object |

**Examples**

| ns |
| --- |
| protected override void OnStateChange() {     if (State == State.DataLoaded)     {         // Charts a 20 period simple moving average to the chart         AddChartIndicator(SMA(20));     } } |

|  |
| --- |
| **Tip**:  If you are adding an indicator which is dependent on the correct [State](https://ninjatrader.com/support/helpGuides/nt8/state.htm) of the indicator, you will need to ensure that you are also calling the indicator from the strategy in [OnBarUpdate()](https://ninjatrader.com/support/helpGuides/nt8/onbarupdate.htm), otherwise your indicator will only process in **State.RealTime** for performance optimizations. |

| ns | |
| --- | --- |
| protected override void OnStateChange() {   if (State == State.DataLoaded)   {     // Charts a 20 period simple moving average to the chart     AddChartIndicator(SMA(20));   } }   protected override void OnBarUpdate() {     // call SMA() historically to ensure the indicator processes its historical states as well   double sma = SMA(20)[0]; } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/support/helpGuides/nt8/common.htm) > [Charts](https://ninjatrader.com/support/helpGuides/nt8/chart.htm) > [Rendering](https://ninjatrader.com/support/helpGuides/nt8/rendering.htm) >  **OnRender()** | | [Previous page](https://ninjatrader.com/support/helpGuides/nt8/oncalculateminmax.htm) [Return to chapter overview](https://ninjatrader.com/support/helpGuides/nt8/rendering.htm) [Next page](https://ninjatrader.com/support/helpGuides/nt8/onrendertargetchanged.htm) |

**Definition**

Used to render custom drawing to a chart from various chart objects, such as an [Indicator](https://ninjatrader.com/support/helpGuides/nt8/indicator.htm), [DrawingTool](https://ninjatrader.com/support/helpGuides/nt8/drawingtool.htm) or [Strategy](https://ninjatrader.com/support/helpGuides/nt8/strategy.htm).

|  |
| --- |
| **Notes**:  1.This method uses the 3rd party SharpDX library to render custom Direct2D Text and Shapes.  For a walk through for using the **SharpDX**, please see the educational resource [Using SharpDX for Custom Chart Rendering](https://ninjatrader.com/support/helpGuides/nt8/using_sharpdx_for_custom_chart_rendering.htm)  2.The **OnRender()**method frequently runs once the [State](https://ninjatrader.com/support/helpGuides/nt8/state.htm) has reached **State.Realtime** in response to market data updates or a user interacting with the chart (e.g., clicking, resizing, rescaling, etc.)  3.For performance optimizations, the timing of the calls to **OnRender()** are buffered to at least 250ms, and re-renders once internal logic determines that values may be out-of-date.  See also [ForceRefresh()](https://ninjatrader.com/support/helpGuides/nt8/forcerefresh.htm) for more details  4.When using the [Strategy Analyzer](https://ninjatrader.com/support/helpGuides/nt8/strategy_analyzer.htm), **OnRender()** does **NOT** call until you switch to the "Chart" display and renders from **State.Terminated**.  As a result, this method should **NOT** be relied on for historical Strategy backtesting logic and should **ONLY** be used for rendering purposes  5.Unlike market data events and strategy order related events, there is **NO** guarantee that the *barsAgo* indexer used for [Series<T>](https://ninjatrader.com/support/helpGuides/nt8/seriest.htm) objects are in sync with the current bars in progress.  As a result, you should favor using an absolute index method to look up values (e.g.,[<series>.GetValueAt()](https://ninjatrader.com/support/helpGuides/nt8/getvalueat.htm), [Bars.GetOpen()](https://ninjatrader.com/support/helpGuides/nt8/getopen.htm), etc)  6.While **OnRender()** is an excellent means for customizing and enhancing indicators and strategies, its application can easily be abused, resulting in unforeseen performance issues which you may not catch until the right conditions (e.g., in the hands of your users during an FOMC event)  7.Please limit any calculations or algorithms you may be tempted run in OnRender() simply to rendering. You should always favor precomputed values and store them for rendering later as the preferred approach to working with the OnRender() method (e.g., reusing brushes, passing values from [OnBarUpdate()](https://ninjatrader.com/support/helpGuides/nt8/onbarupdate.htm), etc.).  See also [OnRenderTargetChanged()](https://ninjatrader.com/support/helpGuides/nt8/onrendertargetchanged.htm) method for more information on reusing Brushes  8.If you are using this method as an opportunity to "hook" onto a user related event, such as when a user selects a 3rd party control, you should alternatively consider using the events of that control independent of official NinjaScript events. See also [TriggerCustomEvent()](https://ninjatrader.com/support/helpGuides/nt8/triggercustomevent.htm) |

**Method Return Value**

This method does not return a value

**Syntax**

protected override void OnRender(ChartControl chartControl, ChartScale chartScale)  
{  
   
}

|  |
| --- |
| **Warning**:  Each DirectX [render target](https://ninjatrader.com/support/helpGuides/nt8/rendertarget.htm) requires its own brushes. You must create a brushes directly in **OnRender()** or using [OnRenderTargetChanged()](https://ninjatrader.com/support/helpGuides/nt8/onrendertargetchanged.htm).  If you do not you will receive an error at run time 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/support/helpGuides/nt8/onrendertargetchanged.htm) for examples of a brush that needs to be recalculated, or the example below of recreating a static brush. |

**Method Parameters**

|  |  |
| --- | --- |
| chartControl | A [ChartControl](https://ninjatrader.com/support/helpGuides/nt8/chartcontrol.htm) object (the chart's bar-related properties and x-axis) |
| chartScale | A [ChartScale](https://ninjatrader.com/support/helpGuides/nt8/chartscale.htm) object (the chart's y-axis) |

|  |
| --- |
| **Tips**:  •Please see the help guide topic on [Working with Brushes](https://ninjatrader.com/support/helpGuides/nt8/working_with_brushes.htm) for general information on using brushes and advanced brush concepts  •If you are using standard [Plots](https://ninjatrader.com/support/helpGuides/nt8/plots.htm) along with custom rendering from an indicator or strategy, you will need to ensure to call the **base.OnRender()** method for those plots to display. |

**Examples**

| ns **Using a static SharpDX Brush to render a rectangle on the chart panel** |
| --- |
| protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   // implicitly recreate and dispose of brush on each render pass   using (SharpDX.Direct2D1.SolidColorBrush dxBrush = new SharpDX.Direct2D1.SolidColorBrush(RenderTarget, SharpDX.Color.Blue))   {     RenderTarget.FillRectangle(new SharpDX.RectangleF(ChartPanel.X, ChartPanel.Y, ChartPanel.W, ChartPanel.H), dxBrush);   } } |

| ns **Calling the base.OnRender() method to ensure Plots are rendered along with custom render logic** |
| --- |
| protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   // call the base.OnRender() to ensure standard Plots work as designed   base.OnRender(chartControl, chartScale);     // custom render logic } |

| ns **Using multiple SharpDX objects to override the default plot appearance** |
| --- |
| protected override void OnRender(ChartControl chartControl, ChartScale chartScale) {   // get the starting and ending bars from what is rendered on the chart   float startX = chartControl.GetXByBarIndex(ChartBars, ChartBars.FromIndex);   float endX = chartControl.GetXByBarIndex(ChartBars, ChartBars.ToIndex);     // Loop through each Plot Values on the chart   for (int seriesCount = 0; seriesCount < Values.Length; seriesCount++)   {     // get the value at the last bar on the chart (if it has been set)     if (Values[seriesCount].IsValidDataPointAt(ChartBars.ToIndex))     {         double plotValue = Values[seriesCount].GetValueAt(ChartBars.ToIndex);           // convert the plot value to the charts "Y" axis point         float chartScaleYValue = chartScale.GetYByValue(plotValue);           // calculate the x and y values for the line to start and end         SharpDX.Vector2 startPoint = new SharpDX.Vector2(startX, chartScaleYValue);         SharpDX.Vector2 endPoint = new SharpDX.Vector2(endX, chartScaleYValue);           // draw a line between the start and end point at each plot using the plots SharpDX Brush color and style         RenderTarget.DrawLine(startPoint, endPoint, Plots[seriesCount].BrushDX,           Plots[seriesCount].Width, Plots[seriesCount].StrokeStyle);           // use the chart control text form to draw plot values along the line         SharpDX.DirectWrite.TextFormat textFormat = chartControl.Properties.LabelFont.ToDirectWriteTextFormat();           // calculate the which will be rendered at each plot using it the plot name and its price         string textToRender = Plots[seriesCount].Name + ": " + plotValue;           // calculate the layout of the text to be drawn         SharpDX.DirectWrite.TextLayout textLayout = new SharpDX.DirectWrite.TextLayout(Core.Globals.DirectWriteFactory,           textToRender, textFormat, 200, textFormat.FontSize);           // draw a line at each plot using the plots SharpDX Brush color at the calculated start point         RenderTarget.DrawTextLayout(startPoint, textLayout, Plots[seriesCount].BrushDX);           // dipose of the unmanaged resources used         textLayout.Dispose();         textFormat.Dispose();     }   } }   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "OnRender Example";     IsOverlay = true;           AddPlot(Brushes.DarkKhaki, "Open");     AddPlot(Brushes.SeaGreen, "High");     AddPlot(Brushes.Crimson, "Low");     AddPlot(Brushes.DodgerBlue, "Close");   } }   protected override void OnBarUpdate() {   Values[0][0] = Open[0];   Values[1][0] = High[0];   Values[2][0] = Low[0];   Values[3][0] = Close[0]; } |