|  |  |
| --- | --- |
| **Navigation:**  [Operaciones](https://ninjatrader.com/es/support/helpGuides/nt8/operations.htm) > [Analizador de mercado](https://ninjatrader.com/es/support/helpGuides/nt8/market_analyzer.htm) >  **Consejos de rendimiento** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_templates.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/market_analyzer.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/reloading_indicators__columns.htm) |

Los siguientes consejos de rendimiento son específicos de la ventana **Market Analyzer** .

**Número de instrumentos y columnas.**

•Market Analyzer no tiene límite para la cantidad de instrumentos que se pueden agregar. Es importante monitorear los recursos de la computadora para comprender el límite de su PC.

•Market Analyzer no tiene límite para la cantidad de columnas, específicamente columnas indicadoras, que se pueden agregar. Dependiendo del indicador y **la** propiedad "Marco de tiempo" que se describe a continuación, puede tomar algunos segundos calcular el indicador. Este tiempo se multiplica por la cantidad de instrumentos en el Market Analyzer que puede resultar en unos minutos de tiempo de carga. Disminuya el número de instrumentos o columnas indicadoras para disminuir este tiempo de carga.

**Columnas Indicadoras**

•La propiedad **Marco de tiempo** ("Barras para cargar", "Días atrás", "Rango personalizado") determina el número mínimo de barras requerido para inicializar correctamente cada columna de indicador. Cuanto mayor sea el número, más tardará en cargar los datos y más memoria (RAM) usará NinjaTrader para mantener los datos en la memoria.

•Cada indicador tiene un parámetro Retrospectiva de barras máximas en la ventana Columnas que determina cuántos valores de indicadores históricos se almacenan para el acceso. Está configurado en TwoHundredFiftySix de forma predeterminada para un rendimiento óptimo. Establecer esto en Infinito tomará más tiempo para calcular y NinjaTrader usará más memoria (RAM) para mantener los valores adicionales en la memoria.

**Indicador Calcular en parámetro de cierre de barra**

•Todos los indicadores agregados a Market Analyzer tienen el parámetro "Calcular" establecido en "Al cerrar la barra" de forma predeterminada, que solo calcula el valor del indicador en la barra cercana para ayudar con el rendimiento de la PC.  Este parámetro se puede configurar opcionalmente en "En cambio de precio", que solo se calculará cuando haya habido un cambio en el precio, o en "En cada tic" que permita un cálculo tick por tick (que utilizará más recursos de CPU).

**Clasificación dinámica y frecuencia de clasificación**

•Dependiendo de la cantidad de instrumentos que haya agregado a supantalla de**Market Analyzer**, el uso de un valor bajo de "[Clasificación automática de segundos](https://ninjatrader.com/es/support/helpGuides/nt8/market_analyzer_properties.htm)" puede hacer que su CPU se dispare a medida que la función de clasificación automática continúa reevaluando la clasificación de la columna que está clasificando.  Por ejemplo, usar un valor de 1 segundo en 100 instrumentos podría sobrecargar su CPU.  Establecer esto en un valor más alto, como cada 10-30 segundos, reducirá la carga de trabajo de la CPU y seguirá manteniendo la ordenación dinámica en un intervalo personalizable.  Debe monitorear la carga de trabajo de su CPU para encontrar la cantidad de segundos que funcionan para su sistema.

|  |  |
| --- | --- |
| **Navigation:**  [Operaciones](https://ninjatrader.com/es/support/helpGuides/nt8/operations.htm) > [Analizador de mercado](https://ninjatrader.com/es/support/helpGuides/nt8/market_analyzer.htm) >  **Propiedades del analizador de mercado** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/creating_cell_and_filter_condi.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/market_analyzer.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_templates.htm) |

Market Analyzer se puede personalizar según sus preferencias en la ventana Propiedades de Market Analyzer.

tog_minus        [Cómo acceder a la ventana de propiedades de Market Analyzer](javascript:HMToggle('toggle','HowToAccessTheMarketAnalyzerPropertiesWindow','HowToAccessTheMarketAnalyzerPropertiesWindow_ICON'))

|  |
| --- |
| Para acceder a la ventana de Propiedades de Market Analyzer, presione el botón derecho del mouse dentro de la ventana de Market Analyzer y seleccione el menú **Propiedades ...** |

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

tog_minus        [Propiedades disponibles y definiciones](javascript:HMToggle('toggle','AvailablePropertiesAndDefinitions','AvailablePropertiesAndDefinitions_ICON'))

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Las siguientes propiedades están disponibles para la configuración dentro de la ventana Propiedades de Market Analyzer    MarketAnalyzer_14    **Definiciones de propiedades**   |  |  | | --- | --- | | **General** |  | | Ordenar automáticamente | Habilita / deshabilita la clasificación y clasificación automática de filas | | Segundos de clasificación automática | Establece el intervalo de tiempo en segundos entre el recurso automático de filas | | Fuente de cuadrícula | Establece la fuente | | Alineación de texto de fila de etiqueta | Establece la alineación para las filas de etiquetas | | Duración de resaltado de cambio de fila (ms) | Establece la duración (en milisegundos) que la celda del instrumento permanecerá resaltada.    **Nota**:  El valor más bajo que surtirá efecto es 1000 (ms) | | Filtro de fila | Activa / desactiva el filtrado automático de filas desde la visualización de la cuadrícula en función de las condiciones de filtro de las columnas. | | Mostrar fila total | Activa / desactiva la fila Total en la cuadrícula de visualización de la ventana de Market Analyzer | | Nombre de la pestaña | Establece el nombre de la pestaña | | **Color** |  | | Fondo de cuadrícula | Establece el color predeterminado del fondo de la cuadrícula de visualización | | Primer plano de cuadrícula | Establece el color predeterminado del texto en una celda | | Líneas de cuadrícula | Establece el color de las líneas de la cuadrícula. | | Fondo de fila de etiqueta | Establece el color predeterminado para el fondo de la fila de etiquetas | | Etiqueta fila primer plano | Establece el color predeterminado para el primer plano de la fila de etiquetas | | La fila cambió el fondo resaltado | Establece el color para el fondo resaltado de cambio de fila (establecido en transparente para deshabilitar) | | La fila cambió el primer plano resaltado | Establece el color del texto en el resaltado de cambio de fila (el valor predeterminado está desactivado, como se establece en primer plano de cuadrícula) | | Fondo de fila total | Establece el color del fondo de la fila Total | | **Ventana** |  | | Mostrar pestañas | Habilita / deshabilita el control de pestaña | | Siempre en la cima | Habilita / deshabilita si la ventana estará siempre encima de otras ventanas. | |

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

tog_minus        [Cómo preestablecer valores predeterminados de propiedad](javascript:HMToggle('toggle','HowToPresetPropertyDefaults','HowToPresetPropertyDefaults_ICON'))

|  |
| --- |
| Una vez que haya configurado sus propiedades según sus preferencias, puede hacer clic con el botón izquierdo del mouse en el texto "preestablecido" ubicado en la parte inferior derecha del cuadro de diálogo de propiedades. Al seleccionar la opción "guardar", se guardarán estas configuraciones como las configuraciones predeterminadas que se usan cada vez que abre una nueva ventana / pestaña.    Si cambia la configuración y luego desea volver a la configuración original, puede hacer clic con el botón izquierdo del mouse en el texto "preestablecido" y seleccionar la opción "restaurar" para volver a la configuración original.    Las columnas de Market Analyzer junto con las propiedades personalizadas se pueden guardar dentro de una [plantilla de Market Analyzer](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_templates.htm) . |

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

|  |  |
| --- | --- |
| **Navigation:**  [Configuración](https://ninjatrader.com/es/support/helpGuides/nt8/configuration.htm) >  **Consejos de rendimiento** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/multiple_connections.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/configuration.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/operations.htm) |

**Factores de rendimiento**

Hay muchas variables que contribuyen al rendimiento general de laaplicaciónNinjaTrader.

•Los diferentes proveedores de conectividad (proveedores de datos del mercado y tecnologías de intermediario) que admite NinjaTrader varían en su nivel de servicio de datos en tiempo real. Por ejemplo, los proveedores que entregan datos de ticks sin filtrar (enviar todos los ticks de mercado) impondrán una mayor carga de procesamiento que un proveedor que proporciona datos de mercado limitados.

•La carga que coloque en laaplicaciónNinjaTrader (la ejecución de 200 gráficos consumirá más poder de procesamiento que la ejecución de solo 20 gráficos)

•La capacidad del hardware de su PC (¿está ejecutando una nueva máquina de última generación o una PC de 4 años con RAM limitada)?

•Una sola "configuración" incorrecta puede causar problemas de rendimiento y una única secuencia de comandos de terceros activa puede causar un problema de rendimiento general. Consulte lasección[Comprender el impacto de instalar complementos](https://ninjatrader.com/es/support/helpGuides/nt8/using_3rd_party_add-ons.htm).

|  |
| --- |
| **Nota**: El uso elevado de memoria puede no estar relacionado con un rendimiento deficiente. Se requieren recursos informáticos para determinar cuándo liberar memoria que ya no se está utilizando. Entonces, en lugar de usar recursos para determinar constantemente qué memoria liberar, el sistema determinará el mejor momento para liberar memoria cuando sea necesario. |

**Optimizar el rendimiento**

Las siguientes son algunas sugerencias que pueden ayudarlo a ajustar lainstalación deNinjaTrader para que se ejecute de manera óptima.

1.Cierre cualquier espacio de trabajo abierto innecesario. Aunque es posible que un área de trabajo no esté a la vista, aún utiliza los recursos del sistema cuando está abierto.

2.Salga de cualquier gráfico innecesario, SuperDOM, analizadores de mercado, etc. en su (s) espacio (s) de trabajo abierto (s). Además, si tiene varias pestañas en las ventanas que no necesita, salga de ellas.

3.Asegúrese de que todos los indicadores aplicados utilicen la configuración Calcular de Cambio de precio o Cierre de barra. No hay ningún beneficio al usar En cada marca a menos que el indicador se ocupe del conteo de volumen o marca.

4.Elimine los indicadores innecesarios de Gráficos, Analizadores de mercado y SuperDOM.

5.Reduzca los días para cargar en gráficos y en las columnas de indicadores de Market Analyzer. Esto puede ser especialmente efectivo con cualquier serie basada en ticks.

6.Use Tick Replay con moderación y solo cuando sea necesario. Por ejemplo, un indicador Pivot simple que solo usa los niveles de precios diarios actuales y anteriores no vería ninguna ventaja al usar la reproducción de ticks. En contraste, un indicador de perfil de volumen que se basa en la secuencia exacta de operaciones para calcular varios niveles se beneficiaría enormemente del uso de la repetición de ticks.

7.Eliminar objetos de dibujo innecesarios. Especialmente con el tiempo, ya que tiene más objetos de dibujo en sus gráficos, el rendimiento puede disminuir ya que hay más que representar.

8.Los espacios de trabajo que tienen un número único o menor de instrumentos no pueden aprovechar los procesadores de varios núcleos. Por ejemplo, tener 30 gráficos con el mismo instrumento no funcionará tan bien como 30 gráficos de diferentes instrumentos en un procesador multi-core. Reducir el número de ventanas con el mismo instrumento puede aumentar el rendimiento. Alternativamente, los espacios de trabajo que tienen uno o solo unos pocos instrumentos se beneficiarían de un procesador con una velocidad de reloj más alta.

9.Reduzca la cantidad de gráficos con la mira global habilitada.

10.Establezca la cruz para dibujar solo el cursor. Los sistemas con GPU menos potentes pueden beneficiarse de la reducción de la ruptura que resulta en este cambio. Esta configuración está dentro de las propiedades del gráfico.

11.Restablecer y borrar el historial de su cuenta de simulación. Una gran cantidad de operaciones simuladas puede aumentar el tamaño de su base de datos. Si no necesita esta información histórica, completar este paso liberará más recursos.

12.ReinicieNinjaTraderdiariamente.

13.Deshabilite la grabación de mercado para Reproducir y Grabar datos en vivo como históricos si no son necesarios.

14.Notas: "Guardar datos de gráficos como históricos" solo debe verificarse si utiliza un proveedor de conectividad que NO proporciona datos históricos. Muchos de los instrumentos de forex y futuros más populares están grabados en nuestros servidores y están disponibles para descargarlos para su reproducción.

15.La propiedad "Mostrar texto de volumen" en la columna Volumen de SuperDOM puede afectar el rendimiento de la PC y la velocidad de representación de los objetos en el SuperDOM. Esta propiedad está deshabilitada de manera predeterminada para minimizar el impacto en el rendimiento y, cuando está deshabilitada, puede colocar el cursor del mouse sobre cualquier fila de Volumen para ver el volumen exacto en esa fila.

16.Algunas configuraciones de computadora proporcionan dos tarjetas gráficas (integradas vs. dedicadas).  Para aprovechar al máximo el rendimiento de representación de gráficos, habilite un procesador gráfico dedicado de alto rendimiento.

17.Utilice una conexión a internet por cable. Las conexiones inalámbricas y por satélite pueden dejar caer paquetes o tener una latencia pobre. Si una conexión de baja latencia no está disponible, un VPS puede ser una gran solución.

18.Configure su conexión para conectarse en el arranque. Esto hará que su espacio de trabajo solo necesite cargarse una vez, en lugar de una vez cuando inicie NinjaTrader, y una segunda vez cuando se descarguen nuevos datos después de conectarse.

19.Reduzca la cantidad de aplicaciones que se ejecutan en su sistema.

20.La indexación de búsqueda de Windows también puede suponer una carga adicional para su procesador cuando ejecutaNinjaTrader, lo que afecta negativamente el rendimiento. También se recomienda excluir de la indexación las carpetas enumeradas a continuación, lo que puede hacerse a través del Panel de control de Windows.

 C:\Users\User\Documents\NinjaTrader 8

 C:\Program Files (x86)\NinjaTrader 8

**Reproducción**

1.Eliminar las suscripciones de instrumentos no utilizados en la reproducción. En su configuración de reproducción, tenga en cuenta los instrumentos que ha agregado (por ejemplo, en un Market Analyzer o mediante Cartas) que tendría datos para reproducir realmente disponibles, ya que cada suscripción de instrumento aquí consumiría ciclos de CPU y, por lo tanto, contribuiría al rendimiento experimentado. Por ejemplo, tener el índice SP500 agregado en su Market Analyzer, pero luego solo se espera que la reproducción de datos de MSFT tenga un rendimiento más bajo en comparación con tener solo este instrumento MSFT incluido en Market Analyzer también.

**Diagnósticos adicionales**

Si aún tiene problemas de rendimiento, siga estos pasos de diagnóstico para aislar el problema.

1.Inicia NinjaTrader en modo seguro.

 El Modo seguro evitará que NinjaTrader:

         Cargando espacios de trabajo.

         Conexión en la puesta en marcha.

         Cargando ensamblajes personalizados.

         Obteniendo actualizaciones de instrumentos desde el servidor.

 Para habilitar el modo seguro, siga los siguientes pasos:

         Salir NinjaTrader.

         Mantenga presionada la tecla CONTROL y haga doble clic en elicono deNinjaTrader.

         Mantenga presionada la tecla CONTROL hasta que vea el Centro de control.

         Puede verificar que está en modo seguro yendo a Ayuda -&gt; Acerca de.

2.Abra algunas ventanas y pruebe el rendimiento sin cargar ningún indicador personalizado.

3.Si todo se ve bien, intenta abrir tu espacio de trabajo. Es posible que deba cerrar su área de trabajo, sin guardarla y volver a abrirla. Si esto funciona bien, los montajes personalizados instalados pueden ser la causa.

4.Si el rendimiento es deficiente con el área de trabajo en modo seguro, podría ser que el área de trabajo tenga demasiados recursos para su sistema, que puede ser de los scripts que se están utilizando.

5.Para determinar si hay secuencias de comandos específicas que tienen como resultado un rendimiento deficiente, deberá volver a crear su área de trabajo y agregar las secuencias de comandos de una en una para ver cuál de ellas puede resultar en un rendimiento deficiente.

6.Para obtener más ayuda o asistencia con cualquiera de estos pasos, escriba aplatformupport@ninjatrader.com

|  |  |
| --- | --- |
| **Navigation:**  [Introducción](https://ninjatrader.com/es/support/helpGuides/nt8/introduction.htm) >  **Uso de complementos de terceros** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/learning_to_use_ninjatrader.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/introduction.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/configuration.htm) |

**Complementos de terceros**

El entorno de desarrollo integral y flexible de NinjaTrader permite a los Desarrolladores de terceros desarrollar aplicaciones ricas e integradas. Estos complementos permiten una personalización infinita

tog_minus        [Instalación de complementos](javascript:HMToggle('toggle','InstallingAdd-Ons','InstallingAdd-Ons_ICON'))

|  |  |
| --- | --- |
| **Instalación de complementos de terceros**  Una vez que haya descargado el complemento de terceros, se pueden importar desde el Centro de control de NinjaTrader.     1. Desde la ventana del Centro de control, seleccione el menú **Herramientas** &gt; **Importar** &gt; **Complemento de NinjaScript ...** para abrir la ventana de diálogo "Importar"   2. Selecciona el archivo que quieres importar.   3. Pulse el botón "**Importar**"     |  | | --- | | **Notas** :  1.Su proveedor puede tener diferentes instrucciones para instalar sus **complementos de terceros** .  Verifique con el proveedor las pautas específicas que puedan necesitar para instalar sus productos.  2.Si recibe un error durante la importación **"Tiene archivos NinjaScript personalizados en su PC que tienen errores de programación..."**, consulte la siguiente publicación para obtener información sobre cómo resolverlo: [¿Cómo resuelvo los errores de programación de NinjaScript?](https://ninjatrader.com/es/support/helpGuides/nt8/how_do_i_resolve_ninjascript_p.htm) | |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?using_3rd_party_add-ons.htm#InstallingAdd-Ons)

tog_minus        [Comprender el impacto de instalar complementos](javascript:HMToggle('toggle','UnderstandingTheImpactOfInstallingAdd-Ons','UnderstandingTheImpactOfInstallingAdd-Ons_ICON'))

|  |
| --- |
| **Comprender el impacto de instalar complementos**  NinjaTrader 8 proporciona un entorno de desarrollo que permite el acceso de bajo nivel a desarrolladores externos para construir indicadores integrados, herramientas de dibujo, estrategias automatizadas y más. Un **complemento** con errores de software puede tener efectos adversos en toda la aplicación NinjaTrader . Estos complementos también de forma nativa ejecutan en el equipo, por lo tanto, es importante instalar sólo **3rd Party Complementos** de fuentes de confianza.    Los siguientes síntomas posteriores a la instalación podrían indicar que se instaló un complemento que causó un impacto negativo:    •Windows se vuelve lento o no responde a la interacción del usuario  •Los datos de mercado se vuelven inusualmente lentos para cargar o actualizar  •Las características estándar no funcionan según lo diseñado  •Otros scripts no funcionan como se diseñaron (los scripts personalizados que funcionan por sí mismos pueden entrar en conflicto entre sí)  •Conexiones perdidas de proveedores de datos de mercado  •Los mensajes de error se generan en varias ocasiones.  •Toda la aplicación se apaga abruptamente y sin previo aviso.    Si se encuentra con alguno de los síntomas anteriores después de la instalación de un **complemento de terceros** , intente desinstalar los **complementos de terceros** para ver si el problema desaparece y póngase en contacto con el desarrollador de terceros para obtener asistencia. |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?using_3rd_party_add-ons.htm#UnderstandingTheImpactOfInstallingAdd-Ons)

tog_minus        [Actualización y eliminación de complementos](javascript:HMToggle('toggle','UpdatingAndRemovingAdd-Ons','UpdatingAndRemovingAdd-Ons_ICON'))

|  |  |
| --- | --- |
| **Actualizando complementos**  Los desarrolladores pueden emitir actualizaciones para solucionar problemas o agregar funcionalidad. Si ha obtenido una copia actualizada de su **complemento de terceros** , puede importar la nueva versión siguiendo los mismos pasos que instaló originalmente para instalar en **Herramientas** > **Importar** > **Complemento de NinjaScript ...** y seleccionando el nuevo archivo.  Durante el proceso de importación, se le dará una opción para reemplazar los **complementos** actuales que existen en su PC, que debe aceptar para cada archivo que desee actualizar.     |  | | --- | | **Nota:**Siempre debe reiniciar NinjaTrader después de instalar una actualización para asegurarse de que está ejecutando el código más reciente. |         **Eliminación de complementos de terceros**  Si identifica un problema, o sospecha que un **complemento de terceros** está causando problemas, es posible que desee eliminar estos archivos de su sistema.  Los pasos exactos para eliminar dependerán de cómo se distribuyó. **Los complementos de terceros** se pueden instalar como un "**Conjunto protegido"** o "**No protegido**".  Por favor, consulte la información a continuación sobre cómo proceder.    **Eliminar conjuntos de complementos de terceros protegidos**  Si ha comprado un **complemento de terceros** , es probable que se distribuya como un conjunto **protegido** .  Estos archivos protegidos se pueden desinstalar yendo a **Herramientas** &gt; **Eliminar ensamblaje de NinjaScript** .  Si no puede encontrar el **complemento de terceros** en este cuadro de diálogo, es muy probable que su **complemento** sea un conjunto **no protegido**.    **Eliminación de complementos de terceros no protegidos**  La mayoría de los **complementos** gratuitos de terceros que se descargan de foros en línea y otras comunidades se distribuyen como scripts c # **no protegidos**. Estos archivos de código abierto se pueden desinstalar usando los siguientes pasos:    1.Desde la ventana del Centro de control, seleccione el menú **Nuevo** > **Editor NinjsScript** para abrir el [Editor NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/editor.htm)  2.En el lado derecho, debajo del "**Explorador de NinjaScript"**, expanda el tipo de carpeta del **complemento de terceros** que está intentando desinstalar.  3.Busque el nombre del **complemento de terceros (Nota:**Los complementos de terceros se pueden instalar en varias subcarpetas **)**  4.Haga clic derecho en la entrada > seleccione "**Eliminar**" |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?using_3rd_party_add-ons.htm#UpdatingAndRemovingAdd-Ons)

tog_minus        [Deshabilitar temporalmente los complementos](javascript:HMToggle('toggle','TemporarilyDisablingAdd-Ons','TemporarilyDisablingAdd-Ons_ICON'))

|  |
| --- |
| **Deshabilitar temporalmente los complementos**  En caso de que empiece a experimentar un problema con NinjaTrader , el primer paso para aislar el problema es determinar si continúa experimentando problemas sin los **complementos** habilitados, lo que se conoce como &#39;Modo seguro&#39;. Para habilitar el modo seguro, siga los siguientes pasos:    1. Salir NinjaTrader  2. Mantenga presionada la tecla CONTROL en su teclado y haga doble clic en el icono de NinjaTrader.  3. Mantenga presionada la tecla CONTROL hasta que vea el Centro de control de NinjaTrader  4.Una vez que vea el Centro de control, puede verificar que está en modo seguro yendo a Ayuda&gt; Acerca de.    Una vez que esté en modo seguro, puede usar NinjaTrader sin complementos de terceros, lo que le permite verificar si un problema ya no está presente o permitirle eliminar un complemento de terceros. |

[permalink](https://ninjatrader.com/es/support/helpGuides/nt8/index.html?using_3rd_party_add-ons.htm#TemporarilyDisablingAdd-Ons)

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) >  **IsInstantiatedOnEachOptimizationIteration** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isfilllimitontouch.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/istradinghoursbreaklinevisible.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/es/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/es/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/es/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:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) >  **NinjaScript Best Practices** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/code_breaking_changes.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/distribution.htm) |

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

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

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

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

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

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

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

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

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

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

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

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

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

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

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Optimization Fitness](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) >  **OnCalculatePerformanceValue()** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness_value.htm) |

**Definition**

This method calculates the value for the Optimization Fitness.

**Syntax**

**protected override void OnCalculatePerformanceValue(StrategyBase strategy)**  
**{**

**}**

**Examples**

| ns | |
| --- | --- |
| protected override void OnCalculatePerformanceValue(StrategyBase strategy)  {      Value = strategy.SystemPerformance.AllTrades.TradesPerformance.Percent.Drawdown;  } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Optimization Fitness](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) >  **Value** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/oncalculateperformancevalue.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/optimizer.htm) |

**Definition**

The value an optimization would be calculating against when using this Optimization Fitness.

**Property Value**

A double value.

**Syntax**

Value

**Examples**

| ns | |
| --- | --- |
| protected override void OnCalculatePerformanceValue(StrategyBase strategy)  {      Value = strategy.SystemPerformance.AllTrades.TradesPerformance.Percent.Drawdown;  } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) >  **IncludeTradeHistoryInBacktest** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/includecommission.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/isadoptaccountpositionaware.htm) |

**Definition**

Determines if the strategy will save orders, trades, and execution history. When this property is set to **false** you will see significant memory savings at the expense of having access to the detailed trading information.

|  |
| --- |
| **Notes**:  •Since trade information is not stored you will only see entry/exit executions plotted on the chart with no connecting PnL trade lines.  •This property is set to **true** by default when a strategy is applied to a chart. However, in any other window (Strategies tab, Strategy Analyzer), it will be set to **false** by default. If you are working with a strategy outside of a chart, and would like to save orders, trades, and execution history for reference in your code, you will need to set **IncludeTradeHistoryInBacktest** to **true**in your script.  •During [Strategy Analyzer Optimization](https://ninjatrader.com/es/support/helpGuides/nt8/optimize_a_strategy.htm) in a 32-bit process, the **IncludeTradeHistoryInBacktest** property is forced to **false** due to the limited resources available in a 32-bit environment.  You must use a 64-bit process if trade history is needed during optimization. |

**Property Value**

This property returns **true** if the strategy will include trade history; otherwise, **false**. Default is set to **true**.  Always **false** during a strategy analyzer optimization on a 32-bit process.

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

**Syntax**

IncludeTradeHistoryInBacktest

**Examples**

| ns | |
| --- | --- |
| protected override void OnStateChange() {     if (State == State.SetDefaults)     {         // Explicitly include trade history in a backtest         IncludeTradeHistoryInBacktest = true;     } }   protected override void OnBarUpdate() {   // Stop taking trades after 10 trades have been taken since the strategy was enabled   if(SystemPerformance.AllTrades.Count >= 10)       return; } | |
| **Navigation:**  [Operaciones](https://ninjatrader.com/es/support/helpGuides/nt8/operations.htm) > [Analizador de estrategia](https://ninjatrader.com/es/support/helpGuides/nt8/strategy_analyzer.htm) >  **Mejoramiento** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/backtest_a_strategy.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/strategy_analyzer.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/genetic_algorithm.htm) |

Puede ajustar los parámetros de entrada de una estrategia a través de la optimización. La optimización es el proceso de probar un rango de valores a través de pruebas repetitivas iterativas para determinar los valores de entrada óptimos durante el período de prueba histórico en función de su capacidad de optimización. Para ejecutar una optimización, necesitará:

•Acceso a [datos históricos.](https://ninjatrader.com/es/support/helpGuides/nt8/data_by_provider.htm)

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

•Una comprensión profunda de las capacidades de [backtesting de](https://ninjatrader.com/es/support/helpGuides/nt8/backtest_a_strategy.htm) Strategy Analyzer

|  |
| --- |
| **Consejo** :  Hay varias estrategias de muestra predefinidas que se instalan con NinjaTrader que puede explorar. |

tog_minus        [Cómo ejecutar una optimización](javascript:HMToggle('toggle','HowToRunAnOptimization','HowToRunAnOptimization_ICON'))

|  |  |
| --- | --- |
| **Ejecutando una optimización**  Para ejecutar una **optimización,** seleccione el **tipo** de Backtest de " **Optimización"** en el panel de configuración de **Strategy Analyzer** .    StrategyAnalyzer_OptimizationRiun     |  | | --- | | **Nota**: Al hacer la selección, se harán visibles parámetros adicionales para configurar su optimización. |       **Establecer el rango de prueba**  Puede probar el rango de parámetros de estrategia a probar haciendo clic izquierdo en el triángulo para expandir los subparámetros de estrategias.    Nota: Si no ve el triángulo, asegúrese de que el **tipo de Backtest** esté configurado en " **Optimización** ".    StrategyAnalyzer_Optimization_Paramters    **Min** . - El valor inicial que desea probar **Max** . - El último valor para probar **Incremento** - El valor de incremento (valor de paso) utilizado para incrementar el valor inicial en cada pasada de optimización posterior    En la imagen de arriba, la entrada "Rápido" tiene un valor inicial (inicial) de 10 y un valor final de 30 con un incremento de 1. Esto significa que el primer valor probado será 10, luego 11, luego 12 hasta 30. La entrada "Lento" tiene un valor inicial de 6, un valor final de 16 con un incremento de 1. En función de esta configuración, se procesarán un total de 200 (20 valores únicos para "Rápido" multiplicado por 10 valores únicos para "Lento") iteraciones de backtest para encontrar la combinación óptima de valores de entrada basada en la mejor aptitud de optimización.    **Establecer la aptitud de optimización**  La optimización se basa en la mejor aptitud de optimización que seleccione. Si establece la propiedad "Optimizar en ..." en "Beneficio neto máximo", el optimizador buscará los valores de entrada óptimos que devuelvan el beneficio máximo posible. Hay más de 10 criterios de optimización diferentes que puede seleccionar y personalizar a través de NinjaScript. Consulte la sección "*Comprender las propiedades de optimización*" a continuación para obtener más información. |

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

tog_minus        [Comprender las propiedades de optimización](javascript:HMToggle('toggle','UnderstandingOptimizationProperties','UnderstandingOptimizationProperties_ICON'))

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Propiedades de optimización**  Además de las propiedades específicas de optimización que se describen a continuación, las propiedades son idénticas a las que se encuentran en la ventana de propiedades del backtest. Para obtener más información, consulte la sección "*Descripción de las propiedades del backtest*" de la página [Backtest a Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/backtest_a_strategy.htm) de la Guía de ayuda.    Las siguientes propiedades específicas de optimización están disponibles:    StrategyAnalyzer_Optimization_OptimizationParamaters       |  | | --- | | **Consejo** :  Opcionalmente, puede "**Optimizar en**" múltiples objetivos mediante el uso de una [optimización de objetivos múltiples](https://ninjatrader.com/es/support/helpGuides/nt8/multi-objective_optimization.htm) |        |  |  | | --- | --- | | Mantenga los mejores # resultados | Establece el número de mejores resultados para mostrar | | Optimizar series de datos | Si se establece en verdadero, la propiedad Valor de la serie de datos estará disponible para la optimización (no compatible con los tipos de período Kagi, Punto y figura, Salto de línea y Heiken Ashi) | | Optimizar en ... | Establece la aptitud de optimización para basar los resultados de optimización en | | Optimizador | Establece el algoritmo de optimización que se usa. NinjaTrader viene con algoritmos optimizadores "predeterminados" y "[genéticos](https://ninjatrader.com/es/support/helpGuides/nt8/genetic_algorithm.htm)". Cuando se selecciona la opción "Genética", los campos de propiedades de optimización del algoritmo genético aparecerán debajo de la selección del Optimizador. Puede programar su propio [algoritmo de optimización](https://ninjatrader.com/es/support/helpGuides/nt8/optimizer.htm) utilizando NinjaScript. | |

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

tog_minus        [Comprender los resultados de optimización](javascript:HMToggle('toggle','UnderstandingOptimizationResults','UnderstandingOptimizationResults_ICON'))

|  |
| --- |
| **Comprender los resultados de optimización**  Una vez que se complete el proceso de optimización, verá aparecer la Cuadrícula de resultados de optimización en la pestaña Analizador. Los resultados se agruparán por instrumento y muestran la combinación de parámetros que logró el mayor rendimiento. La columna "Rendimiento" es dinámica y siempre será el Optimization Fitness que seleccionó para el parámetro " **Optimizar** " cuando ejecutó la optimización.    StrategyAnalyzer_Optimization_BestResults1    **Los mejores resultados de optimización**  La pestaña Optimizer mostrará el número superior de resultados según el valor que establezca para la propiedad "Conservar los mejores # resultados" en la ventana de diálogo Optimizer. La columna Parámetros muestra los valores de entrada optimizados.    StrategyAnalyzer_Optimization_BestResults    1.El valor óptimo para la entrada "Rápida" para la estrategia de demostración utilizada para esta optimización  2.El valor óptimo para la entrada "lenta" para la estrategia de demostración utilizada para esta optimización |

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

tog_minus        [Ejecutando una prueba de canasta](javascript:HMToggle('toggle','RunningABasketTest','RunningABasketTest_ICON'))

|  |  |  |
| --- | --- | --- |
| **Prueba de cesta**  Ejecutar una optimización a través de una lista de instrumentos funciona de manera muy similar a ejecutar un backtest de canasta regular.  Para obtener información general, consulte la página [Cesta de prueba de varios instrumentos.](https://ninjatrader.com/es/support/helpGuides/nt8/basket_test.htm)  Sin embargo, al ejecutar una optimización en varios instrumentos, estará disponible una opción opcional "Agregada".      StrategyAnalyzer_Optimization_Aggregated     |  |  | | --- | --- | | Agregado | Si se establece en True, NinjaTrader intenta encontrar los resultados óptimos para toda la cesta de instrumentos. La fila COMBINADA en la pestaña de resultados mostrará una agregación de resultados en la cesta de instrumentos. (Este parámetro solo está disponible cuando se selecciona una [Lista de instrumentos](https://ninjatrader.com/es/support/helpGuides/nt8/instrument_lists.htm) para la optimización). | |

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

tog_minus        [Comprender los factores que afectan el rendimiento de la optimización.](javascript:HMToggle('toggle','UnderstandingFactorsThatAffectOptimizationPerformance','UnderstandingFactorsThatAffectOptimizationPerformance_ICON'))

|  |
| --- |
| **32 bit vs 64 bit**  Cuando ejecuta una optimización en la versión de **32 bits** de NinjaTrader para consumir menos memoria, no almacenamos ningún dato comercial para cada backtest que se ejecuta. Por lo tanto, si desea hacer un análisis comercial de uno de los resultados de la prueba posterior devueltos por una optimización, NinjaTrader debe volver a ejecutar la prueba posterior para obtener los datos comerciales, esto agrega un pequeño retraso al cambiar entre pruebas. La versión de **64 bits** de NinjaTrader aprovechará la memoria RAM adicional disponible para NinjaTrader y mantendrá los resultados comerciales de cada prueba retrospectiva, lo que le permitirá cambiar rápidamente entre los informes de resultados de la prueba.    **Mantenga los mejores # resultados**  Si descubre que se está quedando sin memoria del sistema durante sus pruebas de retroceso, reduzca este número de resultados para mantener una mejora significativa en la memoria utilizada por NinjaTrader .    **Ejecutando múltiples pruebas a la vez**  No podrá hacer más en un período de tiempo más pequeño al separar múltiples pruebas manualmente y ejecutarlas al mismo tiempo en la misma PC. NinjaTrader utilizará de manera eficiente todos los núcleos de CPU para cualquier optimización y realizar las pruebas más rápidas posibles.    **Recursos de CPU**  Asegúrese de tener tantos recursos del sistema disponibles para la optimización como sea posible, esto generalmente significa asegurarse de que todas las demás aplicaciones estén cerradas. Además, como el motor de optimización NinjaTrader está optimizado para aprovechar la mayor cantidad de recursos del sistema posible, es aconsejable no activar una optimización durante un tiempo en el que necesitaría usar la PC. Por ejemplo, no se recomienda iniciar una optimización mientras gestiona la salida de una operación.    **Datos comerciales históricos**  La propiedad [IncludeTradeHistoryInBacktest](https://ninjatrader.com/es/support/helpGuides/nt8/includetradehistoryinbacktest.htm) se establece en **falso** de forma predeterminada cuando se aplica una estrategia en el **Analizador de estrategias** para la optimización. Esto proporciona un uso de memoria más ágil, pero a expensas de no poder acceder a objetos **comerciales** para intercambios históricos. Por lo tanto, los campos como [SystemPerformance.AllTrades.Count](https://ninjatrader.com/es/support/helpGuides/nt8/alltrades.htm) que dependen de referencias a objetos **Trade** no tendrán tales referencias para trabajar. Si desea guardar estos objetos para referencia en su código, puede establecer **IncludeTradeHistoryInBacktest** en **verdadero** en el estado **Configurar**, pero esto puede resultar en un mayor uso de memoria. Para obtener más información, consulte la página [Trabajar con datos comerciales históricos](https://ninjatrader.com/es/support/helpGuides/nt8/strategyanalyzer_properties_2.htm) .    **Ejecución de optimizaciones eficientes**  Se espera que las optimizaciones de estrategia consuman una buena cantidad de recursos de CPU, simplemente debido a la naturaleza del procesamiento iterativo de datos que realizan. Las estrategias con un número relativamente grande de parámetros para la optimización pueden multiplicar este impacto. Cuando trabaje con estrategias con una gran cantidad de parámetros, evite usar "1" como valor de incremento para el optimizador, para evitar forzar al algoritmo del optimizador a ejecutar la cantidad máxima de permutaciones. Cambiar el valor de incremento a tan solo "2" puede reducir el número de permutaciones a la mitad, y aumentar este valor puede tener un impacto progresivamente menor.    El [algoritmo genético](https://ninjatrader.com/es/support/helpGuides/nt8/genetic_algorithm.htm) puede ofrecer una solución alternativa para aumentar los valores de incremento de parámetros. En lugar de ejecutar pruebas de fuerza bruta iterando sobre todas las permutaciones, el algoritmo genético ignora intencionalmente las combinaciones de parámetros que probablemente produzcan resultados subóptimos.    Especialmente en conjuntos de parámetros más grandes con valores de incremento más finos, los límites superiores de permutaciones potenciales / combinaciones de parámetros podrían alcanzarse para ambos enfoques de optimización: el mensaje de error *"La estrategia necesita al menos un parámetro para optimizar"* sería una indicación para reelaborar el # de parámetros o aumente los valores de incremento para alcanzar un recuento de permutación más significativo.    **Usar un servidor virtual / en la nube**  Si está utilizando un servidor virtual o en la nube como base para su configuración cuando ejecuta pruebas de optimización en el Analizador de estrategias, tenga en cuenta que dichos entornos generalmente pueden asignar los recursos disponibles bajo demanda. NinjaTrader seguirá aprovechando todos los hilos disponibles para su procesamiento, sin embargo, esos recursos disponibles se determinarán al inicio de la plataforma NinjaTrader. Entonces, si sus recursos virtuales hubieran cambiado mientras estaba en una sesión de trabajo, reinicie nuevamente para asegurarse de que el rendimiento sea óptimo. |

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

|  |  |  |
| --- | --- | --- |
| **Navigation:**  [Operaciones](https://ninjatrader.com/es/support/helpGuides/nt8/operations.htm) >  **Listas de instrumentos** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/hot_list_analyzer_properties.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/operations.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_instrument_lists.htm) |
| **Descripción general de las listas de instrumentos**  **Las listas de instrumentos** se pueden configurar haciendo clic con el botón izquierdo del mouse en el menú **Herramientas** dentro del Centro de control de NinjaTrader y seleccionando el elemento del menú **Listas de instrumentos** .  NinjaTrader admite la agrupación de instrumentos en listas de fácil acceso. Hay varios usos para una lista de instrumentos:    •[Backtesting](https://ninjatrader.com/es/support/helpGuides/nt8/backtest_a_strategy.htm) en el [analizador de estrategias](https://ninjatrader.com/es/support/helpGuides/nt8/strategy_analyzer.htm)  •Agregar rápidamente varios instrumentos a la ventana de [Market Analyzer](https://ninjatrader.com/es/support/helpGuides/nt8/market_analyzer.htm)  •Crear listas de los instrumentos desde la ventana [Hotlist Analyzer](https://ninjatrader.com/es/support/helpGuides/nt8/hot_list_analyzer.htm) .  •Organizar instrumentos en el **Selector de instrumentos** para facilitar el acceso.    ›[Trabajando con listas de instrumentos](https://ninjatrader.com/es/support/helpGuides/nt8/working_with_instrument_lists.htm)  ›[Actualización de divisiones y dividendos](https://ninjatrader.com/es/support/helpGuides/nt8/updating_splits_and_dividends.htm) | |

|  |  |
| --- | --- |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) >  **MaximumBarsLookBack** | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isvaliddatapointat.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/onbarupdate.htm) |

**Definition**

Determines memory performance of custom [Series<T>](https://ninjatrader.com/es/support/helpGuides/nt8/seriest.htm) objects (such as Series<double>, Series<long>, etc.).  When using **MaximumBarsLookBack.TwoHundredFiftySix**, only the last 256 values of the series object will be stored in memory and be accessible for reference. This results in significant memory savings when using multiple series objects. In the rare case should you need older values you can use **MaximumBarsLookBack.Infinite** to allow full access of the series.

|  |
| --- |
| **Notes**:  •ISeries<T> objects that hold bar data (such as Close, High, Volume, Time, etc) always use **MaximumBarsLookBack.Infinite** which ensures all data points are always accessible during the lifetime of your NinjaScript indicator or strategy.  •Series<double> objects that hold indicator [plot values](https://ninjatrader.com/es/support/helpGuides/nt8/values.htm) always use **MaximumBarsLookBack.Infinite** which ensures that charts always display the entire indicator's calculated values. |

**Property Value**

A **MaximumBarsLookBack** enum value. Default value is **MaximumBarsLookBack.TwoHundredFiftySix**

Possible values are:

|  |  |
| --- | --- |
| MaximumBarsLookBack.TwoHundredFiftySix | Only the last 256 values of the series object will be stored in memory and accessible for reference (improves memory performance) |
| MaximumBarsLookBack.Infinite | Allow full access of the series, but you will then not be able to utilize the benefits of memory optimization |

|  |
| --- |
| **Tip**:  A **MaximumBarsLookBack.TwoHundredFiftySix** series works as a circular ring buffer, which will "loop" when the series reaches full capacity.  Specifically, once there are 256 entries in the series, new data added to the series overwrite the oldest data. |

**Syntax**

MaximumBarsLookBack

**Examples**

| ns **Setting all custom series to use the default MaximumBarsLookBack** |
| --- |
| Series<double> myDoubleSeries = null; Series<string> myStringSeries = null;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "Example Indicator";     // Store all series values instead of only the last 256 values     MaximumBarsLookBack = MaximumBarsLookBack.Infinite;   }   else if (State == State.DataLoaded)   {     // The custom Series<t> below are all constructed using only the NinjaScriptBase object (i.e., "this")     // therefore, the Series<T> MaximumBarsLookBack is taken from the NinjaScript's configured MaximumBarsLookBack property     myDoubleSeries = new Series<double>(this);     myStringSeries = new Series<string>(this);   } } |

| ns **Optimizing custom series to use unique MaximumBarsLookBack behavior** | |
| --- | --- |
| Series<double> myDoubleSeries = null; Series<string> myStringSeries = null;   protected override void OnStateChange() {   if (State == State.SetDefaults)   {     Name = "Example Indicator";   }   else if (State == State.DataLoaded)   {     // The custom Series<t> below are constructed using MaximumBarsLookBack parameter     // therefore, each Series<t> will use their uniquely specified MaximumBarsLookBack properites     myDoubleSeries = new Series<double>(this, MaximumBarsLookBack.Infinite); // stores all values     myStringSeries = new Series<string>(this, MaximumBarsLookBack.TwoHundredFiftySix); // only the last 256 values (better performance)   } } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Common](https://ninjatrader.com/es/support/helpGuides/nt8/common.htm) > [ISeries<T>](https://ninjatrader.com/es/support/helpGuides/nt8/iseriest.htm) > [PriceSeries<double>](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) >  **Values** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/value.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/priceseries.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/weighted.htm) |

**Definition**

Holds an array of ISeries<double> objects holding hold the indicator's underlying calculated values. ISeries<double> values are added to this array when calling the [AddPlot()](https://ninjatrader.com/es/support/helpGuides/nt8/addplot.htm) method. In case of a [MultiSeries](https://ninjatrader.com/es/support/helpGuides/nt8/multi-time_frame__instruments.htm) indicator synched to the primary series.

**Property Value**

A collection of ISeries<double> objects.

**Syntax**

Values[int *index*]

**Examples**

| ns | |
| --- | --- |
| // OnBarUpdate method of a custom indicator protected override void OnBarUpdate() {     // Ensures we have enough bars loaded for our indicator     if (CurrentBar < 1)         return;       // Evaluates the indicator's secondary value 1 bar ago and sets the value of the indicator     // for the current bar being evaluated     if (Values[1][1] < High[0] - Low[0])         Value[0] = High[0] - Low[0];     else         Value[0] = High[0] - Close[0]; } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Strategy](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) >  **IsInstantiatedOnEachOptimizationIteration** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/isfilllimitontouch.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/strategy.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/istradinghoursbreaklinevisible.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/es/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/es/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/es/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:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Optimization Fitness](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) >  **OnCalculatePerformanceValue()** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness_value.htm) |

**Definition**

This method calculates the value for the Optimization Fitness.

**Syntax**

**protected override void OnCalculatePerformanceValue(StrategyBase strategy)**  
**{**

**}**

**Examples**

| ns | |
| --- | --- |
| protected override void OnCalculatePerformanceValue(StrategyBase strategy)  {      Value = strategy.SystemPerformance.AllTrades.TradesPerformance.Percent.Drawdown;  } | |
| **Navigation:**  [NinjaScript](https://ninjatrader.com/es/support/helpGuides/nt8/ninjascript.htm) > [Language Reference](https://ninjatrader.com/es/support/helpGuides/nt8/language_reference_wip.htm) > [Optimization Fitness](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) >  **Value** | | [Previous page](https://ninjatrader.com/es/support/helpGuides/nt8/oncalculateperformancevalue.htm) [Return to chapter overview](https://ninjatrader.com/es/support/helpGuides/nt8/optimization_fitness.htm) [Next page](https://ninjatrader.com/es/support/helpGuides/nt8/optimizer.htm) |

**Definition**

The value an optimization would be calculating against when using this Optimization Fitness.

**Property Value**

A double value.

**Syntax**

Value

**Examples**

| ns |
| --- |
| protected override void OnCalculatePerformanceValue(StrategyBase strategy)  {      Value = strategy.SystemPerformance.AllTrades.TradesPerformance.Percent.Drawdown;  } |