NinjaScript Null Checks

Prepared by EgoNoBueno

In NinjaScript, it's crucial to **always check for null before accessing members of objects** that might not be initialized. This practice prevents your scripts from crashing due to NullReferenceException errors. Objects like Instrument, ChartControl, and instances of indicators can sometimes be null, especially during certain script states or if they haven't been properly loaded or configured.

Why Null Checks are Essential

Think of a null object as an empty container. If you try to take something out of an empty container (i.e., access a property or call a method on a null object), your script won't know what to do and will throw an error, halting its execution.

Common Scenarios Where Objects Might Be Null:

- **Instrument Object:**
  - When running a strategy on a chart that has no instrument

selected.
    - During the OnStateChange() method if the script is removed from the chart (State.Removed).
    - If data for the instrument is not yet loaded.
- **ChartControl Object:**
    - When a script is run in a context where there is no chart (e.g., in the Strategy Analyzer in certain modes, or during backtesting if chart generation is disabled).
    - During the initial setup phases before the chart is fully available.
- **Indicator Instances:**
    - If an indicator fails to load due to incorrect parameters or missing data.
    - Before the OnStateChange() with State.Configure has completed, or if State.DataLoaded hasn't been reached yet, custom indicators might not be fully initialized.
    - If you declare an indicator variable but don't initialize it with an indicator function (e.g., EMA(14)).

How to Perform Null Checks

The most common way to perform a null check is using an if statement.

Instrument Object Example

C#

```
// Inside a NinjaScript (e.g., an Indicator or Strategy)
```

```
protected override void OnBarUpdate()
{
    // Check if the Instrument object is null or if Bars is null
    if (Instrument == null || Bars == null || Bars.Instrument == null)
    {
        Print("Instrument object is null. Cannot proceed.");
        return; // Exit the method to prevent errors
    }

    // Now it's safe to access Instrument properties
    string instrumentName = Instrument.FullName;
    Print("Current instrument: " + instrumentName);

    if (Bars.Instrument.MasterInstrument.Name == "ES")
    {
        // Do something specific for ES
    }
}
```

Explanation:

- if (Instrument == null): This directly checks if the Instrument property of your NinjaScript is null.
- || Bars == null || Bars.Instrument == null: It's also good practice to check Bars and Bars.Instrument as they are intrinsically linked to the instrument's data.
- return;: If any of these are null, we print a message and exit the OnBarUpdate() method to prevent further processing that could lead to an error.

ChartControl Object Example

C#

```
// Inside a NinjaScript that might interact with the chart UI

protected override void OnStateChange()
{
    if (State == State.SetDefaults)
    {
        Name = "My Null Check Example";
    }
    else if (State == State.Realtime || State == State.Historical)
    {
        // Attempt to access ChartControl
        if (ChartControl == null)
        {
            Print("ChartControl is null. UI operations cannot be performed.");
            return;
        }

        // Now it's safe to use ChartControl
        ChartControl.Dispatcher.InvokeAsync(() =>
        {
            // Example: Change chart background color if it's not black
            if (ChartPanel != null && ChartPanel.BackBrush !=
System.Windows.Media.Brushes.Black)
            {
                // Note: Direct UI manipulation like this should be done carefully
                // and often requires ChartPanel checks as well.
```

```
            Print("ChartControl is available.");
        }
    });
  }
}
```

Explanation:

- if (ChartControl == null): This checks if the ChartControl object, which provides access to chart UI elements, is null.
- This is particularly important if your script tries to draw custom objects directly on the chart or modify chart properties programmatically.

Indicator Instances Example

C#

```
// Inside a Strategy or another Indicator

// Declare an indicator variable
private EMA emaFast;
private RSI rsiSlow;

protected override void OnStateChange()
{
  if (State == State.SetDefaults)
  {
    Name = "Indicator Null Check";
    // Initialize other default parameters
  }
```

```
    else if (State == State.Configure)
    {
        // Add the indicators
        // It's possible for these to fail to initialize under certain rare conditions
        // or if incorrect parameters were somehow passed programmatically.
        emaFast = EMA(14);
        AddChartIndicator(emaFast);

        // Let's imagine rsiSlow might be added conditionally or later
        // rsiSlow = RSI(28);
        // if (someCondition)
        //     AddChartIndicator(rsiSlow);
    }
    else if (State == State.DataLoaded)
    {
        // Ensure indicators are initialized before accessing their values
        if (emaFast == null)
        {
            Print("EMA Fast indicator is null. Cannot access its values.");
            // Potentially halt the strategy or take corrective action
            return;
        }

        // Example for an indicator that might not always be added
        if (rsiSlow == null)
        {
            Print("RSI Slow indicator is not initialized or added. Skipping RSI
logic.");
        }
    }
```

```
}

protected override void OnBarUpdate()
{
    if (CurrentBar < BarsRequiredToTrade)
        return;

    // Null check before accessing indicator values
    if (emaFast == null)
    {
        Print("EMA Fast is null in OnBarUpdate. Exiting.");
        return;
    }

    double emaValue = emaFast[0]; // Access the current value of the EMA

    if (rsiSlow != null) // Check before using rsiSlow
    {
        double rsiValue = rsiSlow[0];
        // Use rsiValue for decisions
        if (emaValue > Open[0] && rsiValue < 30)
        {
            // Enter long
        }
    }
    else
    {
        // Logic if rsiSlow is not available
        if (emaValue > Open[0])
        {
            // Alternative entry logic without RSI
```

```
    }
  }
}
```

Explanation:

- private EMA emaFast;: We declare an indicator variable.
- emaFast = EMA(14);: We initialize it in State.Configure.
- if (emaFast == null): Before using emaFast in State.DataLoaded or OnBarUpdate(), we check if it's null. This is a safeguard, as indicator initialization usually happens successfully if parameters are correct. However, for indicators that are conditionally added or managed more dynamically, this check becomes even more critical.
- The rsiSlow example demonstrates checking an indicator that might be optional or added based on certain conditions.

Consequences of Not Performing Null Checks

If you attempt to access a member of a null object (e.g., nullObject.SomeProperty or nullObject.SomeMethod()), NinjaTrader will throw a NullReferenceException.

- **Script Halts:** This exception will typically stop your indicator or strategy from running further, potentially causing it to be disabled.
- **Missed Signals/Orders:** For strategies, this can mean missed trading signals or failed order submissions.
- **Chart Issues:** For indicators, it can lead to incorrect drawing or the indicator failing to load on the chart.
- **Log Errors:** You'll see error messages in the NinjaScript Output window or Log tab, which can help diagnose the problem but

are best avoided in the first place.

Best Practices 👍

1. **Be Defensive:** Assume any object that *could* be null *will* be null at some point. Write your code to handle this gracefully.
2. **Check Early, Check Often:** Perform null checks as soon as you get a reference to an object and before you try to use it, especially in methods that are called frequently like OnBarUpdate() or event handlers.
3. **Use && (AND) for Chained Checks:** When accessing nested properties, you can use the && operator for concise null checks:
   C#
   ```
   if (Instrument != null && Instrument.MasterInstrument != null && Instrument.MasterInstrument.TickSize > 0)
   {
       // Safe to use TickSize
   }
   ```

4. **Null-Conditional Operator (?. and ?[] - C# 6.0+):** NinjaScript's underlying C# version supports null-conditional operators, which can make null checks more concise. However, be mindful of the NinjaTrader platform's specific C# version support.
   C#
   ```
   // If ChartControl or its properties might be null
   string chartId = ChartControl?.ChartId?.ToString(); // Returns null if ChartControl or ChartId is null, no exception

   // For indicators or collections
   // double? emaValue = emaFast?[0]; // If emaFast is null, emaValue will be null
   ```

While powerful, ensure its compatibility and be aware that complex logic might still benefit from explicit if statements for clarity or for executing alternative code paths when an object is null.

5. **Initialize Variables:** Always initialize your indicator variables (and other custom objects) in an appropriate OnStateChange() state, typically State.Configure or State.DataLoaded.
6. **Logging:** When a null check fails (i.e., an object is found to be null when you didn't expect it), print a descriptive message to the NinjaScript Output window or Log. This helps in debugging.

```C#
if (MyCustomObject == null)
{
    Print(Time[0] + " MyCustomObject is unexpectedly null. State: " + State);
    return;
}
```

By diligently implementing null checks, you'll create more robust, reliable, and error-free NinjaScripts.