SentryOne Test Visual Studio Extension Custom Assemblies

Last Modified on 12 November 2019

Adding Custom Assemblies

Sometimes you need more flexibility than the available assertions provide, and you want to write some C# code that adds more logic to your tests. To do this, you can add code to the user file that SentryOne Test generates. There are four types of code that can be run:

- Before Test
- After Test
- Before Test Element
- After Test Element

For each of those, the following method declarations should be added to the user file.

- static partial void BeforeTest(string testName, Dictionary testResources, ref bool cancel)
- static partial void AfterTest(string testName, Dictionary testResources)
- static partial void BeforeTestElement(string testName, string elementName, Dictionary testResources, ref bool cancel)
- static partial void AfterTestElement(string testName, string elementName, Dictionary testResources)

> **Note:** The **before** methods have the **cancel** parameter. This can be set to **true** within the body of the method, which would cancel the execution of the element.

For example, you might want to execute some code after an element called **Execute**

Query (Grid) in a test called **Test 1** is run.

The method implementation might be similar to the following:

```
 static partial void AfterTestElement(string testName, string elementName, D
ictionary testResources)
       {
           if (testName == "Test 1" && elementName == "Execute Query (Grid)"
)
           {
               if (!testResources.ContainsKey("ExecuteQuery(Grid)_TargetReso
urceKey"))
               {
                   NUnit.Framework.Assert.Fail("There was no grid placed in
the resources by the test element.");
               }
               var grid = (PragmaticWorks.Common.Sql.RowStoreGrid.IGrid)test
Resources["ExecuteQuery(Grid)_TargetResourceKey"];
               if (grid.RowCount < 100)
               {
                   NUnit.Framework.Assert.Fail("We needed 100 or more rows t
o be returned from the execute query element.");
               }
           }
       }
```

First, check that the **testName** and the **elementName** are the ones you want. Find these easily by searching for **AfterTestElement** in the generated code file.

After this, check that the test resources passed to you contain the desired item. We are using the **Resource key** from the execute query (grid) configuration to identify the resource. If the resource is not present, fail the test. This test was built with NUnit, so we are using NUnit.Framework.Assert.

Once you're sure that the resource in place, get it back as a local variable by casting it to the correct type. Grid resources are always of the type PragmaticWorks.Common.Sql.RowStoreGrid.IGrid.

Check that the **RowCount** is 100 or more, and if not fail the test.

> ⚠ **Important:** This code isn't automatically modified when you change the name of tests, elements or resource keys. Make sure that any changes are reflected in the user code.