SentryOne Test Visual Studio Extension Using Parameters

Last Modified on 13 November 2019

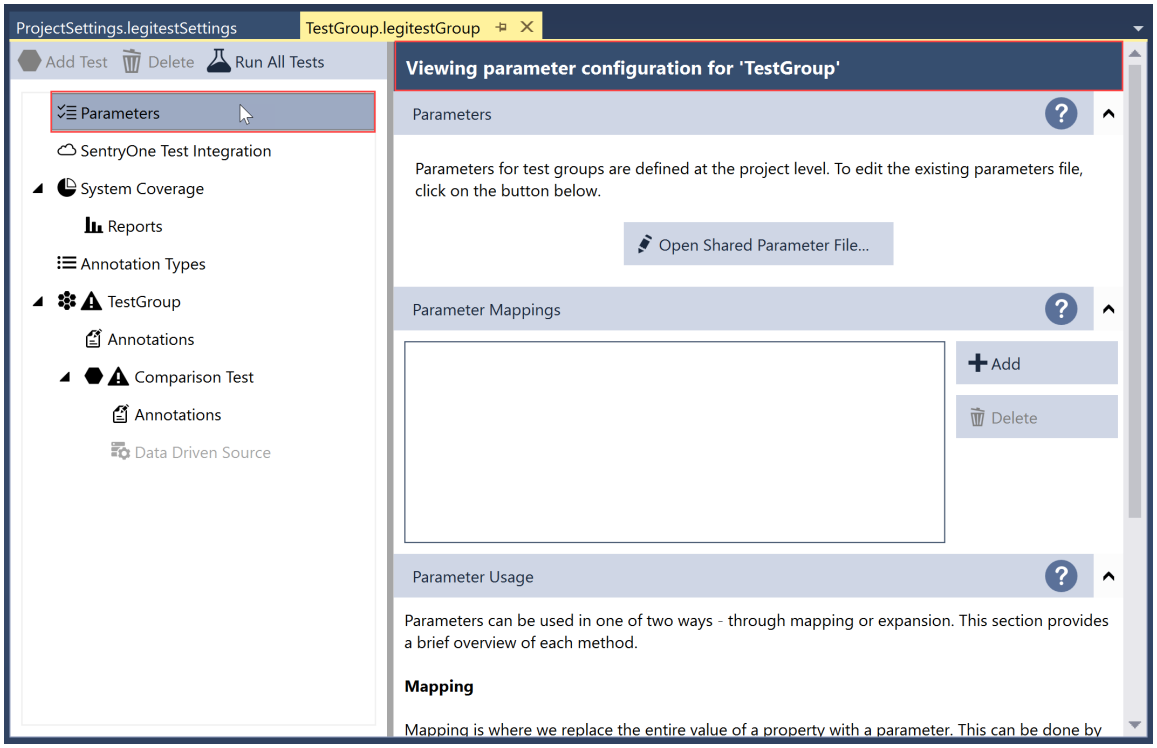
SentryOne Test includes the ability to parameterize tests. Element properties within the tests can use the parameters to allow easy porting between environments. This tutorial uses the Pragmatic Works HigherEd demonstration project. The tutorial creates tests for an SSIS package ensuring it extracts and then loads data into a data warehouse. By the end of this tutorial the tests use parameters to allow for easier deployment.

> **Note:** If you're unfamiliar with the layout in SentryOne Test, see the SentryOne Test Overview.

Overview

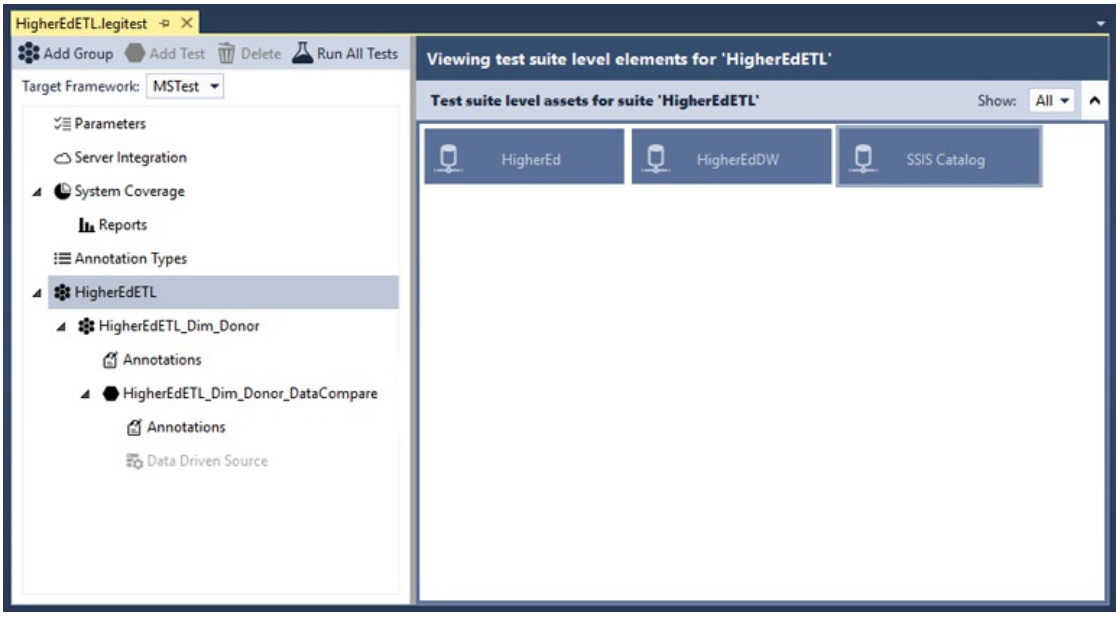
Parameters Overview

Within the test group select the **Parameters** node to view parameter settings. This area allows you to create and manage parameters.

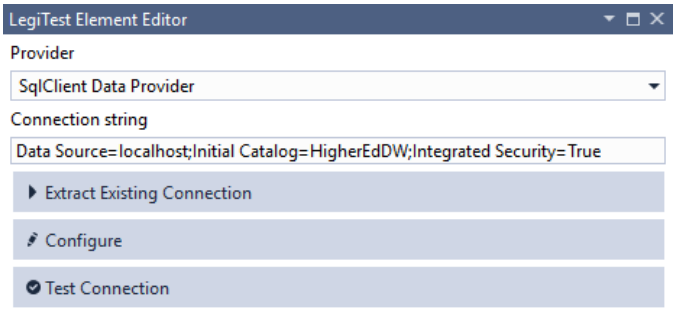


If you select the **HigherEdETL** node, the view changes test group level elements. Test group level assets are available for use across all tests. This test group contains three assets:

Test Object	Description
HigherEd	A Connection asset referencing the HigherEd database.
HigherEdDW	A Connection asset referencing the HigherEdDW database.
SSIS Catalog	A Connection asset referencing the SSIS Catalog that stores the package.

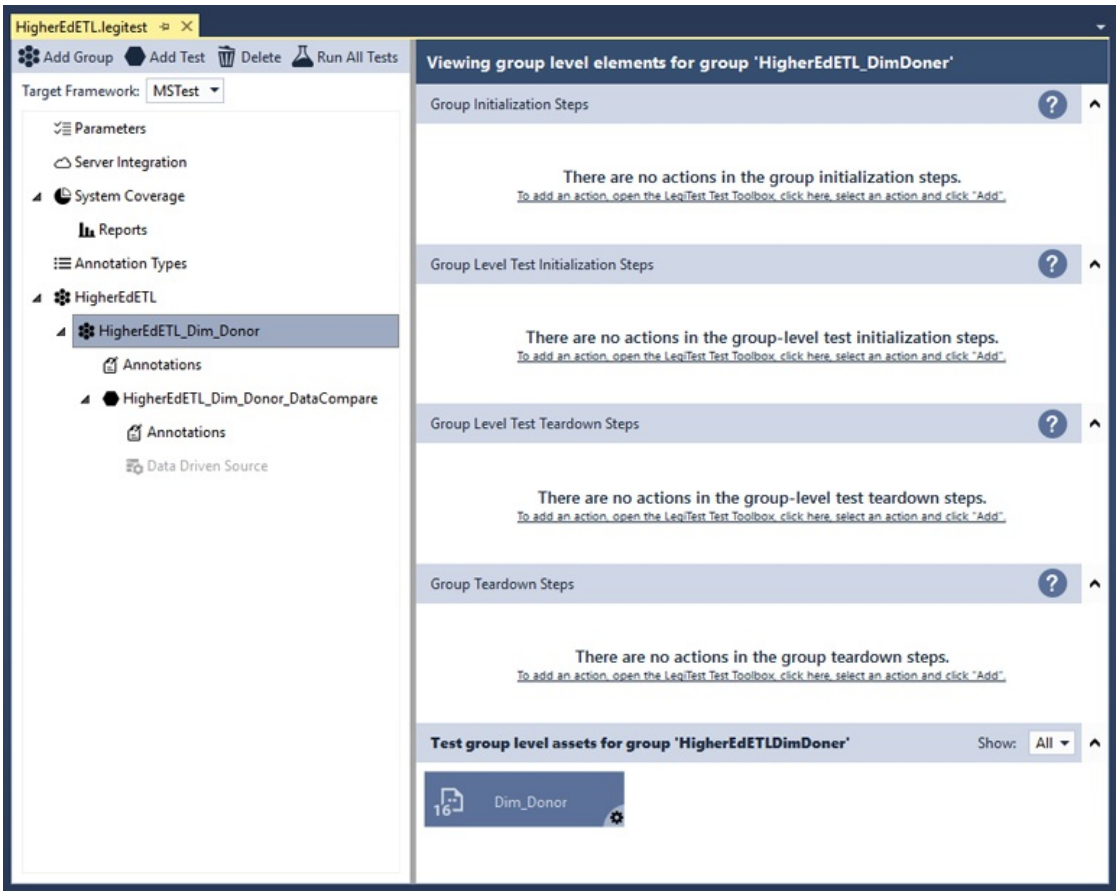


As of now, the **HigherEdDW** connection references our local data source. We can override this with a parameter for easier deployment.



If you select the **HigherEdETL_Dim_Donor** node, the view displays group elements. Group level elements are only available for the specific group and its tests. This group contains a single asset:

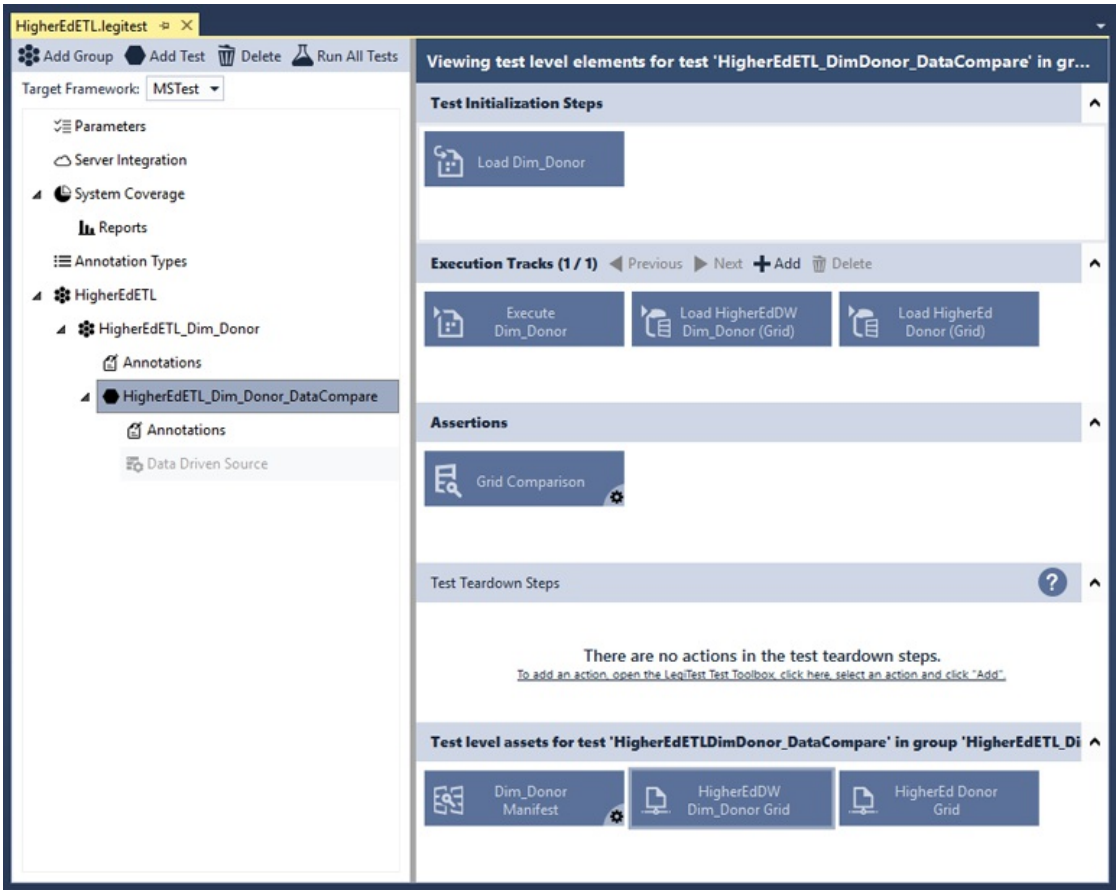
Asset	Description
Dim_Donor	An SSIS Package Reference that references the SSIS package.



Selecting the **HigherEdETL_Dim_Donor_DataCompare** node displays test elements.

> **Note:** Usually, a test group can contain more than one test. This test group has minimal complexity to better illustrate the use of parameters.

The **HigherEdETL_Dim_Donor_DataCompare** contains five main areas, though this test uses only four.



Test Assets	Assets used by all elements within this test. **Example Asset** / **Description** **DimDonor Manifest** — A Comparison Manifest that describes how to compare the Donor and Dim_Donor table. **HigherEdDW Dim_Donor Grid** — The Query Asset containing a T-SQL query that extracts the Dim_Donor table. **HigherEd Donor Grid** — The Query Asset containing a T-SQL query that extracts the Donor table.
Test Initialization Steps	These test initialization steps run before all execution tracks. **Example Test Initialization Step** / **Description** **Load Dim_Donor** — A Load Package action that loads the Dim_Donor package. SentryOne Test needs to load an SSIS Package before it can run within the execution track.
	Each execution track runs asynchronously. Within a single execution track the actions run from left to right. This particular test contains only a single execution track.

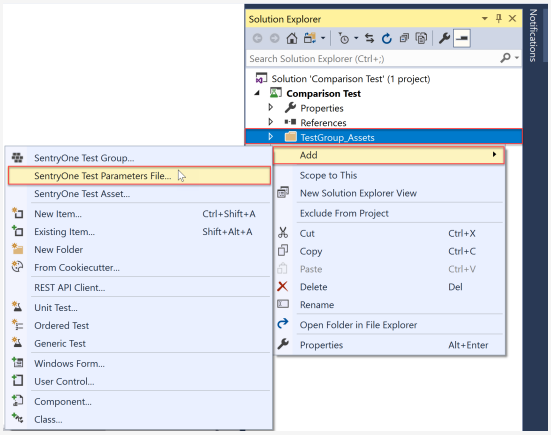
<table>
<tr><td>Execution Tracks</td><td>
<table>
<tr><th>Example Execution Track</th><th>Description</th></tr>
<tr><td>Execute Dim_Donor</td><td>An Execute Package action that executes the Dim_Donor package.</td></tr>
<tr><td>Load HigherEdDW Dim_Donor (Grid)</td><td>The Execute Query (Grid) loads the Dim_Donor table for comparison</td></tr>
<tr><td>Load HigherEd Donor (Grid)</td><td>The Execute Query (Grid) loads the Donor table for comparison.</td></tr>
</table>
</td></tr>
<tr><td>Test Teardown Steps</td><td>Test teardown steps execute after all execution tracks. This sample test contains no elements within this area.</td></tr>
<tr><td>Asserts</td><td>Asserts determine whether the test passes or fails.
<table>
<tr><th>Example Assert</th><th>Description</th></tr>
<tr><td>Grid Comparison</td><td>A Grid Comparison assert that compares whether Dim_Donor and Donor are identical.</td></tr>
</table>
</td></tr>
</table>

Tutorial

Parameter Tutorial

Tests contain hard-coded package and connection references. Rather than using hard-coded references, parameters allow for flexible deployment across environments. This tutorial walks you through the process of configuring the connections to reference parameters.

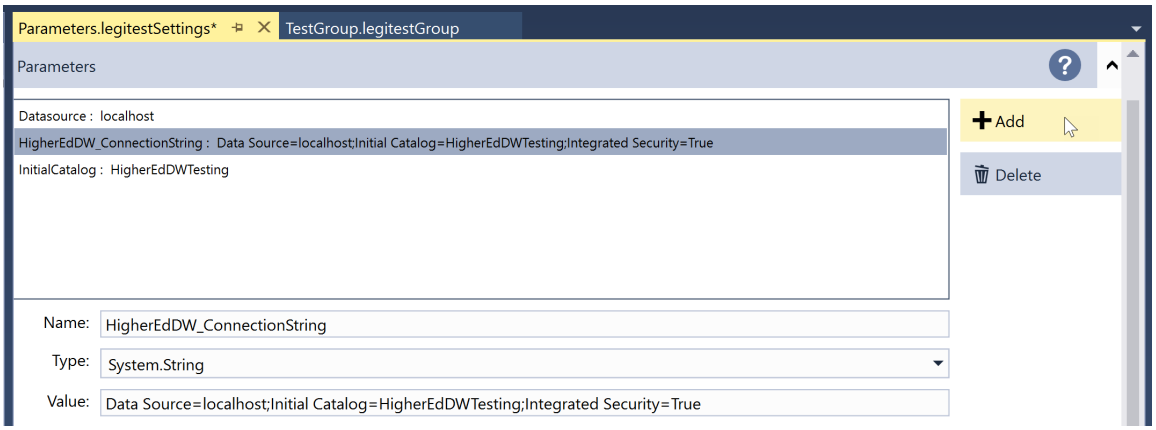
Note: Parameters can be defined both in SentryOne Test groups as well as individual parameter files within the project. To add a parameter file to the project, right click on the project node or group node in solution explorer and then select **Add** > **SentryOne Test Parameters File**.



1. Open the **parameter settings** to display the parameter area, and then select **Add** to create a new parameter.

2. Create the following parameters:

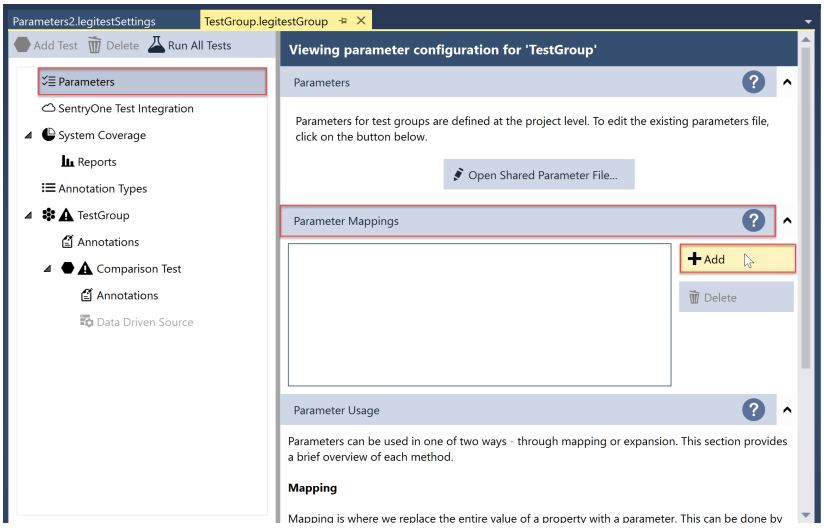
Parameter Name	Type	Value
DataSource	System.String	localhost
HigherEdDW_ConnectionString	System.String	Data Source=localhost;Initial Catalog=HigherEdDWTesting;Integrated Security=True
InitialCatalog	System.String	HigherEdDWTesting



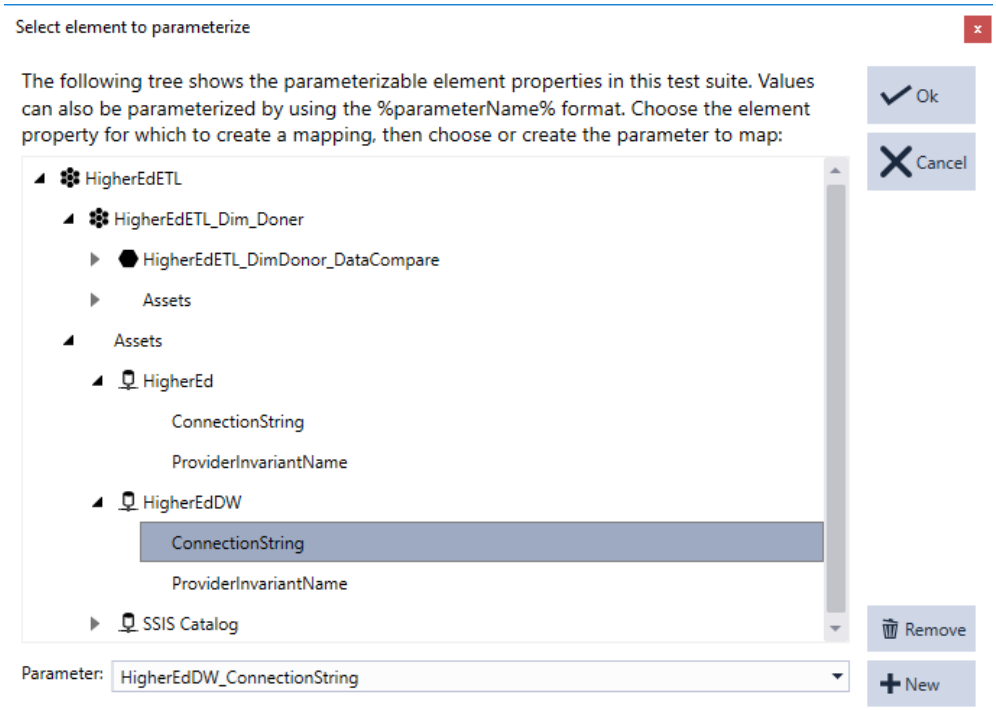
Direct Property Replacement

You can map parameters to properties within the SentryOne Test's Parameter area. This is the most basic form of parameter mapping. Direct property replacement completely replaces the entire property value with the value specified.

1. Select the **Parameter** Node to open the test group's parameter area, and then select the Parameter Mappings **Add** button.



2. The next window displays an organized list of all available element properties. Navigate to the **HigherEdDW** connection asset and select its **ConnectionString property**. Within the parameter drop down list, select the **HigherEdDW_ConnectionString** parameter, and then select **Ok**.



Parameter Token Replacement

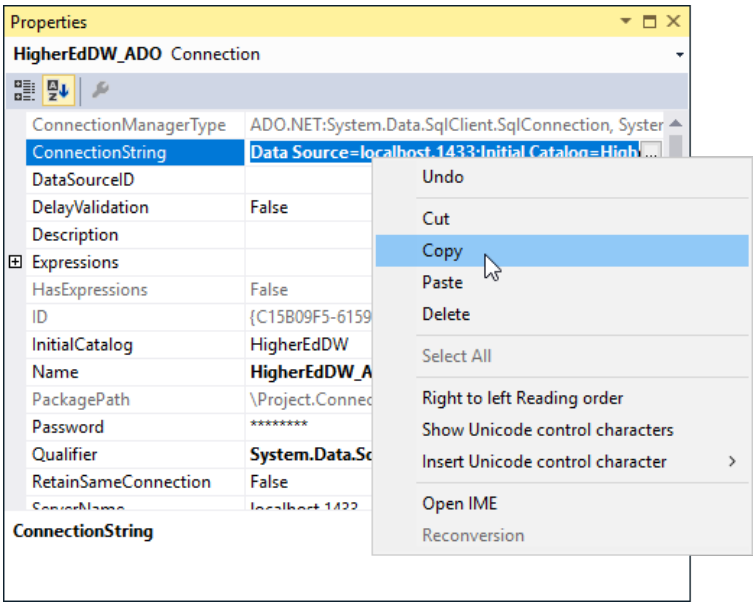
Using parameter token replacement, you can replace portions of a property. This allows you to replace pieces of a property instead of the entire property. When using this method, SentryOne Test interprets these as a parameter.

```
{{ParameterName}}
```

All actions that rely on the **HigherEdDW** asset now use the **HigherEdDW_ConnectionString** parameter. The **Dim_Donor** package still contains connection managers pointing to the original database. The package's connection managers need parameter mappings as well.

1. Within SSDT or BIDS, select each of the connection managers and view its properties.
2. Copy out the connection's connection string property. This connection string property forms the

baseline for the parameter token replacement. Do this for both connection strings.



Below are the original values for the connection strings.

Connection String	Value
HigherEdDW_ADO	Data Source=localhost,1433;Initial Catalog=HigherEdDW;Integrated Security=True;Application Name=SSIS-InvalidPackage-{C15B09F5-6159-4B38-A44F-69C9A98C8612}localhost,1433.HigherEdDW;
HigherEdDW_OLEDB	Data Source=localhost,1433;Initial Catalog=HigherEd;Provider=SQLNCLI11.1;Integrated Security=SSPI;Auto Translate=False;

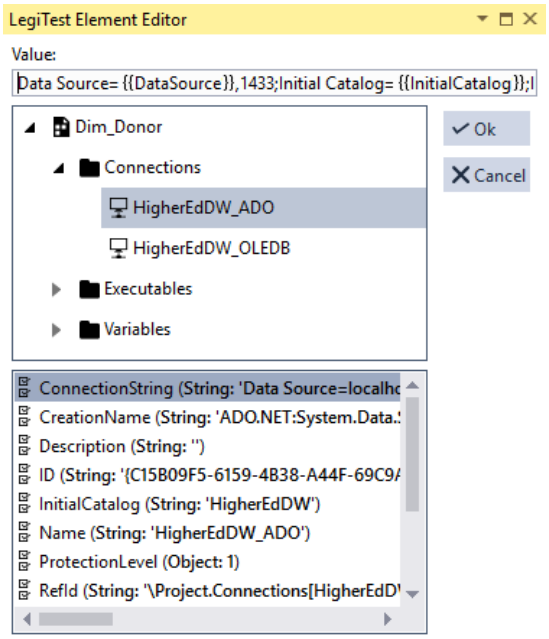
3. Back within SentryOne Test, select the **HigherEdETL_Dim_Donor_DataCompare** node.

4. Add a **Set Property** action within the **Group Initialization** steps.

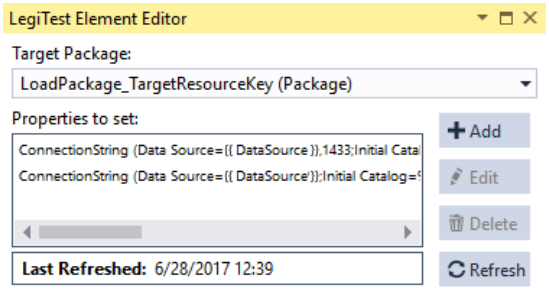
5. Replace the Data Source and Initial Catalog within the connection strings. Below, with emphasis, are the replaced values for the connection strings.

Connection String	Value
HigherEdDW_ADO	Data Source={{DataSource}},1433;Initial Catalog={{InitialCatalog}};Integrated Security=True;Application Name=SSIS-InvalidPackage-{C15B09F5-6159-4B38-A44F-69C9A98C8612}localhost,1433.HigherEdDW;
HigherEdDW_OLEDB	Data Source={{DataSource}};Initial Catalog={{InitialCatalog}};Provider=SQLNCLI11.1;Integrated Security=SSPI;Auto Translate=False;

6. Select both Connections and replace the ConnectionString with their modified value.



7. There should be two ConnectionString property values that were changed. Once for **HigherEdDW_ADO** and again for **HigherEdDW_OLEDB**.



Environment Token Replacement

Using environment token replacement, you can use system environment variables to replace portions of a property. This allows you to replace pieces of a property instead of the entire property. When using this method, SentryOne Test interprets these as a parameter.

```
%EnvironmentVariableName%
```

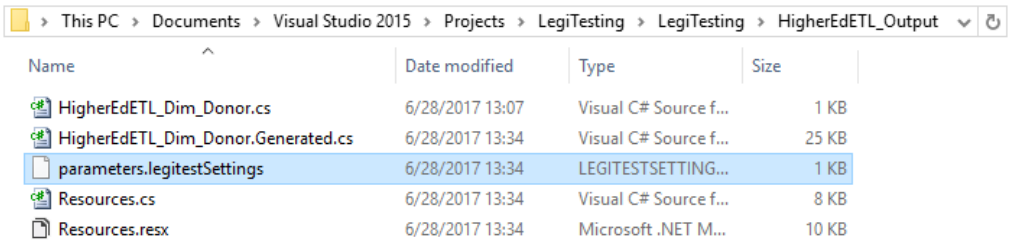
The steps to using environment token replacement are identical to parameter token replacement.

Settings & Global

Settings Files and Global Parameters

Settings files

The previous tutorial explained how to use parameters within Visual Studio. Within the MSTest and NUnit command line SentryOne Test can use setting files. These setting files control parameters during runtime. SentryOne Test even generates the file for you when building the project. This parameters file is located within the output folder.



When a test runs, SentryOne Test looks in the following locations for setting files (in this order):

- The directory where the test assembly is located
- %PROGRAMDATA%\PragmaticWorks\LegiTest (e.g. C:\ProgramData\PragmaticWorks\LegiTest)
- %USERPROFILE%\Documents\PragmaticWorks\LegiTest(e.g.C:\Users\jsmith\Documents\PragmaticWorks\Leg

After this search is done, the same set of directories is searched again, for legitestSettings files that have the correct **applicableTestSuiteId** attribute.

> **Note:** To find the correct **applicableTestSuiteId** for a test group, open up the file in notepad and look for the **uniqueId** attribute on the **Test group** node in the XML.

Any settings that are found in a location later in the list over-ride any that are found previously.

For example, a setting called **TargetServer** in the directory where the assembly is located is over-ridden by a setting called **TargetServer** in the user profile directory. However, a setting called **TargetServer** in a file that has the **applicableTestSuiteId** attribute correctly set and exists in the assembly directory over-rides the same setting from a file in the user profile where the **applicableTestSuiteId** is not present.

To clarify, the following is a list of locations, in increasing order of importance:

- Assembly directory without applicableTestSuiteId attribute
- %PROGRAMDATA% without applicableTestSuiteId attribute
- %USERPROFILE% without applicableTestSuiteId attribute
- Assembly directory with applicableTestSuiteId attribute
- %PROGRAMDATA% with applicableTestSuiteId attribute
- %USERPROFILE% with applicableTestSuiteId attribute

> SentryOne recommends using the Program Data location to store legitestSettings files. These directories can hold many *.legitestSettings files, the file names are not important. SentryOne Test only searches for files with the *.legitestSettings extension. We recommend renaming setting files after the tests they belong to.

Below is sample content of a settings file:

Parameter	Description
applicableTestSuiteId	Associates the collection of parameters and mappings with the test group that has that specific identifier. Only test groups with the corresponding id use this setting file.
targetElementId	Associates a parameter value to a specific element within the test group using the element's unique identifier. The targetPropertyName attribute reference's the specific property that the parameter value will replace.

Global parameters

In the previous tutorial, we parameterized only the connection strings of a test group. SentryOne Test can use parameters throughout all its actions, asserts, and assets. The Query asset is another good example where parameters would be useful.

Through the use of parameters, you can transition between many environments. Parameters help keep the testing process flowing through the various stages of development.

Global Parameters are parameters applied to all test group executions on a machine. When modifying a settings file, removing the applicableTestSuiteId creates a global settings file. The most common use would be between publishing Test assemblies to SentryOne Test.

Global parameters allow the establishment of commonly named parameters across all environments. After publishing to a new environment, future executions use that environment's settings file. The most common use would be redirecting connection strings between database deployments.

Note: SentryOne recommends using the Program Data location for global legitestSettings files. Though you can have more, we recommend only one global file per machine.

Parameter Replacement Methods

Elements, Assets, and Assertions within SentryOne Test can use parameters in one of three replacement methods :

Replacement Method	Description
Direct Property Replacement	Users can map a parameter to a property within the test group.
Parameter Token Replacement	Users can insert tokens within a property value. When using this method, SentryOne Test interprets these as a parameter. • **{{ParameterName}}**
Environment Variable Replacement	Users can insert system environment variables within a property value. When using this method, SentryOne Test interprets these as environment variables. • **%EnvironmentVariableName%**

At test execution, SentryOne Test replaces all parameters with the supplied value. This parameter replacement occurs regardless of the method used. For examples of the parameter replacement methods in use, please see the Parameter Tutorial.

⚠**Important:** Direct Property Replacement is the only method of mapping a parameter to a value that is not a string.

Formatting Parameters

Sometimes it's necessary to format our parameters. For example, consider the SQL Query:

```
SELECT * FROM [MyTable] WHERE [LastModifiedDate] >= '{{ProcessingDate}}'
```

In this example, we would like the parameter to be formatted as four digits for the year, two for the month and two for the day. We can use filters that are defined by the DotLiquid markup to format the parameters as we would like:

```
SELECT * FROM [MyTable] WHERE [LastModifiedDate] >= '{{ProcessingDate | date:"yyyyMMdd"}}'
```

Additional Information: A reference of the available filters can be found on the DotLiquid for Designers page.

Initializing Values

Initializing a Value Once per Run

One-Time Initialization for Values

Sometimes you may want a parameter to apply to all groups or tests in an assembly, and initialize it dynamically once per run. To do that, we use some custom code. To achieve this, add a new file to the project by right-clicking on the project in Solution Explorer, and then selecting **Add** > **Class...** In this example we name

the file **OneTimeInitialization**.

Within this file, we define the parameters that we'd like to initialize once only. In this example we have **ProcessingDate** that we derive from a database and **FileCount** that represents the count of files in a particular directory.

```
using PragmaticWorks.LegiTest.Runtime;
using System;
using System.Data.SqlClient;
using System.IO;namespace LegiTestProject
{
   static class OneTimeInitialization
   {
       public static Lazy ProcessingDate { get; } = new Lazy(LoadProcessingDate, true);       stat
ic object LoadProcessingDate()
       {
           var parameterProvider = new ParameterProvider(Guid.Empty, typeof(OneTimeInitialization)
);
           using (var connection = new SqlConnection(parameterProvider.GetTypedParameter("SystemCo
nnection", string.Empty)))
           {
               connection.Open();
               using (var cmd = new SqlCommand("SELECT MAX(LastProcessedDate) FROM dbo.ProcessTrac
king", connection))
               {
                   return cmd.ExecuteScalar();
               }
           }
       }       public static Lazy FileCount { get; } = new Lazy(LoadFileCount, true);       static
object LoadFileCount()
       {
           var parameterProvider = new ParameterProvider(Guid.Empty, typeof(OneTimeInitialization)
);
           return Directory.GetFiles(parameterProvider.GetTypedParameter("WorkingPath", string.Emp
ty)).Length;
       }
   }
}
```

Here we have two properties **ProcessingDate** and **FileCount** both of type Lazy. This is a system type that s upports loading a value once and once only. There are also methods, one for each property, that are run on ce to initialize the object the first time it's requested.

We are using parameters defined in the test group '**SystemConnection**' and **WorkingPath**. To do this we n eed to create a parameter provider:

```
var parameterProvider = new ParameterProvider(Guid.Empty, typeof(OneTimeInitialization));
```

We can then get the value of the parameter by calling the **GetTypedParameter** method:

```
parameterProvider.GetTypedParameter("WorkingPath", string.Empty)
```

In the examples above, **LoadProcessingData** connects to the SQL Server using the connection string defin ed in the **SystemConnection** parameter, and then we run the query to select the MAX value of **LastProces sedDate** from the **dbo.ProcessTracking** table. The **LoadFileCount** method gets a list of the files from the directory defined in the **WorkingPath** parameter and returns the count.

Using the Initialized Values in Tests

To use these one-time values in our tests, use the **BeforeTest** method to set the values into the resources.

```
```

Note: You need to do this in each group where you want to use those values.

To do this, copy content similar to the following into the user file generated for each test group:

```
namespace LegiTestProject.TestGroup1_Output
{
   using System.Collections.Generic;

   // This file provides a point at which partial methods can be implemented to augment tests.
   // The content of this file is preserved when the test group is regenerated.
   public partial class TestGroup1
   {
       static partial void BeforeTest(string testName, Dictionary testResources, ref bool cancel
)
       {
           testResources["ProcessingDate"] = OneTimeInitialization.ProcessingDate.Value;
           testResources["FileCount"] = OneTimeInitialization.FileCount.Value;
       }
   }
}
```

This code makes the one-time values available as a resource to the test, and they can be referenced like a p arameter, using the {{resourceName}} syntax. For example, if you wanted to use the ProcessingDate value i n a SQL query, you would enter the query as:

```
SELECT * FROM [MyTable] WHERE [LastModifiedDate] >= '{{ProcessingDate | date:"yyyyMMdd"}}'
```

Using this approach, you can set up and use values that are initialized once and only once throughout your test group.