Visual Studio Extension Overview

Last Modified on 07 January 2020

The SentryOne Test Visual Studio Extension allows you to create tests that verify the quality of your data integrity. Using SentryOne Test, you can create testing implements for your data storage solutions that automate the testing procedure. You can create a test once, and reuse the test any time there is a change in the data. For example, if you were transporting data from SQL Server into a data warehouse, you could create a test that verifies the information in a table before the transfer, and after the transfer. You could also create a test that compares the data from the SQL Server and data warehouse sources to ensure that the transfer was done smoothly and accurately.

SentryOne Test also allows you a great deal of granularity through this automated process. You can create tests that test SSIS packages at runtime, or to verify that the packages are successful. You can even use the **Package Wizard** to create a test for each individual task on the package instead of running the package as a whole. The SentryOne Test Visual Studio Extension enables you to automate your testing procedures and ensure the integrity of your data.

Note: The SentryOne Test Visual Studio Extension can be used to compare external sources to an internal database.

Choosing a Test Framework

SentryOne Test is a code generation tool at its core. The tests, groups, and assets that you configure become code in the form of unit tests. There are multiple test frameworks available, each with their own benefits.

Note: SentryOne Test currently supports **NUnit 2**, **NUnit 3**, and **MSTest**.

SentryOne recommends NUnit as a better choice in most scenarios. Some of the specific benefits include:

Benefits of NUnit	**Explanation**
More powerful model for supporting Data Driven Testing	While Data Driven Testing is supported in both NUnit and MSTest, in MSTest your data driven query can be executed at unexpected times because of limitations in the framework.
More powerful assert model	Assertions are a key aspect of testing. They are the part of the test where the code verifies that the result of the test (the actual result) matches the desired result (the expected result). NUnit has a lot more flexibility in this area because the generated code is simpler and it's easier to write your own assertions.
Broader compatibility	NUnit tests operate in a way that means that there are fewer problems with execution of tests. Testing SSIS packages is one key area where NUnit is superior—packages loaded at the group level have issues under MSTest—while no similar problems occur using NUnit.

Note: Both versions of NUnit carry these benefits, and the choice between NUnit 2 and NUnit 3 is available to fit in with your organization's existing usage. If no prior usage exists, SentryOne recommends using NUnit 3.

Project Organization

SentryOne Test projects are organized similarly to C# projects. There are a few different types of files that can be added, and are handled specially within SentryOne Test projects:

File Type	Description
LegiTest group (.legitestGroup)	These files define a single SentryOne Test group. SentryOne Test, in previous releases, used suite files (.legitest) which contained many groups. To allow a better organization of projects the group files were added. This has particular benefits for any teams using source control. With groups in individual files it becomes much less likely that a complicated merge is required when more than one person is working on a project. A group file will emit a single C# class that contains the tests that are defined.
LegiTest settings file (.legitestSettings)	Settings files come in two types; settings and project settings. **File Type** | **Description** **Settings files** | Settings files contain parameters and their values. **Note:** Projects can contain more than one setting file if desired. | Project settings files define both the server with which the project

File Type	Description
Project Settings files	is associated and the test framework type. **Note:** Projects can only contain one project settings file.
Asset file (.asset)	Asset files are used to define assets that are generally useful across multiple groups. These asset files are the same as assets defined within groups, and tests, but are defined at the project level. Asset files have two benefits; they are easier to change in isolation (for example, changing one asset doesn't mean that the group file has to be checked in to source control) and they give us a better way to organize the data within our projects. For more information about Assets in SentryOne Test, see the Assets section.

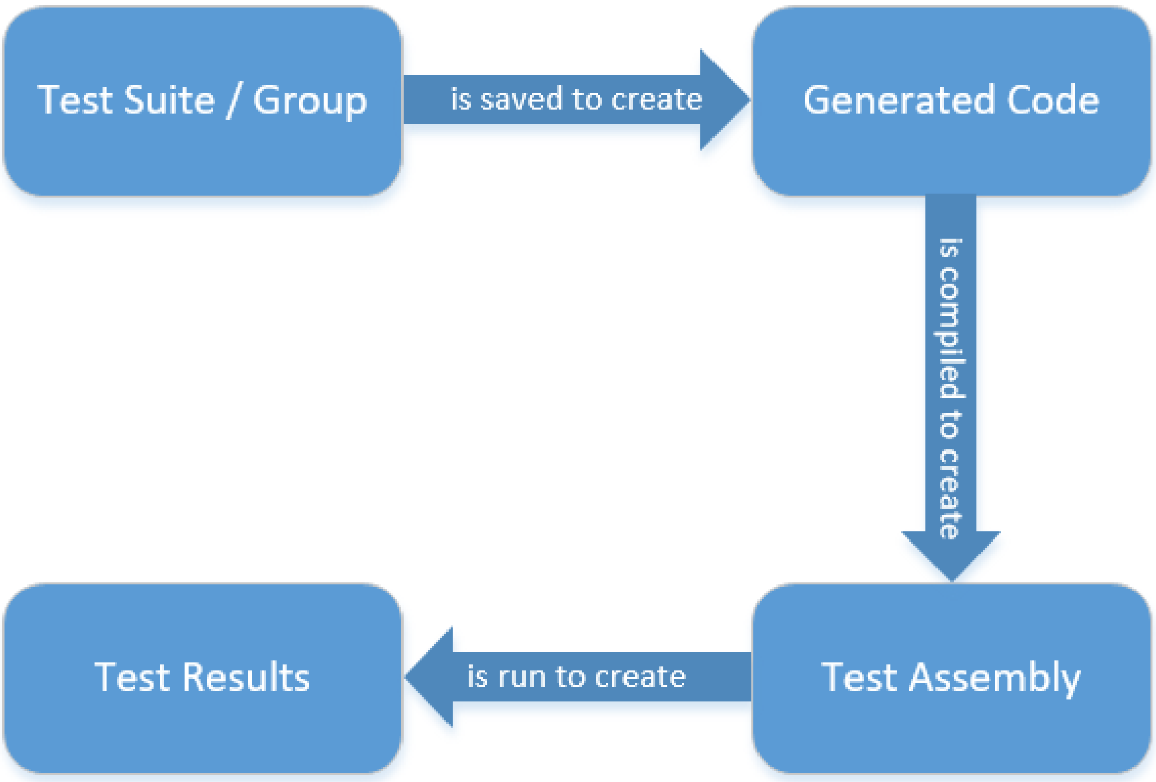
Note: Folders in the solution can be used to group all types of items together. A large testing project with many groups, assets, and settings can become hard to manage unless it's well organized.

Output

Understanding Generated Output

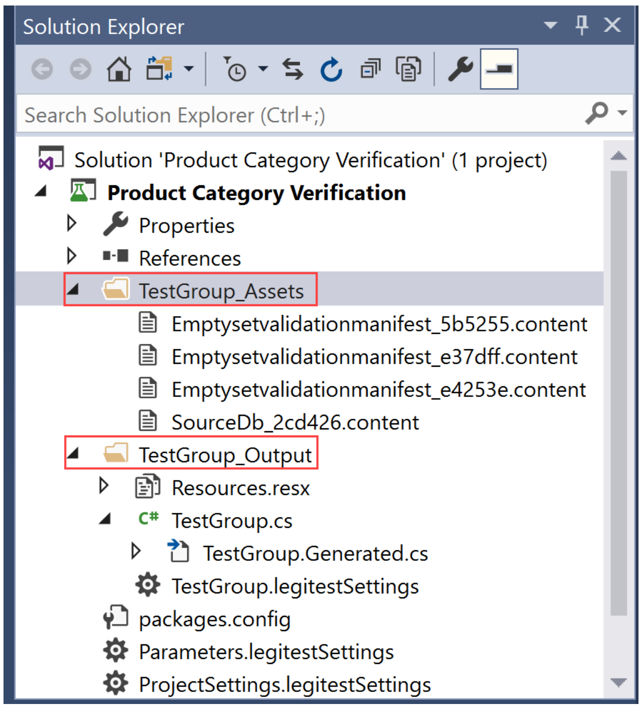
The generated output from SentryOne Test is C# code, along with some additional files. These files are used as the input to the C# compiler that generates an assembly with unit tests. The assembly can then be executed within Visual Studio, SentryOne Test, or a third party product that supports standard .NET unit tests. This topic aims to clarify the process of moving from a design time test group to a run time assembly.

The process follows the diagram below. When a test group is saved, the code is generated. Then the code is compiled by the C# compiler into a test assembly. When the assembly runs, the tests execute, and the results for the tests are available. When running within Visual Studio, the test results appear within the **Test Explorer** window. You can also publish results to SentryOne Test Web Portal.



Generated output

Looking at the output generated by a test group called **TestGroup**, we might see the following in Visual Studio Solution Explorer:



Solution 'Product Category Verification' (1 project)
Product Category Verification
Properties
References
TestGroup_Assets
Emptysetvalidationmanifest_5b5255.content
Emptysetvalidationmanifest_e37dff.content
Emptysetvalidationmanifest_e4253e.content
SourceDb_2cd426.content
TestGroup_Output
Resources.resx
TestGroup.cs
TestGroup.Generated.cs
TestGroup.legitestSettings
packages.config
Parameters.legitestSettings
ProjectSettings.legitestSettings

There are two main folders; the TestGroup_Assets and the TestGroup_Output folder, which are described in the table below.

Warning:

The **_Assets** and the **_Output** folders are used to store test infrastructure files and are not intended to contain additional files. Any non SentryOne Test generated files are removed during the regeneration of the SentryOne Test group(s).

Folder	Description
_Assets Folder	The **_Assets folder** contains the content for your assets at design time. For example, if we have a **Delimited Content** asset, then the content file contains the data that makes up the delimited content. Most assets are made up of two types of data; the metadata and the data itself.

Data Type	Description
Metadata	Metadata is usually small, and is stored within the group file to display the asset type once a SentryOne Test group is opened. It can be a list of output columns or settings, for example.
The data itself	The data itself can be large, and is stored in a separate file so that large amounts of data aren't loaded as soon as you open the file.

For more information on the **Assets** available within SentryOne Test, see the Assets section.

Folder	Description
_Output Folder	The **_Output folder** contains the code that's generated from your group file, along with some other files. The following files are contained in the **_Output folder**:

File	Description
Resources.resx	This file contains the same content as the **_Assets folder**, but is encoded in a way that the C# compiler understands. The asset content becomes part of the compiled assembly and is available to the tests when they are run.
TestGroup.cs	This file is the **user file**. It's usually blank, but code can be added here that's integrated with the generated code. **Note:** You can create methods that are called by the generated test code to perform any action that you want.
TestGroup.Generated.cs	This is the main output file where the code that forms the bulk of the tests is stored. Each test, action, and assertion has a method defined within this file.
TestGroup.legiTestSettings	This file contains the details of any parameters defined, and is used by the tests at runtime to control any parameterized elements. For more information on using parameters within tests, see the Using Parameters topic.

Warning:

The generated files are re-created every time you save a SentryOne Test group.

Any changes made directly to the **.Generated.cs** file will be lost.

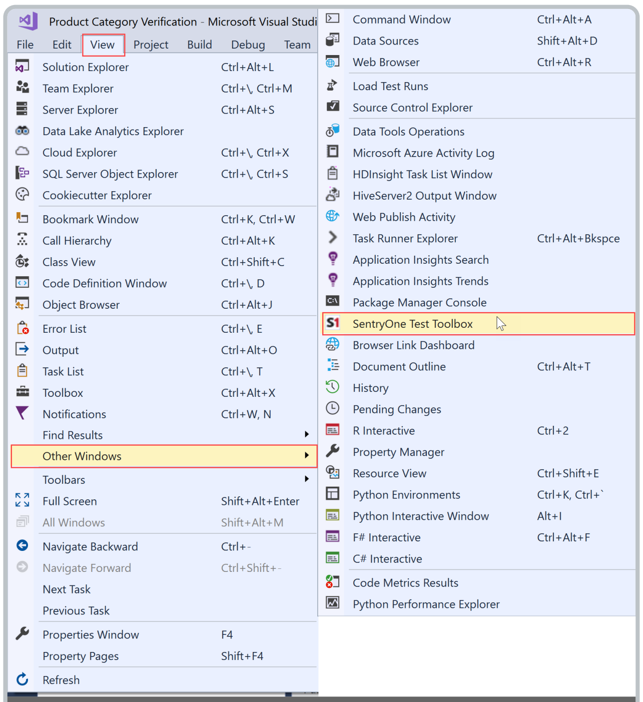
Design

Design Area

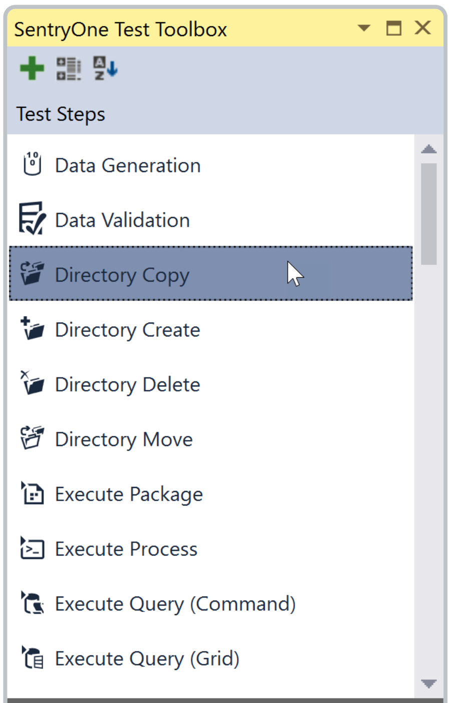
The design area is where you add your test elements. The area is divided into the different test flow steps and changes depending on whether a single test, or a test group is selected. Test elements are executed first from left to right, then top to bottom. Test elements can be placed on the design area by:

- Dragging and dropping from the toolbox
- Double clicking an item in the toolbox
- Right clicking the design area and then using the quick add menu

Open the toolbox by selecting **View** > **Other Windows** > **SentryOne Test Toolbox** on the Visual Studio toolbar.

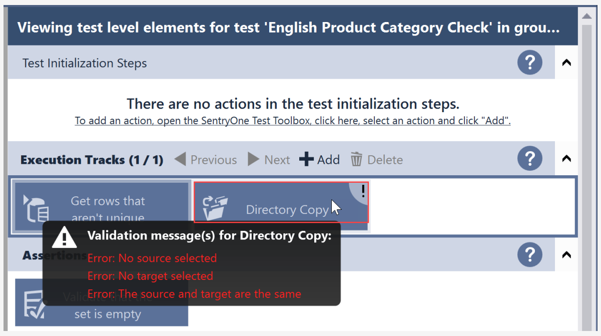

Product Category Verification - Microsoft Visual Studi
File Edit View Project Build Debug Team
Solution Explorer Ctrl+Alt+L
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Data Lake Analytics Explorer
Cloud Explorer Ctrl+\, Ctrl+X
SQL Server Object Explorer Ctrl+\, Ctrl+S
Cookiecutter Explorer
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O
Task List Ctrl+\, T
Toolbox Ctrl+Alt+X
Notifications Ctrl+W, N
Find Results
Other Windows
Toolbars
Full Screen Shift+Alt+Enter
All Windows Shift+Alt+M
Navigate Backward Ctrl+-
Navigate Forward Ctrl+Shift+-
Next Task
Previous Task
Properties Window F4
Property Pages Shift+F4
Refresh
Command Window Ctrl+Alt+A
Data Sources Shift+Alt+D
Web Browser Ctrl+Alt+R
Load Test Runs
Source Control Explorer
Data Tools Operations
Microsoft Azure Activity Log
HDInsight Task List Window
HiveServer2 Output Window
Web Publish Activity
Task Runner Explorer Ctrl+Alt+Bkspce
Application Insights Search
Application Insights Trends
Package Manager Console
SentryOne Test Toolbox
Browser Link Dashboard
Document Outline Ctrl+Alt+T
History
Pending Changes
R Interactive Ctrl+2
Property Manager
Resource View Ctrl+Shift+E
Python Environments Ctrl+K, Ctrl+`
Python Interactive Window Alt+I
F# Interactive Ctrl+Alt+F
C# Interactive
Code Metrics Results
Python Performance Explorer

Open the SentryOne Test Toolbox



SentryOne Test Toolbox

Note: Once an item is placed, an **!** icon may appear on the top right of the element's box. Hover over the element to see any potential errors with the element.


Viewing test level elements for test 'English Product Category Check' in grou...

Test Initialization Steps

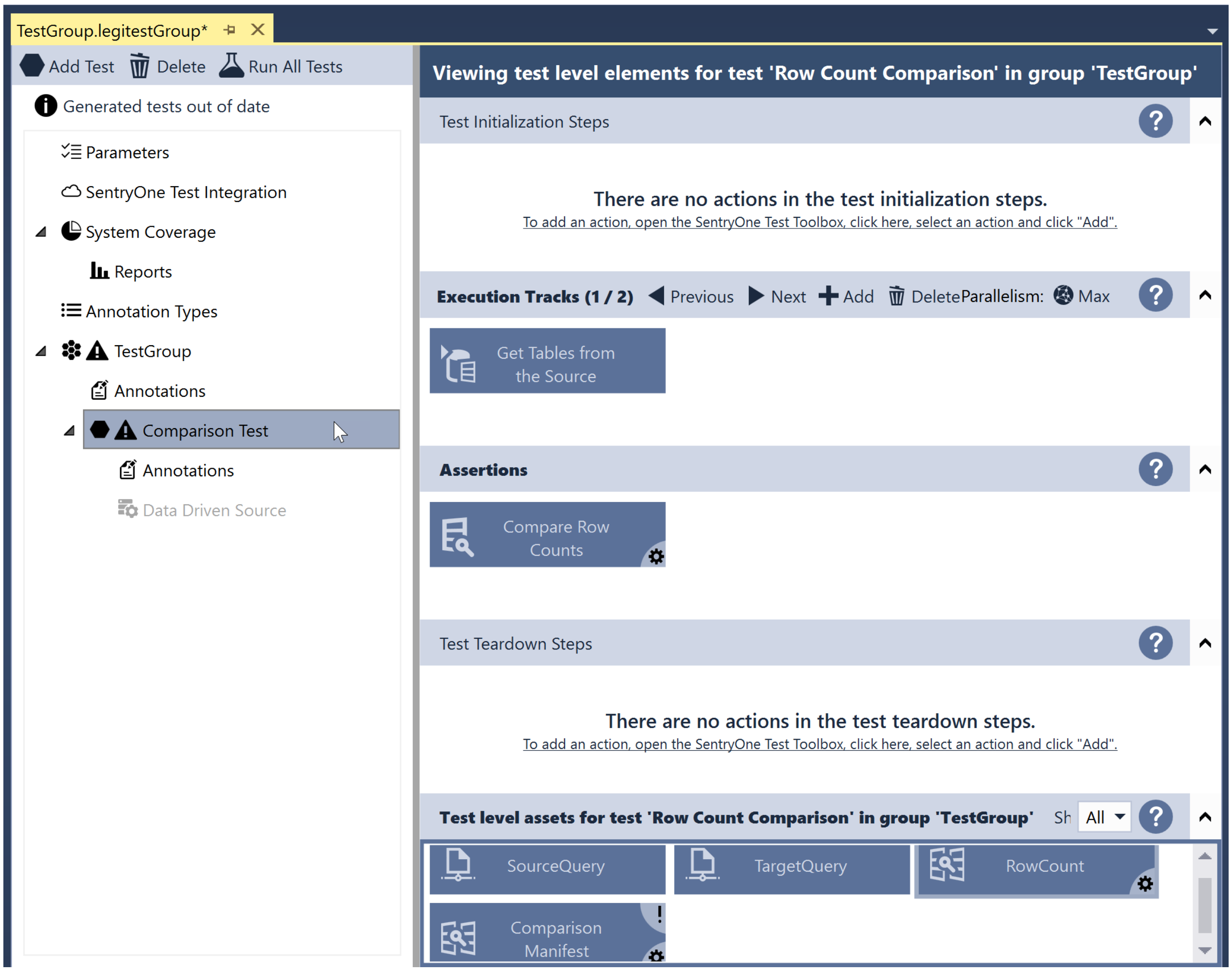
There are no actions in the test initialization steps.
To add an action, open the SentryOne Test Toolbox, click here, select an action and click "Add".

Execution Tracks (1 / 1)

Validation message(s) for Directory Copy:
Error: No source selected
Error: No target selected
Error: The source and target are the same

Once the tests have been set up, saving the project automatically generates the test code. Once the project has been built, the test is discoverable in the **test explorer** window of Visual Studio. For more information on the flow from design to execution, see Understanding Generated Output under the Output tab in this article.

A simple comparison test might look like the following:


TestGroup.legitestGroup*

Add Test | Delete | Run All Tests

Generated tests out of date

- Parameters
- SentryOne Test Integration
- System Coverage
  - Reports
- Annotation Types
- TestGroup
  - Annotations
  - Comparison Test
    - Annotations
    - Data Driven Source

Viewing test level elements for test 'Row Count Comparison' in group 'TestGroup'

Test Initialization Steps

There are no actions in the test initialization steps.
To add an action, open the SentryOne Test Toolbox, click here, select an action and click "Add".

Execution Tracks (1 / 2) Previous Next Add Delete Parallelism: Max

Get Tables from the Source

Assertions

Compare Row Counts

Test Teardown Steps

There are no actions in the test teardown steps.
To add an action, open the SentryOne Test Toolbox, click here, select an action and click "Add".

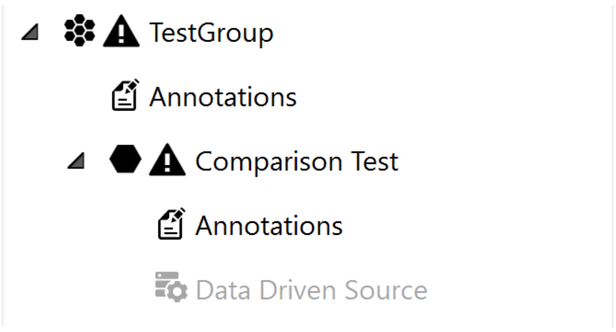
Test level assets for test 'Row Count Comparison' in group 'TestGroup' Sh All

SourceQuery | TargetQuery | RowCount | Comparison Manifest

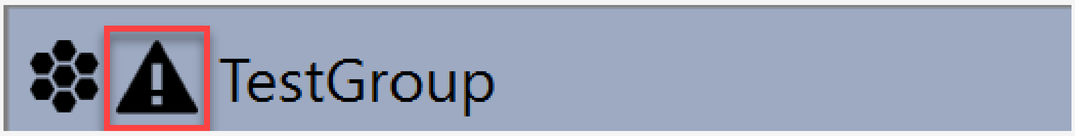
The tree nodes on the left are covered in more detail in the following sections:

Parameters	Parameters
SentryOne Test Integration	SentryOne Test Integration
◢ System Coverage	System Coverage
Reports	Reports
Annotation Types	Annotation Types

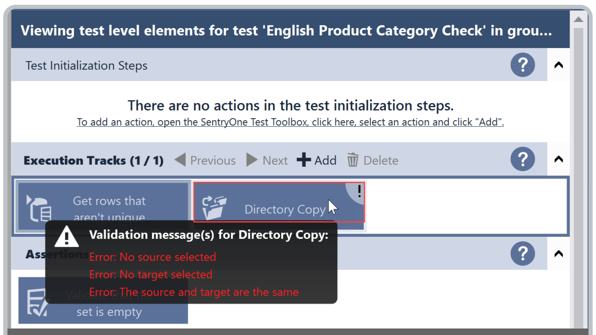
Under that we have the node for the test group, annotations for the group, the test itself, annotations for the test and the data driven configuration for the test.



Note: The test and group nodes have a warning icon next to them if some elements require attention.



Display a brief description of the errors by hovering over the warning icon, or view the errors in more detail by double-clicking on the icon to open the **Inspect errors** window:

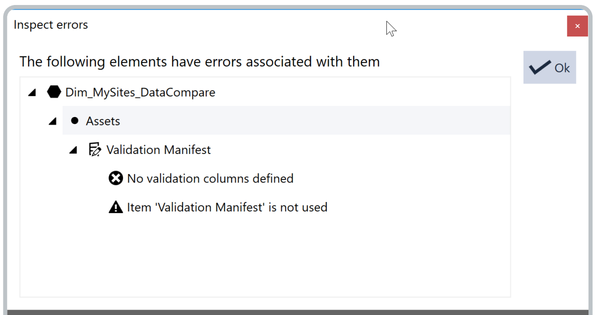


Viewing test level elements for test 'English Product Category Check' in grou...

There are no actions in the test initialization steps.
To add an action, open the SentryOne Test Toolbox, click here, select an action and click "Add".

Validation message(s) for Directory Copy:
Error: No source selected
Error: No target selected
Error: The source and target are the same

Hover over a Test Element icon

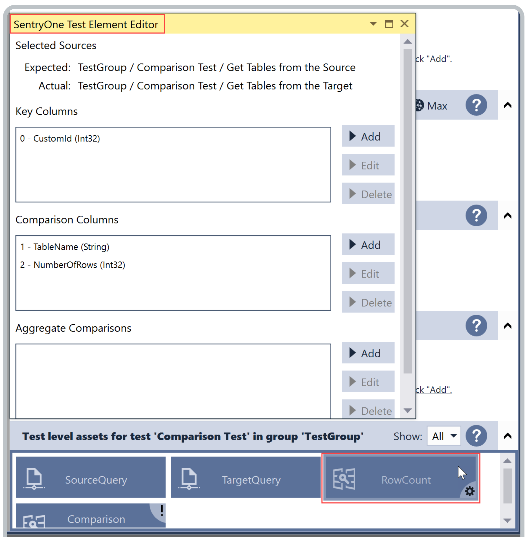


The following elements have errors associated with them

- Dim_MySites_DataCompare
  - Assets
    - Validation Manifest
      - No validation columns defined
      - Item 'Validation Manifest' is not used

Open the Inspect errors window

Double clicking on the icon of an element opens its editor, and double-clicking on the text starts editing the name of the element.


SentryOne Test Element Editor

Selected Sources

Expected: TestGroup / Comparison Test / Get Tables from the Source

Actual: TestGroup / Comparison Test / Get Tables from the Target

Key Columns

0 - CustomId (Int32)

Comparison Columns

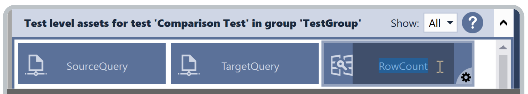
1 - TableName (String)

2 - NumberOfRows (Int32)

Aggregate Comparisons

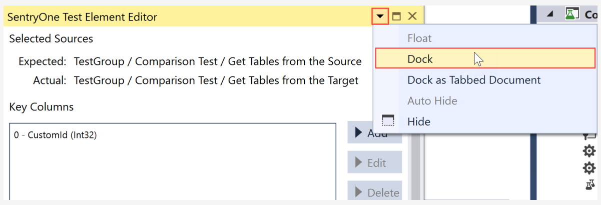
Test level assets for test 'Comparison Test' in group 'TestGroup'   Show: All

Double Click a Test Element to open the SentryOne Test Element Editor


Test level assets for test 'Comparison Test' in group 'TestGroup'   Show: All

Double Click a Test Element text to rename it

Note: The **SentryOne Test Element Editor** is a dockable window within Visual Studio, so you may find it convenient to dock it into a position where it's permanently visible.


SentryOne Test Element Editor

Selected Sources

Expected: TestGroup / Comparison Test / Get Tables from the Source

Actual: TestGroup / Comparison Test / Get Tables from the Target

Key Columns

0 - CustomId (Int32)

Note: Some elements have custom editors that present a more convenient user interface for configuring the element.

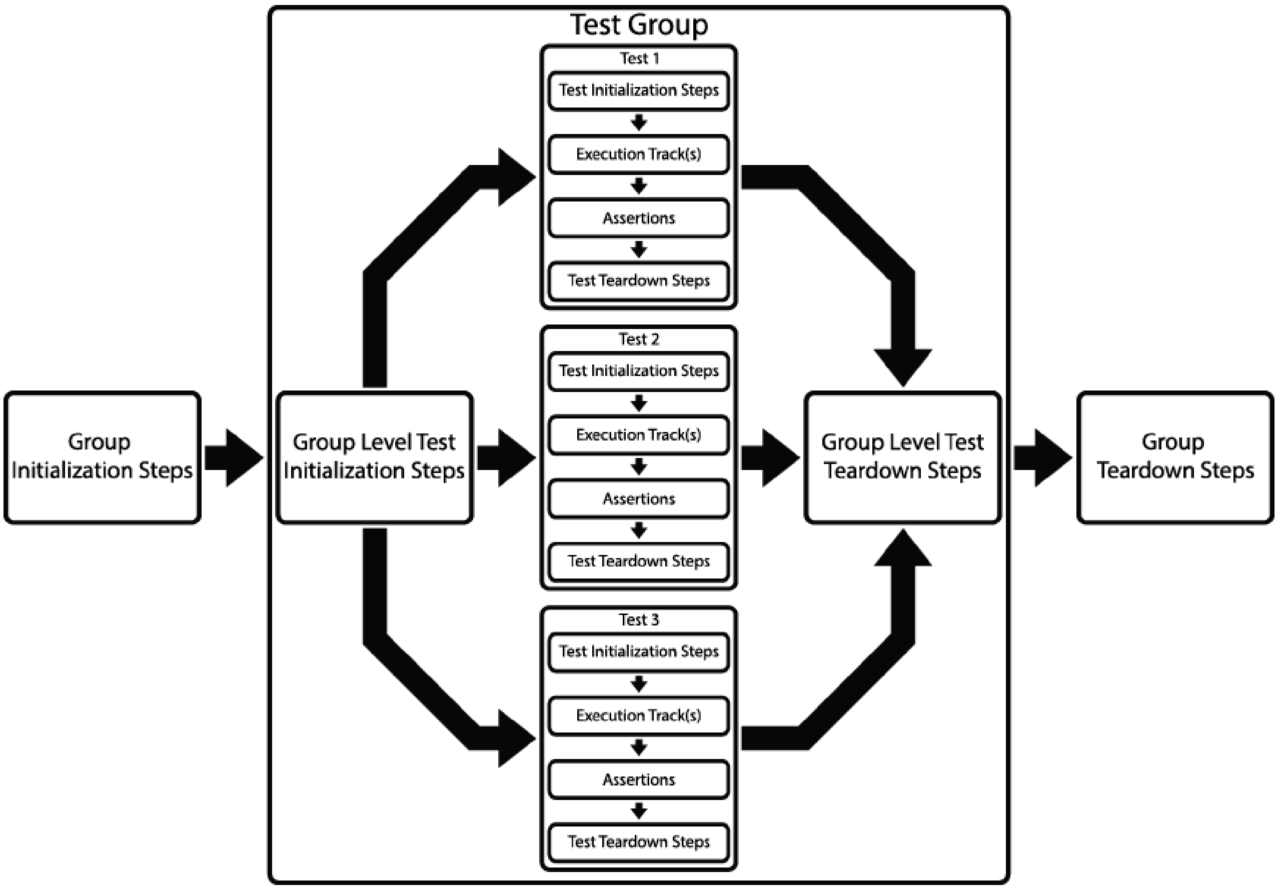
Test Flow

Test Flow

A SentryOne Test Project organizes all test elements into three scopes:

Scope	Description
Project	A test project can contain assets and test groups.
Groups	Groups can contain assets, and multiple tests. Groups can manage the Group and Test Level Initialization, and Group and Test Level Teardown steps.
Tests	Tests can contain assets and manage the Test Initialization Steps, Execution Tracks, Assertions, and Test Teardown Steps.

Note: SentryOne Test now supports group files being added directly within solution explorer. While existing suites are still supported, individual SentryOne Test groups offer greater flexibility in use, and are the preferred method of building tests.



Test Component	Description
Group Initialization Steps	The **Group Initialization Step** executes only once at the beginning of the entire test group. This step is the first to run before any tests are actually executed. This step is the first to run before any tests within the group are actually executed. Each group has its own **Group Initialization Step**. This step configures everything needed to prepare the test environment for the group execution.
Group Level Test Initialization Steps	This steps executes once at the beginning of every test in the group. It runs right before the actual test begins executing. Each group has its own **Group Level Test Initialization** step. This step configures everything needed to prepare the test environment before each test execution.
Test Initialization Steps	This step executes once at the beginning of a single test. Each test contains its own **Test Initialization step**. This step configures unique elements needed to prepare this test for execution.
Executions Tracks	**Execution Tracks** contain the actual elements that compose the test. Each test can hold any number of tracks. By default, the test executes as many tracks as the test environment can handle. You can modify this behavior to only run a set number of tracks at the same time.
Assertions	**Assertions** contain test validations that confirm whether a test passes or fails. Assertions run immediately after all execution tracks for the current test are completed. Assertions compare expected data to actual data generated during the execution tracks. The results of these data comparisons dictate whether the test passes or fails.
Test Teardown Steps	This step executes once at the end of a single test. Each test contains its own **Test Teardown step**. This step configures unique elements that are needed to clean the test environment after test execution.

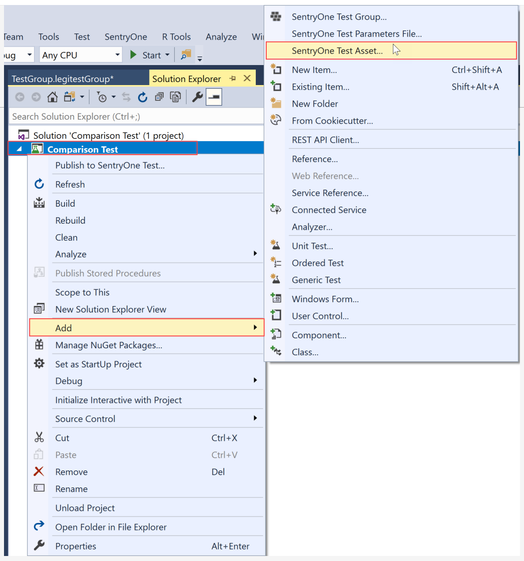
Test Component	Description
Group Level Test Teardown Steps	This steps executes once at the end of every test in the group. It runs right after the actual test finishes executing. Each group has its own **Group Level Test Teardown step**. This step configures everything that's needed to clean the test environment after each test completes.
Group Teardown Steps	The **Group Teardown Step** executes only once at the end of the entire test group. This step is the last to run after all tests in the test group execute. Each group has its own **Group Teardown**. This step configures everything needed to clean the test environment once group execution completes.

Key Concepts

Assets, Actions, Asserts, and Resources

Four key concepts in SentryOne Test are Assets, Actions, Asserts, and Resources.

Concept	Explanation
	Assets are items that are created at design time and used at runtime. For example, you might have a query or a reference to an SSIS package. These items are configured when you are creating tests and used when the tests run. There are many different types of assets available in SentryOne Test, and you can read about those in the Assets section. **Note:** Assets can use parameters; you may want to configure a connection string using the parameter system. Parameters are evaluated at the point that the asset is used. Assets can be part of a test group, or a part of the test project itself. To add an asset to the project, right click on the **project node** in **solution explorer**, and then select **Add** > **SentryOne Test Asset**.

Assets Concept	Explanation
	 Team Tools Test SentryOne R Tools Analyze Any CPU Start TestGroup.legitestGroup* Solution Explorer Search Solution Explorer (Ctrl+;) Solution 'Comparison Test' (1 project) Comparison Test Publish to SentryOne Test... Refresh Build Rebuild Clean Analyze Publish Stored Procedures Scope to This New Solution Explorer View Add Manage NuGet Packages... Set as StartUp Project Debug Initialize Interactive with Project Source Control Cut Ctrl+X Paste Ctrl+V Remove Del Rename Unload Project Open Folder in File Explorer Properties Alt+Enter SentryOne Test Group... SentryOne Test Parameters File... SentryOne Test Asset... New Item... Ctrl+Shift+A Existing Item... Shift+Alt+A New Folder From Cookiecutter... REST API Client... Reference... Web Reference... Service Reference... Connected Service Analyzer... Unit Test... Ordered Test Generic Test Windows Form... User Control... Component... Class... **Note:** Project-level assets can be placed in subfolders of the project to aid in organization, but this doesn't affect their use. Assets at the project level are available to all test groups, despite the folder they are placed in. Assets that are defined within a test group can only be used within that group. Additionally, assets defined at the test level may only be used by that test.
Actions	Actions produce some kind of outcome. For example, they may execute a query to return a row count, process a cube, or execute a package. When an action returns information, such as a row count or data set (grid), it's stored in a resource. Actions often use assets or resources as part of their input (for example, an action to load a grid uses a connection asset and a query asset - an action to filter a previously loaded grid uses the resource). **Note:** Actions can be used for set-up and teardown at both the group and test level because actions can be placed in any of the execution areas except for the assert section. For the list of available actions, see the Actions section.
	Asserts are used to test the results of actions. For example, we might use an integer assert to compare the result of a query action that returns a row count with some predetermined value. Asserts compare

Concept	Explanation
	the values in resource keys to each other, or to a static value entered into the Assert.
Asserts	**Note:** A test that doesn't contain any asserts always yields an **Inconclusive** result. It's important to make sure that all of your tests actually include an assert to determine whether the test passes or fails. Asserts can only be placed in the assert section of a test. For the list of available assertions, please see the Asserts section.
Resources	Resources are items that are available only at runtime. For example, when you execute a query you may get a result back that you wish to compare. Resources are referred to by their **resource key** (a string that uniquely identifies that resource). Resources are usually loaded by actions, (the exception being data driven testing) and can be referenced by other actions or assertions. You can view dependent elements for any element. This displays the elements that follow and uses the resource(s) generated. **Note:** One additional resource is available for each column of data used as the data source when data driven testing is used. For more information, see the data driven testing topic. Resources can also be used in place of parameters by elements that follow them. For example, an execute scalar query might get back a resource called **CutOffDate**. A query might follow where the query is defined similarly to: • **SELECT OrderId FROM Orders WHERE OrderDate > ''** Because parameters are evaluated when an asset is used, the resource **CutOffDate** would be included in the query and used as a filter.