CS8 Development Library UI

User Manual

Spirent

541 Industrial Way West
Eatontown, NJ 07724 USA

Email: sales@spirent.com
Web: http://www.spirent.com

AMERICAS 1-800-SPIRENT • +1-818-676-2683 • sales@spirent.com
EUROPE AND THE MIDDLE EAST +44 (0) 1293 767979 • emeainfo@spirent.com
ASIA AND THE PACIFIC +86-10-8518-2539 • salesasia@spirent.com

This manual applies to CS8 Development Library Version 1.30 or higher.

Page Part Number: 71-006746, Version A1

Table of Contents

1. Introduction

1.1. Overview

This document provides information on the CS8 Development Library UI module. You will become familiar with using the CS8 Development Library UI by following step-by-step procedures.

1.2. Intended Audience

This manual is intended for those who have a working knowledge of wireless communication equipment, and the automated testing of mobile devices. It is assumed that you are familiar with the *Test Manager* GUI environment. Those who are unfamiliar with the Test Manager should refer to the *Test Manager User Manual* before proceeding.

1.3. Before You Get Started

Before getting started with this guide, install all software and power up the test system. The controller PC should have *Test Manager Test Executive*, Development Library Services and their respective prerequisites installed. For Development Library Services prerequisites such as the B-series platform, consult the *Development Library Services User Manual* and *Platform User Manual*. These documents can be found under the Test Manager *Help* menu.

1.4. Security Information

The CS8 Development Library UI is shipped with the appropriate dongle and software/hardware security passwords configured.

To verify the security information:

1. Open Test Manager and select **Help›About**.
   The *About Test Manager* window displays, as shown in Figure 1-1.

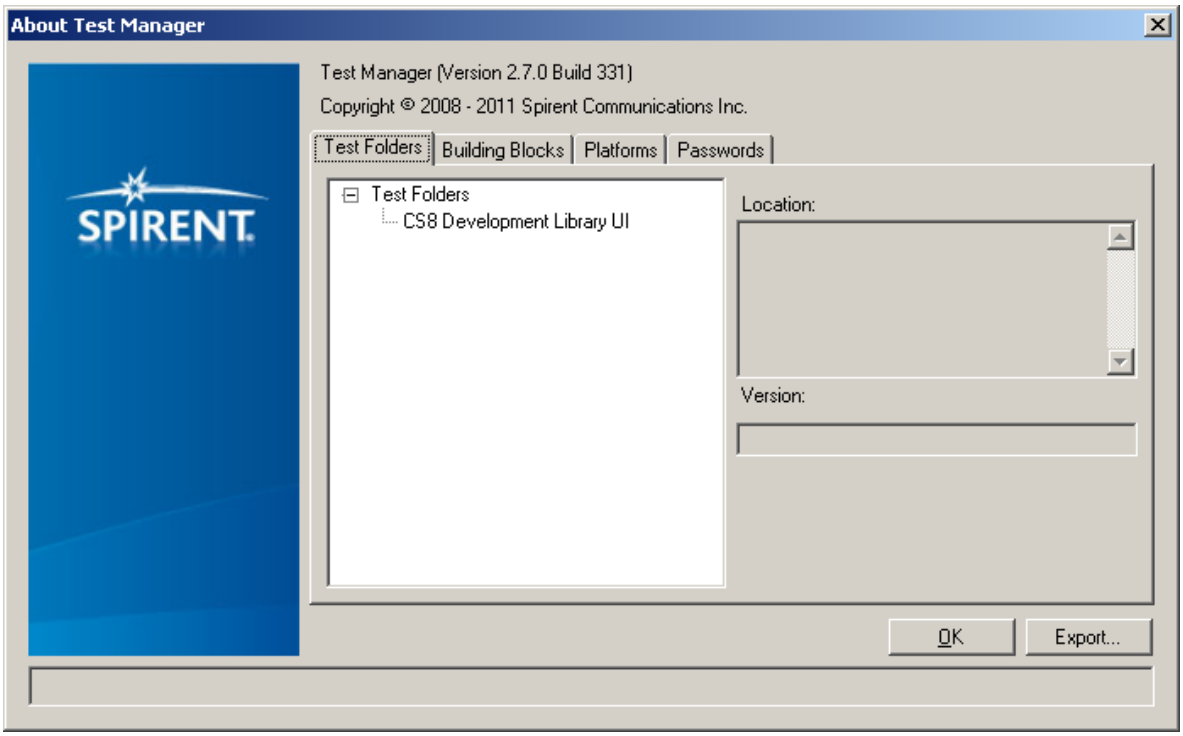

Figure 1-1: Test Manager About Window

2. Select the **Passwords** tab.
3. Under **Installed Features**, confirm the CS8 Development Library UI is installed, as shown in Figure 1-2.

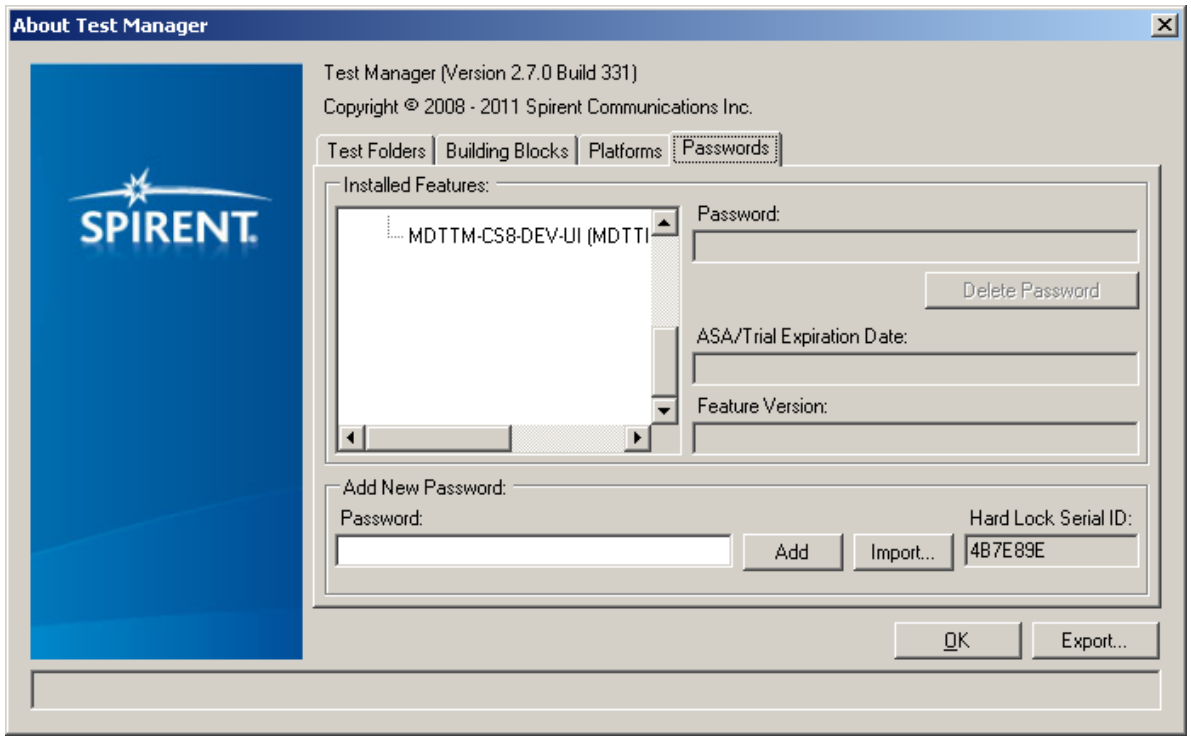

Figure 1-2: About Test Manager Window – Installed Features

Password authentication is a prerequisite to running any tests supported by these Test Packs. This password is tied to security information provided by the USB hard-lock dongle that comes with the module installation.

If you have any questions or concerns, contact Spirent Technical Support at support.spirent.com, or by phone at 1-800-SPIRENT.

1.5. Accessing Documentation

There are two ways to access this document from the Controller PC:

1. Windows Explorer
2. Test Manager

1.5.1. Accessing Documentation from Windows Explorer

Access this manual offline by opening the *CS8 Development Library UI* folder and clicking the **User Manual.pdf** shortcut, as shown in Figure 1-3.

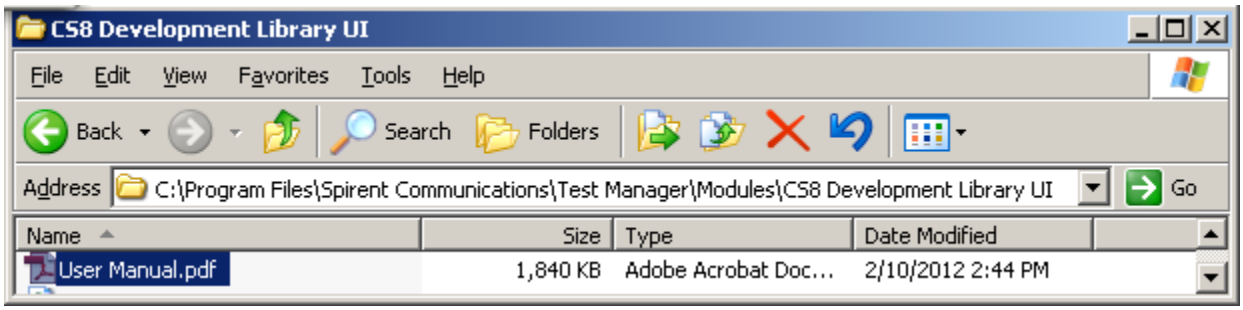

Figure 1-3: Accessing the Manual from Windows Explorer

1.5.2. Accessing Documentation from Test Manager

Access this Manual from the Test Manager menu by selecting **Help›Test Folders›**CS8 Development Library UI**›User Manual**, as shown in Figure 1-4.

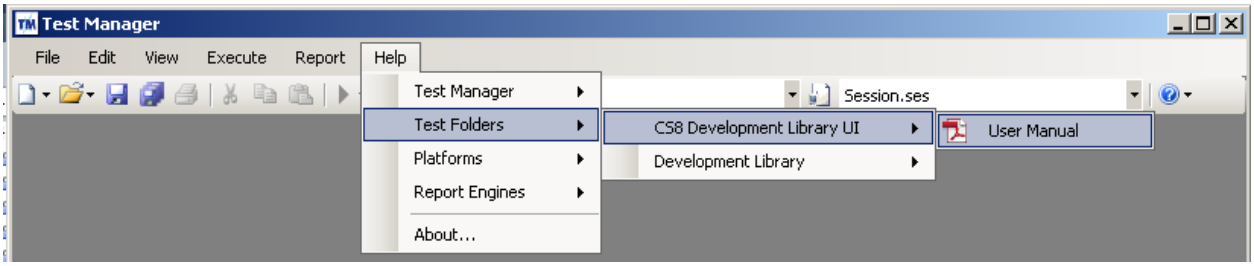

Figure 1-4: Accessing the Manual from Test Manager

2. Using the CS8 Development Library UI

2.1. Overview

This chapter provides a high-level overview of the key concepts involved in configuring the CS8 Development Library UI and associated test packs.

This includes the following:

- Test Case Structure
- Step Structure
- Step Execution and Error Handling Options

2.2. Description

The CS8 Development Library UI Module is an integrated software component that allows you to perform interactive and automated testing of an UE device.

The CS8 Development Library UI module test cases run with Test Manager software. This provides an easy way to automate test sessions with analysis and reporting capabilities.

CS8 Development Library UI provides a Graphic User Interface (GUI) and an automated Development Library Services API that abstracts the interfacing to the B-Series platform instruments and UE-oriented peripherals.

There are two ways to open a CS8 Development Library UI test case:

1. Double-click the **CS8 Development Library UI** icon on the desktop, as shown in Figure 2-1.
2. Navigate to: **Start›All Programs›Spirent Communications›CS8›CS8 Development Library UI**, as shown in Figure 2-2.

Figure 2-1: CS8 Development Library Desktop Icon

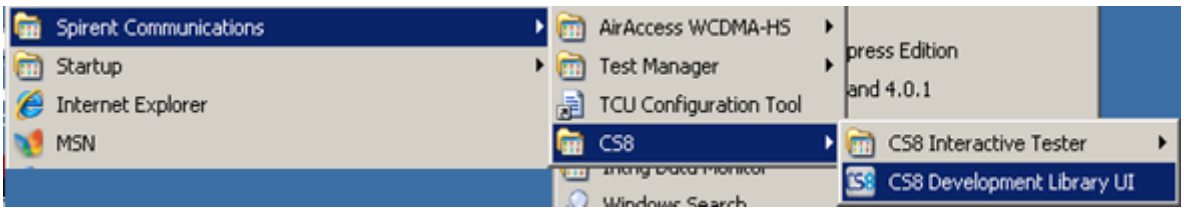

Figure 2-2: CS8 Development Library UI Menu Navigation

After the Test Manager executive GUI is opened, select the **Test Folders** tab, double-click the **CS8 Developer Template** and select the first test case in the Suite Editor, as shown in Figure 2-3.

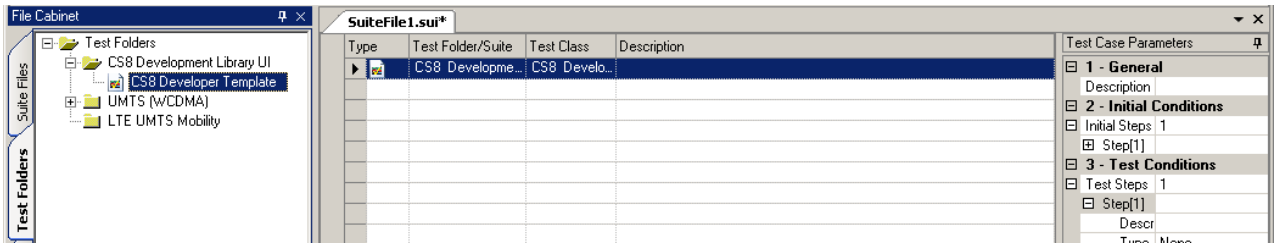

Figure 2-3: Opening a CS8 DL UI Test Case

2.3. Test Case Structure

A test case is divided into five sections, as shown in Figure 2-4.

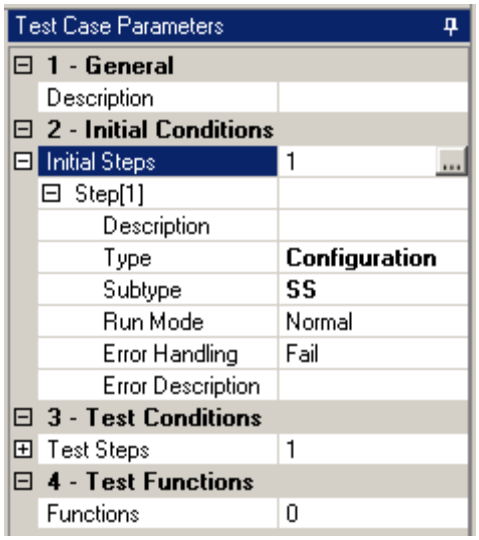

Figure 2-4: Test Case Sections

1. **General**: Contains a short description of the intended functionality of the test case.
2. **Initial Conditions**: Contains steps executed prior to call processing being enabled. The first step is System Settings (SS) used to configure multiple network settings. These configurations can then be shared among different test cases and suites through an external **.ncf** file.

3. **Test Conditions**: Contains steps after the call processing has started and constitutes the main test flow of the case.
4. **Functions**: Contains the set of functions which are steps grouped by functionality. These steps can be accessed from any point in the Initial, Test, or Function sections of the test case.
5. **Test Criteria**: Contains the settings for the pass/fail criteria when test case is executed more than once for statistical confidence purposes.

2.4. Step Structure

APIs executed as steps have several common parameters and a number of proprietary parameters specific to each one of them, given their respective functionalities. Common step properties determine parameters shared among all the steps.

Step Parameter	Description
Description	• Short description of what this specific instance of the step (with its parameters) is functionally intended to perform.
Type	• The type of step per nomenclature described in 2.5
Subtype	• The specific type (class) of the step as described in 2.6
Run Mode	• Mode of execution of the step
Error Handling	• Mode of handling the step failures
Error Description	• In case the step fails, a custom message that gives additional info on context or possible/probable cause

2.5. Step Types

The APIs executed as steps, shown in Figure 2-5, are classified into several categories based on their perceived usage and impact on the system. They reflect the API hierarchy of the Development Library Services and other compatible API assemblies.

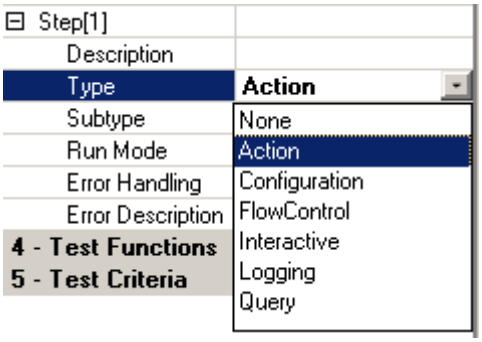

Figure 2-5: API Step Types

1. **None**: This is an empty step used as placeholder until you select a step type, subtype, and other parameters. During run-time it has no functional effect.
2. **Configuration**: These steps allow you to configure the parameters of the system and its underlying components.
3. **Actions**: These steps allow you to perform simple or complex procedures. In some cases, the procedures can take time to complete. Actions can be executed while you are performing the consecutive steps.
4. **Query**: These steps intended to retrieve the state of the system and are used with different capacities. You can retrieve a specific property of the state of the system and check if a certain aspect or condition derived from the state of the system is met. Queries do not alter the state of the system.
5. **Logging**: These steps are used to log acquired data over time in memory and, in some cases, log to files as well. Logging steps do not alter the state of the system.
6. **FlowControl**: These steps are used to alter the flow of the test. The common flow control steps are supported, such as checking conditions, looping, calling functions, error handling, tracing, etc.
7. **Interactive**: This is general purpose step from which you can execute any API step. Interactive steps are primarily used for debugging purposes. The Interactive step type is shown in Figure 2-6.

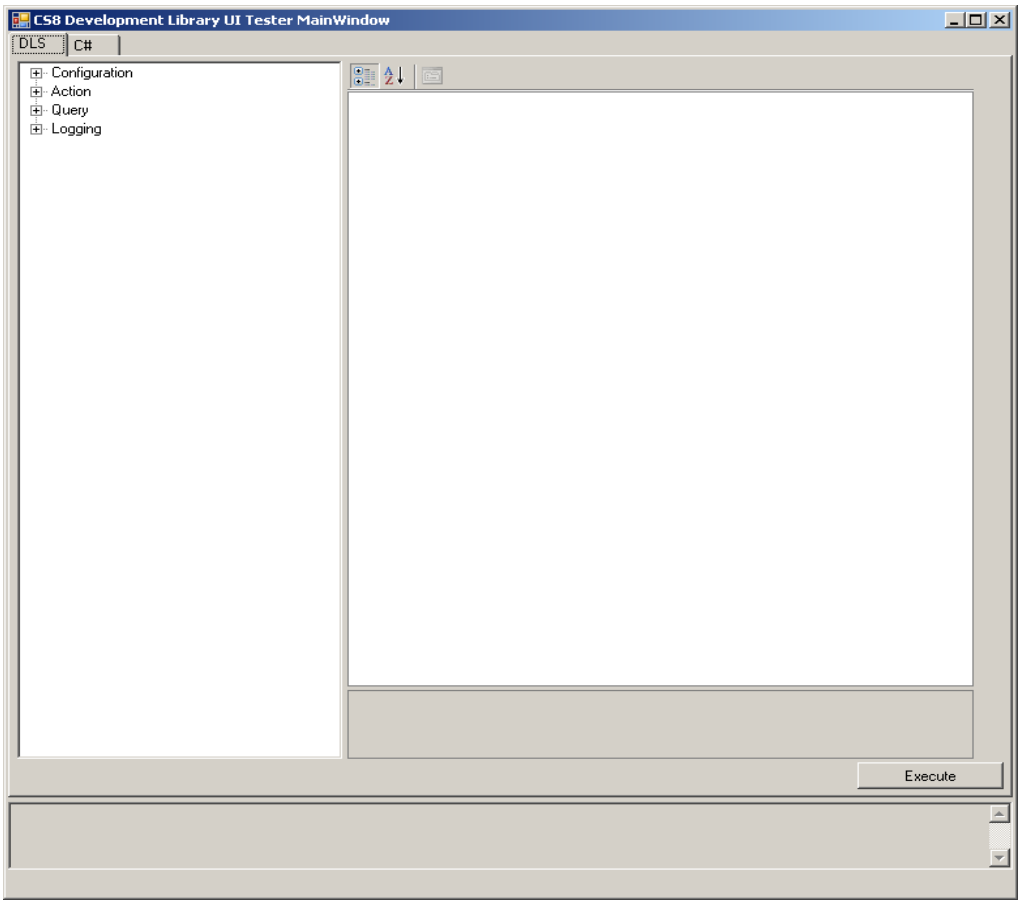

Figure 2-6: Main Window: Interactive Step

2.6. Step Subtypes

Within the step type categorization described in Section 2.5, you can select individual APIs through the Step Subtype field, as shown in Figure 2-7. The GUI editor displays the subtypes. You can select the types and parameter values, which are replayed at run-time.

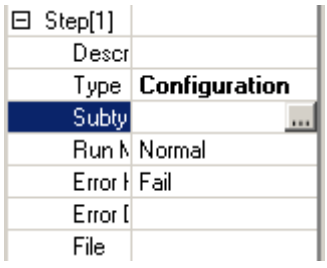

Figure 2-7: Step Subtypes

The following list explains how to work with Subtypes:

- Steps shown in bold font have altered parameters relative to previous state of the system.

- Parameters that are read-only may show warning messages instead of values. This is an indication that the values are not ready to be retrieved in edit mode.

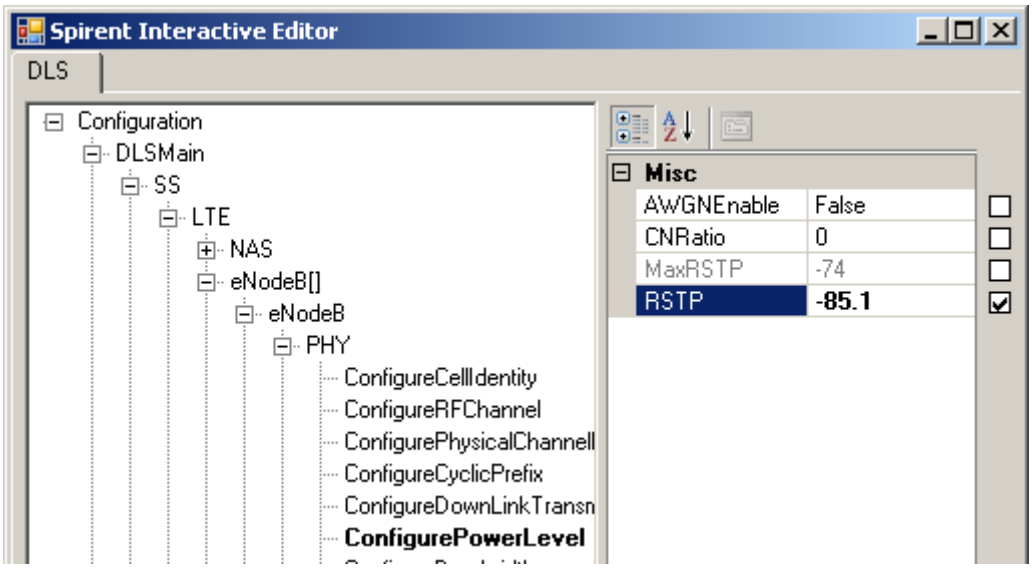

Figure 2-8: Interactive Editor — Selecting API, Parameters, and Values

- After one or more parameters of an API have been changed, the **OK** button becomes enabled and the values are stored.
  If the button is not enabled, temporarily select any other property. Note that only one API with multiple parameters can be stored for each step. The exception is the first Configuration (SS) step in the Initial Conditions section. The SS step can store multiple API settings at the same time and you can load or store them with an external file, as shown in Figure 2-8.
- For the configuration steps, select the check-box corresponding to the changed property to indicate that changes will be saved into the suite file. The property value can always be explicitly saved even for the ones that have not changed by selecting the corresponding check box. This is useful in the cases where the property values are changed indirectly by another step or as part of a later step.

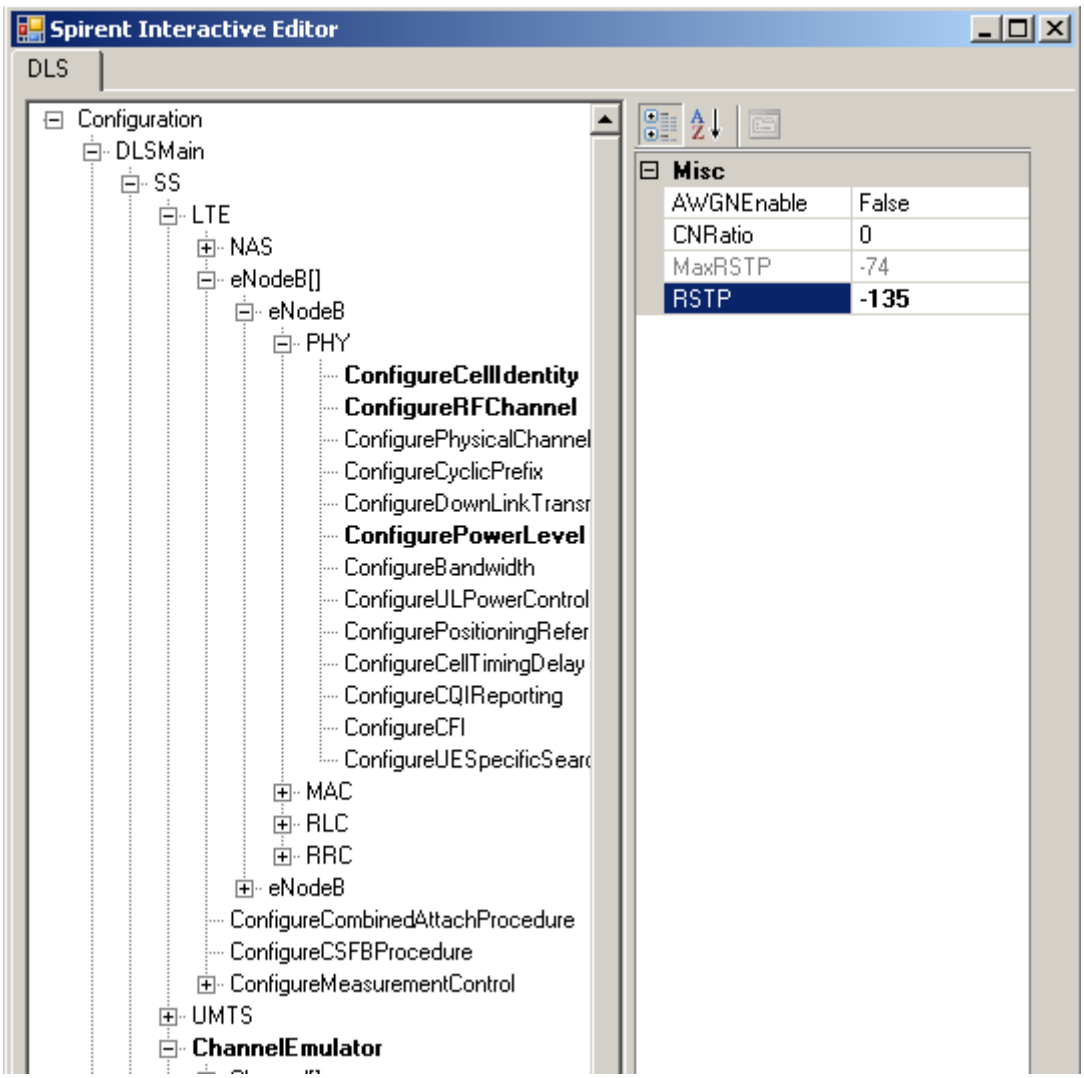

Figure 2-9: Interactive Editor — Multiple Configurations of SS Step

2.7. Step Subtype Hierarchy

Steps are hierarchically grouped by several criteria in the following order:

1. **Step Type**: Refer to Section Step Types2.5.
   Example: Interactive Step, shown in Figure 2-6.

2. **Step Assembly**: The assembly (DLL) the step resides in.
   Example: DLSMain or DLTestCaseSteps, shown in Figure 2-9.

3. **Step Namespace**: The hierarchy derived by the functional grouping of the steps.
   Example: Grouping technologies, topological elements, and protocol layers as shown in Figure 2-8.

2.8. Characteristic Step Properties

There are several properties commonly seen among many steps:

1. **SyncTimeOut**: Designates the maximum period of time (in seconds) a step is allowed to complete. If the set time is exceeded, the step fails.
2. **Blocking**: Designates if test case execution proceeds with the next step while the current step is executing. This property is only available in steps that support concurrent execution.
3. **Done**: Property that indicates if a step has completed (as a get). It also forces premature completion of the step (as a set). Usually this is associated with non-blocking steps.
4. **Error**: Read-only property of a Query step that indicates that a query has failed. The error may be due to either the system state not being ready or the property being retrieved with a value that does not match requested criteria.
   Note that an Error only applies after the Done parameter displays "True".

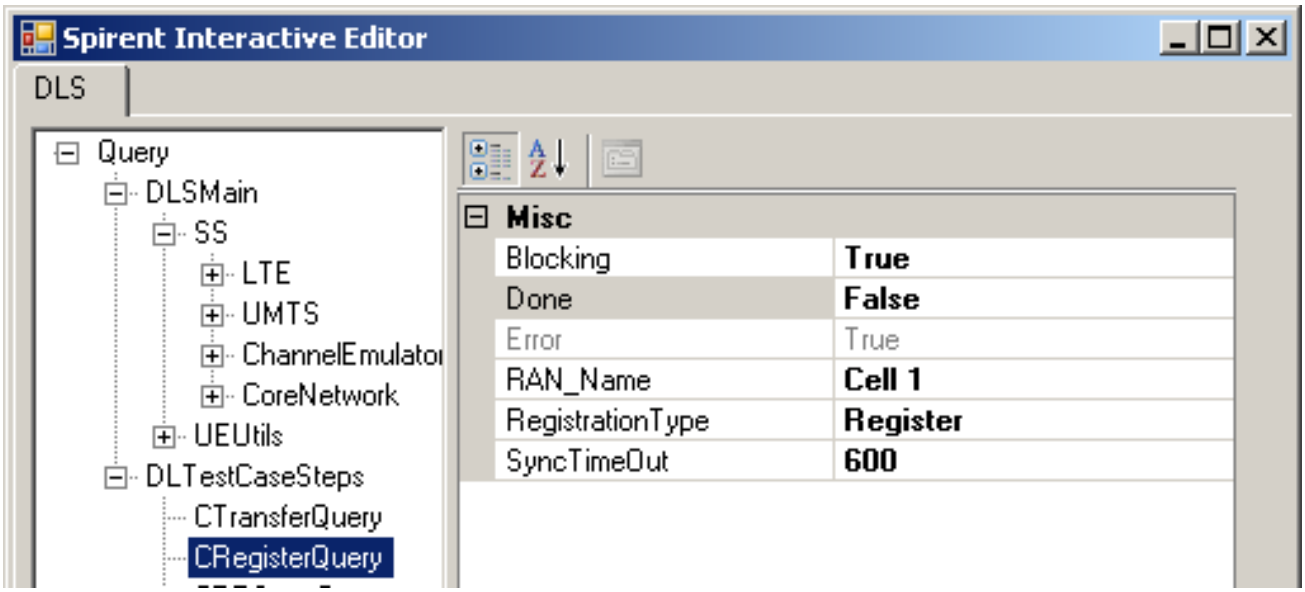

Figure 2-10: Common Step Parameters

5. **File**: Optional property of the first SS Configuration step in the Initial Conditions. This property refers to the external network configuration file (**.ncf**) that contains shared settings between different test cases and suites.
   If left empty, the step settings are stored locally in the suite as all the other steps. This is the default setting.
   To create a new **.ncf** file, enter a desired file path for the new file and click the Subtype to input the settings.
   After making the changes, click **OK** and the settings are written to the file.
   To use an existing file, follow the instructions for creating a new file. In this case, the existing **.ncf** file is overwritten after you click **OK**.

Initial Steps	5
⊟ Step[1]	Configure Network
Description	**Configure Network**
Type	**Configuration**
Subtype	**SS**
Run Mode	Normal
Error Handling	Fail
Error Description	
File	**C:\Program Files\Spirent Communications\Test Manager\Suites\LTE UMTS Mobility.ncf**

Figure 2-11: SS Network Configuration Parameter — Initial Conditions

3. Editing the CS8 Development Library UI

3.1. Step Editing

These step options determine the mode of execution of steps at run-time:

1. Selecting the Step Type
2. Selecting the Subtype
3. Selecting the Execution Options
4. Selecting the Error Handling Options

3.2. Selecting the Step Type

Select the Step Type from the *Type* field, as shown in Figure 3-1.

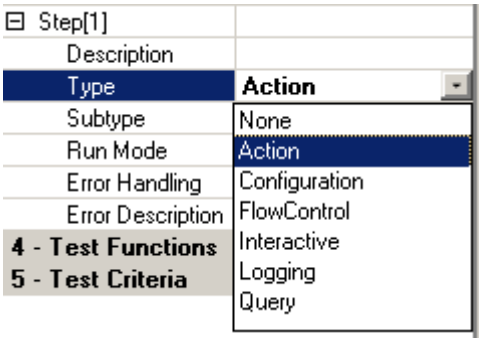

Figure 3-1: API Step Types

3.3. Selecting the Step Subtype

Select a specific step API from the *Subtype* field, as shown in Figure 3-2. This initiates a wizard that allows you to set specific APIs and settings.

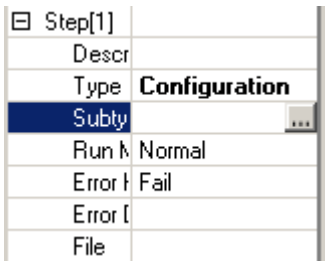

Figure 3-2: Step Subtypes

3.4. Step Execution Properties

3.4.1. Run Mode

These step options determine the mode of execution of steps at run-time:

1. **Normal**: The step is executed.
2. **Skip**: The step is skipped.
3. **Break**: This option is intended for debugging purposes. The Step editor GUI displays and you can execute a step by clicking the **Execute** button.
   The test case is effectively paused until you click **Execute**. After closing the GUI, the test resumes.

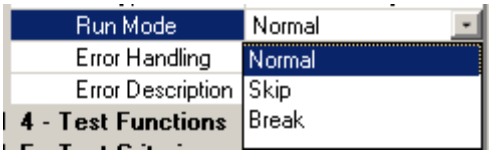

Figure 3-3: Run Modes

3.4.2. Error Handling

This option determines what type of error handling is applied if a step fails, as shown in Figure 3-4.

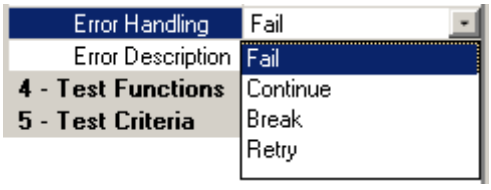

Figure 3-4: Error Handling Modes

1. **Fail**: If a step fails because of invalid parameters or functional reasons, the test case will be stopped at that point and marked as failed.
2. **Continue**: Step failure is ignored and the test case proceeds to the next step.

3. **Break**: This mode is used for debugging purposes. The step editor GUI displays and you can view and change other system properties.

4. **Retry**: You are presented with two additional options, shown in Figure 3-5. If the final retry fails, the step and test case fail as well.
   If the step succeeds during the retry attempts, the step outcome is considered a pass.

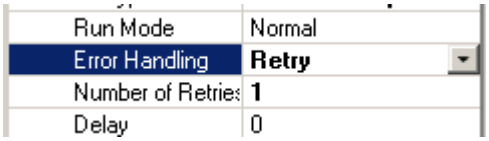

Figure 3-5: Retry Options

Step Parameters	
Number of Retries	• Number of retries while the step fails
Delay	• Delay in seconds between the retries

3.5. FlowControl Steps

FlowControl steps provide test flow management capability and closely emulate the control structures from traditional procedural languages. Some of these steps require matching an End step to enclose the block of steps, because the FlowControl constructs can be nested.

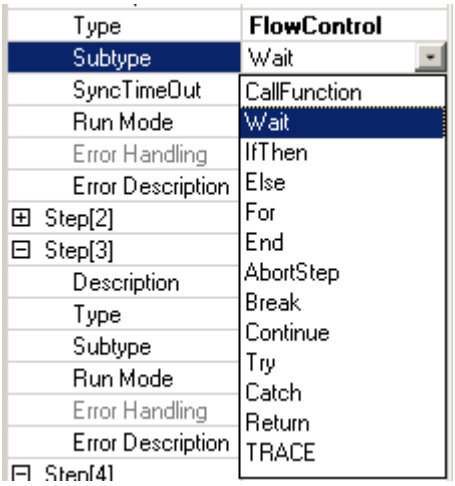

Figure 3-6: FlowControl Subtypes

1. **CallFunction**: This step is used to invoke a function, a named set of steps. The FunctionName has to match the name of the function or it can be a numerical index (starting from 1) of the function in the Functions section.
   The option Blocking determines the mode of invocation, True for blocking and False for non-blocking; in this case, the test execution engine proceeds to the next step without waiting for a function to complete.

Step Parameter	
FunctionName	• Name or index of the Function to call
Blocking	• True - invoke and wait for function to complete • False - just invoke and proceed to the next step

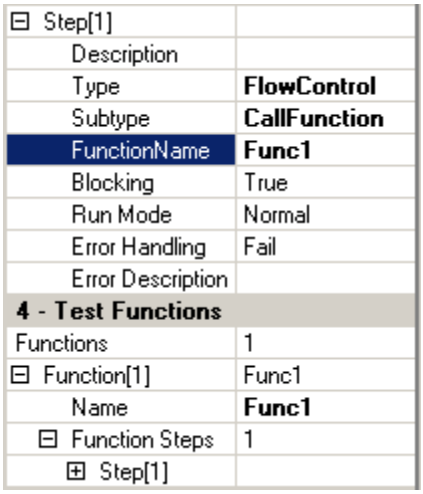

Figure 3-7: Functions

2. **Wait**: This step waits for SyncTimeOut number of seconds until proceeding to next step. Any worker threads of already invoked steps are not affected.

Step Parameter	
SuncTimeOut	• Delay in seconds, can be a fractional number

3. **IfThen**: This step has the condition as an input parameter that can be an expression that evaluates to true or false. The condition expects C/C++/C# style syntax. Note that End step is mandatory to enclose the block of steps, and Else step is optional. The IfThen can be nested each enclosed having its own End statement. Examples of Condition parameter are shown in Figure 3-8.

   a. **Else**: This is an optional step under IfThen and encloses the steps executed when a "False" result is recorded.

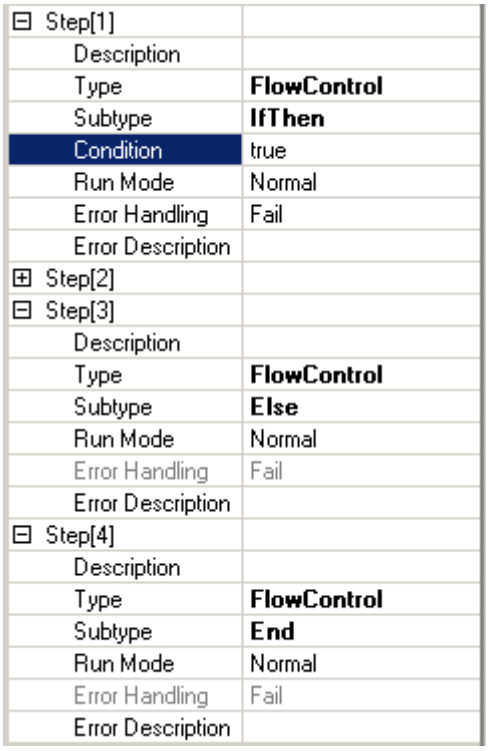

Figure 3-8: IfThen Step

Condition Parameter	
DL2.Error	• Outcome of the last step executed
((IQueryStep)DL2.Step(1)).Error	• Outcome of the query at step 1
DLS.SS.LTE.eNodeB[0].PHY.ConfigurePowerLevel.RSTP == -85	• Check for specific step parameter value
((For)DL2.Step(1)).Iteration == 1	• Check for For Iteration parameter value at step 1

4. **End**: Enclosing step for each IfThen/Else, For, and Try/Catch constructs, as shown in Figure 3-9.

⊟ Step[1]	
Description	
Type	**FlowControl**
Subtype	**IfThen**
Condition	true
Run Mode	Normal
Error Handling	Fail
Error Description	
⊟ Step[2]	
Description	
Type	**FlowControl**
Subtype	**IfThen**
Condition	true
Run Mode	Normal
Error Handling	Fail
Error Description	
⊞ Step[3]	
⊟ Step[4]	
Description	
Type	**FlowControl**
Subtype	**End**
Run Mode	Normal
Error Handling	Fail
Error Description	
⊟ Step[5]	
Description	
Type	**FlowControl**
Subtype	**End**
Run Mode	Normal
Error Handling	Fail
Error Description	

Figure 3-9: Enclosing End Steps

5. **For**: Performs the Loop Over step on the block of statements enclosed with the End step. The For loops can be nested with each construct enclosed with its own End statement. The Break and Continue steps described below are optional:

   a. **Break**: This step is used to exit the For Loop.

   b. **Continue**: This step is used to skip the rest of the steps in one iteration of For Loop.

Step Parameters	
Iterations	• Max number of iterations
Iteration	• Read-only, the current iteration value

⊟ Step[1]	
Description	
Type	**FlowControl**
Subtype	**For**
Iterations	**2**
Run Mode	Normal
Error Handling	Fail
Error Description	
⊞ Step[2]	
⊟ Step[3]	
Description	
Type	**FlowControl**
Subtype	**End**
Run Mode	Normal
Error Handling	Fail
Error Description	

Figure 3-10: For Loop Step

6. **AbortStep**: This step, shown in Figure 3-11, is used to end the execution of any step by index in the current execution section of the test-case.

⊞ Step[1]	
⊟ Step[2]	
Description	
Type	**FlowControl**
Subtype	**AbortStep**
StepIndex	1
Run Mode	Normal
Error Handling	Fail
Error Description	

Figure 3-11: Abort Step

7. **Try**: This step, shown in Figure 3-12, is used to enclose a block of steps, capture the fails or exceptions on those steps, and redirect the execution to the Catch-End section of steps.

   a. **Catch**: Associated with the Try step.

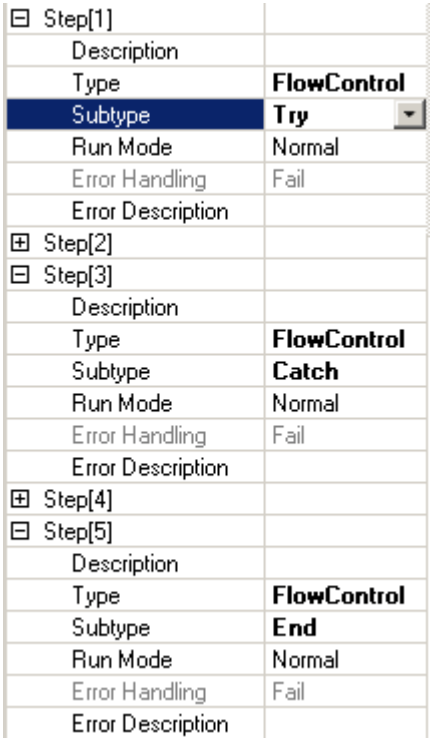

Figure 3-12: Try/Catch Step

8. **Return**: This step, shown in Figure 3-13, returns from the current section of the code with a Pass/Fail ReturnValue.

Step Parameters	
ReturnValue	• True/False

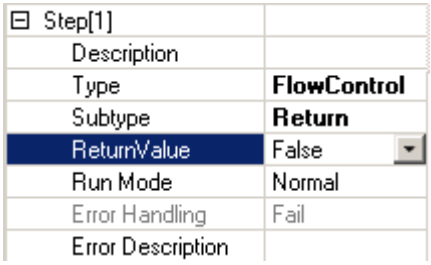

Figure 3-13: Return Step

9. **Trace**: This step posts a message to the TestManager executive. The MessageText can be a literal text string (enclosed in quotes) or an expression that evaluates to a string. In that case, the message text expects a C/C++/C# style syntax.
If the Run Mode is set to "Break", the modal message displays instead and the test case is paused indefinitely until you close the window.

Step Parameters	
TestMessage	• Description
"minus eighty five"	• Literal string

Step Parameters	
DLS.SS.LTE.eNodeB[0].PHY.ConfigurePowerLevel .RSTP == -85	• Expression value

⊟ Step[1]	
Description	
Type	**FlowControl**
Subtype	**TRACE**
MessageText	**DLS.SS.LTE.eNodeB[0].PHY.ConfigurePowerLevel.RSTP**
Run Mode	Normal
Error Handling	Fail
Error Description	

Figure 3-14: Trace Step

3.6. Sample Flow Control Scenario

This sample provides a programmatic equivalent between Error Handling using Retry and performing the retries in an iterative fashion. The CRRCStateQuery step is used, which looks for the RRCState to become RRCState_Connected with a maximum wait time of thirty seconds.

After the condition is met, the Done property displays a “True” value, and depending on if the RRCState is Connected, the Error becomes False or True. In case the step fails because of a timeout or condition not being met, the test case execution engine tries two more times with a three second delay. The third try determines the final outcome of the step and the test-case.

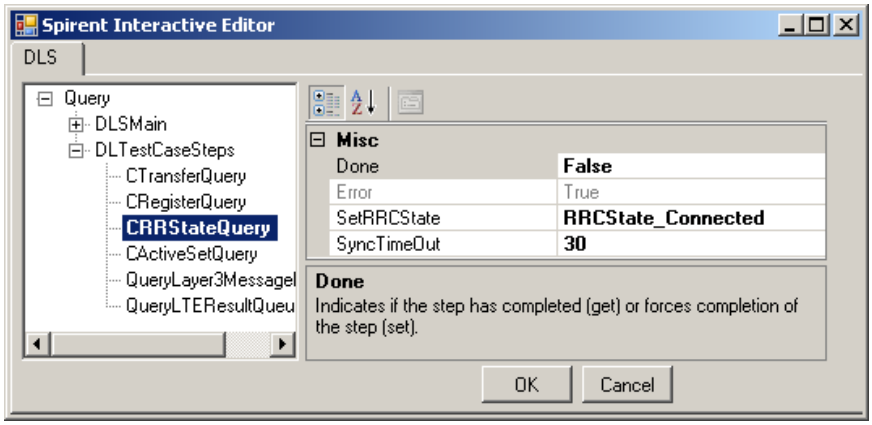

Figure 3-15: Query Step Settings

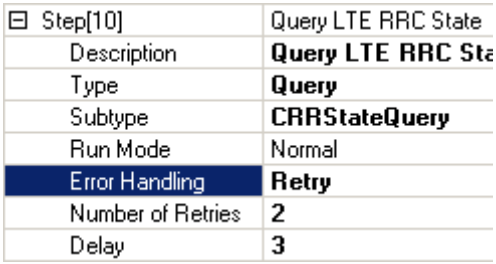

Figure 3-16: Error Handling — Two Retries and Three Second Delay

The same scenario using explicit For loops, Waits, Try/Catch, IfThen/Else, and Return steps.

⊟ Step[12]	
Description	
Type	**FlowControl**
Subtype	**For**
Iterations	**2**
Run Mode	Normal
Error Handling	Fail
Error Description	
⊟ Step[13]	
Description	
Type	**FlowControl**
Subtype	**Try**
Run Mode	Normal
Error Handling	Fail
Error Description	
⊟ Step[14]	Query LTE RRC State
Description	**Query LTE RRC State**
Type	**Query**
Subtype	**CRRStateQuery**
Run Mode	Normal
Error Handling	Fail
Error Description	
⊟ Step[15]	
Description	
Type	**FlowControl**
Subtype	**Catch**
Run Mode	Normal
Error Handling	Fail
Error Description	
⊟ Step[16]	
Description	
Type	**FlowControl**
Subtype	**IfThen**
Condition	**((For)Step(12)).Iteration == 2**
Run Mode	Normal
Error Handling	Fail
Error Description	
⊟ Step[17]	
Description	
Type	**FlowControl**
Subtype	**Return**
ReturnValue	False
Run Mode	Normal
Error Handling	Fail
Error Description	
⊞ Step[18]	End IfThen
⊟ Step[19]	
Description	
Type	**FlowControl**
Subtype	Wait
SyncTimeOut	1
Run Mode	Normal
Error Handling	Fail
Error Description	
⊞ Step[20]	End Try/Catch
⊞ Step[21]	End For

Figure 3-17: Error Handling — Programmatic Retries

3.7. Charting and Annotation Step

The LoggingQuery step provides logging settings for the procedural chart that can display any APIs numerical property value as a data series over time. It also provides a way for you to of annotate certain events usually captured from the Development Library Service trace messages.

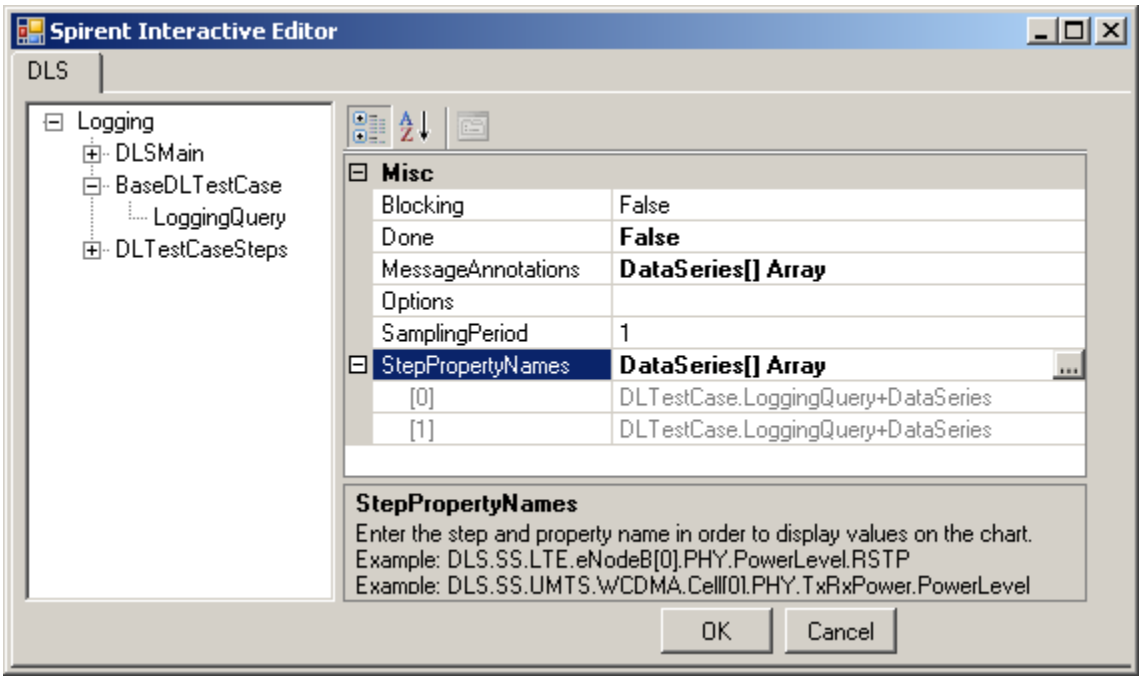

Figure 3-18: Procedure Chart Definition

To configure the data series to be tracked over time, follow the instructions below.

1. Select **Logging›BaseDLTestCase›LoggingQuery**.
2. Select **StepPropertyNames**.
3. Click the **Add** button.
   The window shown in Figure 3-19 displays.
4. Under *Color*, select the color for the line.
5. Under *Label*, type text for the label associated with the data series.
6. Under *Name*, select the data-series source, typically a readable numerical public property of an API step. This field expects a C/C++/C# style syntax.
7. Repeat Steps 3 through 6 if additional data series are needed.
8. Click **OK**.

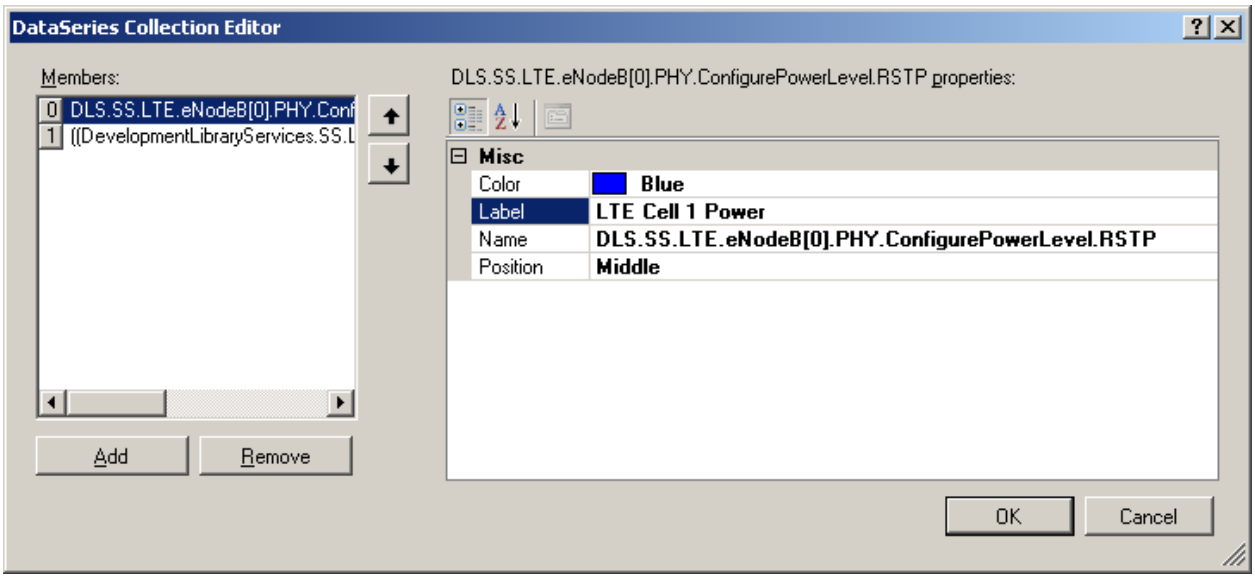

Figure 3-19: Data Series Properties

9. Under the *SamplingPeriod*, select the sampling period (in seconds).

10. Click **OK**.

Step Parameter	
SamplingPeriod	• Sampling period in seconds to read the values
Color	• Color of the line for the data-series
Label	• Text of the label corresponding to the data-series
Name	• The source step property for the data-series • The substring of the text of the trace message for annotations
Position	• The placement of the vertical annotations

Annotation settings, shown in Figure 3-20, configure the occurrence of the events on the chart (shown in vertical green text). You must associate text of certain trace messages and assign it a label.

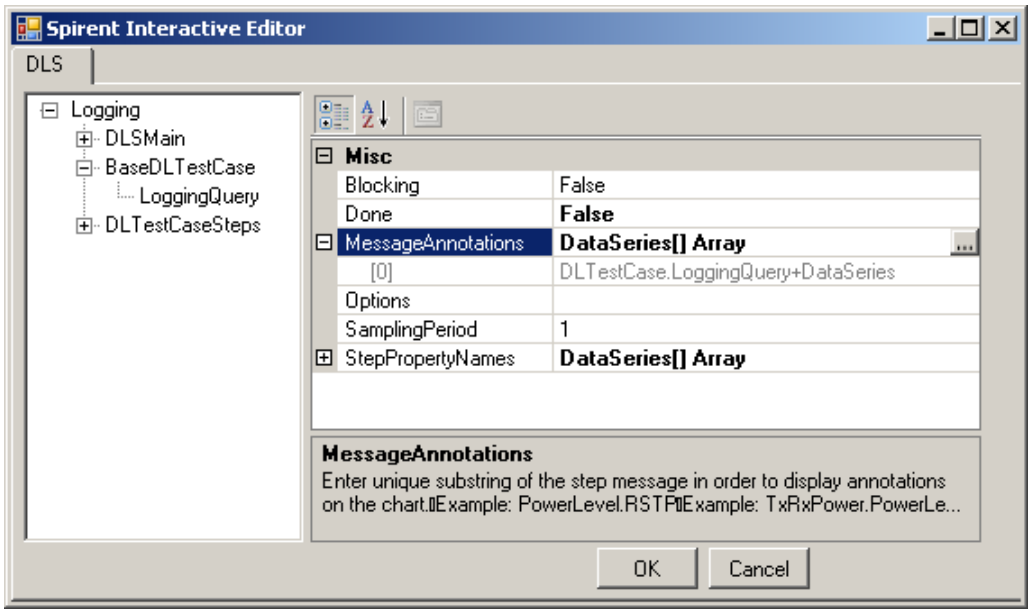

Figure 3-20: Annotation Settings

To configure Annotation settings:

1. Select **MessageAnnotations**.
2. Click the **Add** button.
3. Under *Color*, select the color for the line.
4. Under *Label*, type text for the label associated with the data series.
5. Under *Name*, select the unique substring text of the trace message. The trace messages considered are the Development Library Services DLSLogging.DLSFilter.USR_EVENT type.
6. Under *Position*, select the vertical placement of the annotations.
7. Repeat Steps 2 through 6 if additional event annotations are needed.
8. Click **OK**.

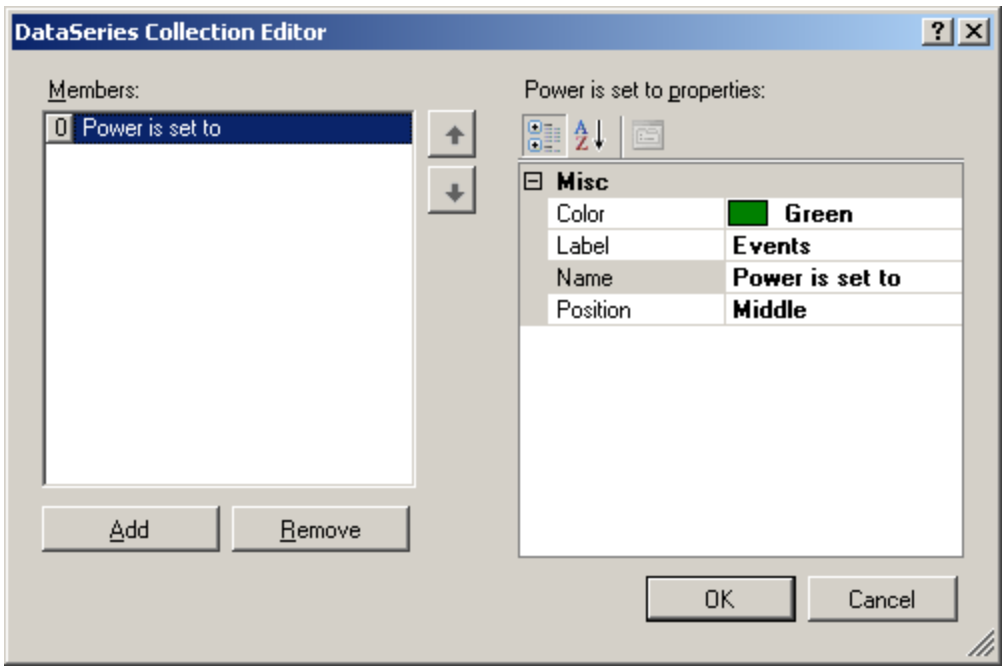

Figure 3-21: Annotation Properties

At run-time, the procedural chart displays as shown in Figure 3-22.

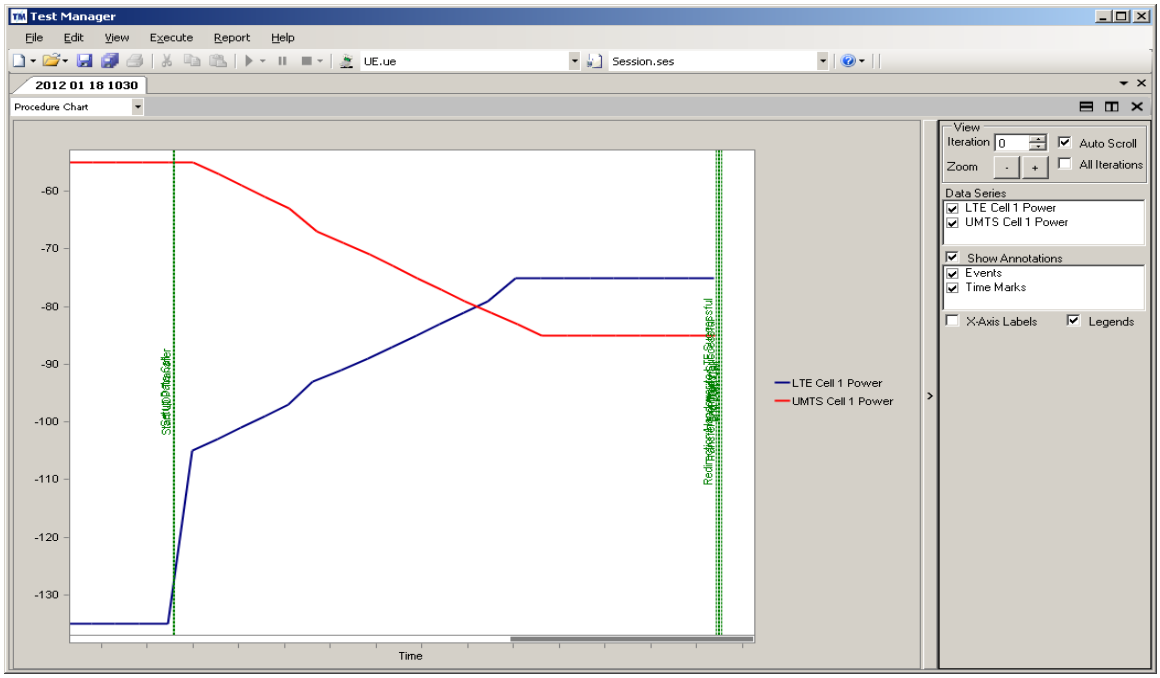

Figure 3-22: Procedure Chart at Run-time

4. Running the CS8 Development Library UI

4.1. Overview

This chapter provides information on using CS8 Development Library UI from *Test Manager*.

This includes the following steps:

1. Locating the Test Cases
2. Creating a Custom Test Suite
3. Configuring Test Case Parameters
4. Selecting the Parameter Files for Session Execution
5. Running a Test Suite

4.2. Locating the Test Suites and Test Cases

4.2.1. Locating Module Test Cases

1. In Test Manager, under the *Test Folders* tab, open the **Test Folders** folder.
2. Open one of the *CS8 Development Library UI Test Pack* subfolders.
   This folder contains CS8 Development Library UI modules such as the LTE-UMTS Mobility Test Cases, as shown in Figure 4-1.

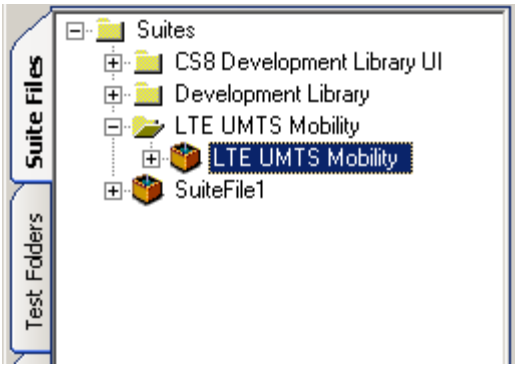

Figure 4-1: LTE-UMTS Mobility test pack Test Cases

4.3. Creating a Custom Test Suite

To set up a custom Development Library UI test, you must first create a Test Suite.

To create a Test Suite:

1. In Test Manager, select **File›New›Suite File**, as shown in Figure 4-2.

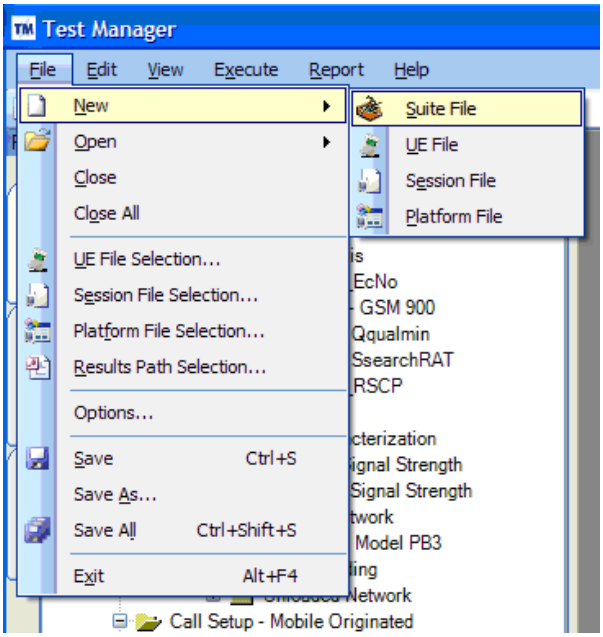

Figure 4-2: Creating a New Suite

2. You can also use the toolbar shortcuts available to create a new file, as shown in Figure 4-3.

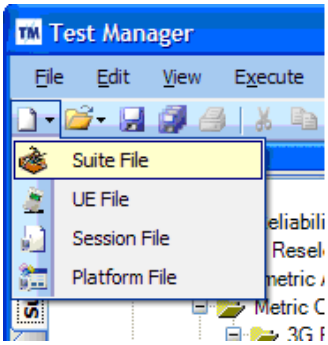

Figure 4-3: Creating a Suite File using the Toolbar Shortcut

3. A new *Test Suite* window opens with a default file name.
   This window allows you to sequence and arrange the tests.

4. To add a test from an existing opened suite, right-click the test-case, select **Copy**, and then Paste into a new suite.
   To add a test from the CS8 Development Library UI Module, drag or double-click the desired test from the *Test Folders* tab of the File Cabinet on the left and move it to the **Suite File** on the right, as shown in Figure 4-4.

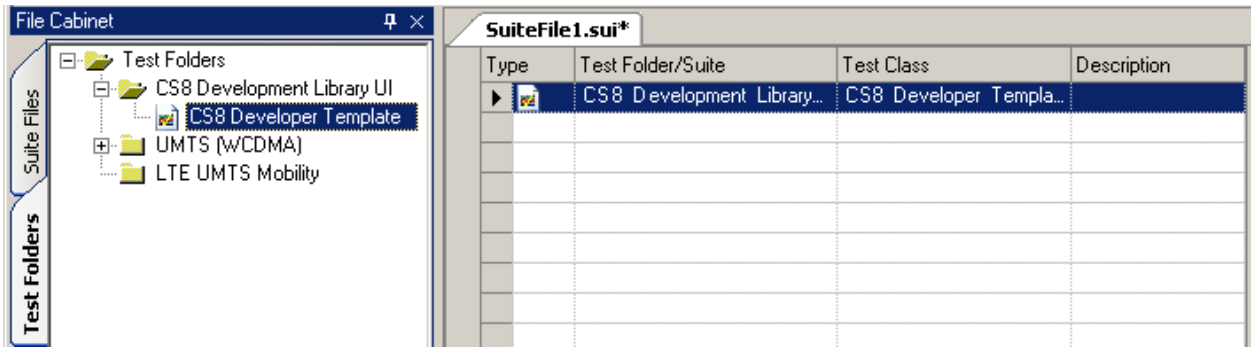

Figure 4-4: Successfully Transferred Test Case

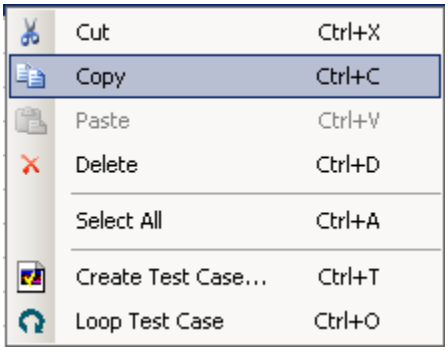

Figure 4-5: Copy/Paste Test Case

5. Configure the parameters for each Test Case based on your testing needs. The steps can be Copied, Pasted, Moved, and Deleted by right-clicking on the Test Case name and selecting an option from the menu shown in Figure 4-6.

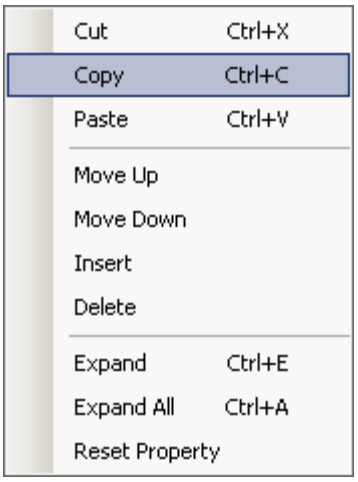

Figure 4-6: Copy/Paste Menu

4.4. Configuring Test Case Parameters

1. In Test Manager, under the *Parameters* tab, select the desired **Test Suite**.
2. In the *Suite Editor*, select the Test Case to configure, as shown in Figure 4-4.

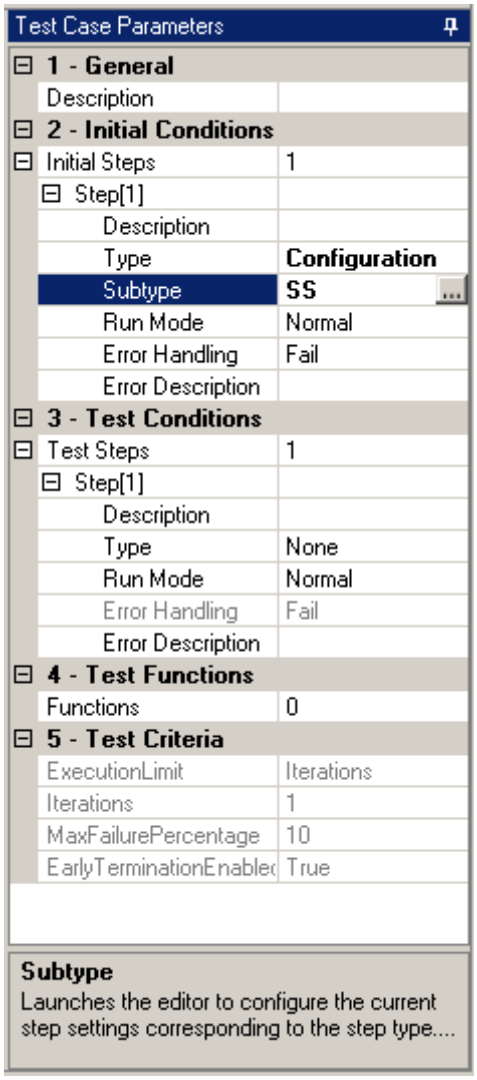

Figure 4-7: Configuring Test Case Parameters

3. Select the appropriate parameters for the test.
4. Save all changes to the Suite File.

The custom Suite File displays in the file cabinet, as shown in Figure 4-8.

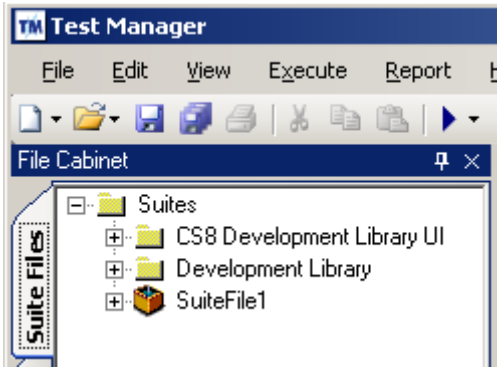

Figure 4-8: Customized Suite Saved to the File Cabinet

4.5. Running a Test Suite

To run the test suite:

1. In the Test Manager, open the desired test suite from the **Suite Files** tab of the File Cabinet, as shown in Figure 4-9.

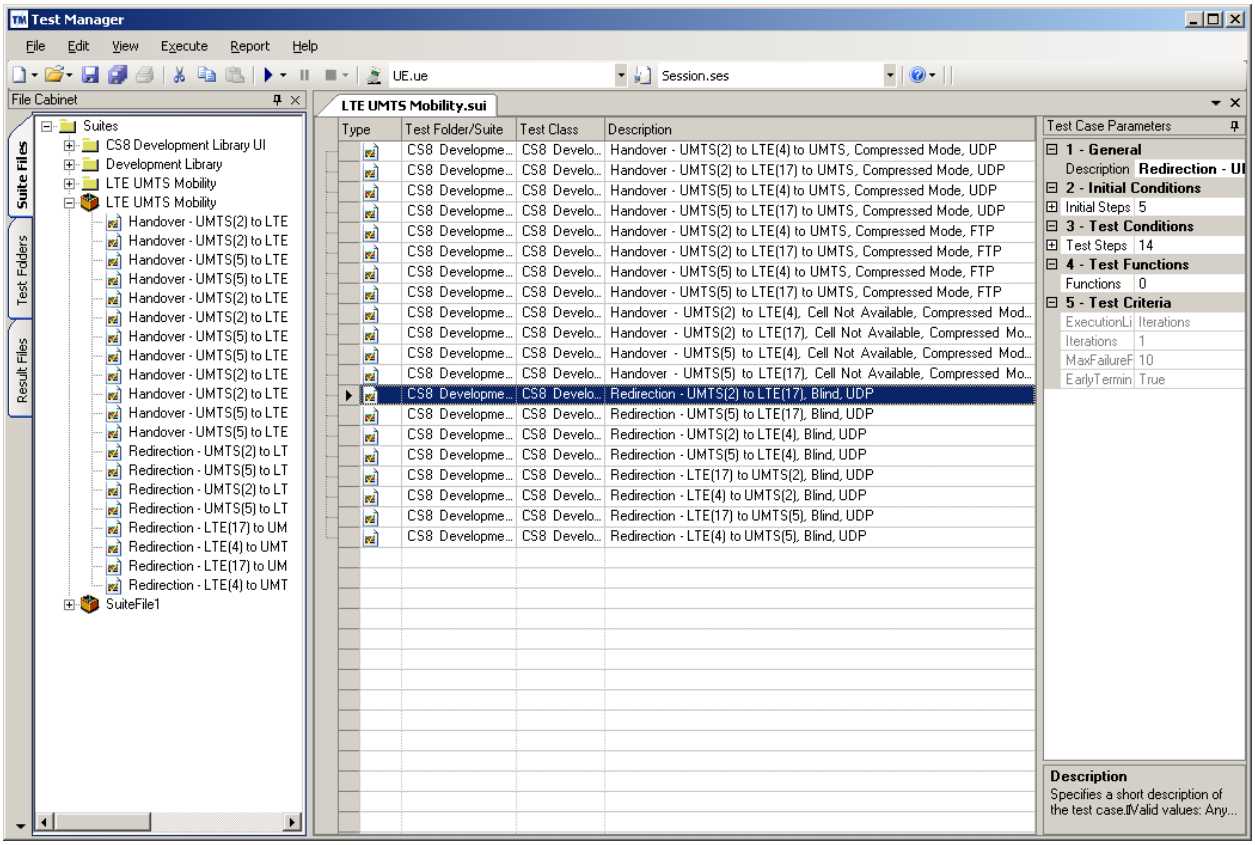

Figure 4-9: Loading a Test Suite

NOTE: Each Test Case in a pre-defined Suite has test case parameters configured to support the test definition. You do not have to alter them unless you are writing your own test suite.

2. In the Test Manager menu, select **Execute›Start Session›Run All** Test Cases to start executing the entire Test Suite.
   You also have the option of executing only the currently selected test case.
   The *Execute Session* window displays.

4.6. Selecting the Parameter Files for Session Execution

After creating the custom test suite and configuring the test case parameters, or selecting a pre-defined test suite, you must ensure the suite is compatible with the platform settings, that is, that the instrument and peripherals exist and are compatible with the functional requirements of the steps in the test case.

As indicated in the *Test Manager User Manual*, the session execution cannot guarantee a successful run of the suite unless valid Platform, Session, and UE Parameter files are specified.

To specify Parameter Files for a Test Suite:

1. In the *Execute Session* window, select the Session File, the UE File, and the Platform File. After selecting the Platform File, select the Platform, as shown in Figure 4-10.

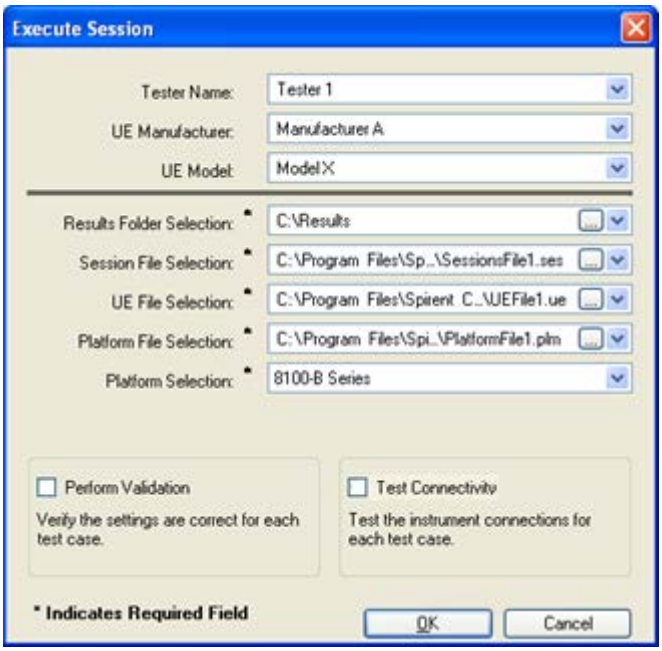

Figure 4-10: Execute Session Window — Selecting Parameter Files

2. Select the appropriate parameter files and platform.

3. The **Tester Name**, **UE Manufacturer**, and **UE Model** provide details about the testing performed. These fields display on all reports you create based on the results of the session. We strongly recommend that you utilize these fields. If they are left blank, it can be difficult to identify which device was used to generate the results.

NOTE: The Results folder and Platform File selections are already set, it is not necessary to make changes to these fields.

4. Click **OK** to begin execution.
   If successful, the test session begins, as shown in Figure 4-11.

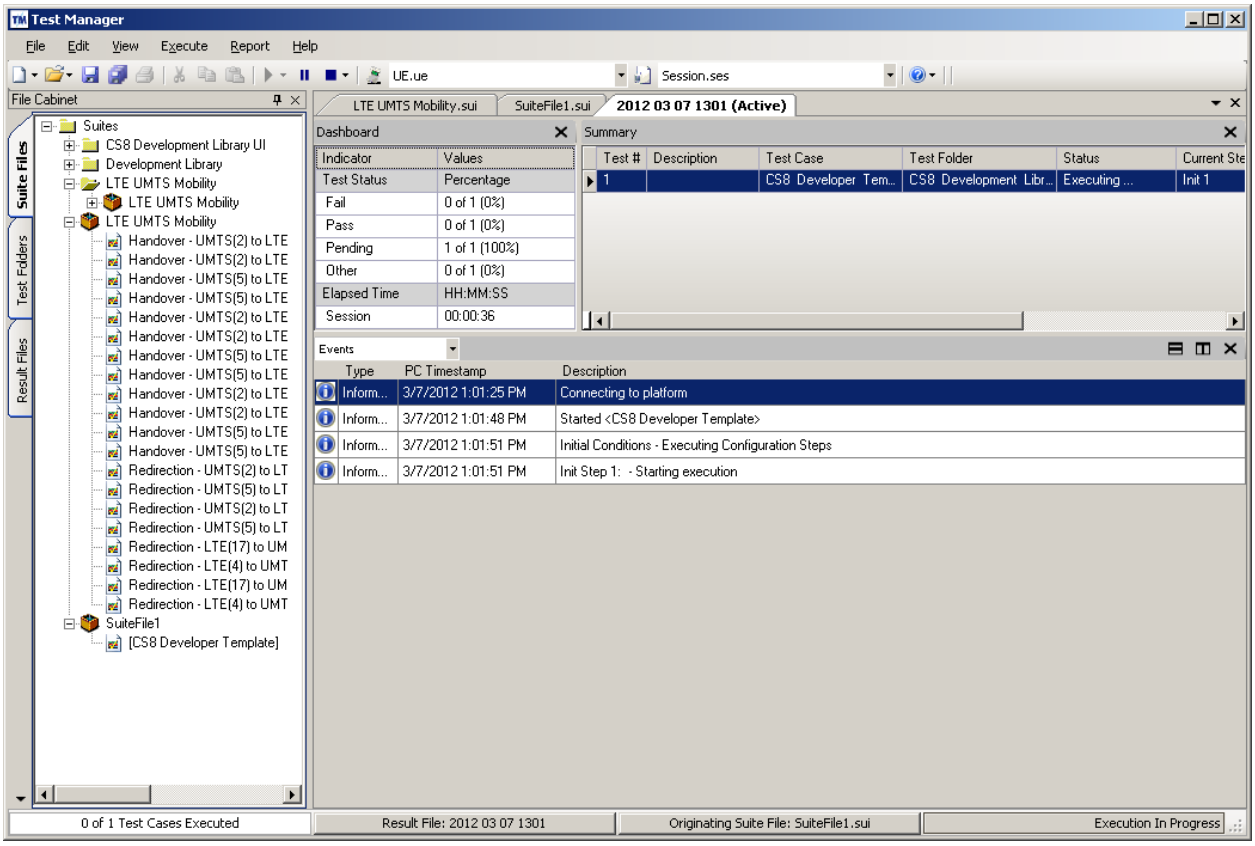

Figure 4-11: Test Session in Progress

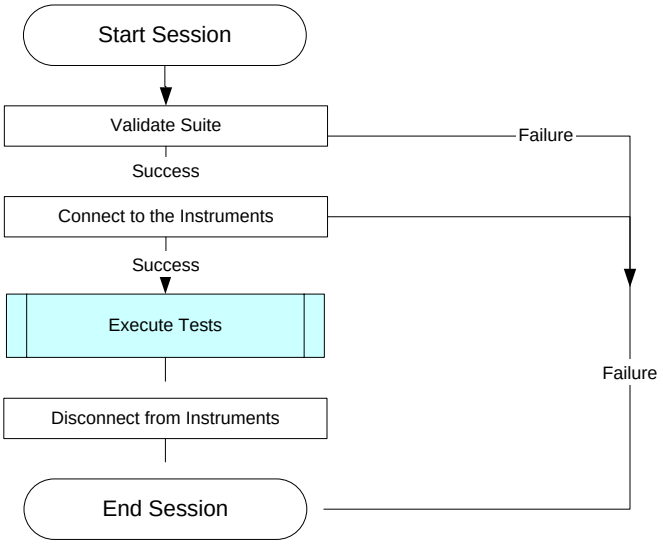

Figure 4-12: Test Suite Execution Flowchart

5. At any point in the Test Case execution you can pause or abort the test, as shown in Figure 4-13.
   Note that some blocking steps might not check for the pause command; this means that test pausing may be delayed until after the step is complete. Also note that some steps do not periodically check if they are paused; they may continue to operate their worker threads.

6. You can either abort the current test case or the entire suite.
   Note that we recommend aborting the test-case between step executions to ensure the coherency of the system. If there is blocking step that does not periodically check for the abort command, it is forcefully aborted after a short grace period.

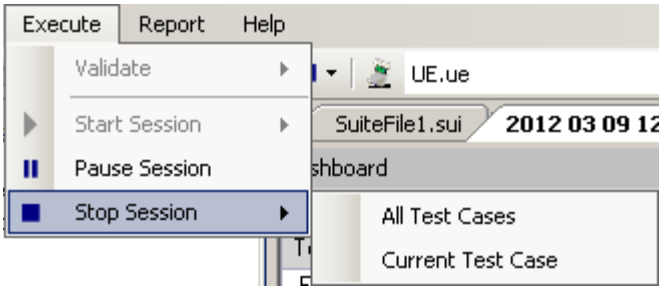

Figure 4-13: Pausing/Aborting the Test

5. Creating Custom CS8 DL UI Steps

5.1. Overview

This chapter provides information on how to develop custom CS8 Development Library UI APIs that can be edited and executed by the CS8 Development Library UI. In general, any assembly containing classes that inherit from Development Library Services or CS8 Development Library UI a certain set of interfaces, can be recognized by the editor and the engine.

This chapter includes the following steps:

1. Creating a C# class library interactively
2. Creating a C# class library project
3. Adding references to CS8 Development Library assembly
4. Deriving from one of the step type specific parent classes
5. Overriding some of the method and properties
6. Testing the step

5.2. Creating a Custom Step Interactively

This section provides information on developing a custom step during an interactive session. It is intended for system debugging and rapid prototyping of light-weight steps. You can save the custom step by selecting the **Save Step** option.

The assembly name is derived from the class name itself, and should display in the editor tree-view hierarchy in the next Test Manager session.

To create a custom step:

1. Insert the interactive step in the suite.
2. Run the suite.
3. When the *Interactive Step Editor* window displays, select the **C#** tab.
4. Make changes to the source code in the edit box and click on the Execute button. If there are no compilation and runtime errors the effects should be observed in the status window on the bottom of the GUI, and in the example below, in the CS8 Interactive Tester, Figure 5-7.

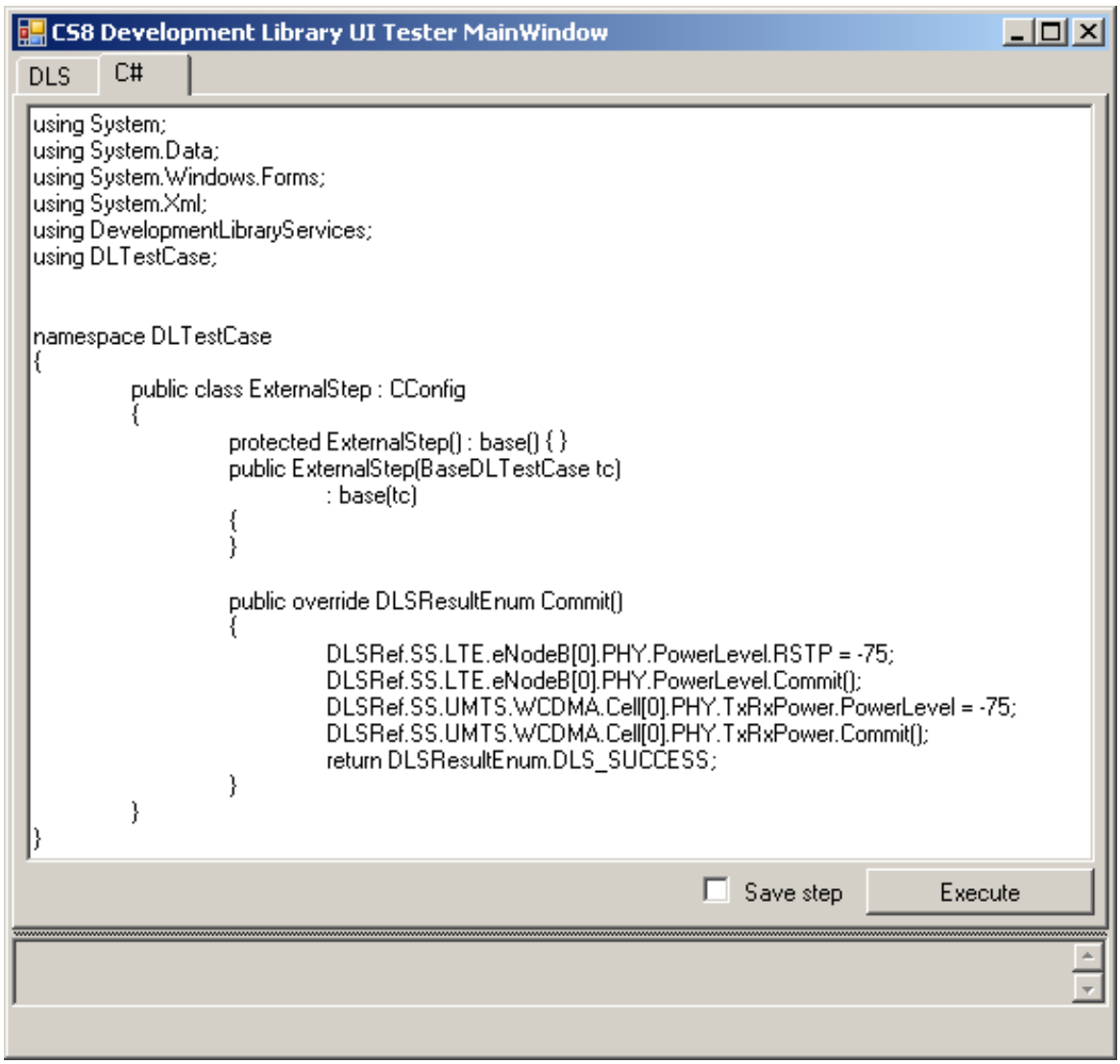

Figure 5-1: Interactive Step Generation Editor

5.3. Creating a Custom Step Project

This section provides instructions on developing custom CS8 Development Library UI steps using Microsoft Visual Studio 2010 IDE. This approach gives you greater control over the attributes of the project and better debugging capability.

To create a custom step project:

1. Open *Visual Studio IDE* and select **File›New›Project**.
   The New Project window displays, as shown in Figure 5-2.

2. Select a **Visual C# Class Library** project.

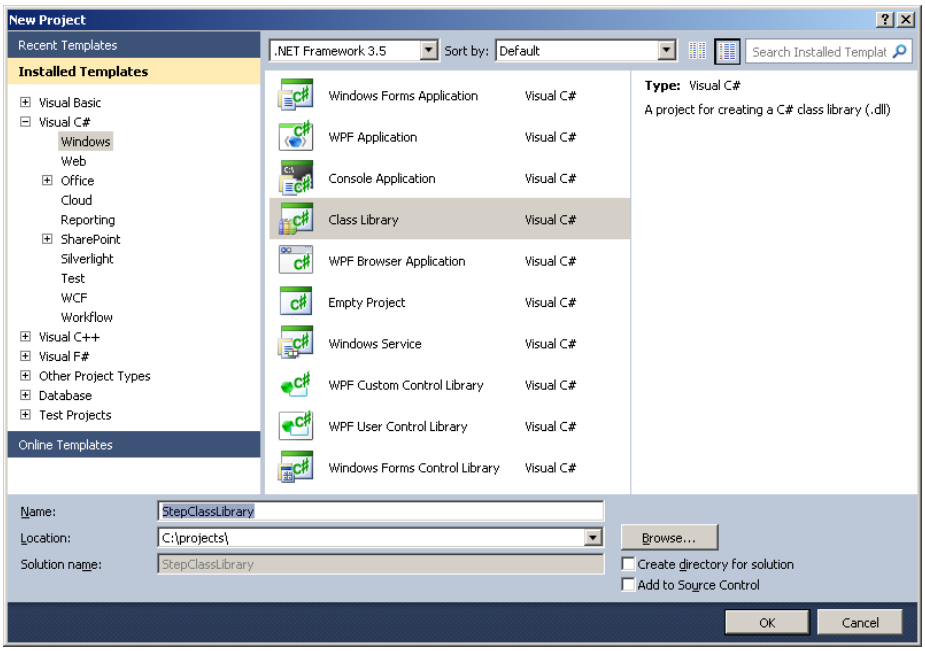

Figure 5-2: Creating a New Project

3. In the Add Reference window, shown in Figure 5-3, select the **CS8 Development Library UI** folder and click the **Browse** tab. Add the following references:

   a. **C:\Program Files\Spirent Communications\Test Manager\Modules\CS8 Development Library UI\ DLTestCase.dll**

   b. **C:\Program Files\Spirent Communications\Development Library Services\bin\DevelopmentLibraryServices.dll**

   c. **C:\Program Files\Spirent Communications\Test Manager\Manager\TestCaseInterface.dll**

   d. **System.Windows.Forms**

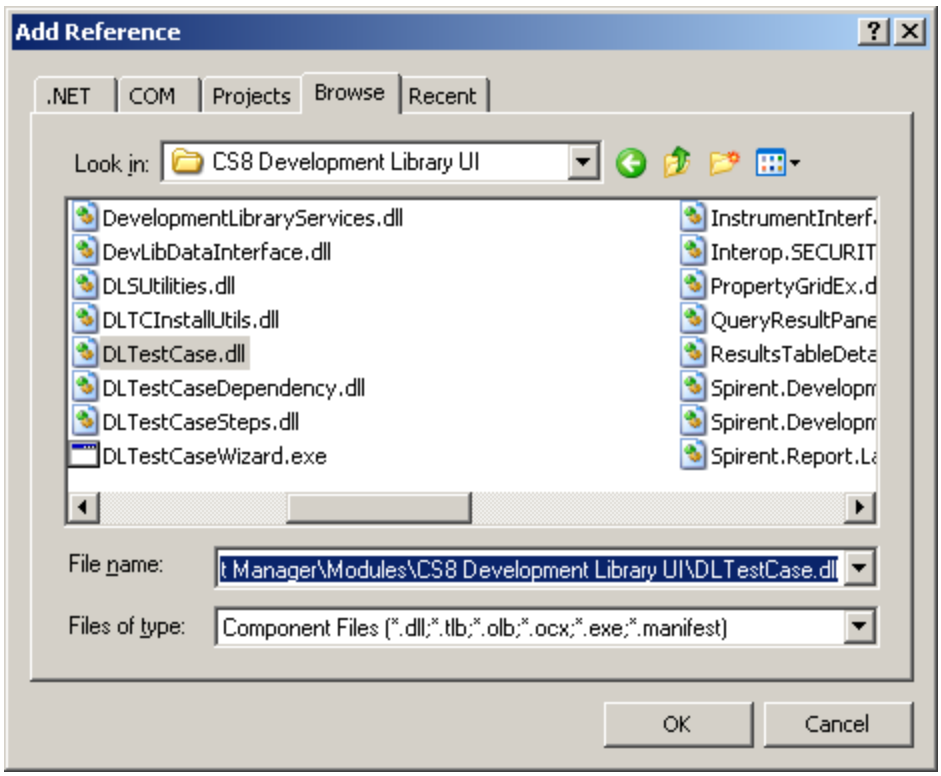

Figure 5-3: Add Reference Window

4. Right-click the **Project Node** and select **Properties**.
5. In the *Properties* window, select the **Applications** tab.
6. Under *Target Framework*, select .NET Framework 3.5, as shown in Figure 5-4. This ensures that the assembly references the .Net CLR that is already present on the target system.

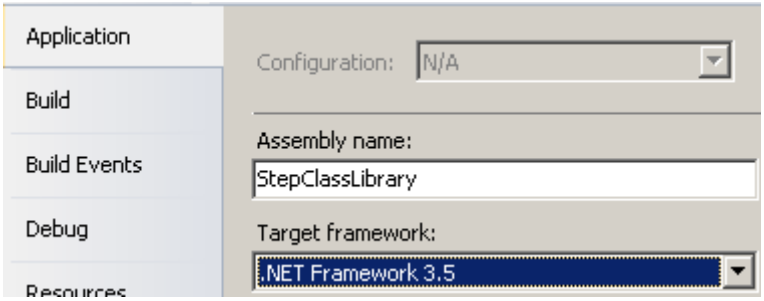

Figure 5-4: Selecting Target Framework

7. Under the *Build Events* tab, select the **Post-build Event Command Line** and insert the *CS Development Library Destination* folder, as shown in Figure 5-5.

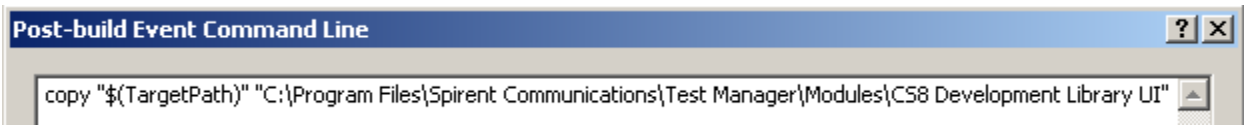

Figure 5-5: Adding the Destination Folder

8. In the **Class1.cs**, make sure the source code uses the #using statements and the class derives, in this case from CConfig. For action steps, this would be CAction and for queries CQuery. The constructor can be left empty.
9. The public properties need to have both read and write capability to display as modifiable in the CS8 Development Library editor. There are two properties in this case.
10. In this case, the Override Commit method performs power assignments. The correct code is shown below.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using DevelopmentLibraryServices;
using DLTestCase;

namespace StepClassLibrary
{
        public class MyCustomStep : CConfig
        {
                protected MyCustomStep() : base() { }
                public MyCustomStep(BaseDLTestCase tc)
                        : base(tc)
                {
                }

                double lte_Power = -75;
                double umts_Power = -75;

                public double LTEPower { get { return lte_Power; } set { lte_Power = value; } }
                public double UMTSPower { get { return umts_Power; } set { umts_Power = value; } }

                public override DLSResultEnum Commit()
                {
                        DLSRef.SS.LTE.eNodeB[0].PHY.PowerLevel.RSTP = lte_Power;
                        DLSRef.SS.LTE.eNodeB[0].PHY.PowerLevel.Commit();
                        DLSRef.SS.UMTS.WCDMA.Cell[0].PHY.TxRxPower.PowerLevel = umts_Power;
                        DLSRef.SS.UMTS.WCDMA.Cell[0].PHY.TxRxPower.Commit();
                        return DLSResultEnum.DLS_SUCCESS;
                }
        }
}
```

12. Build a solution.
13. Make sure that the **.dll** file is present in the **C:\Program Files\Spirent Communications\Test Manager\Modules\CS8 Development Library UI** folder.
14. In Test Manager, create a new suite from the *CS8 Developer Template*, and insert the new configuration step in the Initial or Test Conditions section. The *Interactive Editor* should display the information shown in Figure 5-6.
15. Click **OK**.

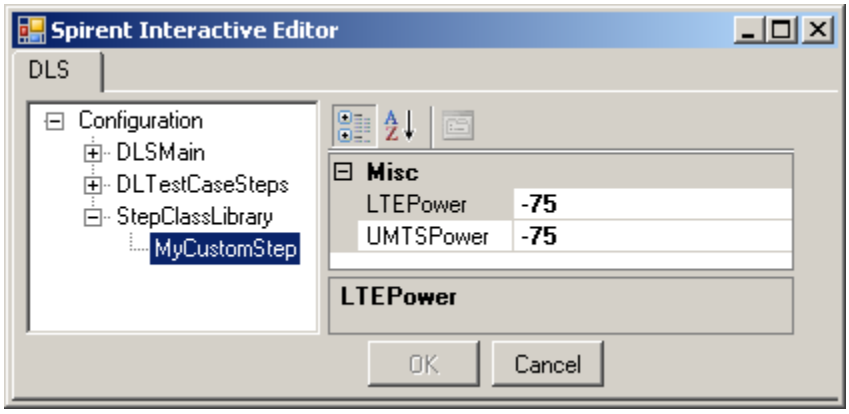

Figure 5-6: Custom Step Properties

16. Run the suite and ensure that the value of the RSTP parameter has changed in the CS8 Interactive Tester GUI, as shown in Figure 5-7.

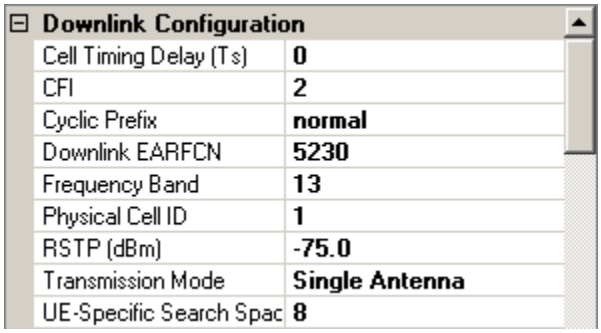

Figure 5-7: CS8 Interactive Tester GUI