HP Solution Test Bench

Plugin Developer Guide v 1.1

Table of Contents

Overview

The Solution Test Bench (STB) is a standalone tool that allows setup and execution of automated test scenarios. The packaged release includes a set of standard STB Plugins, which encode workflow actions and configuration options for the native capabilities of the device and standard office activities. STB additionally will include database support for creating and managing inventories of:

- device information
- documents
- servers and print queues

The STB Plugin SDK can be used to create new plugins to automate workflow testing not included with the standard STB Plugins. These plugins can utilize either the resources included in the SDK or custom third party components, libraries or references.
The STB Plugin SDK consists of:

- STF.Framework.dll, STF.Development.dll, STF.Dependencies.dll
- .chm file documenting the STB API, STB plugin API and included assets and base elements
- Example Plugins with source code
- The Plugin Sandbox Simulator
- Plugin Development template. The template is included in the SDK as a Visual Studio Extension (vsix file).
- HP Solution Test Bench – Plugin Developer's Guide (this document)

This guide includes:

- An overview of the STB Plugin Architecture
- An explanation of the main method calls used by both the plugin and STB to govern their interaction
- A list of the example plugins and their source code included in this SDK
- A brief explanation of some of the salient features of the code from the Pull Print Example plugin provided in the SDK
- Instructions on how to create a new, custom plugin
- Instructions on how to test a new, custom plugin within the Plugin Sandbox simulator
- Instructions on how to add a new, custom plugin to STB so that it may be used by Virtual Workers in Test Scenarios in STB.

Minimum Requirements for STB Plugin SDK

Minimum requirements to run the STB Plugin SDK:

- Installation of STB Server 4.1 or higher
    - Refer to HP Solution Test Bench – Installation and Administration Guide
- MS Visual Studio 2015
- .NET Framework 4.6

Recommended for STB Plugin SDK

Recommended options for improved functionality:

- OXPt.dll – Provides substantial automation support.
  - Refer to Partner Portal Kit's Download page. If not available, contact Partner Support to request access.

STB SDK Plugin File Directory

Once an extraction location is selected, refer to the following file structure:

- HP Solution Test Bench Plugin Developers Guide.docx (this document)
- HPInc_STB_Plugin_Wizard.vsix
- Manifest.txt
- PluginSdkExamples.sln
- **Plugin.SdkGeneralExample**
    - !Readme.txt
    - **Properties**
        - AssemblyInfo.cs
        - Resources.Designer.cs
        - Resources.resx
    - Plugin.SdkGeneralExample.csproj
    - SdkGeneralExampleActivityData.cs
    - SdkGeneralExampleConfigControl.cs
    - SdkGeneralExampleConfigControl.Designer.cs
    - SdkGeneralExampleConfigControl.resx
    - SdkGeneralExampleExecutionControl.cs
    - SdkGeneralExampleExecutionControl.Designer.cs
    - SdkGeneralExampleExecutionControl.resx
- **Plugin.SdkPullPrintExample**
    - !Readme.txt
    - Plugin.SdkPullPrintExample.csproj
    - **Properties**
        - AssemblyInfo.cs
    - SdkPullPrintExampleActivityData.cs
    - SdkPullPrintExampleConfigurationControl.cs
    - SdkPullPrintExampleConfigurationControl.Designer.cs
    - SdkPullPrintExampleConfigurationControl.resx
    - SdkPullPrintExampleExecutionControl.cs
    - SdkPullPrintExampleExecutionControl.Designer.cs
    - SdkPullPrintExampleExecutionControl.resx
- **References**
    - STF.Development.dll
    - STF.Framework.Dependencies.dll
    - STF.Framework.dll
- **SdkPluginSandbox**
    - **Properties**
        - AssemblyInfo.cs
    - !Readme.txt
    - App.config
    - Program.cs
    - SdkPluginSandbox.csproj

STB Plugin Architecture

The STB Plugin architecture consists of the following:

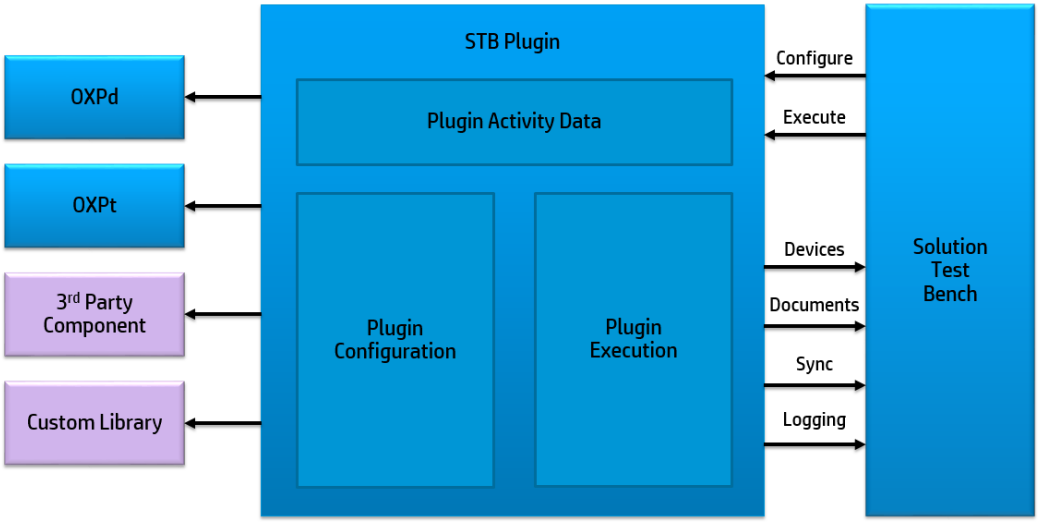


Fig 2.1 – STB Plugin Architecture

At the most basic level, an STB Plugin consists of three components:

- An execution engine that defines an activity that will be executed by the plugin.
- An activity data class that contains configuration options for this plugin that are then used by the execution engine when executing.
- A UI control that collects the configuration information entered by the tester. This UI control is displayed on the Configuration tab of STB during the configuration of a Test Scenario that makes use of this plugin.

STB to Plugin Interaction

The Solution Test Bench will pass configuration data into a plugin and then execute an activity based on that configuration data. Interaction of specific interfaces is defined below.

Plugin Configuration Control Interface

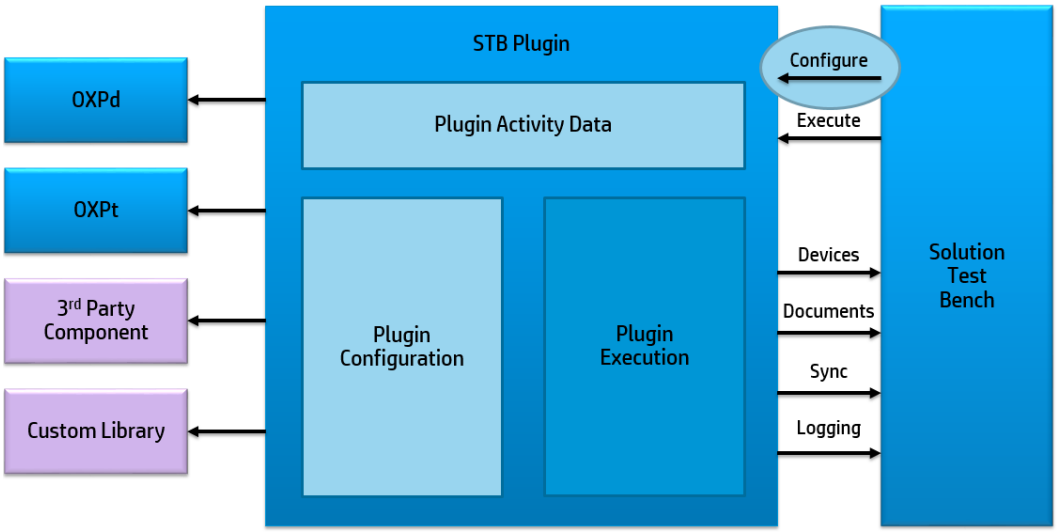


Fig 2.2 – STB Plugin Configuration Interface

The IPluginConfigurationControl Interface:

- The developer defines what "Plugin Activity Data" is necessary for the configuration and the execution of the plugin. This data is persisted during the life of a Test Scenario that includes the developer's plugin.
- Contains UI and logic for configuration data entry. UI elements include radio buttons, spinners, text boxes, etc., these appear in STB when the plugin is selected as part of the Test Scenario. These UI elements display on the "Test Configuration" tab and allow the tester to configure how the activities of the plugin will be executed within this Test Scenario. These controls allow the same plugin to be executed with different configuration parameters in different Test Scenarios, besides the differences in devices, users/user credentials, and servers/print queues provided natively in STB.
- Implementing class is a UI control (WinForms).
- Responsibilities include:
    - initializing UI for either new or existing activity
    - providing UI layout for configuration data entry
    - validating proper user input
    - returning configuration UI elements defined in the plugin for the Test Scenario to the STB database for use during future executions of that Test Scenario

Plugin Execution Engine Interface

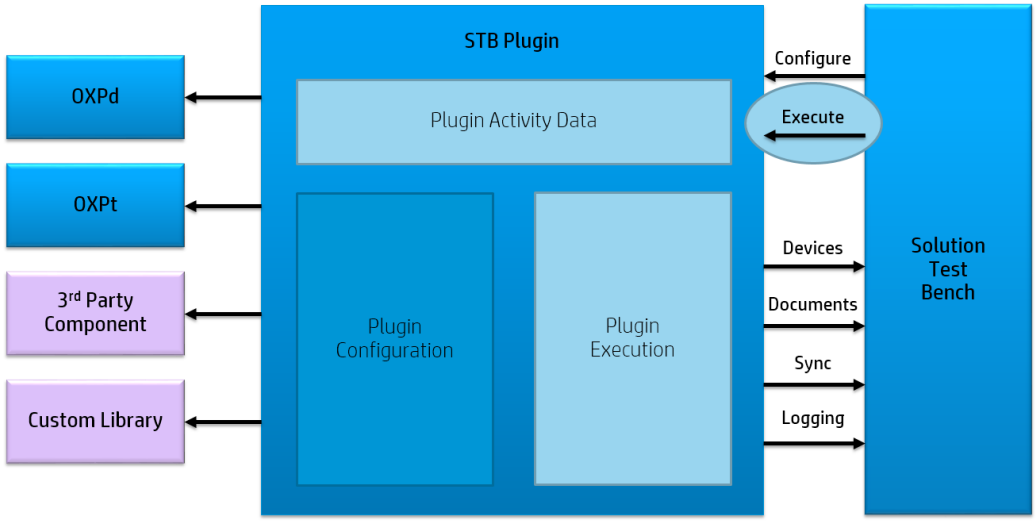


Fig 2.3 – STB Plugin Execution

The IPluginExecutionEngine Interface:

- Executes a plugin activity based on the configuration provided on the "Test Configuration" tab in STB, through the UI elements defined using the IPluginConfigurationControl Interface, which are then stored by STB in the plugin's Activity Data class. The Activity Data class is one of the inputs passed by STB to the plugin's execution interface.
- Implementing class may or may not be UI-based. There is standard information that STB displays during Test Execution. The developer may implement additional UI elements as part of the plugin to show additional information as the plugin executes.
- Responsibilities include:
    - executing test according to configuration data
    - collecting and logging data about execution
    - returning a result to STB for reporting. These are persisted in the STB database.

Plugin to STB Interaction

The STB framework provides a set of services that the plugin can invoke during configuration and execution, which includes but is not limited to accessing the Asset Inventory of available devices, accessing the test Document Library, synchronization methods that allow orchestration of multiple concurrent tests, and logging.

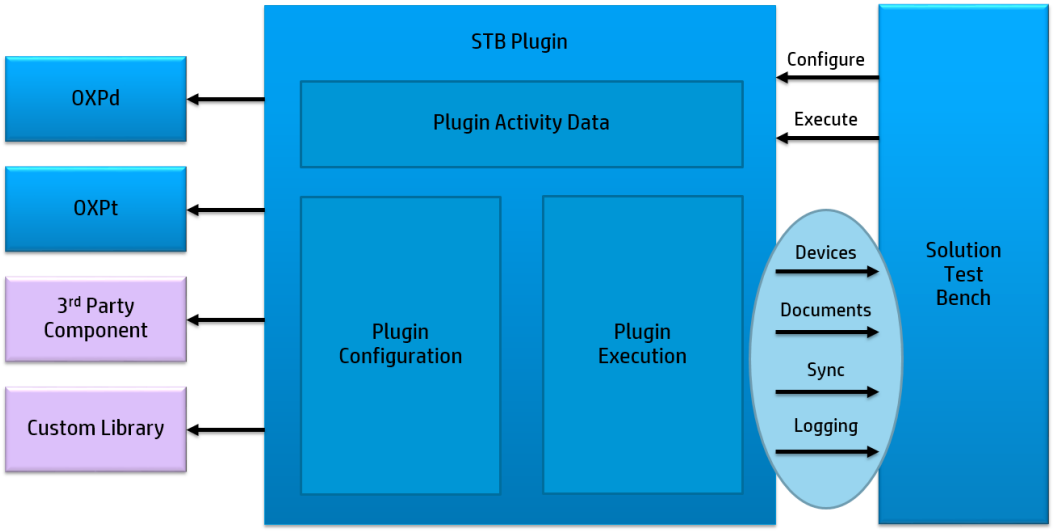


Fig 2.4 – STB Plugin To STB Interaction

Configuration Services

- Asset Inventory
  - Provides access to devices, servers, and print queues
  - Includes UI controls for selecting devices and servers
- Document Library
  - Provides access to a managed library of documents
  - Can select specific documents or select by criteria
  - Includes UI controls for selecting documents
- System Trace
  - Trace logging for debug and triage

Execution Services

- Critical Section
  - Provides system-level locking for test assets (e.g. devices)
  - Allows multiple activities to run without conflicting
- Data Logger
  - Runtime data collection stored for reporting through STB
  - Built-in loggers for common data (e.g. devices, timestamps)
  - Custom logger support for plugin-specific data
- File Repository
  - Manages access to files from public shares (including document library)
- System Trace
  - Trace logging for debug and triage

Example Plugins Included

The following plugins (note that both example plugins follow the Visual Studio project naming convention of Plugin.*.csproj required for the plugins to be usable in STB) are included in the STB Plugin SDK:

Plugin.SdkGeneralExample

The STB Plugin SDK example project "Plugin.SdkGeneralExample" demonstrates usage of common STB plugin tasks such as:

- selecting assets (devices) and putting a lock on them during execution
- selecting and using documents from the test document library
- accessing virtual worker credentials

Plugin.SdkPullPrintExample

The STB Plugin SDK example project "Plugin.SdkPullPrintExample" assumes that the OXPd SDK PullPrint Demo has already been built and installed on a device with the top-level solution button titled "PullPrint Local".

Pre-requisites for use:

- Download OXPd SDK from partner portal
- Compile and install PullPrint Demo to device with top level button titled "PullPrint Local"
- Download OXPt assembly from partner portal
- Add reference to OXPt assembly to this project

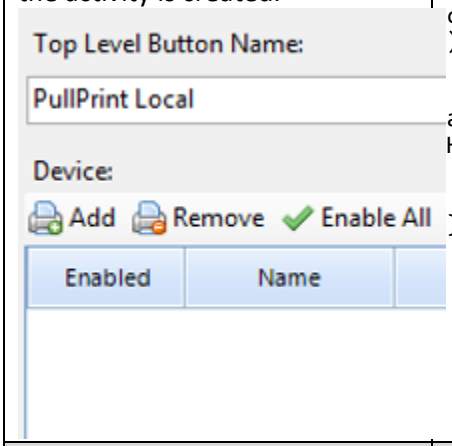
SdkPullPrintExample Plugin Example

In the SdkPullPrintExample included with the SDK package, the developer creates a configuration UI to select a device and enter the top level button name already registered on the device. A pull print job is then executed, with the parameters identified around pass and fail status, and that information is returned to the database to be reported. See example solution for more detailed code and explanations.

Configuration Data Serialization	**Configuration Data Class Example**

To verify that the "Test Configuration" information entered by the tester is correctly serialized going in and out of the database, use the DataContract attribute on the class and the DataMember attribute on any properties that need to be persisted.	`[DataContract]`
	`internal class SdkPullPrintExampleActivityData`
	`{`
	`  [DataMember]`
	`  public string TopLevelButtonName { get; set; }`
	`}`

Create the Configuration UI	**IPluginConfigurationControl.Initialize Example**

The UI used to obtain the configuration data needed to run the activity is created.
 
	`public void Initialize(PluginConfigurationData configuration, PluginEnvironment environment)`
	`{`
	`  var data = configuration.GetMetadata<SdkPullPrintExampleActivityData>();`
	
	`assetSelectionControl.Initialize(configuration.Assets, HP.ScalableTest.Framework.Assets.AssetAttributes.None);`
	
	`  textBoxButtonName.Text = _data.TopLevelButtonName;`
	`}`

Create object with configuration data to send to STB Database	**IPluginConfigurationControl.GetConfiguration Example**

Once collected, the configuration data is provided to the database as an object to be used during execution.	`public PluginConfigurationData GetConfiguration()`
	`{`
	`  _data.TopLevelButtonName = textBoxButtonName.Text;`
	`  return new PluginConfigurationData(_data, "1.0")`
	`  {`
	`  Assets = assetSelectionControl.AssetSelectionData`
	`  };`
	`}`

Execute Plugin Activity	**IPluginExecutionEngine.Execute Example**

A method is then called on to execute the plugin activity, either by starting the activity or iterate a new instance of the activity. The results of the activity are	`public PluginExecutionResult Execute(PluginExecutionData executionData)`
	`{`
	`  var result = new PluginExecutionResult(`
	`  PluginResult.Error, "Unknown");`

then logged and sent back to the STB database.	`_data = executionData.GetMetadata<SdkPullPrintExampleActivityData>();`
	`var device = executionData.Assets.GetRandom() as IDeviceInfo;`
	`IDeviceProxy proxy = DeviceProxyFactory.Create(device.Address, device.AdminPassword);`
	
	`var token = new AssetLockToken(device, TimeSpan.FromMinutes(5), TimeSpan.FromMinutes(5));`
	
	`ExecutionServices.CriticalSection.Run(token, () =>`
	`{`
	`result = PerformPullPrint(executionData, result, device, proxy);`
	`});`
	
	`return result;`
	`}`

Table 2.1 – STB Demo Example

Developing Custom STB Plugins - Getting Started

To develop a new, custom plugin, a new Microsoft Visual Studio solution (i.e., .sln file) must be created. To ensure strong design, the following steps are recommended:

1. Enumerate possible attributes that are needed as inputs to the workflow automated by the plugin, but that will vary between one automated execution of the test to the next.
2. Create the new MS VS solution using the plugin template.
3. Reference any needed libraries or configurations used.
4. Create the Plugin Activity Data
5. Create the Plugin Configuration Controls
6. Create the STB system calls for any devices or documents needed.
7. Add OXPt statements or calls to proprietary libraries to implement automation of the workflow steps that this plugin is automating
8. Configure a project to run the STB SDK Plugin Sandbox.
9. Test the new Plugin using the Plugin Sandbox.
10. Add the new Plugin to the STB.
11. Test the new STB Plugin using a test scenario to verify completion.

Enumerating Attributes of Workflow

The first step in designing the plugin is to determine all components of the workflow that will require automation. This includes detailing all the resources the plugin will need to access, any inputs that will vary from one automated execution of the test to the next (and that therefore need to be configured or entered by the tester before the test begins,) the steps the plugin will execute to implement the workflow being automated, and the results that should be displayed to the tester during the test or logged and saved for reporting:

1. Asset Management:
    a. Are devices needed to execute the workflow? If so, are there specific requirements for the devices that can be used?
    b. Are servers needed to execute the workflow? If so, what kind?
    c. Are print queues needed to execute the workflow?
2. Library Management:
    a. Are documents going to be scanned, emailed, or printed?
    b. Are specific file or document types needed?
3. Configuration data:
    a. What information needs to be passed in during each activity instance to complete the workflow?
    b. What steps of the workflow are configurable, and what steps are strictly controlled by the workflow code? Configurable items will vary from one automated test run to the next. Configurable items allow the tester to specify (i.e. enter) those variables just before execution time for the test.
    c. What configuration data is persistent, or what can be auto generated during tests?
4. Workflow steps:
    a. Are any device buttons being pressed? Are there any fields that need information passed into them or have information generated?
    b. How is authentication completed? Are there any delays between steps? How long should the workflow wait during delays before returning a skip or fail status?
    c. Are there different execution paths that can be used or should be captured?
    d. What should vary from one execution of the activity to the next?
5. Execution Reporting:
    a. What additional runtime status should be displayed for the tester during execution?
        i. Note that STB will produce a standard execution status dialog showing invocation, execution, and termination of activity through the plugins being used by Virtual Workers, and this is sufficient for almost all plugins. However, if additional status display is needed it can be generated.
    b. What results should be captured at the end of the workflow? What information should be passed back?
    c. How should errors encountered be displayed for the tester and logged?

Using the Plugin Template (Visual Studio Extension)

Developers may create a plugin without using the Plugin Template by starting with an empty class library (DLL) and writing the code needed for the plugin to interact with the STB (as illustrated in the training videos for the STB Plugin SDK.) However, the template creates a skeleton of code that pre-populates many required calls, and will compile immediately without modification.

1. Install the vsix template:
    a. Double-click the .vsix file to open the **VSIX Installer**.
    b. Click **Install** to complete the installation.

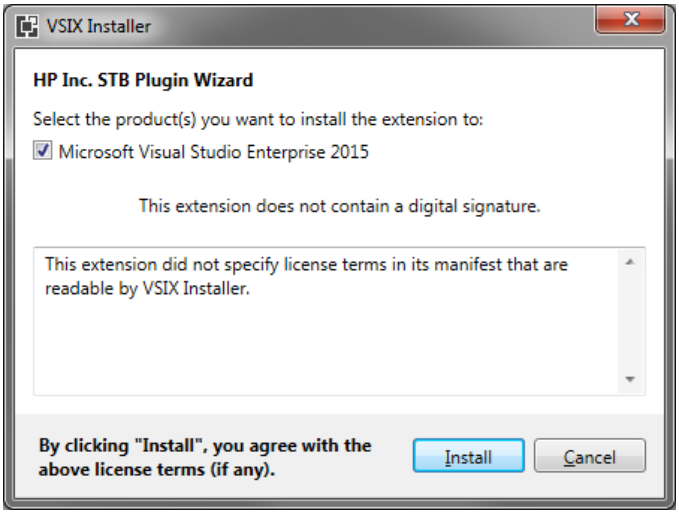


Fig 3.1 – Visx Installer

    c. Click **Close** to complete installation.
2. Open Visual Studio 2015.
3. Select **New** from the **File** drop-down menu, then select **Project** to open the **New Project** window.
4. Verify **.NET Framework 4.6** is selected as the target framework.
    a. **Warning:** Failure to select the proper framework may prevent the template from being displayed and/or working properly.
5. Select **HP Inc. STB Plugin Template** located under **Installed\Templates\Visual C#** file tree in the left panel.
6. Type the name of the plugin into **Name** using format “Plugin.<PluginName>”. <PluginName> must not contain any spaces (see Fig 3.2, *New Plugin Setup.*)
7. Type or browse to location to save plugin information.
8. Click **OK** to open Template Parameters pop-up.

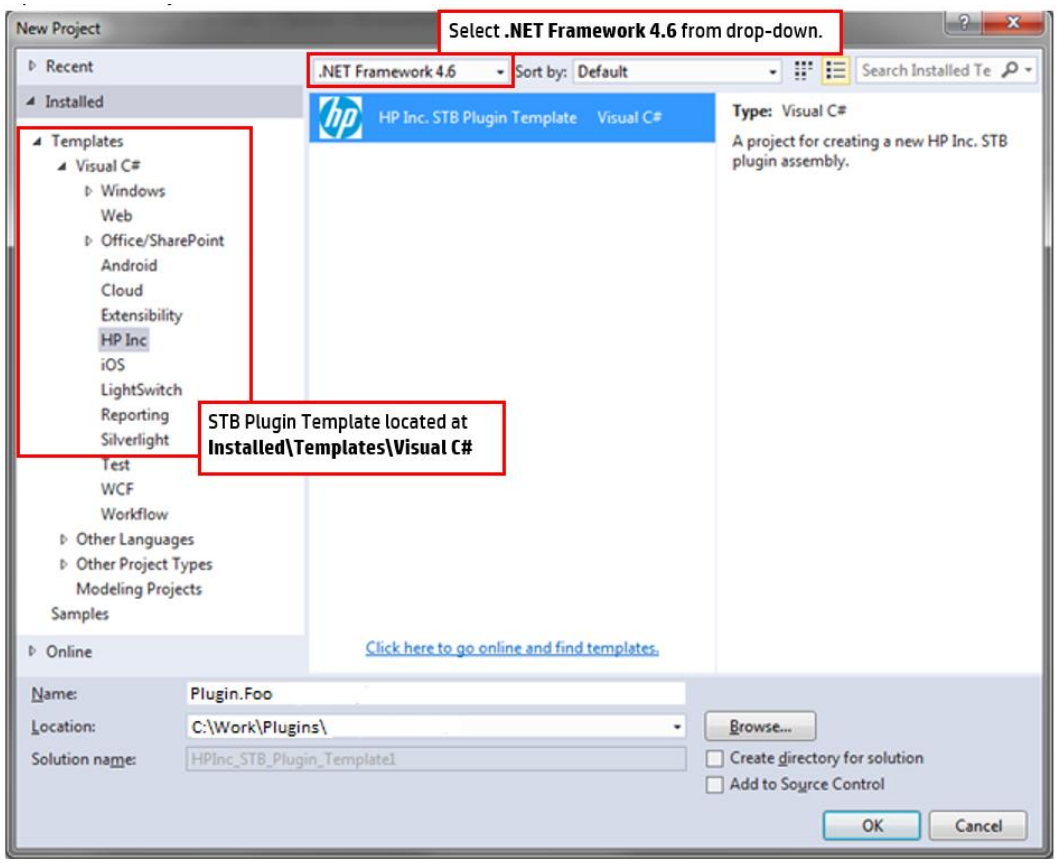


Fig 3.2 – New Plugin Setup

9. If desired, modify the default namespace for the plugin.
10. Browse to the location where you unzipped the PluginSdk.zip file.
11. Select STF.Framework.dll.
12. Click **OK** to create a new project.

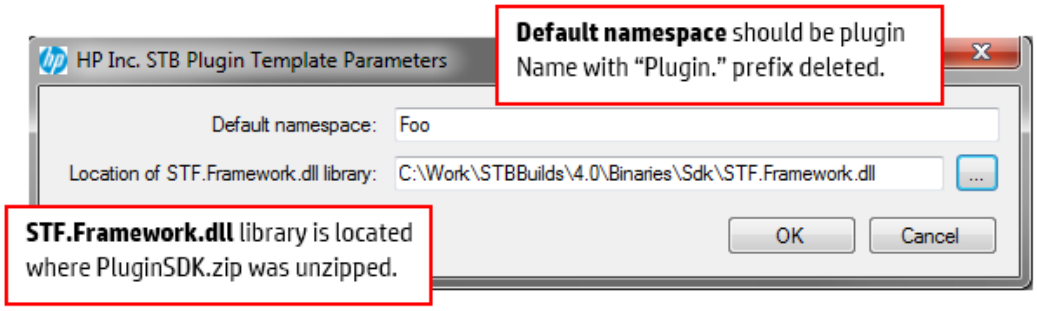


Fig 3.3 – Plugin Namespace and Library location

14. Visual Studio will create a new project with the following files:
    a. Plugin<PluginName>ConfigurationControl.cs file
    b. Plugin<PluginName>ExecutionControl.cs file
    c. Plugin<PluginName>ActivityData.cs file

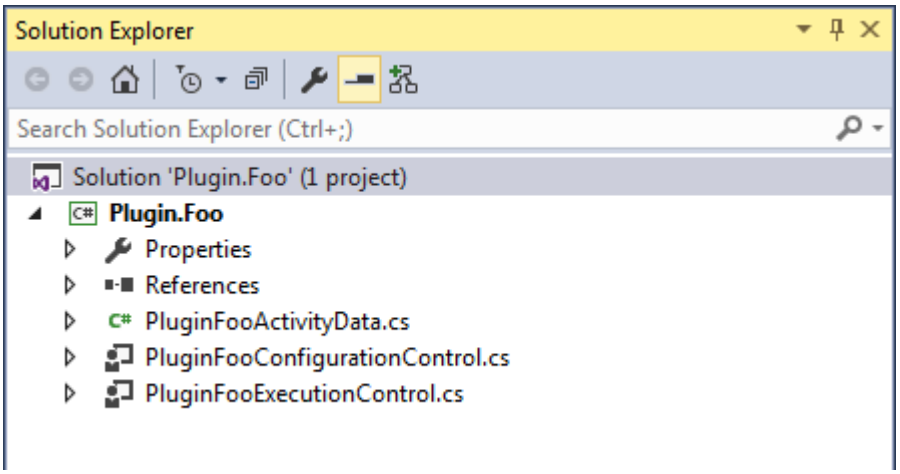


Fig 3.4 – STB Plugin Structure

Manually Set References

Automation workflows do not necessarily reside in the plugin themselves. Plugins can reference external assemblies that contain automation capability. These libraries might include:

- Custom components developed by you
- Components provided by HP (such as OXPt)
- Third party components

The components may reference custom configurations or proprietary libraries that need to be referenced by the plugin. These third party references can include but are not limited to code that:

- prints to a device
- manipulates a device control panel
- interacts with or configures a solution server
- installs a solution to a device
- interacts with a Solution's admin console to change settings for the Solution server or the Solution on the device.

Note: The STB SDK does not inherently provide automation capabilities – these must be obtained from HP, developed independently, or obtained from a third party.

To link a third party library or reference:

1. Open Visual Studios 2015.
2. Open the solution.
3. Right click **References** in the **Solution Explorer** window.
4. Select **Add Reference** from the drop-down menu to open the **Reference Manager** window.
5. Select the reference to be added.

Note: OXPt should be referenced in this way if OXPt will be used to automate activity and workflow steps being executed on an HP device.

Defining Plugin Activity Data

Once needed references have been linked to the new Plugin, define the Activity Data by creating properties for each piece of data needed by the Plugin (enumerated in the earlier design step.)

Items that are stored in STB, such as devices, servers, and documents, have a separate storage mechanism and should not be included in plugin activity data

This class will utilize the DataContract and DataMember attributes to serialize the data when communicating with the STB database.

Example of Enumerated ActivityData	Example of ActivityData code

Each point of enumerated data should be defined and tagged appropriately.
 	
For example, if your plugin was sending emails, some information to be captured might include:
 	
• "To" recipients
• "CC" recipients
• "Subject" of email
	`namespace HP.ScalableTest.Plugin.Email`
	`{`
	`  /// <summary>`
	`  /// Represents the data needed to execute an Email send activity.`
	`  /// </summary>`
	`  [DataContract]`
	`  public class EmailActivityData`
	`  {`
	`  /// <summary>`
	`  /// Gets or sets the number of "To" recipients.`
	`  /// </summary>`
	`  [DataMember]`
	`  public int ToRandomCount { get; set; }`
	
	`  /// <summary>`
	`  /// Gets or sets the number of "CC" recipients.`
	`  /// </summary>`
	`  [DataMember]`
	`  public int CCRandomCount { get; set; }`
	
	`  /// <summary>`
	`  /// Gets or sets the subject line of the e-mail activity.`
	`  /// </summary>`
	`  [DataMember]`
	`  public string Subject { get; set; }`

Table 3.1 – Enumerated Data Example

Designing Plugin Configuration Controls

Once needed data is defined, create the Configuration Control methods which will cause STB to display the UI elements that the tester will use to select or provide configuration data that is stored in the activity data object.

Microsoft Visual Studio includes the Windows Forms Designer, which allows the developer to drag and drop UI elements into a Windows Form, and auto-generates portions of the corresponding code for those elements in Plugin<PluginName>ConfigurationControl.Designer.cs.

It is assumed that developers are familiar with using the Windows Forms Designer feature of MS Visual Studio. Developers needing more guidance with it can start with Walkthrough: Getting Started with the Windows Forms Designer.

Example of Configuration Controls	Example of Plugin Configuration Controls code

Data that must be passed in by the tester must have an appropriate UI developed to pass in this information.
 	
In this instance, the Email Plugin would need to pass information from the Configuration UI to the “To”, “CC” and the “Subject” and “body” fields.
 	
In addition, the tester can select a radio button to determine attachments.
	`public void Initialize(PluginConfigurationData configuration, PluginEnvironment environment)`
	`{`
	`EmailActivityData emailData = configuration.GetMetadata<EmailActivityData>();`
	
	`attachments_DocumentSelectionControl.Initialize(configuration.Documents, GetDocumentFilter());`
	
	`exchange_ServerComboBox.Initialize(configuration.Servers.SelectedServers, "Exchange");`
	
	`toCount_NumericUpDown.Value = emailData.ToRandomCount;`
	`ccCount_NumericUpDown.Value = emailData.CCRandomCount;`
	`subject_TextBox.Text = emailData.Subject;`
	`body_TextBox.Text = emailData.Body;`
	`attachAll_RadioButton.Checked = emailData.SelectAllDocuments;`
	`attachSome_RadioButton.Checked = !emailData.SelectAllDocuments;`
	`documentCount_NumericUpDown.Value = emailData.NumberOfDocuments;`

Table 3.2 – Configuration Control Example

Remember that the STB Framework Services provide a number of controls that allow selection of devices, Servers, and Print queues to test with from Asset Inventory, and Documents to test with from the Document library. Please see the Example Plugins, the STB Plugin SDK training videos, and the API reference file (chm file) for more details on how to use these framework services.

Developers may write validation code within <PluginName>ConfigurationControl that checks that inputs entered by the tester meet certain requirements (for example, that text boxes have a value and are not blank, that an asset selected has certain attributes like a scanner or control panel, etc.)

Writing workflow activity

Once the configuration control has been created, create the workflow that will be executed by the plugin. Each step of the activity must be automated by the appropriate method. For automation that interacts with an HP device, developers are strongly encouraged to take advantage of the automation methods provided in the OXPt SDK.

Example of Workflow Activity	Example of Workflow Activity code

The workflow provides automation for the entire activity.
 	
Once executed, this should encompass everything from creation of the Control UI to final logging of results.
	`public partial class EmailExecutionControl : UserControl, IPluginExecutionEngine`
	`{ /// Constructs the Exchange Email control.`
	`/// </summary>`
	`public EmailExecutionControl()`
	
	`/// Picks random recipients for an e-mail message`
	`private void PickRandomRecipients`
	
	`/// Populate the email form.`
	`/// </summary>`
	`private void PopulateControl`
	
	`/// Adds a list of attachments to the ListBox`
	`/// </summary>`
	`private void AddAttachmentsToList(DocumentCollection attachments)`
	
	`/// User clicked the "Send" button.`
	`/// </summary>`
	`private void emailSendButton_`
	
	`{`
	`//Log Server usage`
	`ActivityExecutionServerUsageLog serverLog = new ActivityExecutionServerUsageLog(executionData, server);`
	`ExecutionServices.DataLogger.Submit(serverLog);`
	`}`
	`}`

Table 3.3 – Workflow Execution Example

Debugging STB Plugins

Prior to adding the plugin to STB, it can be loaded into the STB Plugin Sandbox so it can be executed and tested in an isolated environment.

STB Plugin Sandbox Framework Simulator

To open the Plugin Sandbox:

1. Open SdkPluginSandbox.csproj (this is provided in the SDK zip file.)
2. Build the Plugin Sandbox.
3. Add a reference in the Sandbox to your new plugin.
4. Modify the lines in the Sandbox Program.cs to be as follows:
   simulator.ConfigurationControlType = typeof(<PluginName>ConfigurationControl);
   simulator.ExecutionEngineType = typeof(<PluginName>ExecutionEngine);
5. Execute the Plugin Sandbox. Execution of the Sandbox will simulate what happens when STB loads the configurations control and execution engine of your new plugin.
6. The Sandbox displays the same control that STB would display when configuring a Test Scenario that uses your new plugin.

Plugin Configuration Control

The Plugin Configuration tab will display:

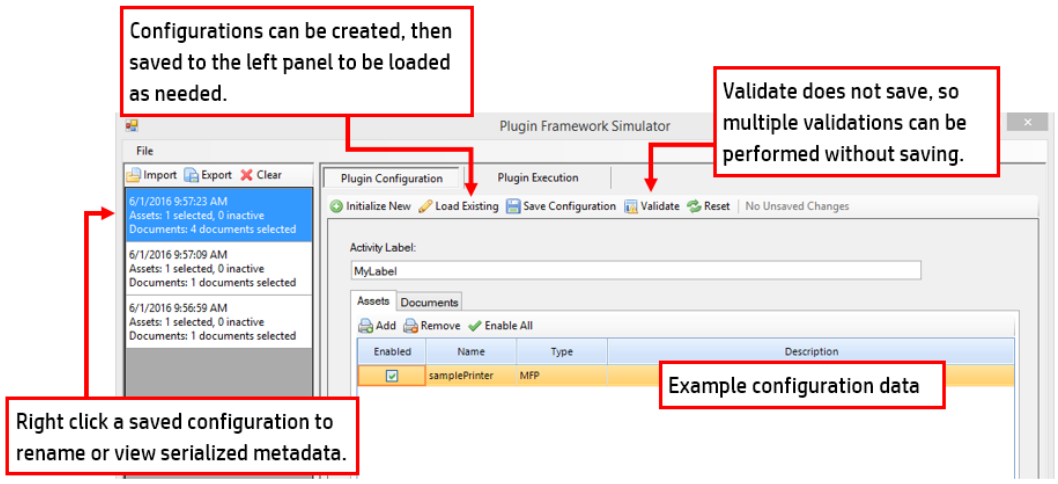


Fig 4.1 – Configuration Control Example

The configuration control screen will allow different configurations to be created and saved to the memory. The configuration will display assets added to be used to debug new Plugins.

Calls to AssetInventory or the Document Library in your plugin call into STB to access STB's database of devices, documents, etc. When testing a plugin using the Plugin Sandbox, these calls do not connect to an actual STB database. Therefore, when testing within the Plugin Sandbox, additional calls can be used to mock or simulate the data returned by a real STB database for the purposes of testing your plugin:

1. Use AssetInfoBuilder to create mock devices as applicable.
2. Use DocumentBuilder to create mock documents as applicable.

To validate configurations:

1. Click **Validate** to validate all fields.
2. The Plugin Sandbox will execute any validation checks on data entered by the user on the UI by executing ValidateConfiguration(). Errors will be thrown on any validation checks that fail because the input did not meet the input requirements encoded in the plugin via the fieldValidator calls the developer has added.

Warning: When using AssetInfoBuilder or DocumentBuilder to simulate or mock the interactions of the plugin with STB's database, remember that the info that otherwise would be returned from STB's database is randomly generated, and therefore, although the data can be saved and loaded, it is easily invalidated and may not match when loaded back in.

Plugin Execution

By clicking on "Plugin Execution", the Plugin Execution tab will display. The Sandbox will now simulate STB's execution of your new plugin:

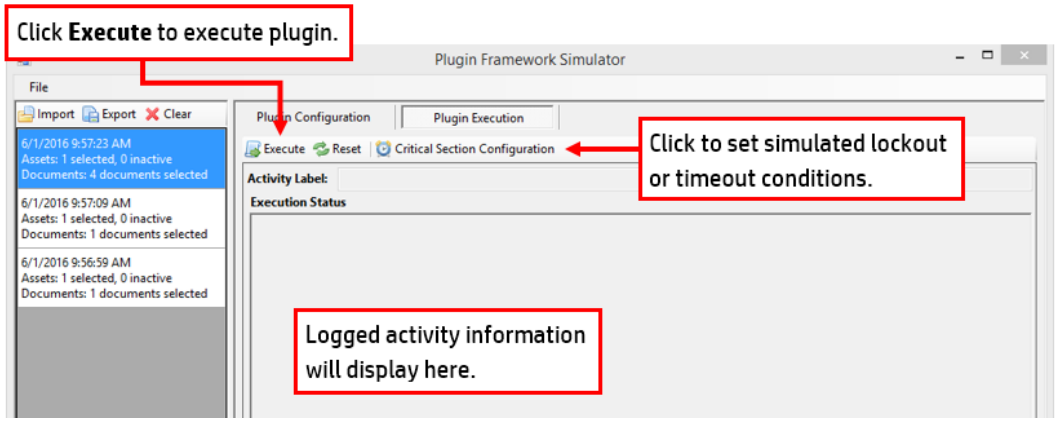


Fig 4.2 – Execution Example

Once a configuration is created and saved, it can be executed. To execute:

1. Select a set of configuration data with which to run in the left hand pane.
2. If applicable, click **Critical Section Configuration** to set simulated lockout conditions.
3. Click **Execute** to pass configuration data in, convert it to execution data, and run the activity.
4. Review the Data Logger Output file representing Data logger output to verify expected results are received.

Compiling STB Plugin assemblies

Once the new plugin is working in the test harness, it can be built/compiled and is ready to be added to STB.

Adding Plugins to STB

Once a plugin is compiled, activate the plugin using the following steps:

1. Verify the plugin assembly file name format is “Plugin.<PluginName>.dll”
2. Verify the plugin assembly implements IPluginConfigurationControl and IPluginExecutionEngine.
3. Open the **STB Admin Control Panel.**
4. Click **Activity Plugin References** to open the **Plugin Reference Management** window.

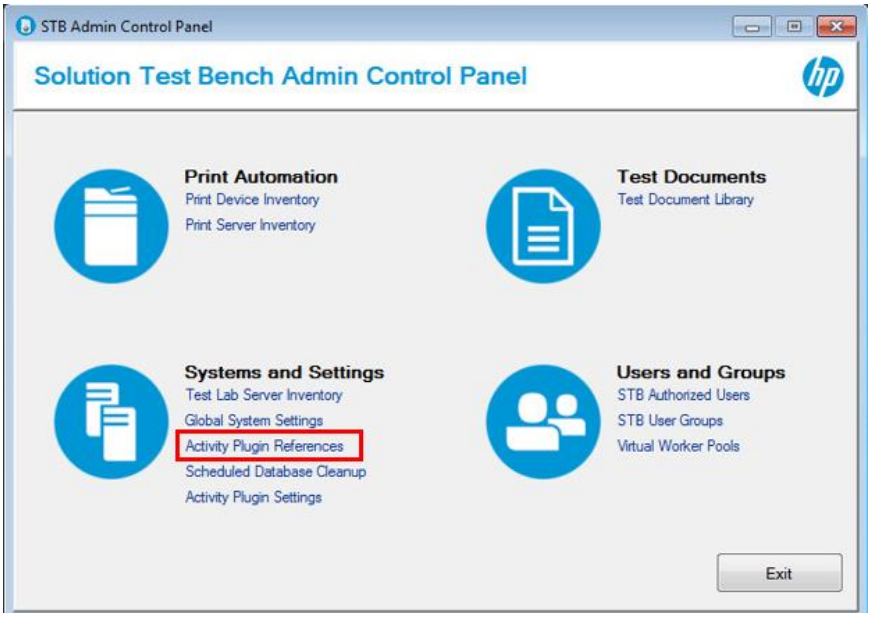


Fig 5.1 – Admin Control Panel

5. Click **New** to open the **Add Plugin Reference** pop-up.
6. Type the name of the plugin being added.
    a. **Warning:** This plugin name must EXACTLY match the latter portion of the plugin assembly file name. For instance, a plugin assembly named "Plugin.Foo.dll" must have the name entered as "Foo".
7. Click **OK** to open the **Plugin Reference** window.

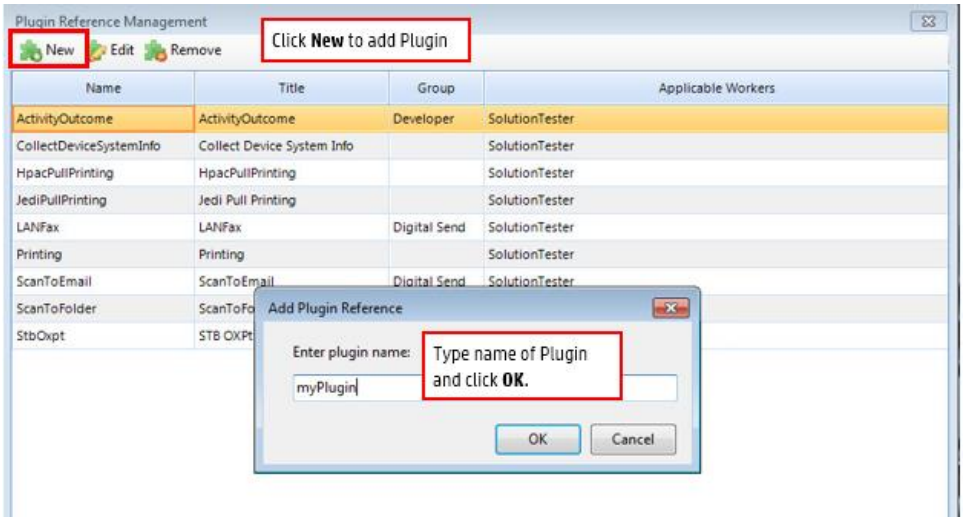


Fig 5.2 – Add Plugin Reference

8. Type a title in the **Plugin Title** field.
9. Select a **Category/Group** if applicable.
10. Click **Solution Tester** checkbox in the **Applicable Resources** field.
11. Click **OK** to return to the **Plugin Reference Management** window.

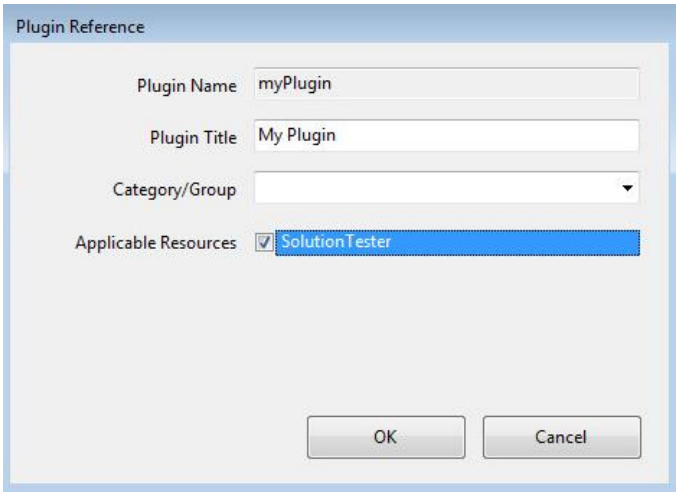


Fig 5.3 – Plugin Reference Management

12. Click **Apply** to complete the plugin registration on the server.
13. Deploy the new plugin by copying the plugin assembly and any necessary dependencies to the STB plugin folder. The typical path is "C:\VirtualResource\Distribution\Plugin", though this can change if you installed the STB to a non-default location.

a. **Note:** This step must be performed on the server and each STB client separately.

14. To verify the plugin has been added:
    a. Open the STB User Console.
    b. Select the **Test Configuration** tab.
    c. Right-click **Test Scenarios**, select **New** and then select **Test Scenario** from the drop-down menu.
    d. Click on **Add** to display a drop down.
    e. Select **Solution Tester**.
    f. Select the **Worker Activities** tab.
    g. Select the **Main Activities** tab.
    h. Click **Add** and verify your plugin is listed as an available option.

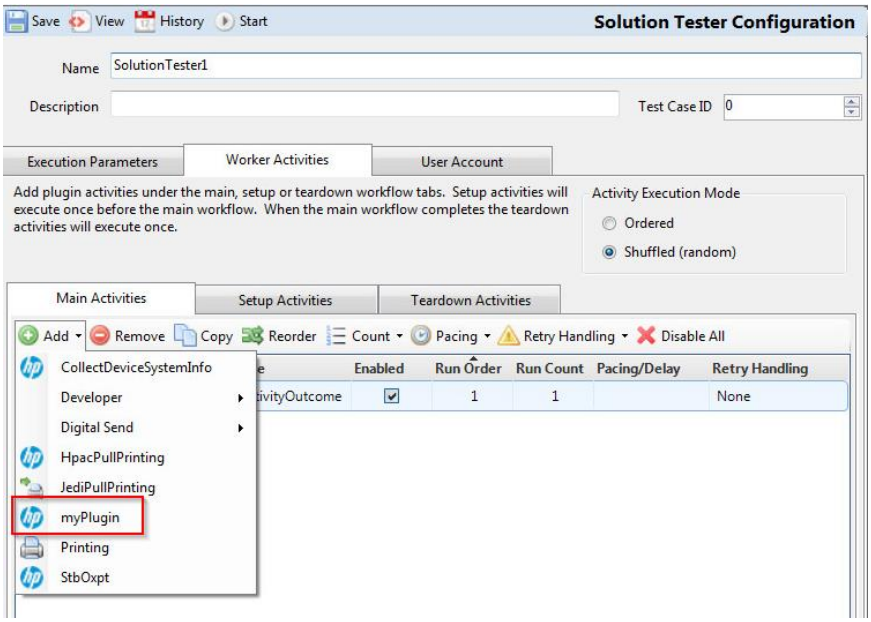


Fig 5.4 – Plugin Example

    i. Select your plugin and verify the correct configuration control is provided when selected.

To verify the plug in works as intended, run it in a STB Test Scenario to verify behavior.