Systinet

Software Version: 10.03
Windows and Linux Operating System

Assertion Editor User Guide

Document Release Date: April 2017
Software Release Date: April 2017

Hewlett Packard Enterprise

Legal Notices

Warranty

The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Documentation Updates

To check for recent updates or to verify that you are using the most recent edition of a document, go to: https://softwaresupport.hpe.com/.

This site requires that you register for an HP Passport and to sign in. To register for an HP Passport ID, click **Register** on the HPE Software Support site or click **Create an Account** on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HPE sales representative for details.

Support

Visit the HPE Software Support site at: https://softwaresupport.hpe.com/.

Most of the support areas require that you register as an HP Passport user and to sign in. Many also require a support contract. To register for an HP Passport ID, click **Register** on the HPE Support site or click **Create an Account** on the HP Passport login page.

To find more information about access levels, go to: https://softwaresupport.hpe.com/web/softwaresupport/access-levels.

HPE Software Solutions Now accesses the HPSW Solution and Integration Portal website. This site enables you to explore HPE Product Solutions to meet your business needs, includes a full list of Integrations between HPE Products, as well as a listing of ITIL Processes. The URL for this website is https://softwaresupport.hpe.com/km/KM01702731.

About this PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

Contents

Chapter 1: Assertion Editor Guide

Welcome to the *Assertion Editor User Guide*. This guide explains how to use Assertion Editor as part of HPE Systinet.

This guide contains the following chapters:

Chapter 2: Assertion Editor

HPE Systinet includes Assertion Editor, a set of features for use with the Policy Manager component of HPE Systinet. Assertion Editor enables you to create, edit, and delete assertions on any number of Policy Manager servers. In addition, you can use Assertion Editor to test an assertion, validating the assertion against a source document.

This chapter introduces Assertion Editor in the following sections:

Overview

Assertions are the building blocks of policy. Each assertion checks a single condition of a policy, returning a true or false result. In Policy Manager, one or more assertions are collected together to form a *technical policy*. The technical policy is a set of assertions that fulfills a management requirement.

HPE Systinet provides tools to test whether sources comply with the relevant policies.

To meet management requirements, a technical policy often needs a new assertion. Changing requirements can also result in existing assertions becoming out of date. Assertion Editor is a tool, built on the widely used Eclipse IDE, to simplify assertion creation and editing.

Assertion Editor makes working with assertions easy.

Use Assertion Editor to do the following:

1. **Create an assertion project.**

   For details, see the following sections:

   - "Creating an Assertion Project File" on page 11
   - "Downloading and Importing Assertions" on page 12

2. **Create and manage assertions.**

   For details, see the following sections:

- "Creating Assertions" on page 15
- "Editing Assertions" on page 16
- "Deleting Assertions" on page 21
- "Comparing Assertion Versions" on page 21

3. **Validate assertions before publishing.**

   For details, see "Testing Assertions" on page 23.

4. **Deploy assertions and manage conflicts.**

   For details, see the following sections:

   - "Publishing Assertions" on page 24
   - "Resolving Conflicts" on page 24

5. **Customize assertions for use with Policy Manager.**

   For details, see "Customizing Assertions" on page 28.

User Interface

The default perspective is split into a number of sections with menu options across the top.

The platform perspective consists of the following views:

- **Project Explorer**

  The tree view of your assertion projects. For details, see "Project Explorer" below.

- **Server Explorer**

  The view listing HPE Systinet server connections to HPE Systinet Workbench. For details, see Server Explorer.

- **Editor**

  The view showing the components of the assertion. For details, see "Editor View" on page 9.

Project Explorer

The Project Explorer contains a hierarchical list of projects, the assertions in each project, and the validation definitions in each assertion.

The Project Explorer contains additional context menu options enabling you to interact with a running HPE Systinet server. Right-click the project name or a particular assertion, and select **HPE Systinet** to view the options listed in the following tables:

Project Context Menu Options

Option	Function
Download Assertions	Import Assertions from HPE Systinet. For details, see "Downloading and Importing Assertions" on page 12.
Upload to Server	Export assertions to the default HPE Systinet server. For details, see "Publishing Assertions" on page 24.
Update from Server	Update assertions from HPE Systinet. For details, see "Editing Assertions" on page 16.
Remove from Server	Delete assertions from HPE Systinet. For details, see "Deleting Assertions" on page 21.
Upload To Other Server	Export assertions to a specified HPE Systinet server.
Build Extension	Create an assertion extension for HPE Systinet containing all the assertions in your project. For details, see "Building an Assertion Extension" on page 26.

Assertion Context Menu Options

Option	Function
Upload to Server	Export an assertion to the default HPE Systinet server. For details, see "Publishing Assertions" on page 24.
Update from Server	Update assertions from HPE Systinet. For details, see "Editing Assertions" on page 16.
Remove from Server	Delete the assertion from HPE Systinet. For details, see "Deleting Assertions" on page 21.
Upload To Other Server	Export an assertion to a specified HPE Systinet server.
Build Extension	Create an assertion extension for HPE Systinet containing all the assertions in your project. For details, see "Building an Assertion Extension" on page 26.

Server Explorer

The Server Explorer displays the HPE Systinet servers connected to HPE Systinet Workbench. The functionality is shared by all the HPE Systinet Workbench editors.

Right-click a server in the Server Explorer to open the context menu described in the following table:

Server Explorer Context Menu Options

Option	Function
New Server	Add a server for downloading assertions and taxonomies (Assertion Editor, Taxonomy Editor, and Customization Editor).
Remove Server	Delete a server from the Server Explorer.
Download Taxonomy	Download a taxonomy from a server (Taxonomy Editor and Customization Editor).
Download Assertion	Download assertions from a platform server (Assertion Editor).
Download Report	Download reports from a reporting server (Report Editor).
Properties	View and edit the server name, URL, username, and password.

Editor View

The Editor view is the main feature of the Assertion Editor UI.

The pane is split into the following tabs:

- **Overview Tab**

  The Overview tab shows the components of the assertion.

  The tab is divided into the following areas:

  - **General Information**

    Name of the assertion and its description.

  - **Implementation**

    List of implementations of validation logic and the artifact types to which they apply.

- **Reference Template**

  Element used to reference this assertion from a WS-Policy document.

- **Implementation Tab**

  The Implementation tab includes a list of implementations.

  Highlighting an implementation opens the XQuery Definition Editor in the window beneath. For details, see "Writing XQuery Definitions" on page 18.

- **Source Tab**

  The Source tab is an XML editor for editing the assertion.

Chapter 3: Getting Started

This chapter describes the prerequisites for working with assertions in Assertion Editor. It contains the following sections:

Creating an Assertion Project File

To work with assertions, you need an Assertion Project. You can create any number of Assertion Projects to help organize your work.

To Create an Assertion Project:

1. Do one of the following:
   a. In the HPE Systinet Workbench Welcome page, click **Create Assertion Project**.
   b. Click **New** to open the Select a Wizard window and select **HPE Systinet > Assertion Project**.
   c. From the menu, select **File > New > Assertion Project**.
   d. Press **Alt+Shift+N**, and then press **R** to open the Select a Wizard window. Then select **HPE Systinet > Assertion Project**.

   The New Assertion Project dialog box is displayed.
2. In the New Assertion Project dialog box, enter the following parameters:

Parameter	Definition
Project Name	The name of your assertion project.
Namespace	The namespace to apply to all assertions in the project.
Create from Existing	Select this option if you want to create a new project from a previous assertion extension. If selected, input the path or browse for the location of the assertion

Parameter	Definition
Extension	extension.
Use Default Location	If selected, Assertion Editor stores the project in your default workspace. If unselected, input the path or browse for an alternative workspace.

3. Click **Next** to select or create a server.

> **Note:** If no servers are currently defined, the dialog box continues to Step 5.

4. Do one of the following:
   a. Select **Create a New Server** and click **Next**. Go to Step 5.
   b. Select **Use an Existing Server**, select the server from the list and input its credentials, and then click **Next**. Go to Step 6.
5. In the New Server dialog box, add the required parameters, and then click **Next**.
6. Select the assertions to download from the server.
7. Select **Download All Taxonomies** to import taxonomies from HPE Systinet to make them available for use as assertion parameters.
8. Click **Finish**.

Downloading and Importing Assertions

Using Assertion Editor, you can download assertions from an HPE Systinet server to edit or test them.

You can download assertions in one of two ways:

- When you create a project, as described in "Creating an Assertion Project File" on the previous page.
- From your local file system, at a later date.

> **Caution:** If you import assertions containing manual validation, Assertion Editor highlights the manual validation as an error with a message instructing you to remove it from the assertion.

To Download Assertions:

1. Right-click the server containing the assertions you need in Server Explorer to open its context menu, and select **Download Assertions**.

The Download Assertion dialog box is displayed.

2. Select the assertions to download, and click **Next**.

   The Choose Location dialog box is displayed.

3. Select the project to add the assertions to, and click **Finish**.

To Import Assertions from a Local File:

1. Right-click at Assertion project name in Project Explorer to open its context menu, and select **Import > Assertions**.

   The Import Assertion dialog box is displayed.

2. Select the assertions to import, and click **Next**.

   The Choose Location dialog box is displayed.

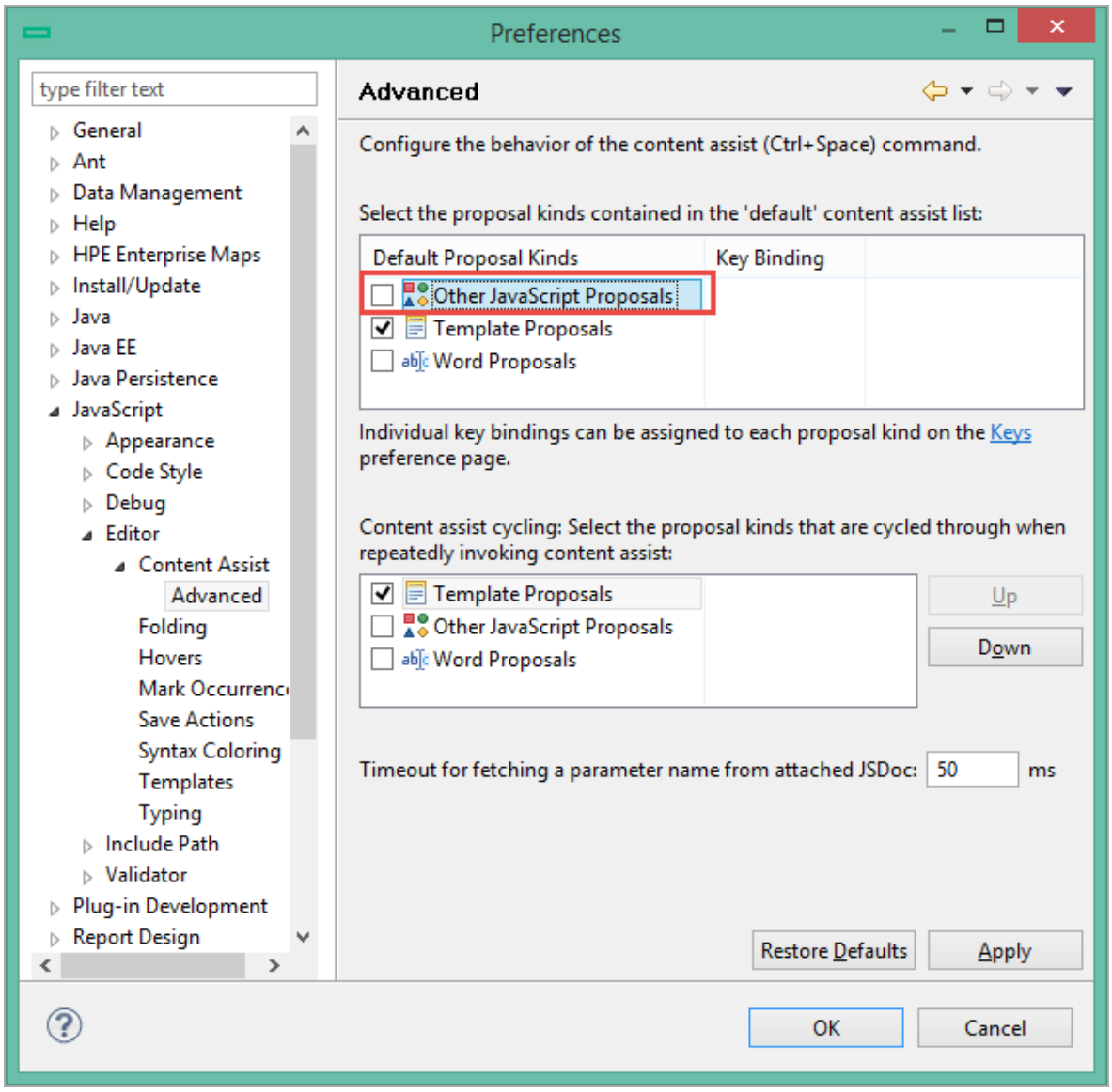
3. Select the project to add the assertions to, and click **Finish**.

The assertions are imported to your project.

Assertion Editor Notes

> **Note:** In case you use HPE Systinet Workbench to implement Javascript assertions, the built-in Javascript editor might throw some unexpected errors.

To resolve this, uncheck **Other JavaScript Proposals** in **Window > Preferences > JavaScript > Editor > Content Assist > Advanced**.


Preferences
type filter text
General
Ant
Data Management
Help
HPE Enterprise Maps
Install/Update
Java
Java EE
Java Persistence
JavaScript
Appearance
Code Style
Debug
Editor
Content Assist
Advanced
Folding
Hovers
Mark Occurrenc
Save Actions
Syntax Coloring
Templates
Typing
Include Path
Validator
Plug-in Development
Report Design
Advanced
Configure the behavior of the content assist (Ctrl+Space) command.
Select the proposal kinds contained in the 'default' content assist list:
Default Proposal Kinds
Key Binding
Other JavaScript Proposals
Template Proposals
Word Proposals
Individual key bindings can be assigned to each proposal kind on the Keys preference page.
Content assist cycling: Select the proposal kinds that are cycled through when repeatedly invoking content assist:
Template Proposals
Other JavaScript Proposals
Word Proposals
Up
Down
Timeout for fetching a parameter name from attached JSDoc:
50
ms
Restore Defaults
Apply
OK
Cancel

Chapter 4: Managing Assertions

This chapter explains how to work with assertions, as detailed in the following sections:

Creating Assertions

In "Creating an Assertion Project File" on page 11, you created an Assertion Project and looked at how to download and import assertions. The following section explains how to create new assertions.

To Create a New Assertion:

1. Do one of the following:
   a. Click **New** to open the New: Select a Wizard dialog. Expand **HPE Systinet > Assertion** and click **Next**.
   b. Select **File > New > Assertion**.
   c. Press **Alt+Shift+N** to open the context menu, and select **Assertion**.

   The New Assertion wizard is displayed.
2. In the New Assertion wizard, enter the required parameters.
3. Click **Finish** to create the assertion.
4. Double-click the assertion in Project Explorer to open it in the Editor, and do the following:
   a. Add an implementation, as described in "Adding and Deleting Implementations" on the next page.
   b. Test the assertion, as described in "Testing Assertions" on page 23.
   c. Publish the assertion, as described in "Publishing Assertions" on page 24.

Editing Assertions

The heart of Assertion Editor's functionality is the ability to edit assertions. To edit an assertion, you must have a local copy.

> **Caution:** If you are editing an assertion that also exists on a server, you must update your local copy before editing it. Editing a local assertion before updating it from the server can result in a revision conflict. Assertion Editor warns you if this is the case. For details, see "Resolving Conflicts" on page 24.

To Update an Assertion from the Server:

1. Right-click the assertion in Project Explorer to open its context menu.
2. Select **HPE Systinet > Update from Server**.

The main functionality of editing assertions is described in the following sections:

Editing General Properties

General properties are the name and text description of the assertion. Changing the name in the editor does not change the file name or reference template local name. These can only be changed in the General Properties section of the Overview tab of the Assertion Editor View. For details, see "Editor View" on page 9.

Adding and Deleting Implementations

An implementation contains a resource type and the code used to validate that resource type. An assertion must contain one or more implementations.

To Add an Implementation:

1. In the Implementation field of the Editor view, click **New**.

   The Define New Implementation wizard is displayed. For details, see "Define New Implementation Wizard" on page 34.

2. Do one of the following:

   a. To use a predefined resource type, select **Select an Existing Type** and select a resource type from the list. Use filter to find a specific resource type and uncheck **Show common types only** to list all available resource types and then click **Next**.

      Go to Step 4.

   b. To manually define a new resource type, select **Define New Type** and then click **Next**.

3. Input a Namespace and Local Name, or click **Load from file** to use a document in your assertion project, and then click **Next**.

4. Select the implementation type from the list and then click **Finish**.

To Delete an Implementation:

1. Open the Editor view and select the **Overview** tab.

   Implementations in your project are displayed in the Implementation window.

2. To delete an implementation, select it and click **Delete**.

After adding the implementation, open the Implementation tab of the Editor and edit the XQuery, JavaScript or XPath definitions to meet your needs. For instructions, see "Writing XPath Definitions" below or "Writing XQuery Definitions" on the next page or "Writing JavaScript Definitions" on page 19.

Writing XPath Definitions

After creating an implementation that uses an XPath validation handler, as described in "Adding and Deleting Implementations" on the previous page, you need to write the XPath definition.

To Write an XPath Definition:

1. Open the Editor view and select the **Implementation** tab.
2. Import a sample XML document of the type to which the assertion applies.
3. In the XPath Definition Editor, under **Load XML Template**, select one of the following links:

a. Click **From Resource** to load a sample XML document from your Assertion Editor project.

b. Click **From File** to load a sample XML document from your local file system.

c. Click **From URL** to load a sample XML document from the Web.

The XML document is displayed in the XML Template tab.

To Add an XPath Expression:

1. Right-click the relevant line in the sample XML document to open its context menu.
2. Select **Generate XPath Expression**.

   The XPath expression is displayed in the XPath Definition Editor field.

   > **Note:** You can have only one XPath expression for each implementation. An artifact passes validation if at least one XML node matches the XPath expression.

3. Modify the XPath expression in the XPath Definition Editor, if necessary.
4. If the XPath contains any unresolved namespace prefixes, an unresolved warning is displayed. If you receive a warning, go to Step 5. Else, go to Step 8.
5. Click the unresolved prefix link.

   The Manage prefix and namespace pane is displayed.
6. Define the namespace of the prefix, as follows:

   a. To add a namespace, click **Add** and then enter the required parameters.

   b. To delete a namespace, select it and click **Remove**.
7. Click **OK**.
8. To test the XPath expression, click **Test Expression**.

   The results of the test appear in the Test Results tab of the XPath Expression Editor.

For more information, see "XPath Assertions" on page 44.

Writing XQuery Definitions

Assertion Editor incorporates syntax highlighting for writing and editing XQueries.

To Write an XQuery Definiton:

1. Open the assertion in the Editor view, open the Implementation tab and click **New**.

   The Define New Implementation dialog box is displayed, as shown in "Define New Implementation Wizard" on page 34.

2. To use a predefined source type:

   a. Select the source type you need from the list and click **Next**.

   b. In the Implementation type, select **XQuery** from the list, and click **OK**.

3. To manually define a source type:

   a. Select the Define new type.

   b. Enter the required parameters and click **Next**.

   c. In the Implementation type, select **XQuery** from the drop-down list, and click **OK**.

   The XQuery Definition is displayed in the Editor view.

4. Edit the XQuery Definition, as described in "Editing XQuery Definitions", and click **Test Assertion**.

   If the assertion passes validation, you can now publish the assertion. For details, see "Publishing Assertions" on page 24.

   If the assertion does not pass validation, you can resolve any problems. For details, see "Resolving Conflicts" on page 24.

For more information, see "XQuery Assertions" on page 45.

Writing JavaScript Definitions

To write JavaScript definition:

1. Open the assertion in Editor view, click the Implementation tab and then click **New**.

   The Define New Implementation dialog box is displayed, as shown in "Define New Implementation Wizard" on page 34.

2. To use a predefined source type:

   a. Select the source type you need from the list and click **Next**.

   b. In the Implementation type, select **JavaScript** from the list and click **OK**.

3. To manually define a source type:

a. Select the Define new Type.

b. Enter the required parameters and click **Next**.

c. In the Implementation type, select **JavaScript** from the list and click **OK**.

Editing Reference Templates

The referencing template defines the element used to reference an assertion from a Technical Policy document. The template can include parameters which represent requirements whose specific values might vary.

To Edit an Assertion's Reference Template:

1. Open the **Overview** tab in the Editor view.
2. In the Reference Template pane, enter the required parameters.
3. Do one of the following:
   a. To add a parameter, click **New**.
   b. To edit an existing assertion, highlight it, and then click **Edit**.

   The Define Parameter wizard is displayed.
4. Input a name, description, select if the parameter is optional or required.
5. Do one of the following:
   a. Select **Primitive** type and the parameter type from the drop-down list.
   b. Select **Taxonomy** and input or **Browse** for the relevant taxonomy.

      > **Note:** The available taxonomies depend on those imported from HPE Systinet when you created the project.

      To Update Taxonomies:

      i. Open the context menu for the assertion project and select **Properties** to open the Project Properties dialog box.

      ii. Select **Taxonomies** to view the list of downloaded taxonomies from the server.

      iii. Click **Download** to update the taxonomies in the project.
6. Click **OK**.

To preview the reference template in a technical policy, click **Preview assertion reference** to open the dialog box, and then enter example parameter values.

Deleting Assertions

If an assertion is no longer useful, you can delete it in one of the following ways:

To Delete a Local Copy of an Assertion:

1. Right-click the assertion in Project Explorer to open its context menu, and select **Delete**.

   > **Note:** Deleting a local copy of an assertion does not affect the version on the server.

To Delete an Assertion on a Server:

1. Right-click the assertion in Server Explorer to open its context menu, and select **Delete Assertion**.

   > **Note:** Deleting the version of an assertion that is on a server does not affect any local versions.

Alternatively, you can delete an assertion from the server directly from the Project Explorer. This gives you the option of deleting the local copy at the same time.

To Delete an Assertion from the Server and the Local Copy:

1. Right-click the assertion in Project Explorer to open its context menu, and select **HPE Systinet > Remove from Server**.
2. When prompted, you can either choose to delete resource from the local file system or retain the resources on the local file system.

Comparing Assertion Versions

Assertion Editor uses the Eclipse Compare function to track version numbers, enabling you to roll back an assertion to a previous version.

To Compare Versions of an Assertion:

1. Right-click the assertion in Project Explorer to open its context menu and select **Replace with > Local History**.

   The Replace with Local History window is displayed.

   > **Note:** Changes to XQuery implementations do not appear in this window. XQueries are held in separate, stand-alone files so they can be accessed by external XML editors. Use your editor's revision control feature for XQueries.

2. Compare the versions.
3. Click **Replace**, if you want to replace the current version with the one to which you are comparing it.

Chapter 5: Validating and Publishing Assertions

This chapter explains how to test assertions and deal with validation conflicts before publishing or exporting them. This is described in detail in the following sections:

Testing Assertions

Before publishing an assertion, you can test it.

To Test an Assertion:

1. Double-click the assertion in Project Explorer to open it in the Editor.
2. Click **Test Assertion**.

   The Run Configurations dialog box is displayed. For details, see "Run Configurations Dialog" on page 35.
3. Enter the required parameters, and click **Apply** to save the parameters, or **Revert** to roll back the changes.
4. Click **Run**.

   The test results appear in the Assertion Console view.

To Test a Different Assertion:

1. Click **Browse**.

   The Select Assertion window is displayed.
2. Browse for the required assertion.

3. Click **OK**.
4. Enter the required parameters and click **Run**.

To Select Source Files for Testing an Assertion:

1. Click **Add Documents** to locate the source file to add.
2. Enter the required parameter values in the Parameters table and then click **Apply** or click **Revert** to use the most recent parameter value.

   For information about assertion reference templates, see "Editing Reference Templates" on page 20.
3. Click **Run**.

Resolving Conflicts

Conflicts occur when there are differences between an updated local copy of an assertion and that on the server. Assertion Editor notifies you of the conflict, and asks if you want to force the update or publication.

Forcing an assertion to be updated overwrites any local changes that have been made. Forcing an assertion to be published overwrites any changes that were made to the version on the server.

The safest way to resolve such conflicts is to either cancel publication or update the assertion.

To Update a Conflicting Assertion:

1. Copy your local version of the assertion to a different location in Project Explorer.
2. Right-click the assertion to open its context menu, and select **HPE Systinet > Update Assertion**.

   A conflict warning is displayed.
3. Click **OK** to update the assertion.

   Assertion Editor overwrites the local copy of the assertion with the version on the server.

Publishing Assertions

After writing, editing, and testing an assertion, you can publish it to an HPE Systinet server.

> **Note:** In Project Explorer, assertions that have not been published are indicated by a question mark (?). Assertions that have been changed locally since they were last synchronized with the version on the server are indicated by a right arrow (>).

To Publish an Assertion:

1. Right-click the assertion in Project Explorer to open its context menu, and select **HPE Systinet > Upload to Server**.

Assertion Editor connects to the server and attempts to publish the assertion.

To Select a Server that is not in the Project:

1. Right-click the assertion to open its context menu, and select **HPE Systinet > Upload to Other Server**.

   The New Server wizard is displayed.

2. Follow the steps for adding a server, as described in "Creating an Assertion Project File" on page 11.

> **Caution:** If changes were made to the version on the server since you last synchronized, a conflict warning is displayed that asks whether you want to force publication. For details on conflict resolution, see "Resolving Conflicts" on the previous page.

Chapter 6: Deploying Assertions

This chapter explains how to deploy a set of assertions as an Extension Project. This is described in detail in the following sections:

Building an Assertion Extension

After publishing assertions, you can copy them to an Assertion extension.

> **Note:** In Project Explorer, assertions that have not been published are indicated by a question mark (?). Assertions that have been changed locally since they were last synchronized with the version on the server are indicated by a right arrow (>).

To Build an Assertion Extension:

1. Right-click the assertion project in Project Explorer to open its context menu, and expand **HPE Systinet > Build Extension** to open the location browser.
2. Enter a name for the extension project and browse for the location you want to save the project to, and then click **Save**.

   All assertions from the selected assertion project are copied to the Assertion extension.

Redeploying the EAR File

After using the Setup Tool to apply extensions or updates, you must redeploy the EAR file to the application server. For JBoss, you can do this using the Setup Tool.

To Redeploy the EAR file to JBoss:

1. Stop the application server.
2. Start the Setup Tool by executing the following command:

SYSTINET_HOME/bin/setup.bat(sh)

3. Select the **Advanced** scenario and click **Next**.
4. Scroll down, select **Deployment**, and then click **Next**.

   When the Setup Tool validates the existence of the JBoss Deployment folder, click **Next**.
5. Click **Finish** to close the Setup Tool.
6. Restart the application server.

Chapter 7: Customizing Assertions

Assertion Editor incorporates predefined elements that are suitable for most use cases. However, you can customize certain elements. Customization of assertions is described in the following sections:

Customizing Source Type

When you define the implementation of an assertion, you can either select from a list provided by Assertion Editor, or you can define your own source type.

To Manipulate Source Types:

1. From the menu, select **Window > Preferences**.

   The Preferences wizard is displayed.

2. Expand **HPE Systinet Assertion Editor** and select **Source Type**.

   A table is displayed with source type names, local names, and namespaces.

3. Do one of the following:

   a. To create a new source type, click **Add** to open the New Source Type window. Enter the required parameters, and click **OK**.

   b. To edit an existing source type, select it and click **Edit** to open the Edit Source Type window. Enter the required parameters, and click **OK**.

   c. To delete an existing source type, select it and click **Delete**.

Adding Policy Extensions

You can extend HPE Systinet with custom-written validation handlers, in addition to the XQuery and XPath handlers that are included in the distribution.

To Add a PM Extension to your Project:

1. Right-click the project in Project Explorer to open its context menu and select **Properties**.

   The Properties for HPE Systinet wizard is displayed.

2. Select **PM Extensions** to open a list of PM extensions in the project.

3. Do one of the following:

   a. Click **Add PM Extension** to open the Select Extension window. Select the required extension, and click **OK**.

   b. Click **Add External PM Extension** to open the Select PM Extension window. Browse for the required extension, and click **OK**.

After adding a PM extension to your Assertion Editor project, apply it to all relevant HPE Systinet servers with the Setup Tool.

Chapter 8: Java Assertion Demo

This demo shows how to create and use a custom assertion validator. You will learn how to:

- Create a custom assertion validator.
- Apply a custom assertion validator into HPE Systinet as an extension.
- Create an assertion in Assertion Editor based on the validator.
- Publish the assertion to the Platform repository.

You can find the demo sources in `SYSTINET_HOME\demos\policymgr\assertionvalidator`. It contains:

- An Eclipse project for developing a custom assertion validator (in the `validator` folder).
- An Assertion Editor project for developing a sample assertion (in the `assertion` folder).
- Demo data for testing our assertion validator (in the `demodata` folder).

The demo is divided into separate procedures, described in the following sections:

Creating the Assertion Validator

A sample Eclipse project is available in `SYSTINET_HOME\demos\policymgr\assertionvalidator\validator`. This project can be imported into Eclipse as an existing project. It has the following structure:

- `src/`

  A directory containing the Java sources.

- `lib/`

A directory containing external libraries used by assertion validator.

- `resources/extension.xml`

  A Policy Manager extension definition file.

- `build.xml`

  An Ant build file to build extension containing the validators.

The `src` folder contains a sample implementation of an assertion validator: `mycompany.validator.demo.ServiceNameValidator`. This validator applies to WSDL documents and checks whether the name of services defined in WSDL starts with the words "dummy," "test," "sample," or "example."

The assertion validator class must implement both methods of the interface `org.em.policy.validation.AssertionValidator`:

- `QName getDialect()` — Must return a QName which will be used in assertions to invoke this validator. The example uses the QName `{http://mycompany/validation}ServiceNameValidator`.
- `void validate(ValidationListener listener, SourceCollection sources, SourceType sourceType, ValidatedAssertion[] assertions, ValidationContext context)` — This is the validation method which is called when a source must be validated against an assertion using this validator. Our sample validator parses the XML content of the WSDL document and checks each service name (under XPath `/wsdl:definitions/wsdl:service/@name`) to see whether it starts with the unwanted words or not.

> **Note:** For more information about the Policy Manager's interfaces see the HPE Systinet Javadoc.

To build an extension from your assertion implementation, you need an extension definition file, which is available at `resources/extension.xml`. It contains a unique identifier (attribute uri) which identifies the extension, the extension name (element name), and the list of contained assertion validators (elements assertion-validator):

```
<?xml version="1.0"?>
<extension version="1.0" uri="mycompany.ext.demo">
  <name>Demo Assertion Validators</name>
  <assertion-validator class="mycompany.validator.demo.ServiceNameValidator" />
</extension>
```

To build the extension you can use Eclipse to start an ANT build using build.xml or use the following procedure:

To Build an Extension for the Assertion Validator:

1. Change your working directory to `SYSTINET_HOME\demos\policymgr\assertionvalidator`.
2. To get help on demo, use `demo.bat` or `demo.sh`.
3. Build the extension with the command **demo make**.
4. Check the created extension in `SYSTINET_HOME\demos\policymgr\assertionvalidator\validator\dist\Demos.jar`.

> **Caution:** If you want to use an Eclipse project for this demo, you must define the `EM_CLIENT_LIB` classpath variable in the Eclipse IDE. The `EM_CLIENT_LIB` variable must point to `SYSTINET_HOME/client/lib`.

Applying the Validator Extension

After you create `Demos.jar`, apply it to HPE Systinet as an extension.

To Apply the Extension to HPE Systinet:

1. Make sure the HPE Systinet server is not running.
2. Copy `SYSTINET_HOME\demos\policymgr\assertionvalidator\validator\dist\Demos.jar` into `SYSTINET_HOME\extensions`.
3. Execute `SYSTINET_HOME\bin\setup` and select the **Apply Extensions** scenario.
4. Start HPE Systinet.

Creating and Deploying the Assertion

There is a sample project for Assertion Editor in `SYSTINET_HOME\demos\policymgr\assertionvalidaton\assertion`.

You can use this sample project to create your own assertions. It already contains a demo assertion named `WSDLServiceNameIsNotDummy`. Browse this assertion to see that it is applicable to WSDL Documents and implements the `ServiceNameValidator` created in "Creating the Assertion Validator" on page 30 (linked through `SYSTINET_HOME\demos\policymgr\assertionvalidator\validator\dist\Demos.jar`).

The XML definition of the assertion is automatically filled out according to the definition in assertion validator (in the method `getDialect()`):

```
<my:ServiceNameValidator xmlns:my="http://mycompany/validation"
```

```
xmlns:pm="http://systinet.com/2005/10/soa/policy"/>
```

After creating the assertion, deploy it. Or deploy the existing `WSDLServiceNameIsNotDummy` assertion.

To Publish the Assertion to HPE Systinet:

1. Open Assertion Editor.
2. Click **File** menu and select **Import > Existing Projects into Workspace** and open the demo project.
3. In the Server Explorer, open the context menu and define a **New Server** pointing to your HPE Systinet server.
4. In the Project Explorer, open the Project context menu and select **Properties > HPE Systinet Server**, and select the server you just defined.
5. Publish the demo assertion from the Project Explorer.

   Open the `WSDLServiceNameIsNotDummy.asr` context menu and select **HPE Systinet > Upload to Server**.

For more details, see the "Publishing Assertions" on page 24.

Testing the Assertion Validator

In the previous sections you created and deployed an assertion validator and an assertion. The next step is to use this new assertion in a validation for a business test case. To do this, perform the following in HPE Systinet:

1. Publish a WSDL containing a service element where the name attribute starts with `Test`. For details, see the *HPE Systinet User Guide*.
2. Create a Technical Policy applicable to WSDLs and add the `WSDLServiceNameIsNotDummy` assertion to it.
3. Create a Policy Report using artifact type, `WSDL`, and the new technical policy. For details, see *How to Create Policy Reports* in the *HPE Systinet User Guide*.
4. Execute the policy report and review the result. The WSDL with the service name attribute `Test` must fail while all others should pass. For details, see the *HPE Systinet User Guide*.

Appendix A: Dialog Box Reference

Each Assertion Editor input dialog is described in the following sections:

Define New Implementation Wizard

The Define New Implementation Wizard enables you to add a new implementation of an assertion either from an existing resource type or by adding a new one.

The wizard contains the following steps.

1. Enter general parameters to define the new implementation.

Parameter	Definition
Define New Type	Select this check-box to manually define the resource type you want to use.
Select an Existing Type	Select a predefined resource type from the drop-down menu.
Filter Text	Use the input to reduce the list of resource types.
Show Common Types only	De-select to show the full list of resource types.

2. If you are defining a new implementation resource type, you must specify the namespace and local name.

Parameter	Definition
Namespace	Namespace of the source type.
Local Name	The local name of the source type.
Load from File	Select to load a source document defining the source type.

3. Define the parameters for a new resource type. Select the implementation type from the available options.

Run Configurations Dialog

Define parameters to test the assertion before publishing.

Parameter	Definition
Name	The name you want to use for the test.
Assertion	Browse for and select the assertion you want to test.
Source	Add or remove a document to test against the assertion.
Parameters	Enter the required parameters for the selected source.

Appendix B: Assertion Document Details

"UDDI BE 01 Assertion XML Document" is the raw XML document of the UDDI BE 01 assertion.

UDDI BE 01 Assertion XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<pm:Assertion xmlns:pm="http://systinet.com/2005/10/soa/policy"
    xmlns:up="http://systinet.com/2005/10/soa/policy/uddi"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <pm:Parameter Name="lang" Type="xs:string" XPointer="xpointer(@RequiredLang)"/>
<!-- template of the instance of the assertion -->
  <pm:Template>
    <up:UDDI_BE_01 RequiredLang="en"/>
  </pm:Template>
  <pm:Validation SourceType="xmlns(ns=urn:uddi-org:api_v2)qname(ns:businessEntity)"
      xmlns:uddi="urn:uddi-org:api_v2"
      xmlns:val="http://systinet.com/2005/10/soa/policy/validation">
<!-- the validation is implemented via xpath expression -->
    <val:XPath>
      count(/uddi:businessEntity/uddi:name[@xml:lang=$lang])&gt;0
    </val:XPath>
  </pm:Validation>
</pm:Assertion>
```

Assertion documents contain the following elements:

- **`pm:Assertion/pm:Template`**: This required element must contain exactly one child element, which is a reference template of how this assertion looks as a WS-Policy document. If there are namespace definitions here, they are included in the reference template. If the assertion has any parameters, you can define default values for them in the reference template. If there are no namespaces or parameters, the reference template can be in the form <name/>.
- **`pm:Assertion/pm:Parameter`**: An assertion in a WS-Policy document may contain parameters including timeouts (in WS-ReliableMessaging), type of authentication, required SOAP header elements, etc. This element gives a definition of such parameters, including the type of the parameter and where the parameter can be found in an instance of the assertion. This information is used by UI console and policy validators.
- **`pm:Assertion/pm:Parameter/@Name`**: The name of the parameter. This name will be shown in the UI.

- **`pm:Assertion/pm:Parameter/@Type`**: Type of the parameter's value.
- **`pm:Assertion/pm:Parameter/@Taxonomy`**: A taxonomy with values that the parameter can adopt. The taxonomy is specified using its tModelKey. This attribute is only required when `Type` has the `pm:taxonomy` value (with pm being the `xmlns:pm="http://systinet.com/2005/10/soa/policy` namespace), otherwise it is ignored (and optional).
- **`pm:Assertion/pm:Parameter/@XPointer`**: In the absence of a `ValueXPointer` attribute, this attribute identifies the place of the parameter in the assertion's template (that is, how the attribute can be obtained from an instance of the assertion). Only a simplified form of the XPointer can be used.

  The evaluation context for the XPointer is the root of the actual assertion. So, for example, `b[1]` is the first "b" child of the assertion's element.

  In this release, an XPath starting with "/" is interpreted to point to the root of the policy document. This behavior will be changed, so do not use absolute XPaths.
- **`pm:Assertion/pm:Parameter/@ValueXPointer`** `ValueXPointer`: Identifies the place of the parameter *relative to* the place identified by the `XPointer` attribute. When the parameter is not set, the element referenced by the `XPointer` attribute is removed from the instance. When the parameter is defined, its value is set to a place identified by the concatenation of the `XPointer` and `ValueXPointer` values. The rationale for this attribute is that there are assertions whose schema requires that either an attribute is set or the attribute's parent element is missing.
- **`pm:Assertion/pm:Parameter/@Optional`**: This attribute tells whether the parameter is optional, that is, if it can be omitted from the assertion instance.
- **`pm:Assertion/pm:Validation`**: The implementation, as described in "Implementations" on page 42.

The key components of the assertion, visible in both the UI and the XML document, are described in the following sections:

- "Reference Templates" below
- "Parameters" on the next page
- "Implementations" on page 42, which includes the validation handler.

Reference Templates

The reference template defines what the assertion looks like instantiated as a WS-Policy document (See the generic `<pm:Template>` element shown in “UDDI BE 01 Assertion XML Document”.). If there

is a namespace to be defined it is included in the reference template. If there are parameters, you can define the default values they point to. If there is no namespace or parameter, the template can be a simple empty tag, like `<assertionName/>`.

The UDDI BE 01 assertion reference template defines the `up` namespace. The assertion has one parameter, `lang`, which points to the `RequiredLang` attribute. The reference template sets the default value of this parameter, `en`. The actual XML of the reference template is:

```
<p:Template>
  <up:UDDI_BE_01 RequiredLang="en"
xmlns:up="http://systinet.com/2005/10/soa/policy/uddi"/>
</p:Template>
```

Reference templates must obey the following rules:

- The template name must be unique.
- The template must be a complete and valid XML element, not a fragment.
- The template can carry a namespace. This is the case with the WS-I BasicProfile assertion reference templates, such as `<wsi:BP1004 xmlns:wsi="http://www.ws-i.org/testing/2004/07/assertions/"/>`

Parameters

Parameters represent requirements whose specific values may vary. They include such things as timeouts, type of authentication, required SOAP header elements, and so on. The value referenced by a parameter can differ between technical policies containing the parameter's parent assertion because each technical policy contains its own instance of the assertion.

Using parameters lets the policy developer reuse assertions. The developer can set a different required value for an assertion in each policy in which the assertion is used. Without parameters, the developer would need a separate assertion for each required value.

“Assertion With Parameter” is an assertion taken from a policy file (namespaces omitted for brevity). Note the attribute `RequiredLang` with the value of "en". This attribute represents the `RequiredLang` parameter. Its default value is "en" for English. This default value is specified in the reference template (see "Reference Templates" on the previous page) but the policy developer can change this value in individual policy files. If the assertion developer does not specify the parameter's default value in the reference template and does not set the parameter as optional, the policy developer must set the parameter value when creating a technical policy with the parameter's parent assertion.

Assertion With Parameter

```
<wsp:Policy xmlns:wsp="..."/>
  <up:UDDI_BE_01 RequiredLang="en" xmlns:up="..."/>
</wsp:Policy>
```

A parameter definition has the following structure:

- `pm:Parameter/@Name`

  Name of the parameter.

- `pm:Parameter/pm:Description`

  Description of the parameter.

- `pm:Parameter/@XPointer`

  Location of the modified attribute (expressed as an XPointer).

- `pm:Parameter/@ValueXPointer`

  Value of the modified attribute (expressed as an XPointer).

  Supported values are most of built-in W3C Schema data types (see http://www.w3.org/TR/xmlschema-2/#built-in-datatypes):

  - xs:string
  - xs:boolean
  - xs:float
  - xs:double
  - xs:duration
  - xs:dateTime
  - xs:time
  - xs:date
  - xs:gYearMonth
  - xs:gYear
  - xs:gMonthDay
  - xs:gDay
  - xs:gMonth
  - xs:hexBinary

- xs:base64Binary
- xs:anyURI
- xs:QName
- xs:integer
- xs:long
- xs:short
- xs:byte
- xs:unsignedLong
- xs:unsignedInt
- xs:unsignedShort
- xs:unsignedByte

Where xs is the `xmlns:xs="http://www.w3.org/2001/XMLSchema"` namespace.

Also supported is the value `pm:taxonomy` (with pm being the `xmlns:pm="http://systinet.com/2005/10/soa/policy` namespace), which specifies that the parameter will take on values from the taxonomy specified by the @Taxonomy attribute.

- `pm:Parameter/@Optional`

  Optional parameter (If the parameter is optional, you need not fill in details).

- `pm:Parameter/@Type`

  Type of the parameter's value.

- `pm:Parameter/@Taxonomy`

  Taxonomy whose values the parameter adopts. Specified with the taxonomy tModelKey. The attribute is required only when Type has the `pm:taxonomy` value, otherwise it is be ignored (and optional). Actual parameter values are specified with keyValues in policy documents.

The following examples demonstrate the use of a taxonomy-based parameter and the corresponding policy document:

Parameter with Taxonomy Type

```
<pm:Parameter Name="artifactType" Optional="true" Type="pm:taxonomy"
    Taxonomy="uddi:systinet.com:soa:model:taxonomies:artifactTypes"
    XPointer="xpointer(@artifactType)"
    xmlns:pm="http://systinet.com/2005/10/soa/policy">
  <pm:Description>Artifact type to restrict applicability.</pm:Description>
</pm:Parameter>
```

Policy Document with a Taxonomy-Based Assertion

```
<wsp:Policy xmlns:ws="...">
  <up:MyAssertion
artifactType="urn:com:systinet:soa:model:artifacts:soa:applicationArtifact"
    xmlns:up="..."/>
</wsp:Policy>
```

The following assertion checks whether communication settings contain a connection timeout set to at least 10 seconds. Additionally, the XML Schema of this assertion specifies that either the "value" must be present, or, to use the default value, the whole `up:ConnectionTimeout` element must be missing.

```
<wsp:Policy xmlns:wsp="..."/>
  <up:Communication xmlns:up="...">
    <up:ConnectionTimeout value="10000"/>
    ...
  </up:Communication>
</wsp:Policy>
```

In this case, a single XPointer referencing the `up:ConnectionTimeout/@value` attribute is not enough, because Policy Manager would not know that the whole element should be removed when the value is not entered. Therefore the parameter is now described in two XPaths:

- Location of the element that should be removed when the value of the parameter is not set
- Location of the value within the element defined above

The location of the element is set in the XPointer and the location of the value within the element is set in a ValueXPointer. For example, “Parameter with ValueXPointer Set at 5000” is a parameter with the ValueXPointer set at 5000. This results in the policy document where “Policy Document with ValueXPointer in Parameter is Set to 5000”. By contrast, if the developer leaves the ValueXPointer blank, the resulting policy document is “Policy Document with Empty ValueXPointer in Parameter”.

Parameter with ValueXPointer Set at 5000

```
<p:Parameter Name="ConnectionTimeout" Optional="false" Type="xsd:integer"
    XPointer="xmlns(up=...)xpointer(up:ConnectionTimeout)"
    ValueXPointer="xpointer(@value)"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <p:Description>Connection timeout in milliseconds.</p:Description>
</p:Parameter>
```

Policy Document with ValueXPointer in Parameter Set to 5000

```
<wsp:Policy xmlns:wsp="..."/>
  <up:Communication xmlns:up="...">
    <up:ConnectionTimeout value="5000"/>
  </up:Communication>
</wsp:Policy>
```

Policy Document with Empty ValueXPointer in Parameter

```
<wsp:Policy xmlns:wsp="..."/>
  <up:Communication xmlns:up="...">
  </up:Communication>
</wsp:Policy>
```

Table “XPointer Combinations and Results” shows the XML representations of various XPointer and ValueXPointer combinations, for optional and required attributes, and whether the value is defined or not. “XPointer” is a correctly defined XPointer.

> **Note:** Only a simplified form of XPointer is recognized in the parameter definition. The rationale is that in this context XPointer is used not only for retrieving data, but also for creating parameters via the UI. This is not possible with general XPointers. The recognized XPointer must have the following structure:
>
> xmlns(prefix1=ns1)*xpointer({/{<prefix>:}?<localname>[<index>]}*)

XPointer Combinations and Results

Optional	Value	XPointer	ValueXPointer	Result in Policy Schema
Yes/No	'ABC'	@P	—	<a P="ABC"/>
Yes	—	@P	—	<a/>
No	—	Prohibited		
Yes	'ABC'	b[1]	@P	<a>
Yes	—	b[1]	@P	<a/> (XPointer is removed.)
Yes	'ABC'	b[1]	—	<a>ABC
Yes	'ABC'	b[1]	c[1]	<a><c>ABC</c>
Yes	—	b[1]	c[1]	<a/> (XPointer is removed.)

XPointer

```
xmlns(soap=http://schemas.xmlsoap.org/soap/envelope/)
xmlns(myns=http://systinet.com/examples/foo)xpointer(soap:Envelope[1]/soap:Body[1]
/myns:Foo)
```

Implementations

An assertion has one implementation for each source type to which the assertion applies. Each implementation is propagated into its own `pm:Validation` element. An implementation contains the

definition of the validation handler, in `p:Validation/##other[1]`, and the type of artifact which the assertion can be used to validate, in `p:Validation/@SourceType`.

Implementations use validation handlers if they do not specify manual validation. Validation handlers are pluggable pieces of code that show Policy Manager how to validate a source document. Validation handlers are usually XPath, JavaScript or XQuery expressions, in which case the source code is included inside the implementation, but they can be custom made. Custom made validation handlers are written in Java and the implementation references the Java class.

Validation handlers and source types are described in the following sections:

- "Source Type" below

  A description of all source types to which an implementation may apply.

- "XPath Assertions" on the next page

  XPath validation handlers.

- "XQuery Assertions" on page 45

  XQuery validation handlers.

- "JavaScript Validator Assertions" on page 47

  JavaScript validation handlers.

Source Type

The `pm:Validation@SourceType` attribute defines the type of artifact validated by the assertion. `SourceType` must be a simplified XPointer identifying the root element of the resource which the assertion validates. If this parameter is omitted, the implementation would apply to sources of any type. However, for performance reasons it is better to map validation to a concrete source type, as narrowly as possible.

`SourceType` can be set as one of the following:

- A general artifact type with the namespace usually defined in the `pm:Validation` element. Please see Table "Source Types Applying to General Resources" for a list of these `SourceType` values and their associated artifacts and namespaces.
- An artifact type. These share the namespace `xmlns:a="http://systinet.com/2005/05/soa/model/artifact"`. A list of these `SourceType` values and their matching artifact types will follow this structure:

`xmlns(ns=http://systinet.com/2005/05/soa/model/artifact)qname(ns:[artifactType])`

For example, xmlns (ns=http://systinet.com/2005/05/soa/model/artifact)qname (ns : applicationComponentArtifact)

Source Types Applying to General Resources

Resource	SourceType value
Any resource	xmlns(ns=http://systinet.com/2005/05/soa/resource)qname(ns:resource)
SOAP message	xmlns(ns=http://schemas.xmlsoap.org/soap/envelope/)qname (ns:Envelope)
UDDI v3 Business Entity	xmlns(ns=urn:uddi-org:api_v3)qname(ns:businessEntity)
HTTP Message Document	xmlns(ns=http://systinet.com/2005/10/soa/policy/report)qname (ns:Message)
WSDL Definition	xmlns(wsdl=http://schemas.xmlsoap.org/wsdl/)wsdl:definitions
XML Schema	xmlns(xsd=http://www.w3.org/2001/XMLSchema)xsd:schema
Conversation Document	xmlns(ns=http://systinet.com/2005/10/soa/policy/report)qname (ns:Conversation)

XPath Assertions

Example “XPath Expression” is an XPath that applies to UDDI business entities and returns every `name` element whose `lang`attribute is set to the same value as the value of the `lang` parameter. If the XPath returns a non-empty list, the source document is considered to be valid against the assertion. If the returned node list is empty, validation has failed.

XPath Expression

```
<val:XPath>
  count(/uddi:businessEntity/uddi:name[@xml:lang=$lang])&gt;0
</val:XPath>
```

You must take the following points into account when writing XPath assertions:

- Namespace

  The element `val:XPath` is the namespace context for the XPath expression. If you need to define a prefix-namespace mapping, do it on this element or its ancestors.

- Type system

  The XPath engine used in this enforcer is the free version of the Saxon-B 8.5.1 XSLT/XPath/XQuery engine. Although this version does not contain XML Schema parsing, it still checks for type conformance. For example, if you need to check that the value of attribute "xyz" is greater than 5, include in your XPath expression:

  ```
  xs:integer(@xyz) > 5
  ```

  If you fail to retype to integer, the XPath expression will never be fulfilled and no warning will be returned.

- Parameter type

  In this release, assertion parameters are always passed as strings, regardless of the schema type written in the parameter definition. For this reason you have to explicitly cast the parameter in numerical comparisons. For example, the following XPath expression would be used in an assertion which checks that the message's body has at most a given number of elements (defined as a parameter named MaxElements):

  ```
  count(soap:Body//*) <=xs:integer($MaxElements)
  ```

XQuery Assertions

XQuery expression can be represented as follows:

XQuery Expression

```
<val:XQuery>
    declare namespace rest="http://systinet.com/2005/05/soa/resource";
    declare namespace a="http://systinet.com/2005/05/soa/model/artifact";
    declare namespace p="http://systinet.com/2005/05/soa/model/property";
    declare namespace val="http://systinet.com/2005/10/soa/policy/validation";
    declare variable $metadata.source.url external;
    if (exists
(rest:resource/rest:descriptor/a:businessServiceArtifact/p:productionStage)) then
        val:assertionOK()
    else
        val:assertionFailed(concat('This service is not assigned a category from a
lifecycle taxonomy. ',
        'To fix this problem, go to <a href="', $metadata.source.url, '&view">the
service</a>, ',
        'click on "Edit" and assign the category.'))
</val:XQuery>
```

The XQuery in “XQuery Expression” comes from the Service Supports Lifecycle assertion. The XQuery applies to business services and checks that each service has a lifecycle stage assigned to it. In the HPE Systinet 2 use of XQueries, the `assertionOK` function is called only one time per tested artifact if the artifact passes validation, whereas if the artifact fails, the `assertionFailed` function is called for each individual violation. For the XQuery in “XQuery Expression” there is no logical need to call `assertionFailed` more than once, since the artifact either has one lifecycle stage or none at all. In “XQuery Reporting Multiple Failures”, the XQuery checks each `include` and `import` element and makes sure they use relative references. The `assertionFailed` function is called for each element that does not use relative references.

XQuery Reporting Multiple Failures

```
declare namespace xs = &amp;quot;http://www.w3.org/2001/XMLSchema&amp;quot;;
    declare namespace
val=&amp;quot;http://systinet.com/2005/10/soa/policy/validation&amp;quot;;
    let $errors :=
    for $el in //xs:*[local-name() = 'include' or local-name() = 'import'] where
($el/@schemaLocation and contains($el/@schemaLocation, ':'))
    return
    val:assertionFailed(concat('This xs:', local-name($el), ' uses absolute
reference to another schema.'), $el)
    return
    if (empty($errors)) then
        val:assertionOK()
    else
        ()
```

> **Note:** Namespaces are not propagated from parent elements but defined via standard XQuery declarations.

Together with the source document, XQuery assertions can be called with additional parameters. For example, these parameters can be used by the assertion to perform additional checks or output the location of the problem back to the user. The parameters are added to the XQuery expression of the assertion. A metadata parameter is shown in “XQuery Expression”.

Parameter name	Description
metadata.source.url	The URL of the source of validation. In the case of HTTP request/response, this points to the request/response message. For one-way messages, WSDL documents etc. it points to the resource being validated.
metadata.description.url	The URL of the associated description document (for example, WSDL associated to a log of messages).
metadata.source.is.subdocument	Detects subdocuments. Returns "false" if document is standalone, "true" if document is part of a larger document.

If you want to write a new XQuery assertion or modify an existing one, follow these guidelines:

- The XQuery engine used in this enforcer is the free version of the Saxon-B 8.5.1 XSLT/XPath/XQuery engine. Although this version does not contain XML Schema parsing, it still checks for type conformance. For example, if you need to check that the value of attribute "xyz" is greater than 5, write:

  ```
  xs:integer(@xyz) > 5
  ```

  Failing to do so, the XQuery expression might never be fulfilled. If this happens, no warning will be returned.

- In this release, assertion parameters are always passed as strings, regardless of the schema type written in the parameter definition. Because of this you must explicitly cast the parameter in numerical comparisons. For example, the following expression would be used in an assertion which checks that the message's body has at most a given number of elements (defined as a parameter named `MaxElements`):

  ```
  count(soap:Body//*) <= xs:integer($MaxElements)
  ```

JavaScript Validator Assertions

JavaScript Validator expression can be represented as follows:

JavaScript Validator Expression

```
/**
    * This function is called iteratively by the java script validator,
    * There are some java script objects pre-initialized in the scripting context
(e.g. repositoryService ...)
    * You can find others and configure more in the managed script environment.js
    * @param  resource
    *         an external resource or an artifact
    * @param  params
    *         a map of params
    */
        function validate(resource, params) {
        return new JsValidationResult(JsValidationCode.OK);
        }
```

To write a JavaScript Validator assertion, see HPE Systinet Java Client API Java documentation.

Appendix C: Integrating XQuery Function Libraries

You can integrate a user-defined XQuery Library into Assertion Editor.

To Integrate an XQuery Function Library:

1. Your JAR file must have the following structure:
   - <*your*>-lib.jar
     - META-INF
       - <*your*>-XQueryContext.xml
     - com
       - <*your*>-class

   Where <*your*>, is the name you give to the XQuery function library.

   `<your>-XQueryContext.xml` should match the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:util="http://www.springframework.org/schema/util"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop.xsd
    http://www.springframework.org/schema/jee
    http://www.springframework.org/schema/jee/spring-jee.xsd
    http://www.springframework.org/schema/lang
    http://www.springframework.org/schema/lang/spring-lang.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx.xsd
    http://www.springframework.org/schema/util
    http://www.springframework.org/schema/util/spring-util.xsd
    http://www.springframework.org/schema/context
```

```
    http://www.springframework.org/schema/context/spring-context.xsd">
    <context:annotation-config/>
    <!--STM Lifecycle Policies: XQuery extension for Policy Manager,
        depends on lifecycle API-->
    <bean id="your-bean-id" class="your-class-XQueryExtension"
scope="singleton"/>
</beans>
```

2. Open `AE_LIB/META-INF/eclipseBeanRefContext.xml` in a text editor.

> **Note:** `AE_LIB` refers to `WB_HOME/plugins/com.em.tools.assertioneditor.lib_1.0.n.xxx` for standalone version or `WB_HOME/dropins/sw/eclipse/plugins/com.hp.em.tools.ae.lib_1.0.n.xxx` for the plugin version.

3. Add your mapping file to the `constructor-arg` element:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="com.hp.soa.em"
      class="com.hp.em.spring.ClassPathXmlApplicationContext">
    <constructor-arg>
      <list>
        <value>classpath*:META-INF/${mapping_file.xml}</value>
        <value>classpath*:META-INF/pmContext.xml</value>
        <value>classpath*:stmContext.xml</value>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

4. Add the XQuery library to the `AE_LIB/lib/` folder.
5. Modify `AE_LIB/META-INF/MANIFEST.MF` to set the Eclipse classpath. Add the XQuery library to the `Bundle-ClassPath` item. For example, `lib/lifecycle-xquery.jar`.
6. Restart HPE Systinet Workbench with command, **start.exe -clean**.