HP SOA Systinet Workbench

Software Version: 4.00

Assertion Editor Guide

Legal Notices

Warranty

The only warranties for HP 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. HP 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 HP 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 Notices

Contents

About this Guide

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

This guide contains the following chapters:

- Chapter 1, Assertion Editor

  Provides an overview of the main features of Assertion Editor.

- Chapter 2, Getting Started

  Describes the installation of the main features, and shows you how to create an assertion project in Assertion Editor.

- Chapter 3, Manipulating Assertions

  Explains how to create, download, edit, and compare assertions using Assertion Editor.

- Chapter 4, Validating and Publishing Assertions

  Shows how to test, publish, and resolve conflicts in assertions using Assertion Editor.

- Chapter 5, Deploying Assertions

  Shows how to build an Assertion extension project using Assertion Editor.

- Chapter 6, Customizing Assertions

  Explains how to customize the source type and add PM extensions in Assertion Editor.

- Chapter 7, Java Assertion Demo

  Demonstrates the creation of a custom assertion validator and its use with Assertion Editor and HP SOA Systinet.

- Appendix A, Dialog Boxes

  Dialog boxes reference.

- Appendix B, Assertion Developer Reference

  Assertion schema reference.

- Appendix C, Eclipse Plug-in Requirements

  Required plug-ins when installing Workbench as an update.

- Appendix D, Integrating XQuery Function Libraries

  Integrating custom XQuery libraries with Assertion Editor.

Document Conventions

This document uses the following typographical conventions:

run.bat make	Script name or other executable command plus mandatory arguments.
[--help]	Command-line option.
either | or	Choice of arguments.
replace_value	Command-line argument that should be replaced with an actual value.
{arg1 | arg2}	Choice between two command-line arguments where one or the other is mandatory.
`java -jar hpsystinet.jar`	User input.
`C:\System.ini`	File names, directory names, paths, and package names.
`a.append(b);`	Program source code.
`server.Version`	Inline Java class name.
`getVersion()`	Inline Java method name.
Shift+**N**	Combination of keystrokes.
Service View	Label, word, or phrase in a GUI window, often clickable.
OK	Button in a user interface.
New>Service	Menu option.

Documentation Updates

This guide's title page contains the following identifying information:

- Software version number, which indicates the software version

- Document release date, which changes each time the document is updated

- Software release date, which indicates the release date of this version of the software

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport logon page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. For details, contact your HP sales representative.

Support

You can visit the HP Software Support Web site at:

http://www.hp.com/go/hpsoftwaresupport

HP Software Support Online provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the HP Software Support web site to:

• Search for knowledge documents of interest

• Submit and track support cases and enhancement requests

• Download software patches

• Manage support contracts

• Look up HP support contacts

• Review information about available services

• Enter into discussions with other software customers

• Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

1 Assertion Editor

HP SOA Systinet Workbench includes Assertion Editor, a set of features for use with the Policy Manager component of HP SOA 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:

- Workbench Suite on page 11

- Overview on page 12

- User Interface on page 13

Workbench Suite

HP SOA Systinet Workbench is a suite of editor tools enabling you to customize your deployment of HP SOA Systinet 4.00.

Workbench consists of the following editor tools, distributed as a single Eclipse development platform:

- **Customization Editor**

  Customizes the underlying SOA Definition Model (SDM) within HP SOA Systinet.

- **Taxonomy Editor**

  Customizes the taxonomies used to categorize artifacts in HP SOA Systinet.

- **Assertion Editor**

  Customizes the conditions applied by your business policies within HP SOA Systinet.

- **Report Editor**

Customizes report definitions for use with HP SOA Systinet.

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 fulfils a management requirement.

HP SOA Systinet provides tools for testing 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 on page 26

   • Downloading and Importing Assertions on page 27

2 **Create and manage assertions.**

   For details, see the following sections:

   • Creating Assertions on page 31

   • Editing Assertions on page 32

   • Deleting Assertions on page 39

   • Comparing Assertion Versions on page 40

3 **Validate assertions before publishing**.

   For details, see Testing Assertions on page 43.

4 **Deploy assertions and manage conflicts**.

   For details, see the following sections:

   • Publishing Assertions on page 45

   • Resolving Conflicts on page 44

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

   For details, see Chapter 6, Customizing Assertions.

User Interface

The default perspective is split into a number of sections with menu options across the top, as shown in Figure 1, "Assertion Editor UI".

Figure 1. Assertion Editor UI
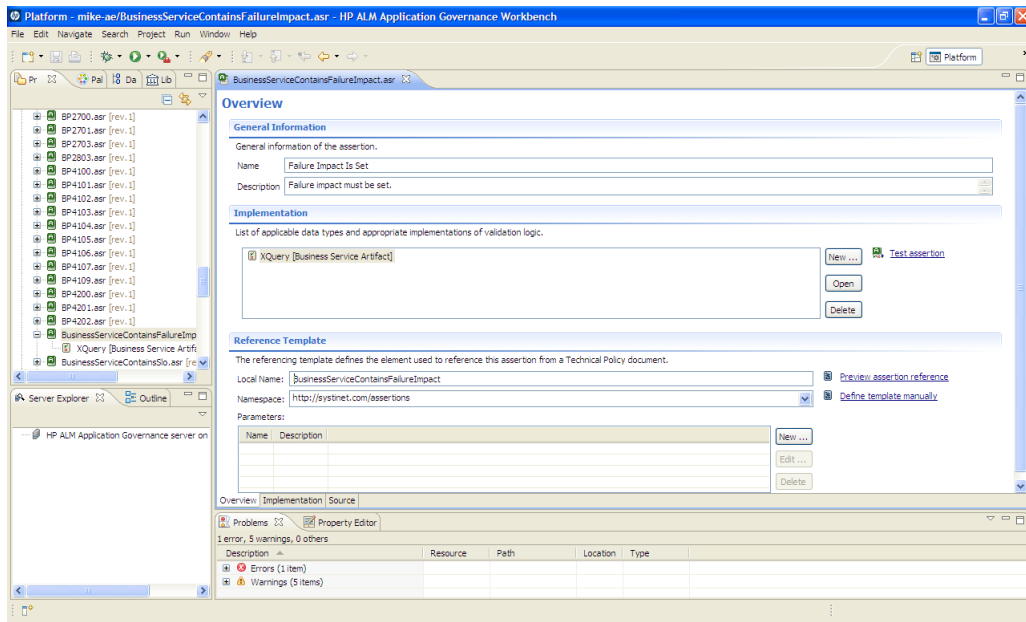
The platform perspective consists of the following views:

- **Project Explorer**

  The tree view of your assertion projects. For details, see Project Explorer on page 14.

- **Server Explorer**

  The view listing HP SOA Systinet server connections to Workbench. For details, see Server Explorer on page 16.

- **Editor**

  The view showing the components of the assertion. For details, see Editor Pane on page 16.

Project Explorer

Project Explorer contains a hierarchical list of projects, the assertions in each project, and the validation definitions in each assertion, as shown in Figure 2, "Project Explorer".

Figure 2. Project Explorer

The Project Explorer contains additional context menu options enabling you to interact with a running HP SOA Systinet server. Right-click the project name or a particular assertion, and select **HP SOA Systinet** to view the options listed in Table 1, "Project Context Menu Options" and Table 2, "Assertion Context Menu Options".

Table 1. Project Context Menu Options

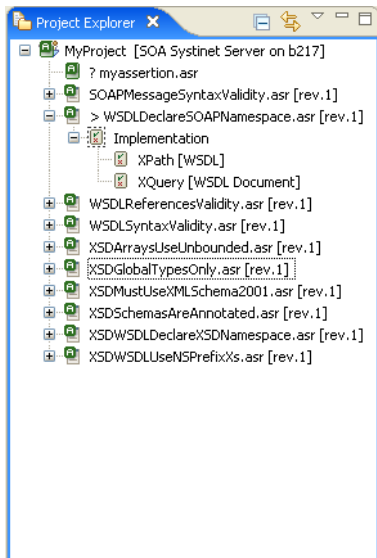| Option	Function
Download Assertions	Import Assertions from HP SOA Systinet. For details, see Downloading and Importing Assertions on page 27.
Upload to Server	Export assertions to the default HP SOA Systinet server. For details, see Publishing Assertions on page 45.
Update from Server	Update assertions from HP SOA Systinet. For details, see Editing Assertions on page 32.
Remove from Server	Delete assertions from HP SOA Systinet. For details, see Deleting Assertions on page 39.
Upload To Other Server	Export assertions to a specified HP SOA Systinet server.
Build Extension	Create an assertion extension for HP SOA Systinet containing all the assertions in your project. For details, see Building an Assertion Extension on page 47.

Table 2. Assertion Context Menu Options

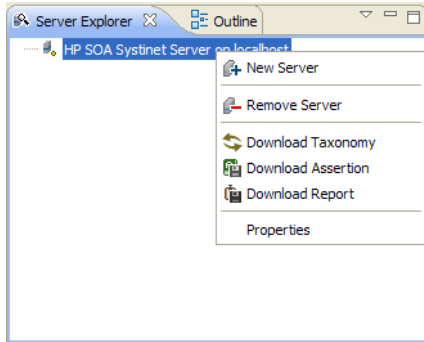| Option	Function
Upload to Server	Export an assertion to the default HP SOA Systinet server. For details, see Publishing Assertions on page 45.
Update from Server	Update assertions from HP SOA Systinet. For details, see Editing Assertions on page 32.
Remove from Server	Delete the assertion from HP SOA Systinet. For details, see Deleting Assertions on page 39.
Upload To Other Server	Export an assertion to a specified HP SOA Systinet server.
Build Extension	Create an assertion extension for HP SOA Systinet containing all the assertions in your project. For details, see Building an Assertion Extension on page 47.

Server Explorer

The Server Explorer displays the HP SOA Systinet servers connected to Workbench, as shown in Figure 3, "Server Explorer View". The functionality is shared by all the Workbench editors.

Figure 3. Server Explorer View

Right-click a server in the Server Explorer to open the context menu described in Table 3, "Server Explorer Context Menu Options".

Table 3. 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 platform 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 Pane

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

The pane is split into tabs, described in the following sections:

- Overview Tab on page 17

- Implementation Tab on page 18

- Source Tab on page 19

Overview Tab

The Overview tab shows the components of the assertion, as shown in Figure 4, "Overview Tab".

Figure 4. Overview Tab
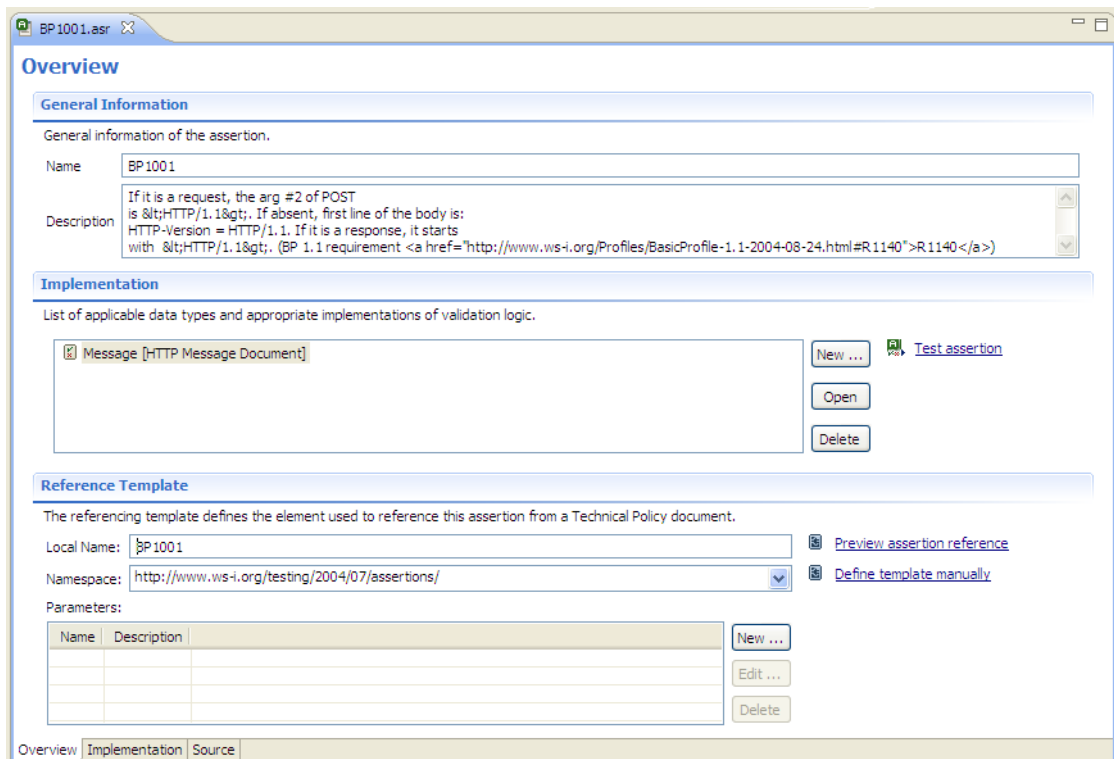
The tab is divided into the following areas:

- **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, as shown in Figure 5, "Assertion Editor UI: Editor Implementation Tab".

Figure 5. Assertion Editor UI: Editor Implementation Tab
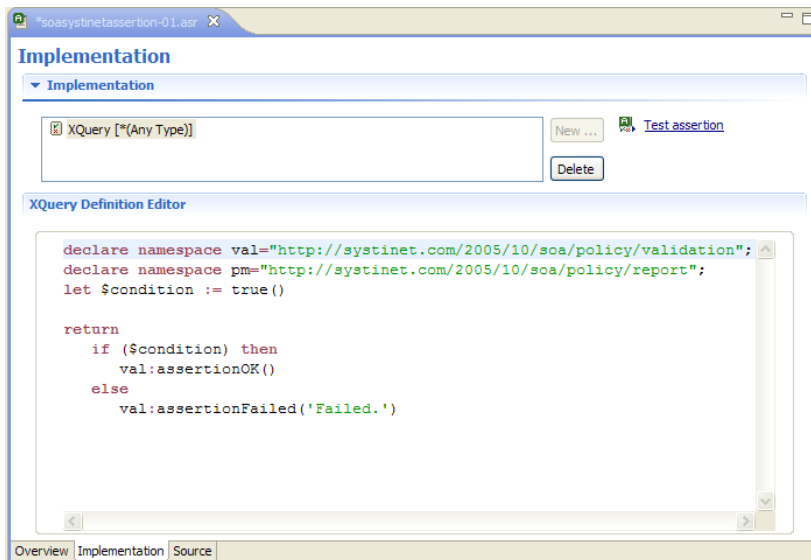
Highlighting an implementation opens the XQuery Definition Editor in the window beneath. For details, see Writing XQuery Definitions on page 35.

Source Tab

The Source tab is an XML editor for editing the assertion, as shown in Figure 6, "Source Tab".

Figure 6. Source Tab
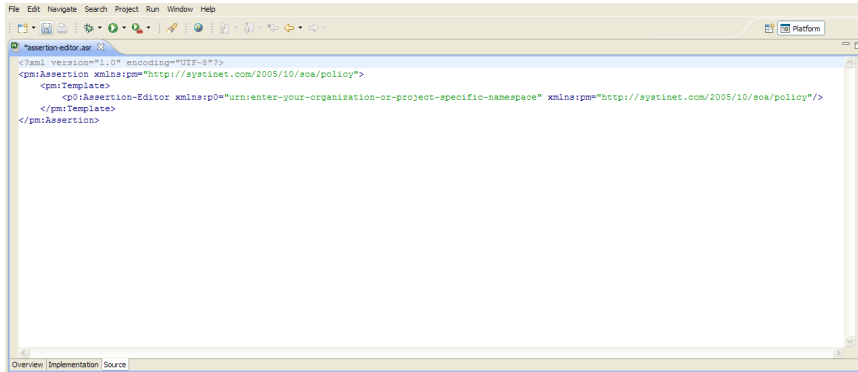
2 Getting Started

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

- Installing Workbench on page 21

- SSL Configuration on page 25

- Creating an Assertion Project on page 26

- Downloading and Importing Assertions on page 27

Installing Workbench

HP SOA Systinet Workbench is an Eclipse development platform distributed as a zip file, `hp-soa-systinet-workbench-4.00-win32.zip` or as a plugin for an existing Eclipse environment, `hp-soa-systinet-workbench-4.00-plugin.zip`.

> For supported platforms and known issues, see `readme.txt` alongside the archive.

To install HP SOA Systinet Workbench as a new Eclipse platform:

- Extract the archive to your required location, referred to in this document as `WB_HOME`.

> The path must not be longer than 97 characters.

To install HP SOA Systinet Workbench to an existing Eclipse platform:

1 Ensure that your Eclipse platform contains the necessary plug-ins. For details, see Appendix C, Eclipse Plug-in Requirements.

2   In your current Eclipse SDK (3.3 or later), use the software updates feature to install HP SOA Systinet Workbench.

Select **Help>Software Updates>Find and Install...**.

The Install/Update dialog opens.

3   In the Install/Update dialog, select **Search for new features to install**, and click **Next**.

The Install – Update Sites to Visit dialog opens.

4   In the Update Sites to Visit dialog, click **New Archived Site**.

The Select Local Archive Site dialog opens.

5   Locate and select `hp-soa-systinet-workbench-4.00-plugin.zip`, and then click **Open**.

The Edit Local Site dialog opens.

6   In the Edit Local Site dialog, if required, rename the local archive name, and click **OK**.

7   In the Install – Update Sites to Visit dialog, select the new local archive, and then click **Finish**.

The Updates – Search Results dialog opens.

8   Select the modules from the archive that you want to install:

- **Workbench Extra 4.00**

  HP SOA Systinet Workbench splash screen.

- **Taxonomy Editor 4.00**

     Required for Customization Editor.

- **Customization Editor 4.00**

- **Assertion Editor 4.00**

- **Report Editor 4.00**

- **Common Plugin 4.00**

  Shared components used by the editors.

  Click **Next**.

  The Install – Feature License dialog opens.

9   In the Feature License dialog, select **I accept the terms in the license agreements**, and click **Next**.

  The Install – Installation dialog opens.

10   In the Installation dialog, if required, change the installation location, and then click **Finish**.

11   If you install `Workbench Extra 4.00` make the following configuration changes:

- Remove `-showsplash org.eclipse.platform` from `ECLIPSE_HOME/eclipse.ini`.

- Edit ECLIPSE_HOME/configuration/config.ini and make the following changes:

  - Set `osgi.splashPath=platform:/base/plugins/com.systinet.tools.workbench` .

  - Set `eclipse.product=com.systinet.tools.workbench.ide`

To start HP SOA Systinet Workbench:

•   Execute **WB_HOME/workbench/start.exe**.

The first time you start Workbench, the welcome screen opens, as shown in Figure 7, "Workbench Welcome Screen".

Figure 7. Workbench Welcome Screen
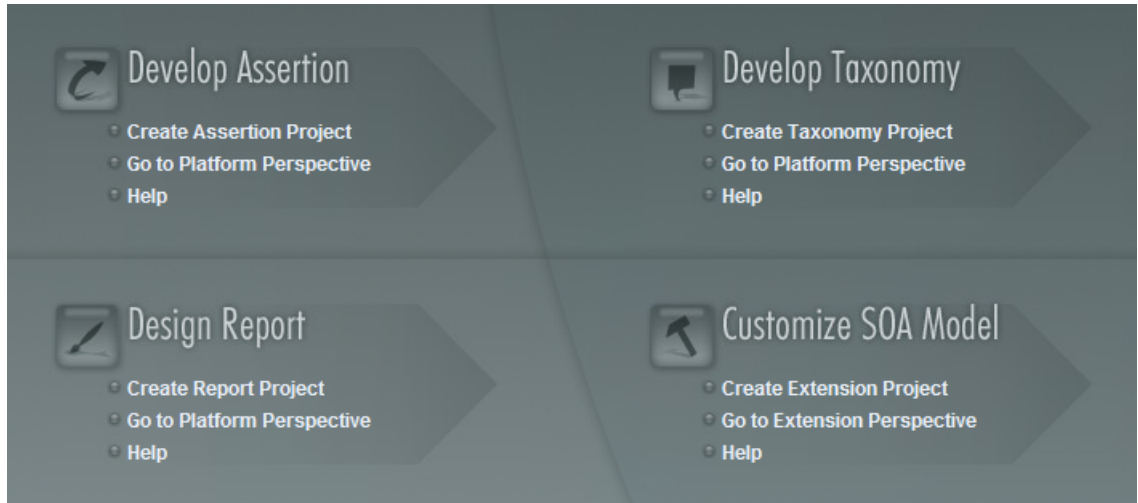
Select one of the options to open one of the editor tools, start a new editing project, or view the documentation set.

You can return to the welcome screen from any of the editor tools by selecting **Help>Welcome** from the menu options.

By default, Workbench runs in 'normal' mode which prevents users from uploading system taxonomies (IDs start with *uddi:systinet.com:soa:model:taxonomies*) and the Report Editor .rptlibrary file to HP SOA Systinet servers. If you need to work with system taxonomies or want to upload the .rptlibrary file you can switch Workbench into 'admin' mode.

> ⚠️ Be extremely careful when working with system taxonomies, HP SOA Systinet uses some hard-coded values from system taxonomies, changing or removing them may cause errors.

To Switch Workbench to Admin Mode

1   Open `WB_HOME/configuration/config.ini` with a text editor.

2   Add *mode=admin* to `config.ini`.

3   Restart Workbench.

> HP SOA Systinet Workbench requires Java SE Development Kit (JDK) 1.5.0 or higher.
> You must include the path to this version of the JDK in the *JAVA_HOME* environment
> variable.

> HP SOA Systinet Workbench is memory-intensive. If you experience performance issues,
> HP recommends increasing the memory allocation.

To increase the memory allocation for HP SOA Systinet Workbench:

1   Open `WB_HOME/workbench/start.ini` for editing.

2   Set these new values:

- `-Xms128m`

- `-Xmx1024m`

3   Save your changes.

4   Restart Workbench.

SSL Configuration

By default, Workbench trusts all HP SOA Systinet server certificates. You may want Workbench to
verify HP SOA Systinet certificates.

To verify HP SOA Systinet server certificates:

- Add the following options to `WB_HOME/workbench/start.ini`:

```
-Dcom.hp.systinet.security.ssl.verifyCert=true
-Djavax.net.ssl.trustStore=USER_TRUSTSTORE
-Djavax.net.ssl.trustStorePassword=TRUSTSTORE_PASS
-Djavax.net.ssl.trustStoreType=TRUSTSTORE_FORMAT
```

If HP SOA Systinet is configured for 2-way SSL, you must provide Workbench certificates to HP SOA
Systinet.

To provide Workbench client certificates to HP SOA Systinet:

• Add the following options to `WB_HOME/workbench/start.ini`:

```
-Djavax.net.ssl.keyStore=USER_KEYSTORE
-Djavax.net.ssl.keyStorePassword=KEYSTORE_PASS
-Djavax.net.ssl.keyStoreType=KEYSTORE_FORMAT
```

Creating an Assertion Project

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:

   • In the Workbench Welcome page, click **Create Assertion Project**.

   • Click **New** to open the Select a Wizard window, and select **HP SOA Systinet>Assertion Project**.

   • From the menu, select **File>New>Assertion Project**.

   • Press **Alt**+**Shift**+**N**, and then press **R**, to open the Select a Wizard window. Then select **HP SOA Systinet>Assertion Project**.

   The New Assertion Project dialog box opens.

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 Extension	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 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.

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

4   Do one of the following:

   • Select **Create a New Server**, and click **Next**.

     Continue to Step 5.

   • Select **Use an Existing Server**, select the server from the list and input its credentials, and then click **Next**.

     Continue 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 HP SOA 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 a Policy Manager 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 on page 26.

• From your local file system, at a later date.

> ⚠ 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 opens.

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

   The Choose Location dialog box opens.

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

To import assertions from a local file:

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

   The Import Assertion dialog box opens.

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

   The Choose Location dialog box opens.

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

The assertions are imported to your project.

3 Manipulating Assertions

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

Creating Assertions

In Creating an Assertion Project on page 26, 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:

   - Click **New** ⬚ to open the New: Select a Wizard dialog, and expand **HP SOA Systinet>Assertion**, and then, click **Next**.

   - Select **File>New>Assertion**.

   - Press **Alt**+**Shift**+**N** to open the context menu, and select **Assertion**.

     The New Assertion wizard opens.

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:

- Add an implementation, as described in Adding and Deleting Implementations on page 33.

- Test the assertion, as described in Testing Assertions on page 43.

- Publish the assertion, as described in Publishing Assertions on page 45.

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.

> 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 44.

To update an assertion from the server:

1   Right-click the assertion in Project Explorer to open its context menu.

2   Select **HP SOA Systinet>Update from Server**.

The main functionality of Assertion Editor is described in the following sections:

- Editing General Properties on page 32

- Adding and Deleting Implementations on page 33

- Writing XPath Definitions on page 34

- Writing XQuery Definitions on page 35

- Editing XQuery Definitions on page 36

- Editing Reference Templates on page 38

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. For details, see Overview Tab on page 17.

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 opens. For details, see Define New Implementation Wizard on page 62.

2   Do one of the following:

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

Skip to Step 4.

- 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 or XPath definitions to meet your needs. For instructions, see Writing XPath Definitions on page 34 or Writing XQuery Definitions on page 35.

Writing XPath Definitions

After creating an implementation that uses an XPath validation handler, as described in Adding and Deleting Implementations on page 33, 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:

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

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

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

   The XML document appears 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 appears in the XPath Definition Editor field.

> 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 ⊗ appears.

   • If you receive a warning, go to Step 5.

   • If you do not receive a warning, go to Step 8.

5   Click the unresolved prefix link.

     The Manage prefix and namespace pane opens.

6   Define the namespace of the prefix, as follows:

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

     • 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

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 and click **New** in the Implementation pane.

     The Define New Implementation dialog box opens, as shown in

2   To use a predefined source type:

     • In the Predefined field, select the source type you need from the drop-down list.

     • In the Dialect field, select **XQuery** from the drop-down list, and click **OK**.

3   To manually define a source type:

     • Select the Manual Define check-box.

     • Enter the required parameters.

- In the Dialect field, select **XQuery** from the drop-down list, and click **OK**.

The XQuery Definition opens in the Editor view.

4   Edit the XQuery Definition, as described in Editing XQuery Definitions on page 36, and click **Test Assertion**.

If the assertion passes validation, you can now publish the assertion. For details, see Publishing Assertions on page 45.

If the assertion does not pass validation, you can resolve any problems. For details, see Resolving Conflicts on page 44.

For more information, see XQuery Assertions on page 78.

Editing XQuery Definitions

Assertion Editor also supports external XML editors.

To use an external XQuery editor with Assertion Editor, you must first add the Saxon extension to the external editor:

- **Folder**

  ```
  WORKBENCH_HOME/plugins/com.systinet.tools.assertioneditor.lib_version-number/lib/
  saxon-extensions/
  ```

- **Extension**

  ```
  pm-extension-functions.jar
  ```

To edit an XQuery Definition:

1   In Project Explorer, right-click the XQuery to open its context menu, select **Open With**, and then select from the following options:

- **Text Editor**

  To edit the XQuery with a plain text editor.

- **System Editor**

  To edit the XQuery with an editor currently used by your system.

- **In-place Editor**

  To edit the XQuery with an OLE editor.

- **Default Editor**

  To edit the XQuery with the default editor provided with Assertion Editor.

- **Other**

  To edit the XQuery with an editor not previously defined.

2   Edit the XQuery as required and save your changes.

Instructions on how to add the Saxon extension to the most popular XML editors are given in the following sections:

- Editing XQueries in oXygen on page 37
- Editing XQueries in Stylus Studio on page 38

Editing XQueries in oXygen

To set up oXygen™ to edit XQueries:

1   Open or create the XQuery file in oXygen.

2   Click **Configure Transformation Scenario** to open the Configure Transformation Scenario wizard.

3   Select **Execute XQuery**, and click **New** to open the **Edit Scenario** pane.

4   In the **Transformer** field, select **Saxon 8B**.

5   Click **Extensions** to open the Extensions dialog box, and click **Add** to open the **Add Extension** dialog box.

6   Type in or browse for the path to `pm-extension-functions.jar`.

7   Click **OK** in all wizard panes to save the transformation scenario.

When you open any other XQuery files, you must always choose this transformation scenario and then edit the XQuery file to force oXygen to rebuild it.

This procedure was created for oXygen 8.1. Other versions can be used but some details may differ.

Editing XQueries in Stylus Studio

To set up Stylus Studio™ to edit XQueries:

1   Select **Tools>Options** to open the **Options** dialogue.

2   Expand **Module Setting+XQuery>Processor Settings** from the tree menu.

3   In the **Processor** drop-down list, select **Saxon 9.0.0.2**, and then, click the **Use as default processor** checkbox.

4   Click **OK**.

5   Select **Project>Set Classpath** and add the path to `pm-extension-functions.jar`.

To open an XQuery in Stylus Studio from Assertion Editor:

1   In the **Project Explorer** of the Assertion Editor UI, right-click the XQuery.

2   Select **Open With>System Editor**.

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:

    • To add a parameter, click **New**.

    • To edit an existing assertion, highlight it, and then click **Edit**.

    The Define Parameter wizard opens.

4   Input a name, description, select if the parameter is optional or required.

5   Do one of the following:

    • Select **Primitive** type and the parameter type from the drop-down list.

    • Select **Taxonomy** and input or **Browse** for the relevant taxonomy.

> The available taxonomies depend on those imported from HP SOA Systinet when you created the project.
>
> **To Update Taxonomies**
>
> 1   Open the context menu for the assertion project and select **Properties** to open the Project Properties dialog box.
>
> 2   Expand **HP SOA Systinet>HP SOA Systinet** and select **Taxonomies** to view the list of downloaded taxonomies from the server.
>
> 3   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:

Deleting Local Assertions

Deleting a local copy of an assertion does not affect the version on the server.

To delete a local copy of an assertion:

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

Deleting Assertions on the Server

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

To delete an assertion on a server:

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

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 **HP SOA Systinet>Remove from Server.**

2   When prompted, select one of the following:

• Also delete resources from local file system.

• Do not delete resources on 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 opens.

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.

4 Validating and Publishing Assertions

This chapter explains how to test assertions and deal with validation conflicts before publishing or exporting them, as detailed 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 dialog box opens. For details, see Run on page 65.

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 opens

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   Do one of the following:

  - Click **Add File** to browse Assertion Editor projects.

  - Click **Add External Files** to browse the local file system.

  - Click **Add URL** and type in the URL of a source file.

2   Enter the required parameter values in the Parameters table, and then do one of the following:

  - Click **Apply**.

  - Click **Revert** to use the most recent parameter value.

    For information about assertion reference templates, see Editing Reference Templates on page 38.

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 **HP SOA Systinet>Update Assertion**.

A conflict warning appears.

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 HP SOA Systinet server.

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:

•   Right-click the assertion in Project Explorer to open its context menu, and select **HP SOA 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 **HP SOA Systinet>Upload to Other Server**.

The New Server wizard opens.

2   Follow the steps for adding a server, as described in Creating an Assertion Project on page 26.

If changes were made to the version on the server since you last synchronized, a conflict warning appears that asks whether you want to force publication. For details on conflict resolution, see Resolving Conflicts on page 44.

5 Deploying Assertions

This chapter explains how to test assertions and deal with validation conflicts before publishing or exporting them, as detailed in the following sections:

- Building an Assertion Extension on page 47

- Applying Extensions on page 47

- Redeploying the EAR File on page 52

Building an Assertion Extension

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

> 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 **HP SOA 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.

Applying Extensions

You can extend HP SOA Systinet by adding libraries or JSPs to the deployed EAR files, by modifying the data model, by configuring the appearance of the UI, and by importing prepackaged data.

Extensions to HP SOA Systinet come from the following sources:

- **Customization Editor**

  Typical extensions created by Customization Editor contain modifications to the data model and artifact appearance, and possibly data required by the customization (taxonomies). They may also contain new web components, which may include custom JSP and Java code.

  > ⚠ If your extension contains new artifact types, HP SOA Systinet does not create default ACLs for them. Set default ACLs for the new artifact types in HP SOA Systinet using the functionality described in "How to Manage Default Access Rights" in the *Administration Guide*.

- **Assertion Editor, Report Editor, and Taxonomy Editor**

  These extensions contain assertion, reporting, and taxonomy data only. They do not involve changes to the data model.

The Setup Tool opens the EAR files, applies the extensions, and then repacks the EAR files.

Apply extensions according to one of the following scenarios:

- Single-Step Scenario on page 49

  The Setup Tool performs all the processes involved in applying extensions, including any database alterations, as a single step.

- Decoupled DB Scenario on page 51

  Database SQL scripts are run manually. The Setup Tool performs the other processes as individual steps that are executable on demand. This scenario is useful in organizations where the user applying extensions does not have the right to alter the database, which is done by a database administrator.

  > ⚠ In some specific circumstances (underscores and numbers in property names), extension application may fail because HP SOA Systinet cannot create short enough database table names (31 character maximum for most databases).
  >
  > The error in `setup.log` resembles the following:
  > `[java] --- Nested Exception ---`

```
[java] java.lang.RuntimeException: cannot reduce length of identifier
  'ry_c_es_Artifact02s_c_priEspPty01Group_c_priEspPty01',
  rename identifier elements or improve the squeezing algorithm
[java] at com.systinet.platform.rdbms.design.decomposition.naming.impl.

 BlizzardNameProviderImpl.getUniqueLimitedLengthName(BlizzardNameProviderImpl.java:432
[java] at com.systinet.platform.rdbms.design.decomposition.naming.impl.
  BlizzardNameProviderImpl.filterTableName(BlizzardNameProviderImpl.java:374)
```

This is due to HP SOA Systinet using an older table naming algorithm in order to preserve backward compatibility with HP SOA Systinet 3.00 and older versions.

If you do not require backwards compatibility with these older versions, you can change the table naming algorithm.

To change the table naming algorithm:

1   Open `SOA_HOME/lib/pl-repository-old.jar#META-INF/rdbPlatformContext.xml` with a text editor.

2   In the `rdb-nameProvider` bean element, edit the following property element:

   <property name="platform250Compatible" value="false"/>

3   Save `rdbPlatformContext.xml`

This solution only impacts properties with multiple cardinality. If the problem persists or you need to preserve backwards compatibility, then review the property naming conventions in your extension.

Single-Step Scenario

Follow this scenario if you have permission to alter the database used for HP SOA Systinet.

To apply extensions to HP SOA Systinet in a single step:

1   Make sure that all extensions are in the following directory:

   `SOA_HOME/extensions`

   The Setup Tool automatically applies all extensions in that directory.

> If you are applying extensions to another server, substitute the relevant home directory for SOA_HOME.

2   Stop the server.

3   Start the Setup Tool by executing the following command:

    SOA_HOME/bin/setup.bat(sh)

4   Select the **Apply Extensions** scenario, and click **Next**.

    The Setup Tool automatically validates the step by connecting to the server, copying the extensions, and merging the SDM configuration.

5   Click **Next** for each of the validation steps and the setup execution.

> This process takes some time.

6   Click **Finish** to end the process.

7   Deploy the EAR file:

    • **JBoss**

    The Setup Tool deploys the EAR file automatically.

    If you need to deploy the EAR file to JBoss manually, see Redeploying the EAR File on page 52.

    • **Other Application Servers**

    You must deploy the EAR file manually.

    For application server-specific details, see "Deploying the EAR File" in the *HP SOA Systinet Installation and Deployment Guide*.

8   Restart the server.

> ⚠️ Applying an extension that modifies the SDM model may drop your full text indices.
>
> `SOA_HOME/log/setup.log` contains the following line in these cases:
>
> ```
> Could not apply alteration scripts, application will continue with slower DB
> drop/create/restore scenario. ... .
> ```
>
> In these cases, reapply full text indices as described in the "Enabling Full Text Search"
> section of the *HP SOA Systinet Installation and Deployment Guide*.

Decoupled DB Scenario

Follow this scenario if the user who applies extensions does not have permission to modify the database.

To apply extensions and modify the database separately:

1   Make sure that all extensions are in the following directory:

     `SOA_HOME/extensions`

     The Setup Tool automatically applies all extensions in that directory.

2   Stop the server.

3   Start the Setup Tool by executing the following command:

     SOA_HOME/bin/setup -a.

4   Select the **Apply Extensions** scenario, and click **Next**.

5   Click **Next**, to execute the extension application, and exit the Setup Tool.

6   Provide the scripts from `SOA_HOME/sql` to the database administrator.

     The database administrator can use `all.sql` to execute the scripts that drop and recreate the
     database schema.

7   Execute the Setup Tool in command-line mode to finish the extension application:

SOA_HOME/bin/setup -c

8   Redeploy the EAR file:

  • **JBoss**

     The Setup Tool deploys the EAR file automatically.

     If you need to deploy the EAR file to JBoss manually, see Redeploying the EAR File on page 52.

  • **Other Application Servers**

     You must deploy the EAR file manually.

     For application server-specific details, see "Deploying the EAR File" in the *HP SOA Systinet Installation and Deployment Guide*.

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.

> For other application servers, follow the EAR deployment procedures described in the "Deploying the EAR File" in the *HP SOA Systinet Installation and Deployment Guide*.

To redeploy the EAR file to JBoss:

1   Stop the application server.

2   Start the Setup Tool by executing the following command:

     SOA_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.

6 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 opens.

2   Expand **HP SOA SystinetAssertion Editor**, and select **Source Type**.

    A table opens displaying source type names, local names, and namespaces.

3   Do one of the following:

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

    - 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**.

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

Adding PM Extensions

You can extend Policy Manager 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 HP SOA Systinet wizard opens.

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

3   Do one of the following:

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

   • 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 Policy Manager servers with the Setup Tool. For information about the Setup Tool, see the *HP SOA Systinet Administrator Guide*.

7 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 Policy Manager as an extension.

- Create an assertion in Assertion Editor (a feature of HP SOA Systinet Workbench) based on our validator.

- Publish the assertion to the Platform repository.

You can find the demo sources in `SOA_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 `SOA_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.systinet.policy.validation.AssertionValidator`:

- `QName getDialect()` — Must return a QName which will be used in assertions to invoke this validator. Our 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.

> For more information about Policy Manager's interfaces see the 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
<?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 `SOA_HOME\demos\policymgr\assertionvalidator`.

2   To get help run `run.bat` or `run.sh`.

3   Build the extension with the command **run.bat|.sh make**.

4   Check the created extension in `SOA_HOME\demos\policymgr\assertionvalidator\validator\dist\mycompany.ext.demo.jar`.

> If you want use an Eclipse project for this demo you must define the `SOA_CLIENT_LIB` classpath variable in the Eclipse IDE. The `SOA_CLIENT_LIB` variable must point to `SOA_HOME/client/lib`.

Applying the Extension

After you create `mycompany.ext.demo.jar`, apply it to HP SOA Systinet as an extension.

To apply the extension to HP SOA Systinet:

1   Make sure the HP SOA Systinet server is not running.

2   Copy `SOA_HOME\demos\policymgr\assertionvalidator\validator\dist\mycompany.ext.demo.jar` into `SOA_HOME\extensions`.

3   Execute `SOA_HOME\bin\setup` and select the **Apply Extensions** scenario.

4   Start HP SOA Systinet.

For details, see "Applying Extensions" in the *HP SOA Systinet Administrator Guide*.

Creating and Deploying the Assertion

There is a sample project for Assertion Editor in `SOA_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 57 (linked through `SOA_HOME\demos\policymgr\assertionvalidator\validator\dist \mycompany.ext.demo.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 an assertion to HP SOA Systinet:

1   Open Assertion Editor.

2   Open the 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 HP SOA Systinet server.

4   In the Project Explorer, open the Project context menu and select **Properties>HP SOA Systinet Server**, and select the server you defined in Step 3.

5   Publish the demo assertion from the Project Explorer.

     Open the WSDLServiceNameIsNotDummy.asr context menu and select **HP SOA Systinet Server>Publish Assertion**.

For more details, see the *HP HP SOA Systinet Assertion Editor Guide*.

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, you need to do the following in HP SOA Systinet:

- Publish a WSDL containing a service element where the name attribute starts with `Test`. For details, see "How to Publish Content" in the *HP SOA Systinet User Guide*.

- Create a Technical Policy applicable to WSDLs and add the `WSDLServiceNameIsNotDummy` assertion to it. For details, see "How to Manage Technical Policies" in the *HP SOA Systinet Administration Guide*.

- Create a Policy Report using artifact type, `WSDL`, and the new technical policy. For details, see "How to Create Policy Reports" in the *HP SOA Systinet User Guide*.

- 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 "How to Review Policy Reports" in the *HP SOA Systinet User Guide*.

A Dialog Boxes

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

- Define New Implementation Wizard on page 62

- Run on page 65

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 Define New Implementation: Select Resource Type on page 62

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

   Define New Implementation: New Type on page 64

3 Define New Implementation: Implementation Type on page 64

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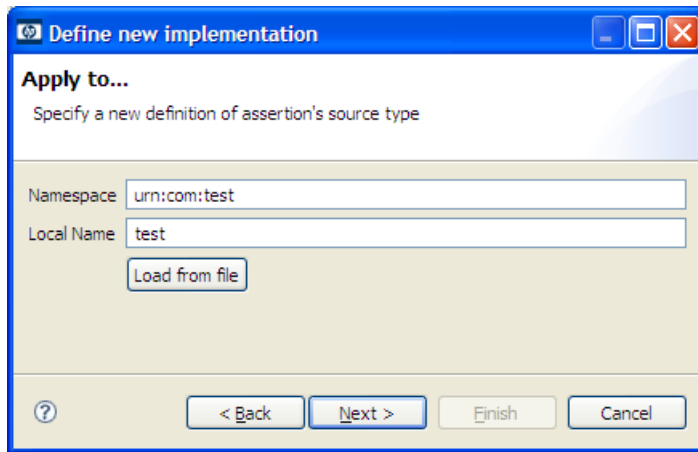
Enter general parameters to define the new implementation.
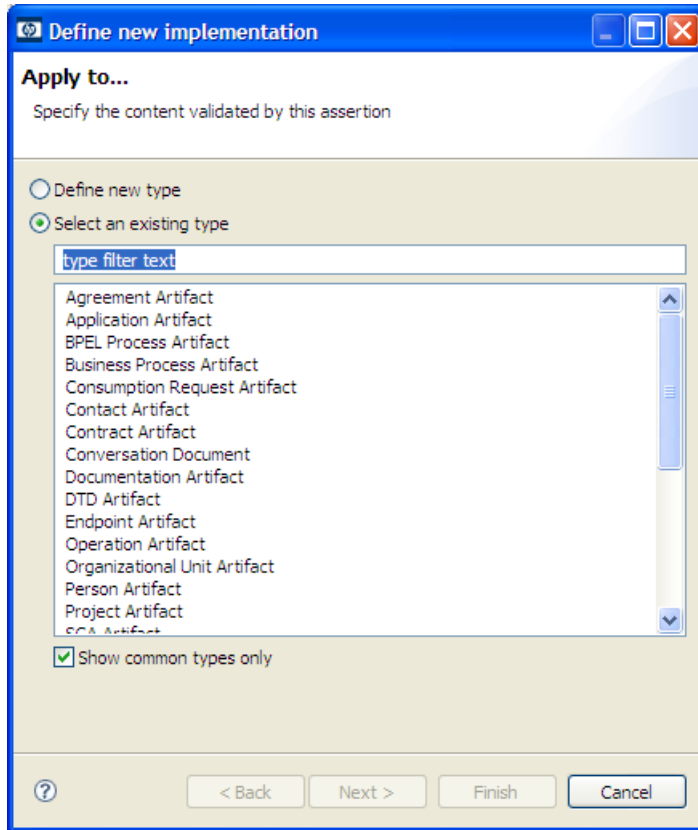
Parameter	Definition
Define New Type	Select this check-box to manually define the resource type you want to use.
Select an Existing Type	Select a predefined resource type from the drop-down menu.
Filter Text	Use the input to reduce the list of resource types.
Show Common Types only	De-select to show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

Define New Implementation: New Type

Define the parameters for a new resource type.

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.

Define New Implementation: Implementation Type

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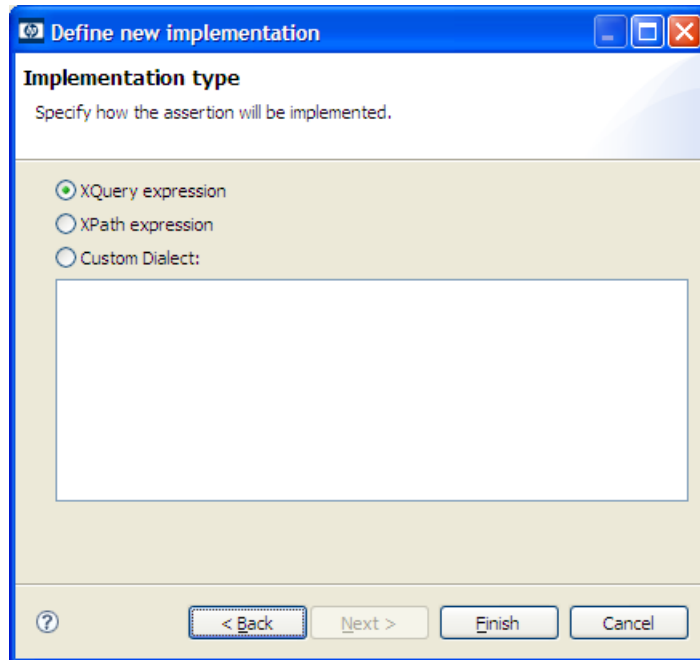
Select the implementation type from the available options.

Run

Define parameters to test the assertion before publishing.
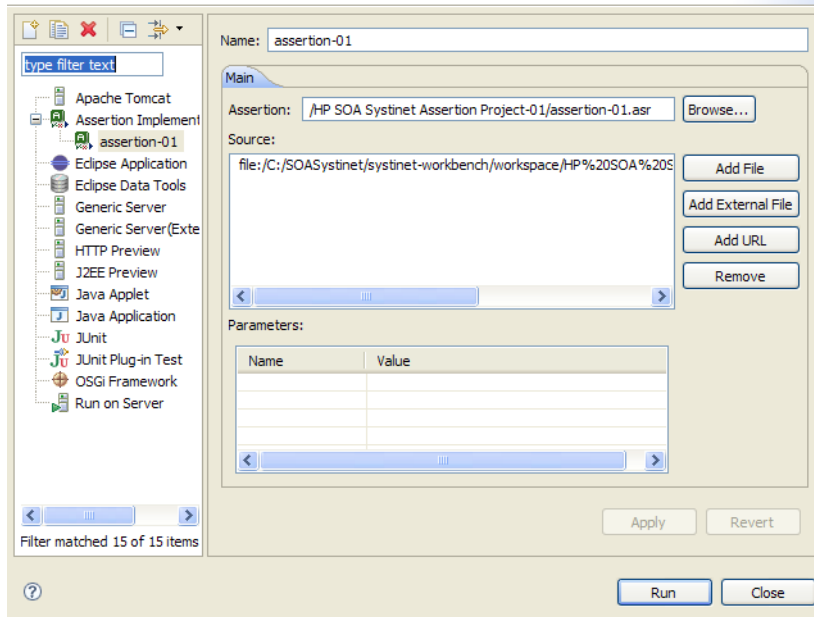
Parameter	Definition
Name	The name you want to use for the test.
Assertion	Browse for and select the assertion you want to test.
Source	Add or remove a local file, external file, or endpoint URL to test against the assertion.
Parameters	Enter the required parameters for the selected source.

B Assertion Developer Reference

Assertion Document Details

Example B.1 on page 67 is the raw XML document of the UDDI BE 01 assertion.

Example B.1. 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 both by the UI console and by 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 75.

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

- Reference Template

- Parameters

- Implementation, 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 Example B.1 on page 67.). 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.

Example B.2 on page 70 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 page 69) 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.

Example B.2. 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`**

  Location of the modified attribute (expressed as an XPointer). See below for details.

- **`pm:Parameter/@Optional`**

  Optionality of the parameter (if it is optional, it might be left unfilled).

- **`pm:Parameter/@Type`**

  Type of the parameter's value. 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/@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.

Example B.3. 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>
```

Example B.4. 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, Example B.5 on page 73 is a parameter with the ValueXPointer set at 5000. This results in the policy document in Example B.6 on page 73. By contrast, if the developer leaves the ValueXPointer blank, the resulting policy document is Example B.7 on page 74.

Example B.5. 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>
```

Example B.6. 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>
```

Example B.7. Policy Document with Empty ValueXPointer in Parameter

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

Table 4, "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. Example B.8 on page 74 is a correctly defined XPointer.

> 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>]}*)

Table 4. 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><b P='ABC'/>
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.)

Example B.8. 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 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 on page 75. A description of all source types to which an implementation may apply.

XPath Assertions on page 77. XPath validation handlers.

XQuery Assertions on page 78. XQuery 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 5, "Source Types Applying to General Resources" for a list of these `SourceType`values and their associated artifacts and namespaces.

- A HP SOA Systinet artifact type. These share the namespace `xmlns:a="http://systinet.com/2005/05/soa/model/artifact"`. They are described in "SOA Definition Model" in the HP SOA Systinet Reference Guide. A list of these `SourceType` values and their matching HP SOA Systinet artifact types is given in Table 6, "SourceTypes Applying to HP SOA Systinet Artifacts ".

Table 5. Source Types Applying to General Resources

Resource	SourceType value
Any HP SOA Systinet resource	xmlns(rest=http://systinet.com/2005/05/soa/resource)rest:resource
SOAP message	xmlns(soap=http://schemas.xmlsoap.org/soap/envelope/)soap:Envelope
UDDI v3 Business Entity	xmlns(uddi=urn:uddi-org:api_v3)uddi:businessEntity
WSDL Definition	xmlns(wsdl=http://schemas.xmlsoap.org/wsdl/)wsdl:definitions
XML Schema	xmlns(xsd=http://www.w3.org/2001/XMLSchema)xsd:schema

Table 6. SourceTypes Applying to HP SOA Systinet Artifacts

HP SOA Systinet artifact	SourceType Value
Agreement	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)agreementArtifact
Application	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)hpsoaApplicationArtifact
Business Policy	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)businessPolicyArtifact
Business Service	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)businessServiceArtifact
Consumption Request	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)contractRequestArtifact
Contact	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)contactArtifact
Contract	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)contractArtifact
Conversation Document	xmlns(a=http://systinet.com/2005/10/soa/policy/report)Conversation
Documentation	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)documentationArtifact
HTTP Message Document	xmlns(a=http://systinet.com/2005/10/soa/policy/report)Message
Person	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)personArtifact
Policy	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)policyArtifact
Registry	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)registryArtifact

HP SOA Systinet artifact	SourceType Value
Report	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)reportArtifact
SOAP Service	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)webServiceArtifact
SLO	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)sloArtifact
Schema	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)schemaArtifact
Taxonomy	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)taxonomyArtifact
UDDI Channel	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)uddiChannelArtifact
UDDI Entity	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)uddiEntityArtifact
UDDI Registry	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)uddiRegistryArtifact
WS-Policy	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)wsPolicyArtifact
WSDL	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)wsdlArtifact
Web Application	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)webArtifact
XML Schema	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)xmlSchemaArtifact
XML Service	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)xmlServiceArtifact
XSLT	xmlns(a=http://systinet.com/2005/05/soa/model/artifact)xsltArtifact

XPath Assertions

Example B.9 on page 77 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..

Example B.9. 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 [http://www.saxonica.com] 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 shown in :

Example B.10. 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 Example B.10 on page 79 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 HP SOA Systinet 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 Example B.10 on page 79 there is no logical
need to call assertionFailed more than once, since the artifact either has one lifecycle stage or none at
all. In Example B.11 on page 80, 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.

Example B.11. XQuery Reporting Multiple Failures

```
declare namespace xs = &quot;http://www.w3.org/2001/XMLSchema&quot;;
            declare namespace val=&quot;http://systinet.com/2005/10/soa/policy/
validation&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
            ()
```

> 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 .

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 [http://www.saxonica.com] 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)
```

C Eclipse Plug-in Requirements

Ensure that the following prerequisite plug-ins are added to your Eclipse development platform based on the required Workbench components:

- DTP SDK 1.5

  Required for Customization Editor and Report Editor.

  http://www.eclipse.org/datatools/downloads.php

- EMF SDO Runtime 2.3.0

  Required for Customization Editor and Report Editor.

  http://www.eclipse.org/modeling/emf/downloads/

- GEF Runtime 3.3

  Required for Customization Editor and Report Editor.

  http://archive.eclipse.org/tools/gef/downloads/drops/R-3.3-200706281000/index.php

- WTP 2.0

  Required for Customization Editor and Report Editor.

  http://www.eclipse.org/webtools/releases/2.0/

- XSD Runtime 2.3

  Required for Customization Editor and Report Editor

  http://www.eclipse.org/modeling/mdt/downloads/?project=xsd

- BIRT 2.2.0

Required for Report Editor.

http://download.eclipse.org/birt/downloads/build_list.php

D Integrating XQuery Function Libraries

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

To integrate an XQuery function library:

1   You JAR file must have the following structure:

- your-lib.jar

    - META-INF

        - your-XQueryContext.xml

    - com

        - your-class

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

```xml
<?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.

> `AE_LIB` refers to `WB_HOME/plugins/`
> `com.systinet.tools.assertioneditor.lib_4.0.0.xxx`.

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.systinet"
 class="com.hp.systinet.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 Workbench with command, **start.exe -clean**.