

HP SOA Systinet Workbench

Software Version: 3.00

Assertion Editor Guide

Document Release Date: June 2008
Software Release Date: June 2008



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.

Third-Party Web Sites

HP provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. HP makes no representations or warranties whatsoever as to site content or availability.

Copyright Notices

© Copyright 2003-2008 Hewlett-Packard Development Company, L.P.

Trademark Notices

Java™ is a US trademark of Sun Microsystems, Inc. Microsoft®, Windows® and Windows XP® are U.S. registered trademarks of Microsoft Corporation. IBM®, AIX® and WebSphere® are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries. BEA® and WebLogic® are registered trademarks of BEA Systems, Inc.

Contents

About this Guide.	5
Document Conventions.	6
Documentation Updates.	7
Support.	8
1 Assertion Editor.	9
Workbench Suite.	9
Overview.	10
User Interface.	11
2 Getting Started.	21
Installing Workbench.	21
Creating an Assertion Project.	23
Downloading and Importing Assertions.	24
3 Manipulating Assertions.	27
Creating Assertions.	27
Editing Assertions.	28
Deleting Assertions.	35
Comparing Assertion Versions.	36
4 Validating and Publishing Assertions.	39
Testing Assertions.	39
Resolving Conflicts.	40
Publishing Assertions.	41
5 Deploying Assertions.	43
Building an Assertion Extension.	43
Applying Extensions.	44

Redeploying the EAR File.	47
6 Customizing Assertions.	49
Customizing Source Type.	49
Adding PM Extensions.	50
A Dialog Boxes.	51
Define New Implementation Wizard.	51
Run.	52
B Assertion Developer Reference.	55
Assertion Document Details.	55



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.

Document Conventions

This document uses the following typographical conventions:

run.bat make	Script name or other executable command plus mandatory arguments.
<code>[-help]</code>	Command-line option.
either or	Choice of arguments.
<i>replace_value</i>	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.
<code>rmdir /S /Q System32</code>	User input.
<code>C:\System.ini</code>	Filenames, directory names, paths and package names.
<code>a.append(b);</code>	Program source code.
<code>server.Version</code>	Inline Java class name.
<code>getVersion()</code>	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 login page.

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

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 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 9](#)
- [Overview on page 10](#)
- [User Interface on page 11](#)

Workbench Suite

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

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

- **Customization Editor**
Customizes the underlying SOA Definition Model (SDM) and the appearance of these artifacts within SOA Systinet.
- **Taxonomy Editor**
Customizes the taxonomies used to categorize artifacts in SOA Systinet.
- **Assertion Editor**
Customizes the conditions applied by your business policies within SOA Systinet.

- **Report Editor**

Customizes report definitions for use with 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 WS-Policy document called a *technical policy*. The technical policy is a set of assertions that fulfils a management requirement. Technical policies, in turn, are associated with specific artifacts or artifact types to form a WS-PolicyAttachments document called a *business policy*. This is the top level of a policy, embodying specific business requirements.

SOA Systinet provides tools for testing whether sources comply with the relevant business 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 23](#)
- [Downloading and Importing Assertions on page 24](#)

- 2 **Create and manage assertions.**

For details, see the following sections:

- [Creating Assertions on page 27](#)
- [Editing Assertions on page 28](#)

- [Deleting Assertions on page 35](#)
- [Comparing Assertion Versions on page 36](#)

3 **Validate assertions before publishing.**

For details, see [Testing Assertions on page 39](#).

4 **Deploy assertions and manage conflicts.**

For details, see the following sections:

- [Publishing Assertions on page 41](#)
- [Resolving Conflicts on page 40](#)

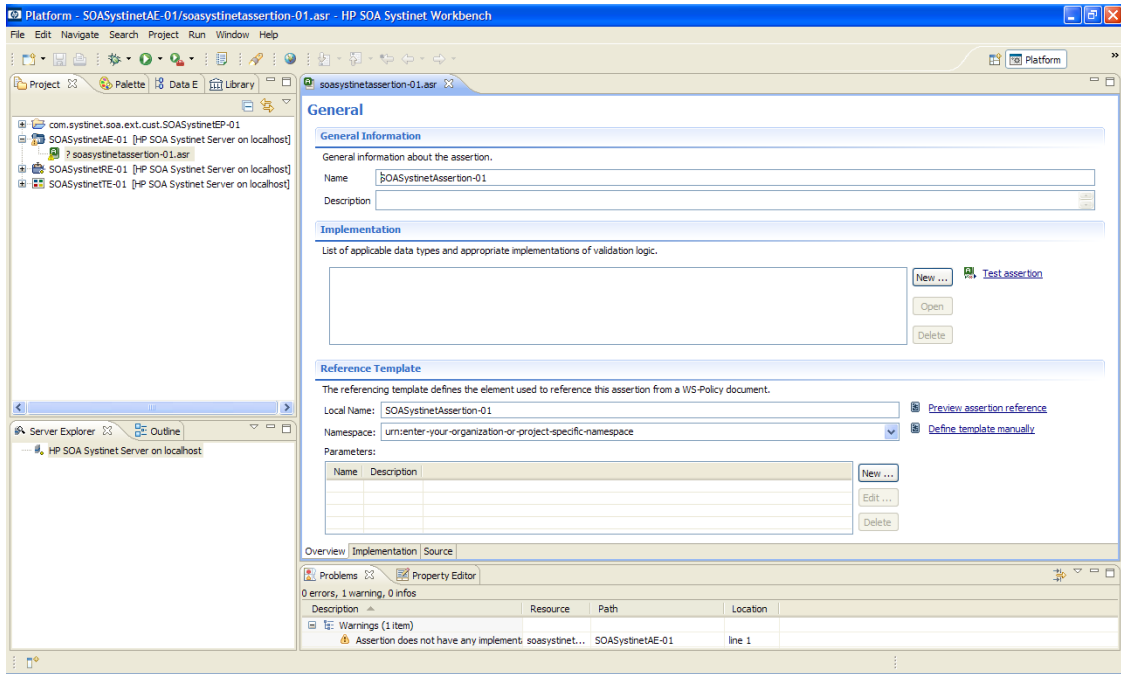
5 **Customize assertions for use with Policy Manager.**

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](#).

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 13.

- **Server Explorer**

The view listing SOA Systinet server connections to Workbench. For details, see [Server Explorer](#) on page 15.

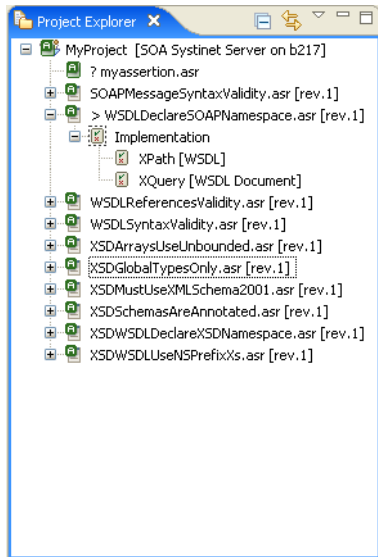
- **Editor**

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](#).

Figure 2. Project Explorer



Right-click elements in the project to view their context menus, as described in [Table 1](#).

Table 1. Project Explorer Context Menu Options

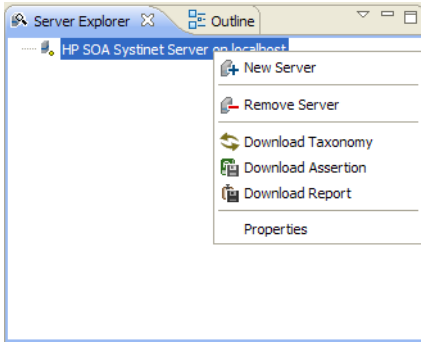
Option	Description
New	Opens a new project, file, or folder. You can also create a new assertion.
Open	Opens the assertion in the default Editor.
Open With	Opens the assertion in editors other than the default Editor.
Copy	Copies the selected item to the clipboard.
Paste	Pastes a copied item from the clipboard into Project Explorer.
Delete	Deletes the selected item.
Move	Moves the selected item to a different location.
Rename	Renames the selected item.
Import	Imports an assertion from a local file into the project.
Export	Exports a project or assertion to a file or Team Project Set.
Refresh	Reflects changes made outside the workspace. The local version does not change. This is the standard Eclipse Refresh action.
Open Project	Opens the project to users.
Close Project	Closes the project to users.
Validate	Runs a compliance check on the selected item.
Run As	Opens the Run dialog box.
Debug As	Opens the Debug dialog box.
Profile As	Opens the Profile dialog box.
Team	Applies a Patch from a team project, or shares your project with a team.
Compare with	Compares the current version of the assertion with the history of its local changes.
Restore from Local History	Restores deleted resources from local history.

Option	Description
HP SOA Systinet	<p>Does one of the following:</p> <ul style="list-style-type: none"> • Downloads an assertion from the server, as described in Downloading and Importing Assertions on page 24. • Updates the assertion with the version on the server, as described in Editing Assertions on page 28. • Publishes the assertion to an SOA Systinet server, as described in Publishing Assertions on page 41. • Removes the assertion from the server, as described in Deleting Assertions on page 35. • Publishes the assertion to a different server, as described in Publishing Assertions on page 41. • Builds an extension, as described in Building an Assertion Extension on page 43.
PDE Tools	Converts projects to plug-in projects.
Properties	General properties of the project, assertion, or implementation. Here you can set read-only status.

Server Explorer

The Server Explorer displays the SOA Systinet servers connected to Workbench, as shown in [Figure 3](#). 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 2](#).

Table 2. 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 18

Overview Tab

The Overview tab shows the components of the assertion, as shown in [Figure 4](#).

Figure 4. Overview Tab

General

General Information
General information about the assertion.

Name: SOAAae01

Description:

Implementation
List of applicable data types and appropriate implementations of validation logic.

New ... Test assertion
Open
Delete

Reference Template
The referencing template defines the element used to reference this assertion from a WS-Policy document.

Local Name: SOAAae01 Preview assertion reference
Namespace: urn:enter-your-organization-or-project-specific-namespace Define template manually

Parameters:

Name	Description

New ...
Edit ...
Delete

Overview | Implementation | Source

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.

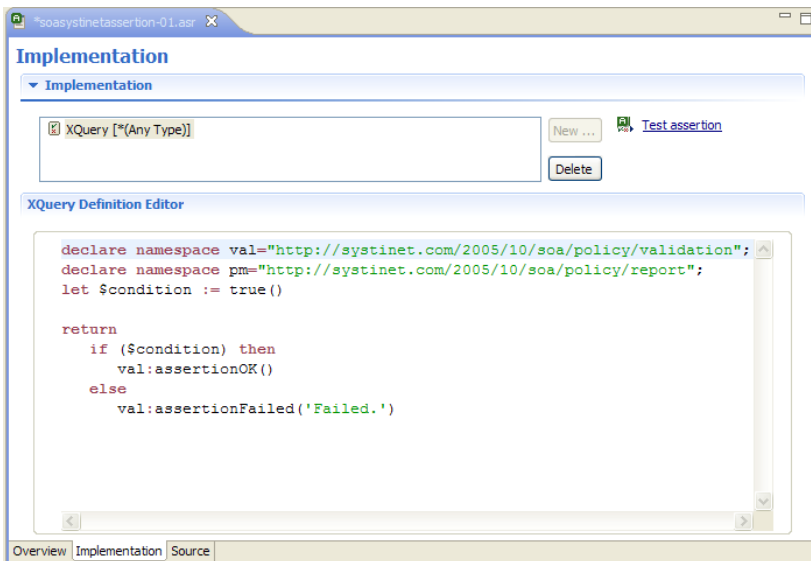
- **Reference Template**

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](#).

Figure 5. Assertion Editor UI: Editor Implementation Tab

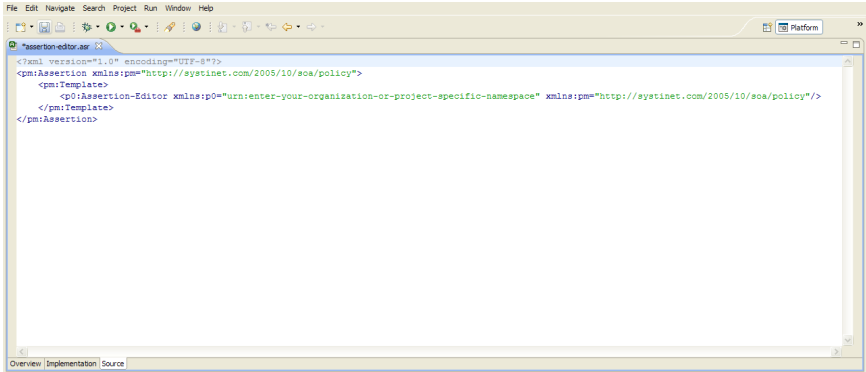


Highlighting an implementation opens the XQuery Definition Editor in the window beneath. For details, see [Writing XQuery Definitions on page 31](#).

Source Tab

The Source tab is an XML editor for editing the assertion, as shown in [Figure 6](#).

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](#)
- [Creating an Assertion Project on page 23](#)
- [Downloading and Importing Assertions on page 24](#)

Installing Workbench

HP SOA Systinet Workbench is an Eclipse development platform distributed as a zip file, `hp-soa-systinet-workbench-3.00-win32.zip`.



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

To install HP SOA Systinet Workbench:

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

To start HP SOA Systinet Workbench:

- Execute `WB_HOME/systinet-workbench/start.exe`.

The first time you start Workbench, the welcome screen opens, as shown in [Figure 7](#).

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.



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/start.ini` for editing.
- 2 Set these new values:
 - `-Xms128m`
 - `-Xmx1024m`

- 3 Save your changes.
- 4 Restart Workbench.

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, add the required parameters.
- 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 assertions to download from the server.
- 7 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 23](#).
- From your local file system, at a later date.

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 on page 27](#)
- [Editing Assertions on page 28](#)
- [Deleting Assertions on page 35](#)
- [Comparing Assertion Versions on page 36](#)

Creating Assertions

In [Creating an Assertion Project on page 23](#), 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 29](#).
 - Test the assertion, as described in [Testing Assertions on page 39](#).
 - Publish the assertion, as described in [Publishing Assertions on page 41](#).

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 40](#).

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 Assertion**.

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

- [Editing General Properties on page 29](#)
- [Adding and Deleting Implementations on page 29](#)
- [Writing XPath Definitions on page 30](#)
- [Writing XQuery Definitions on page 31](#)
- [Editing XQuery Definitions on page 32](#)
- [Editing Reference Templates on page 35](#)

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 51](#).

- 2 Do one of the following:
 - To add a predefined implementation, select it from the **Predefined** drop-down list, and then select the required **Dialect** from the drop-down list.
 - To manually define an implementation, select the **Manual Define** check box, and then enter the required parameters.
- 3 Click **OK**.

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 30](#) or [Writing XQuery Definitions on page 31](#).

Writing XPath Definitions

After creating an implementation that uses an XPath validation handler, as described in [Adding and Deleting Implementations on page 29](#), 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.

Writing XQuery Definitions

Assertion Editor incorporates syntax highlighting for writing and editing XQueries.

To write an XQuery definition:

- 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 [Define New Implementation Wizard on page 51](#).
- 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 32](#), and click **Test Assertion**.

If the assertion passes validation, you can now publish the assertion. For details, see [Publishing Assertions on page 41](#).

If the assertion does not pass validation, you can resolve any problems. For details, see [Resolving Conflicts on page 40](#).

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 33](#)
- [Editing XQueries in Stylus Studio on page 34](#)

[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 WS-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 Enter the required parameters.
- 5 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 on page 35](#)
- [Deleting Assertions on the Server on page 36](#)

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**→**Delete Assertion**.
- 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 on page 39](#)
- [Resolving Conflicts on page 40](#)
- [Publishing Assertions on page 41](#)

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 52](#).

- 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 35](#).

- 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 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**→**Publish Assertion**.

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**→**Publish 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 23](#).



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 40](#).

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 43
- Applying Extensions on page 44
- Redeploying the EAR File on page 47

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 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 pre-packaged data.

Extensions to 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.

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

These extensions contain assertion, reporting, and taxonomy data only and 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 44](#)

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

- [Decoupled DB Scenario on page 46](#)

Database SQL scripts are run manually, and the setup tool performs the other processes as individual steps that are executable on demand. This 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.

Single-Step Scenario

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

To apply extensions to 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.



If your extension does not contain data model changes, select **Apply Extensions Don't Touch DB**.

- 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 47](#).

- For other application servers, 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.



The setup tool normally applies ALTER scripts if database changes are required for an extension. If the ALTER script cannot be used, then a DROP/CREATE process may be used instead. In these cases, you must recreate indices on the database.

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. ....
```

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 47](#).

- For other application servers, 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 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**.
- 5 When the setup tool validates the existence of the JBoss Deployment folder, click **Next**.
- 6 Click **Finish** to close the setup tool.
- 7 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 on page 49
- Adding PM Extensions on page 50

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 System Assertion 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 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 *SOA Systinet Administration Guide*.

A Dialog Boxes

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

- Define New Implementation Wizard on page 51
- Run on page 52

Define New Implementation Wizard

Enter general parameters to define the new implementation.

The dialog box is titled "Define new implementation" and contains the following elements:

- Title Bar:** Includes a help icon (question mark) and a close button (X).
- Section Header:** "Define new implementation" with the subtitle "Specify source type and dialect type".
- Source type section:**
 - Predefined:** A dropdown menu currently showing "Contact Artifact".
 - Manual Define:** An unchecked checkbox.
 - Namespace:** A text input field.
 - Local Name:** A text input field with a "Browse" button to its right.
- Dialect:** A dropdown menu currently showing "XQuery".
- Buttons:** "OK" and "Cancel" buttons at the bottom right.
- Help:** A question mark icon at the bottom left.

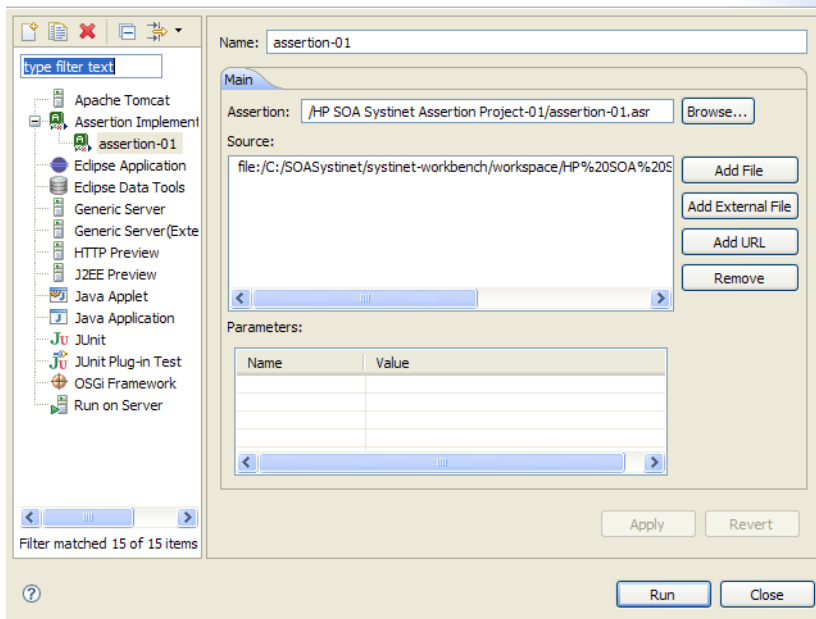
Parameter	Definition
Predefined	Select a predefined source type from the drop-down menu.
Manual Define	Select this check-box to manually define the source type you want to use.
Namespace	Enter the namespace of the source type you want to use.

Parameter	Definition
Local Name	Browse for and select the local name of the source type you want to use.
Dialect	Select the dialect you want to use (default is XQuery).

Run

Define parameters to test the assertion before publishing.

Create, manage, and run configurations



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.

Parameter	Definition
Parameters	Enter the required parameters for the selected source.

B Assertion Developer Reference

Assertion Document Details

Example B.1 on page 55 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>
```

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

- Reference Template
- Parameter
- 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 55.](#)) 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 57](#) 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 56](#)) 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).

Another example:

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

This 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.

In this case, a single XPath pointing to 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 XPath and the location of the value within the element is set in a ValueXPath. For example, [Example B.3 on page 58](#) is a parameter with the ValueXPath set at 5000. This results in the policy document in [Example B.4 on page 58](#). By contrast, if the developer leaves the ValueXPath blank, the resulting policy document is [Example B.5 on page 59](#).

Example B.3: Parameter with ValueXPath Set at 5000

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

Example B.4: Policy Document with ValueXPath in Parameter Set to 5000

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

Example B.5: Policy Document with Empty ValueXPather in Parameter

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

Table 3 shows the XML representations of various XPather and ValueXPather combinations, for optional and required attributes, and whether the value is defined or not. Example B.6 on page 60 is a correctly defined XPather.



Only a simplified form of XPather is recognized in the parameter definition. The rationale is that in this context XPather is used not only for retrieving data, but also for creating parameters via the UI. This is not possible with general XPathers. The recognized XPather must have the following structure:

```
xmlns(prefix1=ns1)*xpather({/{<prefix:>?<localname>[<index>]}*)
```

Table 3. XPather Combinations and Results

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

Example B.6: 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 60](#). A description of all source types to which an implementation may apply.

[XPath Assertions on page 63](#). XPath validation handlers.

[XQuery Assertions on page 64](#). 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 4](#) for a list of these `SourceType` values and their associated artifacts and namespaces.

- A 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 SOA Systinet artifact types is given in [Table 5](#).

Table 4. Source Types Applying to General Resources

SourceType value	Resource
<code>xmlns(soap=http://schemas.xmlsoap.org/soap/envelope/)soap:Envelope</code>	SOAP message
<code>xmlns(wSDL=http://schemas.xmlsoap.org/wSDL/)wSDL:definitions</code>	WSDL Definition
<code>xmlns(xsd=http://www.w3.org/2001/XMLSchema)xsd:schema</code>	XML Schema
<code>xmlns(uddi=urn:uddi-org:api_v2)uddi:businessEntity</code>	UDDI v2 Business Entity
<code>xmlns(uddi=urn:uddi-org:api_v3)uddi:businessEntity</code>	UDDI v3 Business Entity
<code>xmlns(rest=http://systinet.com/2005/05/soa/resource)rest:resource</code>	Any SOA Systinet resource

Table 5. SourceTypes Applying to SOA Systinet Artifacts

SourceType Value	SOA Systinet artifact
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)agreementArtifact	Agreement
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)businessPolicyArtifact	Business Policy
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)businessServiceArtifact	Business Service
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)contactArtifact	Contact
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)contractArtifact	Contract
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)contractRequestArtifact	Consumption Request
xmlns(a=http://systinet.com/2005/10/soa/policy/report)Conversation	Conversation Document
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)documentationArtifact	Documentation
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)hpsoaApplicationArtifact	Application
xmlns(a=http://systinet.com/2005/10/soa/policy/report)Message	HTTP Message Document
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)personArtifact	Person
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)policyArtifact	Policy
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)registryArtifact	Registry
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)reportArtifact	Report
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)schemaArtifact	Schema
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)sloArtifact	SLO
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)taxonomyArtifact	Taxonomy
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)uddiChannelArtifact	UDDI Channel
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)uddiEntityArtifact	UDDI Entity
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)uddiRegistryArtifact	UDDI Registry
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)webArtifact	Web Application
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)webServiceArtifact	SOAP Service
xmlns(a=http://systinet.com/2005/05/soa/model/artifact)wsPolicyArtifact	WS-Policy

SourceType Value	SOA Systinet artifact
<code>xmlns(a=http://systinet.com/2005/05/soa/model/artifact)wslArtifact</code>	WSDL
<code>xmlns(a=http://systinet.com/2005/05/soa/model/artifact)xmlSchemaArtifact</code>	XML Schema
<code>xmlns(a=http://systinet.com/2005/05/soa/model/artifact)xmlServiceArtifact</code>	XML Service
<code>xmlns(a=http://systinet.com/2005/05/soa/model/artifact)xsltArtifact</code>	XSLT

XPath Assertions

[Example B.7 on page 63](#) 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.7: XPath Expression

```
<val:XPath>
    count(/uddi:businessEntity/uddi:name[@xml:lang=$lang])>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) [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.8 on page 64](#):

Example B.8: 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.8 on page 64](#) 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 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.8 on page 64](#) 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.9 on page 65](#), 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.9: 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 [Example B.8 on page 64](#).

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).

Parameter name	Description
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) [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)
```