HP Service Test

for the Windows operating system

Software Version: 9.50

User Guide

Manufacturing Part Number:T6553-90008
Document Release Date: January 2009
Software Release Date: January 2009

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

Trademark Notices

Java™ is a US trademark of Sun Microsystems, Inc.

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

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

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

This web site provides contact information and details about the products, services, and support that HP Software offers.

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

Table of Contents

PART IV: PARAMETERS

PART V: APPENDIX

Welcome to This Guide

Welcome to HP Service Test, HP's tools for creating scripts, with a focus on SOA and Web Service tests. The scripts act like virtual users, allowing you emulate real-life situations. You can use Service Test to develop a script by recording a user performing typical business processes, or you can generate automatic scripts that address your testing requirements.

Using Service Test, you can check your script's functionality using the built-in validation tools and reports. By incorporating scripts into *LoadRunner*, HP's load testing tool, you can check your service's performance under load.

Service Test integrates with *Quality Center*, HP's tool for advanced test management. Using all of Quality Center's capabilities, you can arrange your services and tests in a repository. You can create test plans, set requirements, and track defects for your Web Services.

This chapter includes:

How This Guide Is Organized

This guide contains the following parts:

Part I Working with Service Test

Introduces Service Test scripts and general settings. It also provides an overview of the steps in developing a Vuser script.

Part II Preparing Web Services Scripts

Discusses the ways to add content to a Web Services script both manually and through wizards.

Part III Running Web Service Scripts

Describes how to run a Web Services test and the settings to configure before running the script.

Part IV Parameters

Provides information about creating parameters and setting their values.

Part V Appendixes

Contains keyboard shortcuts for working with Service Test.

Who Should Read This Guide

This guide is for the following users of Service Test:

- Script developers
- Functional Testers
- Load Testers

This document assumes that you are moderately knowledgeable about your enterprise application.

Online Documentation

Service Test includes the following online documentation:

Readme provides last-minute news and information about Service Test and Virtual User Generator. You access the Readme from the **Start** menu.

Books Online/Printer-Friendly Documentation includes PDF versions of the documents. Click the **Help** button and choose **Books Online**.

Online Help is available from specific Service Test windows by clicking in the window and pressing **F1** or clicking the **Help** button.

Online Help includes:

- **Error Codes Troubleshooting** provides clear explanations and troubleshooting tips for Controller connectivity and Web protocol errors as well as general troubleshooting tips for WinSocket, SAPGUI and Citrix protocols.
- **LoadRunner Agent Configuration Tool Online Help** provides help on the Agent Configuration Tool, accessed by clicking the **Help** button in the Agent Configuration dialog box which is accessed via the **Start** menu.

Function Reference gives you online access to all of LoadRunner's functions that you can use when creating Vuser scripts, including examples of how to use the functions. Check HP's Customer Support Web site for updates to the online *Online Function Reference*.

Additional Online Resources

For additional help, refer to one of the following resources:

HP Software Self-solve knowledge base. This link enables you to browse the HP Software Self-solve knowledge base and add your own articles. The URL for this Web site is http://h20230.www2.hp.com/selfsolve/documents.

HP Software Support Web site. This site enables you to access the HP Software Self-solve knowledge base, post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. The URL for this Web site is http://www.hp.com/go/hpsoftwaresupport. You can also access this Web site from the **Help** menu.

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 user ID, go to:
http://h20229.www2.hp.com/passport-registration.html

HP Software Web site. This site provides you with the most up-to-date information on HP Software products. This includes new software releases, seminars and trade shows, customer support, and more. The URL for this Web site is www.hp.com/go/software. You can also access this Web site from the **Help** menu.

Documentation Updates

HP Software is continually updating its product documentation with new information.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to the HP Software Product Manuals Web site (http://h20230.www2.hp.com/selfsolve/manuals).

Part I

Working with Service Test

1

Introducing Service Test

Service Test helps you develop Vuser scripts for a variety of application types and communication protocols.

This chapter includes:

About Service Test

Service Test, also referred to as the Virtual User Generator, *VuGen*, is the primary tool for developing Vuser scripts.

VuGen not only records Vuser scripts, but also runs them. Running scripts from VuGen is useful for debugging. It enables you to emulate how a Vuser script will run when executed as part of a larger test.

When you record a Vuser script, VuGen generates various functions that define the actions that you perform during the recording session. VuGen inserts these functions into the VuGen editor to create a basic Vuser script.

VuGen records Vuser scripts on Windows platforms only. However, a recorded Vuser script can be run on both Windows and UNIX platforms.

Starting VuGen

To open Service Test, select **Start** > **Programs** > **HP Service Test** > **Service Test.** The Service Test Start Page opens.

The following actions are available from the start page:

Action	Procedure
To open an existing script	Click **Open Existing Script**
To open an existing script from a zip archive	Select **File** > **Zip Operations** > **Import From Zip File**
To create a new script	Click **New Vuser Script**

Customizing the Commands

You can customize the appearance of VuGen by configuring toolbars and shortcuts from the Customize dialog box using one of the following tabs:

- Commands Tab
- Toolbars Tab
- Tools Tab
- Keyboard Tab
- Options Tab

Commands Tab

Select a toolbar in the **Categories** section (left pane). The toolbar's buttons are displayed in the right pane, **Commands**. Click on a command to view its description. Drag the desired items from the right pane into the desired open toolbar.

Toolbars Tab

The **Toolbars** tab lets you indicate which toolbars to display. Select the check box for each toolbar you want to display. The available toolbars are Standard, Edit, View, Record, Vuser, Debug, Tools, Tree, and Advanced.

Additionally, the following buttons in this tab allow you to configure the toolbar settings:

- **Reset.** Returns to the default toolbar view settings for toolbar selected in the left pane.
- **Reset All.** Returns to the default toolbar view settings for all toolbars. VuGen issues a warning before reverting back to the default settings.
- **New.** Adds a custom toolbar to the list of shown toolbars. Drag this toolbar to the desired location in the VuGen window. Afterwards, use the Commands tab to add icons to the toolbar.
- **Rename.** Allows you to rename the selected toolbar. This button is only enabled when you select a custom toolbar.
- **Delete.** Deletes the selected toolbar. This button is only enabled when you select a custom toolbar.
- **Show text labels.** Shows text labels for the selected toolbar. For example, if you enable this option for the Record toolbar, the Stop button will display the word Stop and the Run button will display Run.

Tools Tab

The **Tools** tab lets you add custom commands to VuGen toolbars. The command is usually an executable, batch, or *.com* file. In addition you can indicate a specific document or PDF file that you may want to open from within VuGen. You define one or more tools in the Tools tab Tool list. VuGen lists these tools under the Commands tab **Tools** menu. You select the desired command from the Tool menu, and drag it onto the desired toolbar. For example, you can specify *calc.exe*, the Windows calculator, to allow you to open it from VuGen.

Menu Contents

➤ **New.** (left button) Add an item to the Menu Contents list. VuGen creates a new item in the Menu contents list and the cursor moves to the beginning of the line. Type in the name of the command.

➤ **Delete.** (second button) Deletes the selected item from the Menu Contents list.

Up Arrow / Down Arrow. Move the selected item up or down.

Item Details

➤ **Command.** Type in the name of an executable or batch file. Alternatively, click the Browse button to the right of the field, and locate the desired file. If you specify a non-executable file, make sure that the file extension is associated with a program that is installed on the machine.

➤ **Arguments.** Specify run-time arguments to be executed with the specified file.

➤ **Initial Directory.** The initial directory in which to run the custom tool.

Keyboard Tab

The Keyboard tab lets you assign keyboard shortcuts to menu commands. You can assign shortcuts to both standard and custom commands.

To assign a keyboard shortcut:

1 Select a menu category from the Category box.

2 Select a command within the chosen category from the **Commands** box.

3 Click in the **Press New Shortcut Key** box and press the shortcut key or key combination. VuGen displays the key or key combination in the box.

4 Click **Assign**.

5 To remove an assignment, select the command and view its assigned shortcuts. Select the shortcut you want to remove in the **Current Keys** box, and click **Remove**.

6 To restore all default assignments, click **Reset All**. Note that this also deletes all custom keyboard assignments.

Options Tab

Toolbar

You can set several display options that apply to all of the toolbars:

- **Show ScreenTips on toolbars:** Show the screen tips when moving the cursor over the toolbar buttons (enabled by default).
  - **Show Shortcut Keys in Screen Tips:** Show the keyboard shortcut in the tooltips (disabled by default).
- **Large Icons:** Display large buttons on the toolbar (disabled by default).

Personalized Menu and Toolbars

You can personalize the way commands are shown in the menus and toolbars:

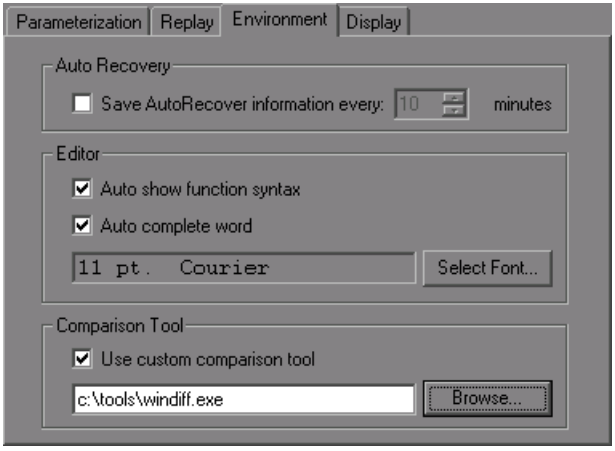
- **Menus show recently used commands first:** The menus show the commands used most recently, at the top of the menu (enabled by default).
  - **Show full menu after a short delay:** For expandable menus, show the full menu after a short delay.

Restore Defaults

➤ **Reset my usage data.** Delete all of the custom commands and restore the default set of visible commands to the menus and toolbars. Does not undo any explicit customization.

Understanding the VuGen Environment Options

You can set up your VuGen working environment in order to customize the auto recovery and editor settings. You set these options from the Tools > General Options > Environment tab.



Auto Recovery

The auto recovery options allow you to restore your script's settings in the event of a crash or power outage. To allow auto recovery, select the **Save AutoRecover Information** check box and specify the time between the saves in minutes.

Editor

You can set the editor options to select a font and enable VuGen's Intellisense capabilities which automatically fill in words and function syntax.

- **Auto show function syntax.** When you type the opening parenthesis of a function, VuGen shows the syntax of the function with its arguments and prototypes. To enable this function globally, select the check box. If you do not check this box, you can still enable this function manually by pressing Ctrl+Shift+Space or choosing **Edit** > **Show Function Syntax** after typing the opening parenthesis in the editor.
- **Auto complete word.** When you type the first underscore of a function, VuGen opens a list of functions allowing you to select the exact function without having to manually type in the entire function. To enable this function globally, select the check box. If you do not check this box, you can enable this function manually by pressing Ctrl+Space or choosing **Edit** > **Complete Word** while typing in the editor.
- **Select Font.** To set the editor font, click **Select Font**. The Font Configuration dialog box opens. Select the desired font, style, and size and click **OK**. Note that only fixed size fonts (Courier, Lucida Console, FixedSys, etc.) are available.

Default Environment Settings

Show Function Syntax and **Auto complete word** are enabled globally by default. Auto Recovery is set to 10 seconds. If you manually changed these settings, you can restore the default values by clicking **Use Defaults**.

Comparison Tool

Vuser scripts can be compared and displayed side by side using the comparison tool.

To compare Vuser scripts:

1. Open the first Vuser script that you want to compare.
2. Select **Tools** > **Compare with Script.**
3. Select the second Vuser script. The Vuser scripts are displayed in a new window, side by side. Differences are highlighted in yellow.

Setting the Environment Options

To set the environment-related options:

1. Select **Tools** > **General Options** and click the Environment tab.
2. To save the current script information for auto recovery, select the **Save AutoRecover Information** option and specify the time in minutes between the saves.
3. To set the editor font, click **Select Font**. The Font dialog box opens. Select the desired font, style, and size and click **OK**. Note that only fixed size fonts (Courier, Lucida Console, FixedSys, and so on) are available.
4. To use a comparison tool other than WDiff, select **Use custom comparison tool** and then specify or browse for the desired tool. Make sure that the executable file that you specify is a valid comparison tool and that the path is correct. An executable file which is not a comparison tool will lead to unexpected results.
5. Click **OK** to accept the settings and close the General Options dialog box.

Viewing and Modifying Vuser Scripts

VuGen provides several views for examining the contents of your script: a text-based Script view, an icon-based Tree view with snapshots, or a icon-based Thumbnail view.

The Script and Tree views are available for most Vuser types. Many protocols also support the Thumbnail view.

Viewing the Code in Script View

The Script view lets you view the actual API functions that were recorded or inserted into the script. This view is for advanced users who want to edit the script by adding "C" or Vuser API functions as well as control flow statements.

To Display the Script View:

From the VuGen main menu, select **View** > **Script View**, or click the **Script** toolbar button. The script is displayed in the text-based Script view. If you are already in the Script view, the menu item is disabled.

You can expand and collapse the functions by clicking the minus or plus sign in the margin to the left of the script. This make the script neater and easier to read.

When working in Script view, you can add steps to the script using the **Insert** > **New Step** command. Alternatively, you can manually enter functions using the Complete Word and Show Function Syntax features. For more information, see "Getting Help on Vuser Functions" on page 42.

Note: If you make changes to a Vuser script while in the Script view, VuGen makes the corresponding changes in the Tree view of the Vuser script. If VuGen is unable to interpret the text-based changes that were made, it will not convert the Script view into Tree or Thumbnail view.

Viewing a Script in Tree View

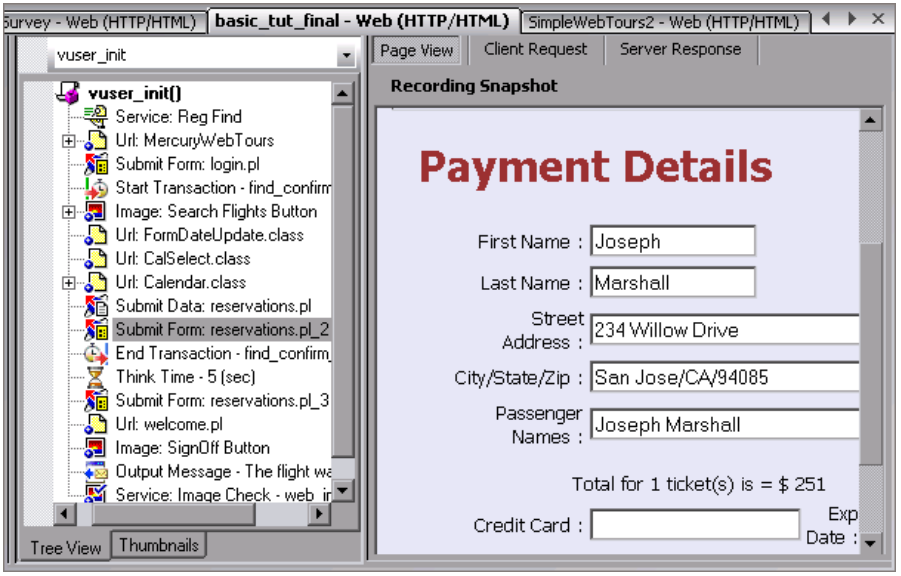
VuGen's Tree view shows the Vuser script in an icon-based format, with each step represented by a different icon.

To display the Tree view:

From VuGen's main menu, select **View** > **Tree View**, or click the **View Tree** button.

The Actions section of the Vuser script is displayed in the icon-based Tree view. To display a different section, select that section in the drop-down list, above the tree.

If you are already in Tree view, the menu item is disabled.



Within the Tree view, you can manipulate steps by dragging them to the desired location. You can also add additional steps between existing steps in the tree hierarchy.

To insert a step in Tree view:

1 Click on a step.

2 Select **Insert Before** or **Insert After** from the right-click menu. The Add Step dialog box opens.

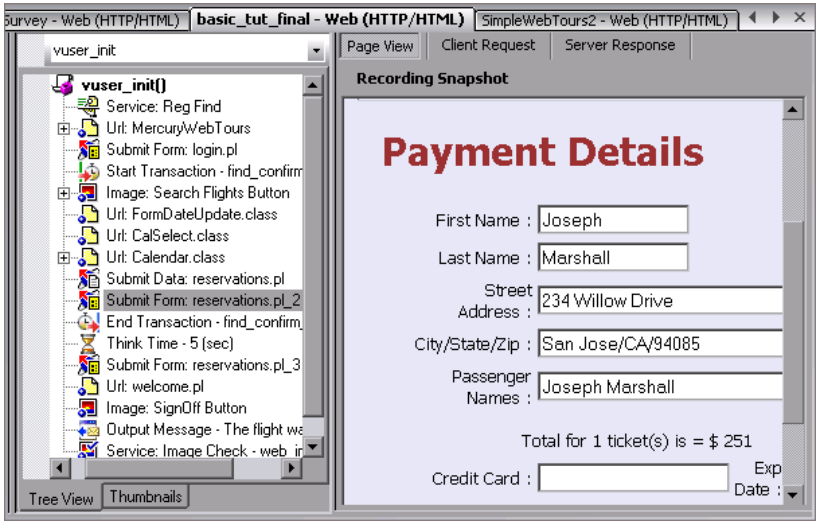
3 Select a step and click **OK**. The Properties dialog box opens.

4 Specify the properties and click **OK**. VuGen inserts the step before or after the current step.

Note: Inserting a step after a parent node results in the detachment of the children nodes from the parent node.

Understanding Snapshots

A snapshot is a graphical representation of the current step. When working in Tree view, VuGen displays the snapshot of the selected step in the right pane. The snapshot shows the client window after the step was executed.



VuGen captures a base snapshot during recording and another one during replay. You compare the Record and Replay snapshots to determine the dynamic values that need to be correlated in order to run the script. For more information, see the section on correlation after recording.

The following toolbar buttons let you show or hide the various snapshot windows.


Show a full window of the recorded snapshot


Show a split window of the recorded and replayed snapshot


Show a full window of the replayed snapshot

To view or hide snapshots:

1. Make sure you are in Tree view. If not, then switch to Tree view (**View > Tree View**).
2. Select **View > Snapshot > View Snapshot**. VuGen shows the snapshot of the client window. If the snapshot is already visible, VuGen hides it.
3. Use the expanded menu of **View > Snapshot** to view the recorded and/or replayed snapshots. You can also use the shortcut toolbar buttons to display the desired view:

Each time you replay the script, VuGen saves another Replay snapshot to the script's result directory: **Iteration1**, **Iteration2**, and so forth.

By default, VuGen compares the recording snapshot to the first replay snapshot. You may, however, select a different snapshot for comparison. To select a specific replay snapshot, select the expanded menu of **View > Snapshot > Select Iteration**. Select a set of results and click **OK.**

Multiple Snapshots

In several protocols such as Microsoft Remote Desktop Protocol (RDP), you can view multiple snapshots for a single step. This occurs when a mismatch occurs during replay and you choose to append the new image to the step. For more information, see the RDP protocol section.

Troubleshooting Snapshots

If you encounter a step without a snapshot, follow these guidelines to determine why it is not available. Note that not all steps are associated with snapshots—only steps with screen operations or for Web, showing browser window content, have snapshots.

Several protocols allow you to disable the capturing of snapshots during recording using the Recording options.

If there is no **Record** snapshot displayed for the selected step, it may be due to one of the following reasons:

- The script was recorded with a VuGen version 6.02 or earlier.
- Snapshots are not generated for certain types of steps.
- The imported actions do not contain snapshots.

If there is no **Replay** snapshot displayed for the selected step, it may be due to one of the following reasons:

- The script was recorded with VuGen version 6.02 or earlier.
- The imported actions do not contain snapshots.
- The Vuser files are stored in a read-only directory, and VuGen could not save the replay snapshots.
- The step represents navigation to a resource.

Snapshot Files

Each time you replay the script, VuGen saves the snapshots in the *script* directory with an *.inf* extension. The replay snapshots are located in the script's result directory: **Iteration1**, **Iteration2**, and so forth, for each set of results.

To determine the name of the snapshot file for a Web Vuser, switch to Script view (**View** > **Script View**). In the following example, the snapshot information is represented by t1.inf.

```
web_url("WebTours",
        "URL=http://localhost/WebTours/",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=",
        "Snapshot=t1.inf",
        "Mode=HTML",
        LAST);
```

For Citrix Vuser scripts, VuGen saves snapshots as .png files in the script's directory. To determine the name of the snapshot file, check the function's arguments in Script view.

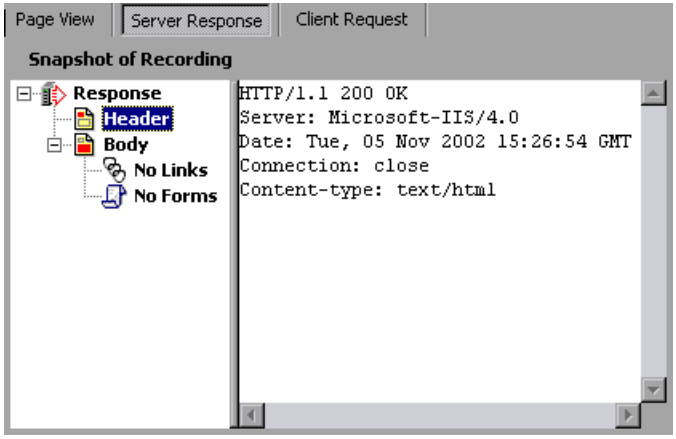
```
ctrx_sync_on_window("ICA Administrator Toolbar", ACTIVATE, 768, 0, 33,
                573, "snapshot12", CTRX_LAST);
```

Web Vuser Snapshot Tabs

In the Snapshot window for Web Vusers, the following tabs are available:

- **Page View.** Display the snapshot in HTML as it would appear in a browser. This button is available for both the recording and replay snapshots. Use this view to make sure you are viewing the correct snapshot. In this view, however, you do not see the values that need to be correlated.
- **Server Response.** Displays the server response HTML code of the snapshot. This button is available for both the recorded and replayed snapshots. The HTML view also shows a tree hierarchy of the script in the left pane, with a breakdown of the document's components: Header and Body with the title, links, forms, and so forth.

To find the HTML text of an element in the source in the right pane, select the element in the left pane of the Server Response, and select **Find Element** from the right-click menu.



➤ **Client Request.** Displays the client request HTML code of the snapshot. This button is available for both the recorded and replayed snapshots. The HTML view also shows a tree hierarchy of the script in the left pane, with a breakdown of the document's components: Header and Body and their sub-components.

Viewing Script Thumbnails

For several Vuser types such as Web, SAPGUI and Citrix, you can view thumbnail representations of the snapshots. You can view thumbnails in either Tree view or through the Transaction Editor.

Viewing Thumbnails in Tree View

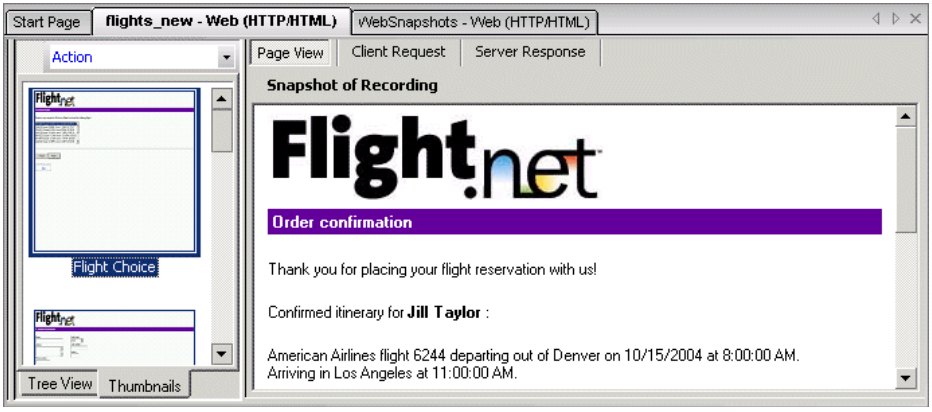
In Tree view, the **Thumbnail** tab appears at the bottom of the Tree view window.

By default, the thumbnail view only shows primary steps in your script. To show all thumbnails, select **View > Show All Thumbnails**. VuGen shows the thumbnails for all of the steps in the script.

Note: For multiple iterations, the VuGen shows the replay thumbnails for the last iteration. To show the thumbnails of a specific iteration, select **View > Snapshot > Select Iteration** and select the desired iteration.

To view the thumbnails from Tree View:

1. Click the **Thumbnail** tab at the bottom of the left pane.
2. Click the desired thumbnail image to open the thumbnail's snapshot in the right pane.



3 Double-click on a thumbnail image to view a larger image. A separate window opens showing a larger view of the thumbnail.

Setting the Mode for Viewing Actions

You can instruct VuGen how to view the script by its actions (**View > Actions**). By default, VuGen displays the script actions in the left pane when in Script view only. You can keep the default setting (**Open Automatically**), or open the action pane in the Tree view and/or Script view.

- **Open Automatically.** In Script view only, VuGen displays the script actions in the left pane. This is the default setting. Clear the selection to hide the actions pane.
- **Open in Tree Mode.** View the actions in Tree view. VuGen opens three panes: the actions, the steps, and the snapshot (if Snapshot view is enabled). You can adjust the width of each of these panes by dragging its border in the desired direction. Clear the selection to hide the actions pane.
- **Open in Script Mode.** View the actions in Script view. VuGen opens two panes: the actions, and the script view. You can adjust the width of the actions pane by dragging its border in the desired direction. Clear the selection to hide the actions pane.

Viewing Thumbnails in the Workflow Wizard

You can view the snapshots through the Transaction Editor. This view sorts the thumbnails by actions and provides you with an flat thumbnail view of all of the script's steps.

To view thumbnails in the Transaction Editor:

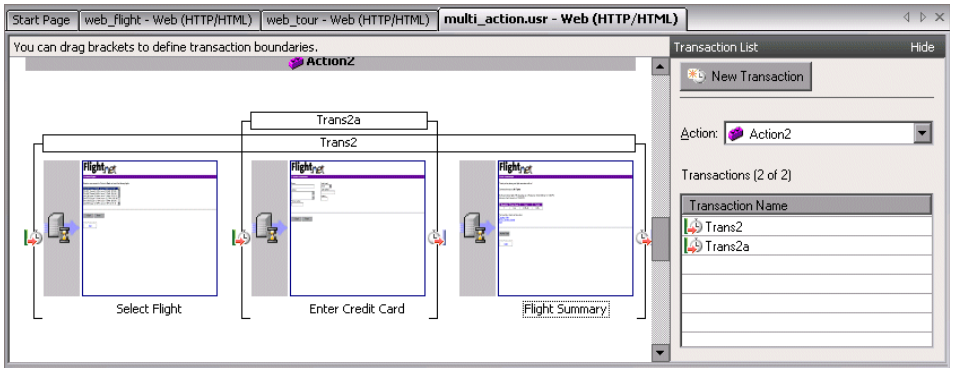
1 Click the **Tasks** button on the toolbar to open the task list pane.

2 Click the **Enhancements** > **Transactions** link. The Transaction Editor opens in the middle and right panes.

For a more encompassing view, click **Tasks** to hide the Task list.

3 In the right pane, select the action that you want to view. VuGen displays the action that you selected.

In the following example, **Action2** was selected.



Working with Thumbnails

VuGen lets you work with thumbnails by renaming them, annotating them, and viewing them in a larger size.

To view a thumbnail as a larger image:

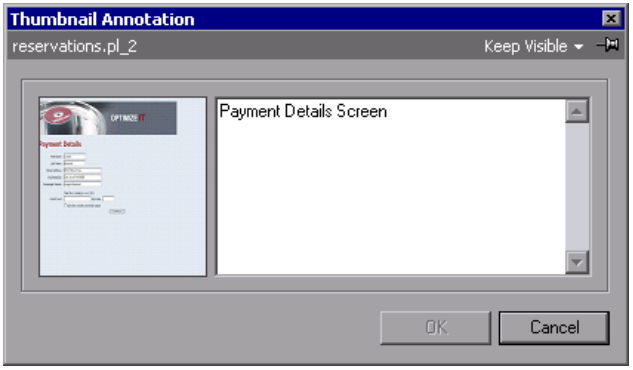
Select **View Larger Image** from the right-click menu or press **Alt+F6**. A separate window opens showing a larger view of the thumbnail.

To rename a thumbnail:

1. Select the thumbnail and select **Rename** from the right-click menu or press F2.
2. Type in the desired text.

To annotate a thumbnail:

1 Select a thumbnail and select **Annotate** from the right-click menu or press Alt+F2. The Thumbnail Annotation dialog box opens.

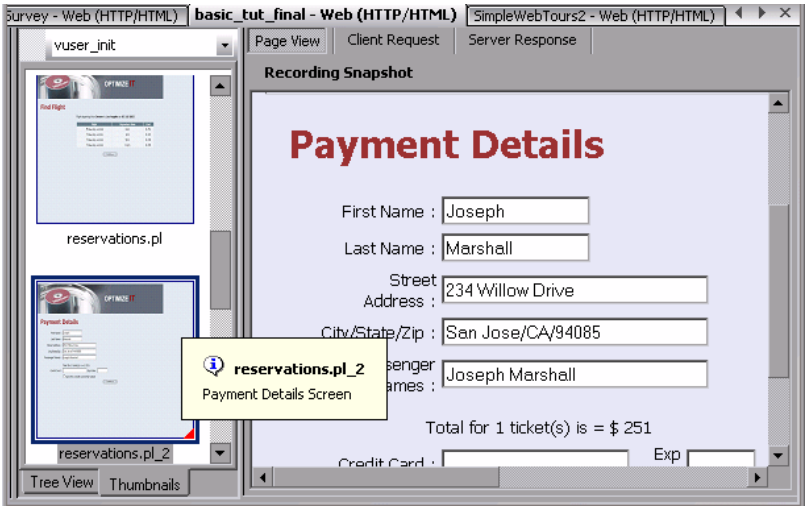


2 Type text into the right pane of the Thumbnail Annotation dialog box.

3 Click **OK** to save the annotation and close the dialog box.

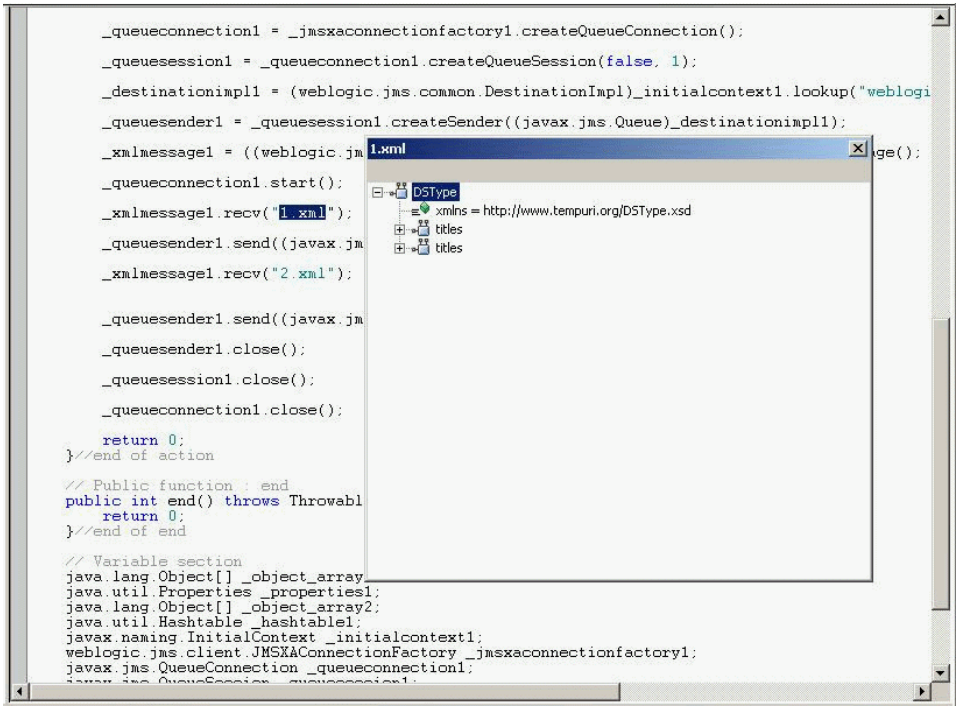
To leave the Annotation box open in order to add more text or work with other snapshots, select **Keep Visible** from the upper right corner. The **OK** button changes to **Apply**.

After you insert an annotation for a thumbnail, VuGen places a red mark in the bottom right corner to indicate that an annotation exists. If you move your mouse over the thumbnail, VuGen shows a popup of the annotation text.



Using the XML Viewer

For certain protocols using XML files such as JMS, VuGen lets you view an XML structure directly from the editor window. A viewer displays the XML elements, and allows you to collapse or expand each of the nodes.



To view an file in the XML viewer:

1. In Tree view, select a step with the XML file and select **View** > **XML**.
2. In Script view, right-click the XML file name and select **View XML**.

Running Vuser Scripts with VuGen

In order to perform testing or monitor an application with your Vuser script, you need to incorporate it into a LoadRunner scenario or Business Process Monitor profile. Before doing this, you check the script's functionality by running it from VuGen. For more information, see Chapter 3, "Running Vuser Scripts in Standalone Mode."

If the script replay is successful, you can then integrate it into your environment: a LoadRunner scenario, Performance Center load test, or Business Process Monitor profile. For more information, see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availibility Center User Guides*.

Before you run a Vuser script, you can modify its run-time settings. These settings include the number of iterations that the Vuser performs, and the pacing and the think time that will be applied to the Vuser when the script is run. For more information on configuring run-time settings, see Chapter 2, "Configuring Run-Time Settings."

When you run a Vuser script, it is processed by an interpreter and then executed. You do not need to compile the script. If you modify a script, any syntax errors introduced into the script are noted by the interpreter. You can also call external functions from your script that can be recognized and executed by the interpreter. For more information, see Appendix 83, "Calling External Functions."

Advanced users can compile a recorded script to create an executable program. For more information, see Chapter 5, "Enhancing Vuser Scripts."

Understanding VuGen Code

When you record a Vuser script, VuGen generates Vuser functions and inserts them into the script. There are two types of Vuser functions:

- General Vuser Functions
- Protocol-Specific Vuser Functions

The *general* Vuser functions and the *protocol-specific* functions together form the LoadRunner API. This API enables Vusers to communicate directly with a server. VuGen displays a list of all of the supported protocols when you create a new script. For syntax information about all of the Vuser functions, see the *Online Function Reference* (**Help > Function Reference**).

General Vuser Functions

The general Vuser functions are also called LR functions because each LR function has an **lr** prefix. The LR functions can be used in any type of Vuser script. The LR functions enable you to:

- Get run-time information about a Vuser, its Vuser Group, and its host.
- Add transactions and synchronization points to a Vuser script. For example, the **lr_start_transaction** (**lr.start_transaction** in Java) function marks the beginning of a transaction, and **the lr_end_transaction** (**lr.end_transaction** in Java) function marks the end of a transaction. See Chapter 5, "Enhancing Vuser Scripts" for more information.
- Send messages to the output, indicating an error or a warning.

See "Getting Help on Vuser Functions" on page 42 for a list of LR functions, and for details see the *Online Function Reference* (**Help > Function Reference**).

Protocol-Specific Vuser Functions

In addition to the general Vuser functions, VuGen also generates and inserts protocol-specific functions into the Vuser script while you record.

The protocol-specific functions are particular to the type of Vuser that you are recording. For example, VuGen inserts LRD functions into a database script, LRT functions into a Tuxedo script, and LRS functions into a Windows Sockets script.

By default, VuGen's automatic script generator creates Vuser scripts in C for most protocols, and in Java for Java type protocols. You can instruct VuGen to generate code in Visual Basic or Javascript. For more information, see Chapter 77, "Setting Script Generation Preferences."

All standard conventions apply to the scripts, including control flow and syntax. You can add comments and conditional statements to the script just as you do in other programming languages.

The following segment from a Web Vuser script shows several functions that VuGen recorded and generated in a script:

```
#include "as_web.h"


Action1()
{

    web_add_cookie("nav=140; DOMAIN=dogbert");

    web_url("dogbert",
        "URL=http://dogbert/",
        "RecContentType=text/html",
        LAST);

    web_image("Library",
        "Alt=Library",
        LAST);

    web_link("1 Book Search:",
        "Text=1 Book Search:",
        LAST);

    lr_start_transaction("Purchase_Order");
…
```

For more information about using C functions in your Vuser scripts, see Chapter 5, "Enhancing Vuser Scripts." For more information about modifying a Java script, see Chapter 42, "Programming Java Scripts."

Note: The C Interpreter used for running Vuser scripts written in C, only supports the ANSI C language. It does not support the Microsoft extensions to ANSI C.

Getting Help on Vuser Functions

You can add API Vuser functions to any script in order to enhance its capabilities. VuGen generates Vuser functions while you record. If required, you can manually insert additional functions into a script after recording. For information about typical enhancements, see Chapter 5, "Enhancing Vuser Scripts."

You can get help for VuGen's API functions in several ways:

- Online Function Reference
- Word Completion
- Show Function Syntax
- Header File

In addition, you can use the standard Search feature (**Edit > Find**) to locate functions within a script, or the Find In Files feature on page 86 to search all of the files in the script.

Online Function Reference

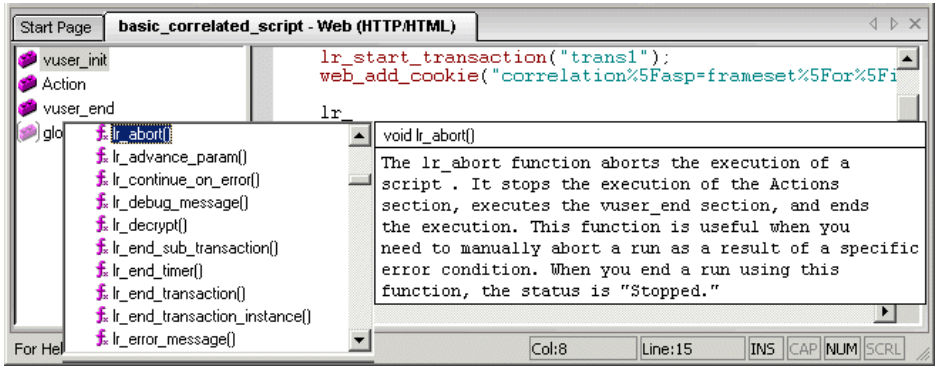
The *Online Function Reference* contains detailed syntax information about all of the VuGen functions. It also provides examples for the functions. You can search for a function by its name, or find it through a categorical or alphabetical listing.

To open the *Online Function Reference*, select **Help > Function Reference** from the VuGen interface. Then select a protocol and the desired category.

To obtain information about a specific function that is already in your script, place your cursor on the function in the VuGen editor, and press the F1 key.

Word Completion

As part of the *IntelliSense* enhancements, the VuGen editor incorporates the *Word Completion* feature. When you begin typing a function, after you type the first underscore, VuGen opens a list box displaying all available matches to the function prefix, along with the function's syntax and description.

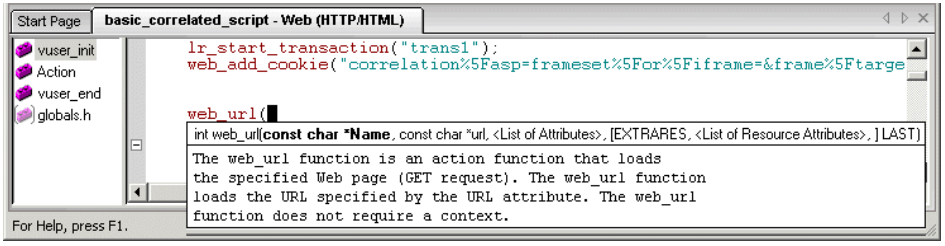


To use one of the displayed functions, select it, or scroll to the desired item and then select it. VuGen inserts the function at the location of the cursor. To close the list box, press the Esc key.

By default, VuGen uses word completion globally. To disable word completion, select **Tools** > **General Options** and select the Environment tab. Clear the check box adjacent to the **Auto complete word** option. If you disable word completion globally, you can still bring up the list box of functions by pressing Ctrl+Space or choosing **Edit** > **Complete Word** while typing in the editor.

Show Function Syntax

An additional feature of VuGen's Intellisense, is **Show Function Syntax**. When you type the opening parenthesis of a function, VuGen shows the syntax of the function with its arguments and prototypes and a brief description.



By default, **Show Function Syntax** is enabled globally. To disable this feature, select **Tools** > **General Options** and select the Environment tab. Clear the check box adjacent to the **Auto show function syntax** option.

If you disable **Show Function Syntax** globally, you can still bring up the syntax by pressing Ctrl+Shift+Space or choosing **Edit** > **Show Function Syntax** after typing the opening parenthesis in the editor.

Header File

All of the non-Java function prototypes are listed in the library header files. The header files are located within the *include* directory of the product installation. They include detailed syntax information and return values. They also include definitions of constants, availability, and other advanced information that may not have been included in the Function Reference.

In most cases, the name of the header file corresponds to the prefix of the protocol. For example, Database functions that begin with an **lrd** prefix, are listed in the **lrd.h** file.

The following table lists the header files associated with the most commonly used protocols:

Protocol	File
AJAX (Click and Script)	**web_ajax.h**
Citrix	**ctrxfuncs.h**
COM/DCOM	**lrc.h**
Database	**lrd.h**
FTP	**mic_ftp.h**
General C function	**lrun.h**
IMAP	**mic_imap.h**
LDAP	**mic_mldap.h**
MAPI	**mic_mapi.h**
Oracle NCA	**orafuncs.h**
POP3	**mic_pop3.h**
RDP	**lrrdp.h**
SAPGUI	**as_sapgui.h**
SAP (Click and Script)	**sap_api.h**
Siebel	**lrdsiebel.h**
SMTP	**mic_smtp.h**
Terminal Emulator	**lrrte.h**
WAP	**as_wap.h**
Web (HTML\HTTP)	**as_web.h**
Web (Click and Script)	**web_api.h**
Web Services	**wssoap.h**
Windows Sockets	**lrs.h**

2

Configuring Run-Time Settings

After you record a Vuser script, you configure the run-time settings for the script. These settings specify how the script behaves when it runs.

This chapter includes:

About Run-Time Settings

After you record a Vuser script, you can configure its run-time settings. The run-time settings define the way that the script runs. These settings are stored in the file *default.cfg,* located in the Vuser script directory. Run-time settings are applied to Vusers when you run a script using VuGen, the Controller, or Business Process Monitor.

Configuring run-time settings allows you to emulate different kinds of user activity. For example, you could emulate a user who responds immediately to output from the server, or a user who stops and thinks before each response. You can also configure the run-time settings to specify how many times the Vuser should repeat its set of actions.

You use the Run-Time Settings dialog box to display and configure the run-time settings tree. You can open these settings in one of the following ways:

- Click the **Run-Time Settings** button on the VuGen toolbar.
- Press the keyboard shortcut key **F4**.
- Select **Vuser** > **Run-Time Settings**.

You can also modify the run-time settings from the LoadRunner Controller. For more information, see the product's documentation.

Note: For LoadRunner, the default run-time setting support the debugging environment of VuGen and the load testing environment of the Controller. The default settings are:

- **Think Time.** Off in VuGen and Replay as Recorded in the Controller.
- **Log.** Standard in VuGen and off in the Controller.
- **Download non-HTML resources.** Enabled in VuGen and the Controller.

The General run-time settings described in this chapter, apply to all types of Vuser scripts. They include:

- Run Logic (Iterations)
  - Pacing
  - Log
  - Think Time
  - Miscellaneous
  - Additional Attributes

For protocols that do NOT support multiple actions, such as WinSocket and Database (Oracle 2-tier, Sybase, MSSQL, and so on), the Iteration and Pacing options are both handled from the Pacing tab. Many protocols have additional run-time settings. For information about the specific run-time settings for these protocols, see the appropriate sections.

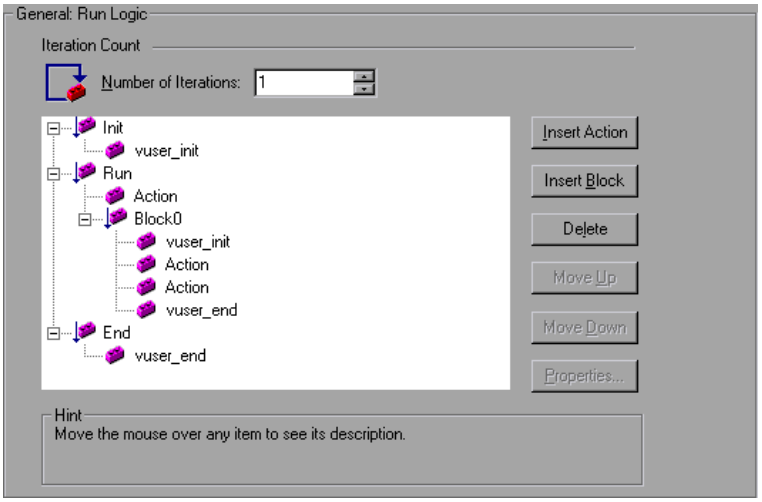
Configuring Run Logic Run-Time Settings (multi-action)

Note: The following section only applies to protocols that work with multiple actions. If the **Run Logic** node exists under the run-time settings, it is a multiple action protocol. For single action protocols, see "Setting Pacing and Run Logic Options (single action)" on page 56.

Every Vuser script contains three sections: *vuser_init*, *Run (Actions)*, and *vuser_end*. You can instruct a Vuser to repeat the *Run* section when you run the script. Each repetition is known as an *iteration*.

The *vuser_init* and *vuser_end* sections of a Vuser script are not repeated when you run multiple iterations.

Open the Run-Time Settings and select the **General:Run Logic** node.



➤ **Number of Iterations.** The number of iterations. The Vusers repeat all of the Actions the specified number of times.

Note: For the LoadRunner Controller: If you specify a scenario duration in the Scheduling settings, they override the Vuser iteration settings. This means that if the duration is set to five minutes (the default setting), the Vusers will continue to run as many iterations as required for five minutes, even if the run-time settings specify only one iteration.

When you run scripts with multiple actions, you can indicate how to execute the actions, and how the Vuser executes them:

➤ **Action Blocks.** Action blocks are groups of actions within your script. You can set the properties of each block independently—its sequence, iterations, and weighting.

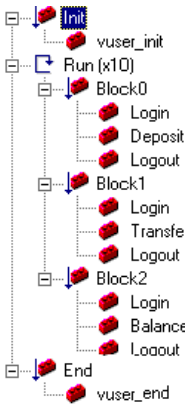
➤ **Sequence.** You can set the order of actions within your script. You can also indicate whether to perform actions sequentially or randomly.

- **Iterations.** In addition to setting the number of iterations for the entire *Run* section, you can set iterations for individual actions or action blocks. This is useful, for example, in emulating a commercial site where you perform many queries to locate a product, but only one purchase.
- **Weighting.** For action blocks running their actions randomly, you can set the *weight* or percentage of each action within a block.

Creating Action Blocks

Action blocks are groups of actions within the Vuser script. You can create separate action blocks for groups of actions, adding the same action to several blocks. You can instruct VuGen to execute action blocks or individual actions sequentially or randomly. In the default sequential mode, the Vuser executes the blocks or actions in the order in which they appear in the iteration tree view.

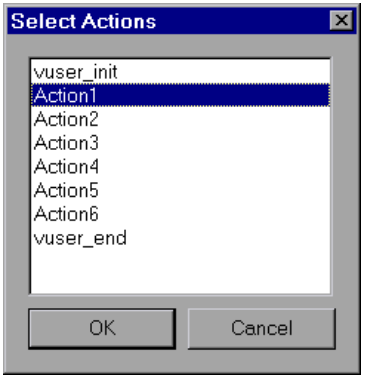
In the following example, *Block0* performs a deposit, *Block1* performs a transfer, and *Block2* submits a balance request. The *Login* and *Logout* actions are common to the three blocks.



You configure each block independently—its sequence and iterations.

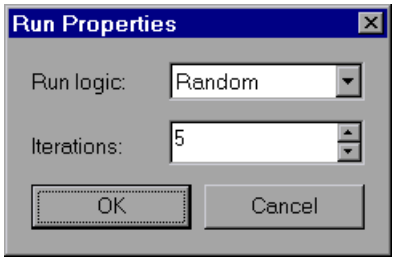
To configure actions and action blocks:

1. Create all of the desired actions through recording or programming.
2. Open the Run-Time setting. Select the **General:Run Logic** node.
3. Add a new action block. Click **Insert Block**. VuGen inserts a new Action block at the insertion point with the next available index (*Block0*, *Block1*, *Block2*).
4. Add actions to the block. Click **Insert Action**. The Select Actions list opens.



5. Select an action to add to the block and click **OK**. VuGen inserts a new action into the current block or section.
6. Repeat step 3 for each action you want to add to the block.
7. To remove an action or an action block, select it and click **Delete**.
8. Click **Move Up** or **Move Down** to modify an item's position.

9 Click **Properties** to set the number of iterations and run logic of the actions. The Run Properties dialog opens.

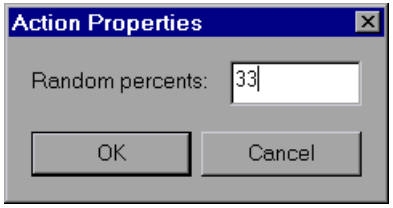


10 Select *Sequential* or *Random* from the **Run Logic** list, indicating to VuGen whether to run the actions sequentially or randomly.

11 Specify the number of iterations in the **Iterations** box. Note that if you define parameters within the action block, and you instruct VuGen to update their values each iteration, it refers to the global iteration—not the individual block iteration.

12 Click **OK**.

13 For blocks with Random run logic, set the weighting of each action. Right-click an action and select **Properties**. The Action Properties dialog opens.



Specify the desired percent for the selected block or action. In the **Random Percents** box, specify a percentage for the current action. The sum of all percentages must equal 100.

14 Repeat the above steps for each element whose properties you want to set.

Pacing Run-Time Settings

Note: The following section only applies to protocols that work with multiple actions. If the **Run Logic** node exists under the run-time settings, it is a multiple action protocol. For single action protocols, see "Setting Pacing and Run Logic Options (single action)" on page 56.

The Pacing Run-Time settings let you control the time between iterations. The pace tells the Vuser how long to wait between iterations of your actions. You instruct the Vusers to start each iteration using one of the following methods:

- **As soon as the previous iteration ends.** The new iteration begins as soon as possible after the previous iteration ends.
- **After the previous iteration ends with a fixed or random delay of …** Starts each new iteration a specified amount of time after the end of the previous iteration. Specify either an exact number of seconds or a range of time. For example, you can specify to begin a new iteration at any time between 60 and 90 seconds after the previous iteration ends.

  When you run the script, VuGen shows the time the Vuser waited between the end of one iteration and the start of the next one, in the Execution Log.
- **At fixed or random intervals, every … [to …] seconds.** You specify the time between iteration—either a fixed number of seconds or a range of seconds from the beginning of the previous iteration. For example, you can specify to begin a new iteration every 30 seconds, or at a random rate ranging from 30 to 45 seconds from the beginning of the previous iteration. Each scheduled iterations will only begin when the previous iteration is complete.

Each scheduled iteration will only begin when the previous iteration is complete. When you run the script, VuGen shows the time the Vuser waited between the end of one iteration and the start of the next one, in the Execution Log.

For example, assume that you specify to start a new iteration every four seconds:

- If the first iteration takes three seconds, the Vuser waits one second.
- If the first iteration takes two seconds to complete, the Vuser waits two seconds.
- If the first iteration takes 8 seconds to complete, the second iteration will start 8 seconds after the first iteration began. VuGen displays a message in the Execution Log to indicate that the iteration pacing could not be achieved.

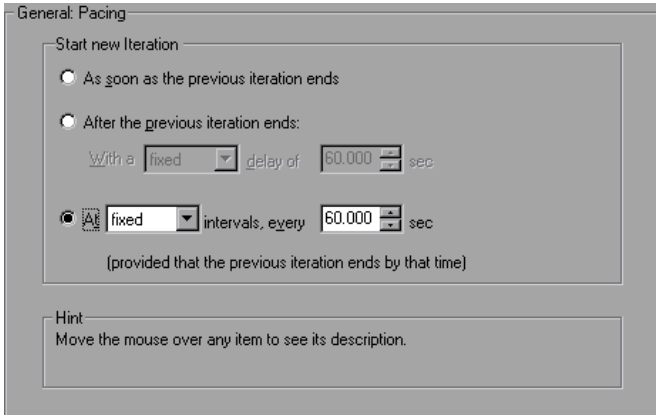
For further instructions about setting the Pacing options, see "Configuring Pacing Run-Time Settings (multi-action)" on page 55.

Configuring Pacing Run-Time Settings (multi-action)

You use the Pacing options to pace your actions by setting the time intervals between iterations.

To set the pacing between iterations:

1 Open the Run-Time Settings and select the **General:Pacing** node.



2 In the **Start New Iteration** section, select one of the following options:

- ➤ As soon as the previous iteration ends
- ➤ After the previous iteration ends
- ➤ At fixed or random intervals

3 For the **After the previous iteration ends** option:

- ➤ Select a delay type: **fixed** or **random**.
- ➤ Specify a value for fixed, or a range of values for the random delay.

4 For the **At … intervals** option:

- ➤ Select a interval type: **fixed** or **random**.
- ➤ Specify a value for fixed, or a range of values for the random interval.

5 Click **OK**.

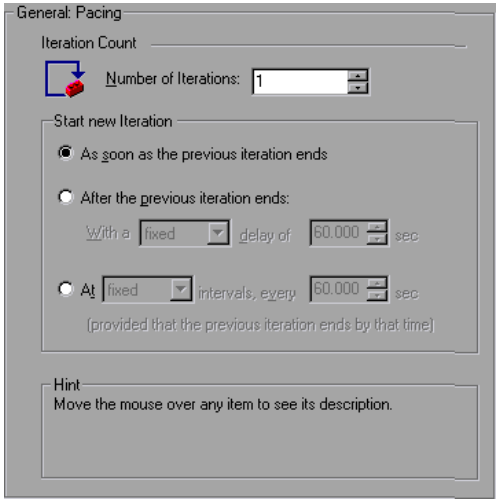
Setting Pacing and Run Logic Options (single action)

Note: The following section only applies to protocols that work with single actions—not multiple actions. If there is a **Pacing** node and not a **Run Logic** node under the General run-time settings, it is a single action protocol.

You can instruct a Vuser to repeat the *Action* section when you run the script. Each repetition is known as an *iteration*. The *vuser_init* and *vuser_end* sections of a Vuser script are not repeated when you run multiple iterations.

To set the iteration and pacing preferences:

1 Click the **Run-Time Settings** button on the VuGen toolbar or select **Vuser > Run-Time Settings**. Click the **Pacing** node to display the iteration and pacing options.



2 Specify the number of iterations in the **Iteration Count** box. The Vuser repeats all of the Actions the specified number of times.

3 In the **Start New Iteration** section, select one of the following options:

- As soon as the previous iteration ends
- After the previous iteration ends
- At fixed or random intervals

4 For the **After the previous iteration ends** option:

- Select a delay type: **fixed** or **random**.
- Specify a value for fixed, or a range of values for the random delay.

5 For the **At … intervals** option:

- Select a interval type: **fixed** or **random**.
- Specify a value for fixed, or a range of values for the random interval.

6 Click **OK**.

For an overview of the pacing options, see "Pacing Run-Time Settings" on page 54.

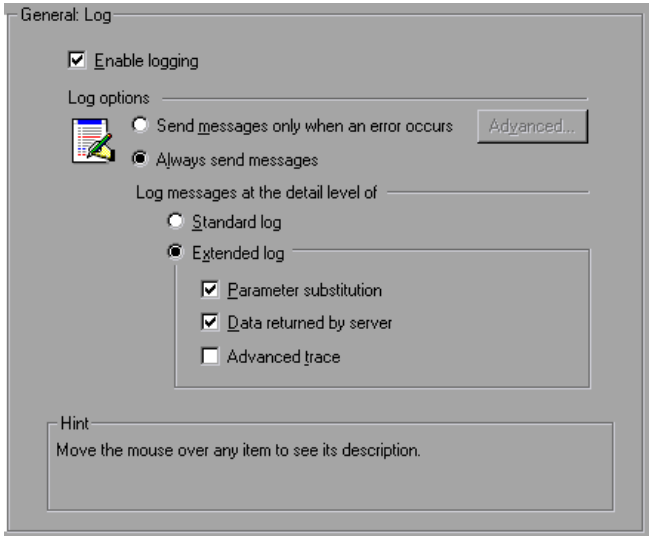
Configuring the Log Run-Time Settings

During execution, Vusers log information about themselves and their communication with the server. In a Windows environment, this information is stored in a file called *output.txt* in the script directory. In UNIX environments, the information is directed to the standard output. The log information is useful for debugging purposes.

The Log run-time settings let you determine how much information is logged to the output. You can select **Standard** or **Extended** log, or you can disable logging completely. Disabling the log is useful when working with many Vusers. If you have tens or hundreds of Vusers logging their run-time information to disk, the system may work slower than normal. During development, enable logging so that you will have information about the replay. You should only disable logging after verifying that the script is functional.

Note: You can program a Vuser script to send messages to an output log by using the **lr_error_message** and **lr_output_message** functions.

Click the **Run-Time Settings** button on select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Select the **General:Log** node to display the log options.



Enable Logging

This option enables automatic logging during replay—VuGen writes log messages that you can view in the Execution log. This option only affects automatic logging and log messages issued through **lr_log_message**. Messages sent manually, using **lr_message**, **lr_output_message**, and **lr_error_message**, are still issued.

Log Options

The Log run-time settings allows you to adjust the logging level depending on your development stage.

You can indicate when to send log messages to the log: **Send messages only when an error occurs** or **Always send messages**. During development, you can enable all logging. Once you debug your script and verify that it is functional, you can enable logging for errors only.

If you choose to send messages only when errors occur, also known as JIT, (Just in Time) messaging, you can set an advanced option, indicating the size of the log cache. See "Setting the Log Cache Size" on page 61.

Setting the Log Detail Level

You can specify the type of information that is logged, or you can disable logging altogether.

Note: If you set **Error Handling** to "Continue on error" in the **General Run-Time Settings** folder, error messages are still sent to the Output window.

If you modify the script's Log Detail Level, the behavior of the **lr_message**, **lr_output_message**, and **lr_log_message** functions will not change—they will continue to send messages.

- **Standard Log**. Creates a standard log of functions and messages sent during script execution to use for debugging. Disable this option for large load testing scenarios or profiles.

  If the logging level is set to **Standard**, the logging mode is automatically set to **JIT logging** when adding it to a scenario or profile. If, however, the logging mode was disabled or set to **Extended**, then adding the script to a scenario or profile will not affect its logging settings.

- **Extended Log.** Creates an extended log, including warnings and other messages. Disable this option for large load testing scenarios or profiles.

You can specify which additional information should be added to the extended log using the Extended log options:

- **Parameter substitution.** Select this option to log all parameters assigned to the script along with their values. For more information on parameters, see Chapter 20, "Working with VuGen Parameters."
- **Data returned by server.** Select this option to log all of the data returned by the server.

➤ **Advanced trace.** Select this option to log all of the functions and messages sent by the Vuser during the session. This option is useful when you debug a Vuser script.

The degree to which VuGen logs events (Standard, Parameter substitution, and so forth) is also known as the *message class*. There are five message classes: Brief, Extended, Parameters, Result Data, and Full Trace.

You can manually set the message class within your script using the **lr_set_debug_message** function. This is useful if you to want to receive debug information about a small section of the script only.

For example, suppose you set Log run-time settings to Standard log and you want to get an Extended log for a specific section of the script. You would then use the **lr_set_debug_message** function to set the Extended message class at the desired point in your script. You must call the function again to specify what type of extended mode (Parameter, Result Data, or Full Trace). Return to the Standard log mode by calling **lr_set_debug_message**, specifying Brief mode. For more information about setting the message class, see the *Online Function Reference* (**Help** > **Function Reference**).

Setting the Log Cache Size

The Advanced options for the Log Run-Time settings, let you indicate the size of the log cache. The log cache stores raw data about the test execution, to make it available should an error occur. When the contents of the cache exceed the specified size, it deletes the oldest items. The default size is 1KB.

The following is the sequence of the logging:

1. You indicate to VuGen to log messages only when an error occurs, by selecting **Send messages only when an error occurs**.
2. VuGen stores information about the test execution in the log cache without writing it to a file. If this information exceeds 1 KB, it overwrites the oldest data. The Execution Log tab also remains empty, since it is a dump of the log file's contents.
3. When an error occurs (either an internal error or a programmed error using **lr_error_message**), VuGen places the contents of the cache into the log file and Execution Log tab. This allows you to see the events that led up to the error.

When an error occurs and VuGen dumps its stored cache into the log file, the actual file size will be greater than the cache size. For example, if your cache size is 1KB, the log file size may be 50 KB. This is normal and only reflects the overhead required for formatting the raw data into meaningful sentences.

Note that in JIT mode, the output of **lr_message** and **lr_log_message**, are only sent to the Output window or log file, if their output was in the log cache at the time of the error. Check the Execution Log for the specified message strings.

Logging CtLib Server Messages

When you run a CtLib Vuser script, (Sybase CtLib, under the Client Server type protocols), all messages generated by the CtLib client are logged in the standard log and in the output file. By default, server messages are not logged. To enable logging of server messages (for debugging purposes), insert the following line into your Vuser script:

```
LRD_CTLIB_DB_SERVER_MSG_LOG;
```

VuGen logs all server messages in the Standard log.

To send the server messages to the output (in addition to the Standard log), type:

```
LRD_CTLIB_DB_SERVER_MSG_ERR;
```

To return to the default mode of not logging server errors, type the following line into your script:

```
LRD_CTLIB_DB_SERVER_MSG_NONE;
```

Note: Activate server message logging for only a specific block of code within your script, since the generated server messages are long and the logging can slow down your system.

Configuring the Think Time Settings

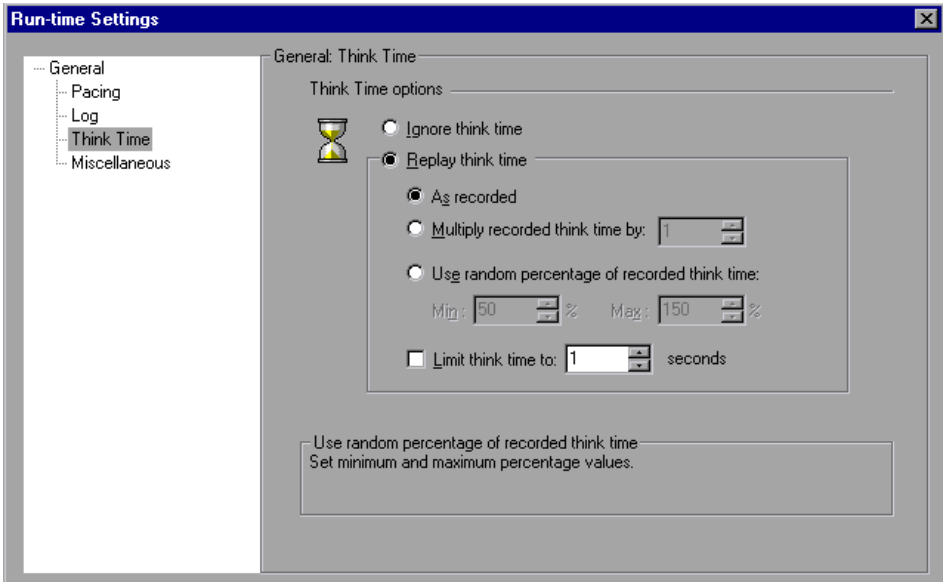
Vuser *think time* emulates the time that a real user waits between actions. For example, when a user receives data from a server, the user may wait several seconds to review the data before responding. This delay is known as the *think time*. VuGen uses **lr_think_time** functions to record think time values into your Vuser scripts. The following recorded function indicates that the user waited 8 seconds before performing the next action:

```
lr_think_time(8);
```

When you run the Vuser script and the Vuser encounters the above **lr_think_time** statement, by default, the Vuser waits 8 seconds before performing the next action. You can use the Think Time run-time settings to influence how the Vuser uses the recorded think time when you run the script.

For more information about the **lr_think_time** function and how to modify it manually, see the *Online Function Reference* (**Help > Function Reference**).

Click the **Run-Time Settings** button on the VuGen toolbar or select **Vuser** > **Run-Time Settings**. Select the **General:Think Time** node to display the Think Time options:



Think Time Options

By default, when you run a Vuser script, the Vuser uses the think time values that were recorded into the script during the recording session. VuGen allows you to use the recorded think time, ignore it, or use a value related to the recorded time:

- **Ignore think time.** Ignore the recorded think time—replay the script ignoring all **lr_think_time** functions.
- **Replay the think time.** The second set of think times options let you use the recorded think time:
  - **As recorded.** During replay, use the argument that appears in the lr_think_time function. For example, lr_think_time(10) waits ten seconds.

- **Multiply recorded think time by.** During replay, use a multiple of the recorded think time. This can increase or decrease the think time applied during playback. For example, if a think time of four seconds was recorded, you can instruct your Vuser to multiply that value by two, for a total of eight seconds. To reduce the think time to two seconds, multiply the recorded time by 0.5.
- **Use random percentage of the recorded think time.** Use a random percentage of the recorded think time. You set a range for the think time value by specifying a range for the think time. For example, if the think time argument is 4, and you specify a minimum of 50% and a maximum of 150%, the lowest think time can be two (50%) and the highest value six (150%).
- **Limit think time to.** Limit the think time's maximum value.

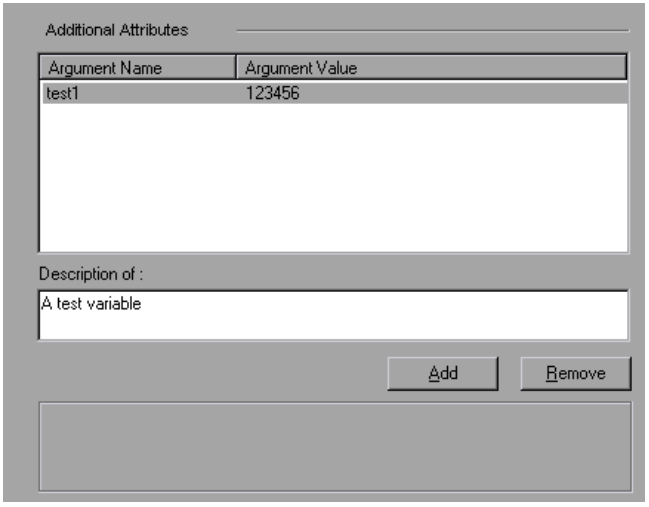
Configuring Additional Attributes Run-Time Settings

You can use the Additional Attributes node to provide additional arguments for a Vuser script. The Additional Attributes settings apply to all Vuser script types.

You specify command line arguments that you can retrieve at a later point during the test run, using **lr_get_attrib_string**. Using this node, you can pass external parameters to prepared scripts.

To set additional attributes:

1 Click the Run-Time Settings button or select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Select the **General:Additional Attributes** node from the tree in the left pane.



2 Click **Add** to add a new command line argument entry. Enter the argument name and its value.

3 Click **Remove** to remove the selected argument.

Configuring Miscellaneous Run-Time Settings

You can set the following Miscellaneous run-time options for a Vuser script: Note that the Multithreading and Automatic Transaction options are not applicable to HP Business Availability Center.

- Error Handling
- Multithreading
- Automatic Transactions

Click the **Run-Time Settings button** or select **Vuser** > **Run-Time Settings** to display the Run-Time Settings dialog box. Select the **General:Miscellaneous** node from the tree in the left pane.

The *Miscellaneous* settings apply to all Vuser script types.

Error Handling

- **Continue on Error.** This setting instructs Vusers to continue script execution when an error occurs. This option is turned off by default, indicating that the Vuser will exit if an error occurs.
- **Fail open transactions on lr_error_message.** This option instructs VuGen to mark all transactions in which an **lr_error_message** function was issued, as *Failed*. The **lr_error_message** function is issued through a programmed *If* statement, when a certain condition is met.
- **Generate Snapshot on Error.** This option generates a snapshot when an error occurs. You can see the snapshot by viewing the Vuser Log and double-clicking on the line at which the error occurred.

It is not recommended to enable both the **Continue on Error** and **Generate Snapshot on Error** options in a load test environment. This configuration may adversely affect the Vusers' performance.

Error Handling for Database Vusers

When working with database protocols (LRD), you can control error handling for a specific segment of a script. To mark a segment, enclose it with LRD_ON_ERROR_CONTINUE and LRD_ON_ERROR_EXIT statements. The Vuser applies the new error setting to the whole segment. If you specify Continue on Error, VuGen issues a messages indicating that it encountered an error and is ignoring it.

For example, if you enable the Continue on Error feature and the Vuser encounters an error during replay of the following script segment, it continues executing the script.

```
lrd_stmt(Csr1, "select…"…);
lrd_exec(…);
```

To instruct the Vuser to continue on error for the entire script except for a specific segment, select the Continue on Error option and enclose the segment with LRD_ON_ERROR_EXIT and LRD_ON_ERROR_CONTINUE statements:

```
LRD_ON_ERROR_EXIT;
lrd_stmt(Csr1, "select…"…);
lrd_exec(…);
LRD_ON_ERROR_CONTINUE;
```

In addition to the LRD_ON_ERROR statements, you can control error handling using *severity levels*. LRD_ON_ERROR statements detect all types of errors—database related, invalid parameters, and so on. If you want the Vuser to terminate only when a database operation error occurs (Error Code 2009), you can set a function's severity level. All functions that perform a database operation use severity levels, indicated by the function's final parameter, *miDBErrorSeverity*.

VuGen supports the following severity levels:

Definition	Meaning	Value
LRD_DB_ERROR_SEVERITY_ERROR	Terminate script execution upon database access errors. (default)	0
LRD_DB_ERROR_SEVERITY_WARNING	Continue script execution upon database access errors, but issue a warning.	1

For example, if the following database statement fails (e.g. the table does not exist), the script execution terminates.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1, 1, 0);
```

To instruct VuGen to continue script execution, even when a database operation error occurs, change the statement's severity level from 0 to 1.

```
lrd_stmt(Csr1, "insert into EMP values ('Smith',301)\n", -1, 1, 1, 1);
```

Note: When you enable Continue on Error, it overrides the "0" severity level; script execution continues even when database errors occur. However, if you disable Continue on Error, but you specify a severity level of "1", script execution continues when database errors occur.

Error Handling for RTE Vusers

When working with RTE Vusers, you can control error handling for specific functions. You insert an **lr_continue_on_error(0);** statement before the function whose behavior you want to change. The Vuser uses the new setting until the end of the script execution or until another **lr_continue_on_error** statement is issued.

For example, if you enable the Continue on Error feature and the Vuser encounters an error during replay of the following script segment, it continues executing the script.

```
TE_wait_sync();
TE_type(...);
```

To instruct the Vuser to continue on error for the entire script, except for the following segment, select the Continue on Error option and enclose the segment with lr_continue_on_error statements, using 0 to turn off Continue on Error and 1 to turn it back on:

```
lr_continue_on_error(0);
TE_wait_sync();
lr_continue_on_error(1);
....
```

Multithreading

Vusers support multithread environments. The primary advantage of a multithread environment is the ability to run more Vusers per load generator. Only threadsafe protocols should be run as threads. (not applicable to HP Business Availability Center)

Note: The following protocols are not threadsafe: Sybase-Ctlib, Sybase-Dblib, Informix, Tuxedo, and PeopleSoft-Tuxedo.

- To enable multithreading, click **Run Vuser as a thread**.
- To disable multithreading and run each Vuser as a separate process, click **Run Vuser as a process**.

The Controller uses a driver program (such as *mdrv.exe* or *r3vuser.exe*) to run your Vusers. If you run each Vuser as a process, then the same driver program is launched (and loaded) into the memory again and again for every instance of the Vuser. Loading the same driver program into memory uses up large amounts of RAM (random access memory) and other system resources. This limits the numbers of Vusers that can be run on any load generator.

Alternatively, if you run each Vuser as a thread, the Controller launches only one instance of the driver program (such as *mdrv.exe*), for every 50 Vusers (by default). This driver process/program launches several Vusers, each Vuser running as a thread. These threaded Vusers share segments of the memory of the parent driver process. This eliminates the need for multiple re-loading of the driver program/process saves much memory space, thereby enabling more Vusers to be run on a single load generator.

Automatic Transactions

You can instruct LoadRunner (not applicable to HP Business Availability Center) to handle every step or action in a Vuser script as a transaction. This is called using automatic transactions. LoadRunner assigns the step or action name as the name of the transaction. By default, automatic transactions per action are enabled.

- To disable automatic transactions per action, clear the **Define each action as a transaction** check box. (enabled by default)
- To enable automatic transactions per step, check the **Define each step as a transaction** check box. (disabled by default)

If you disable automatic transactions, you can still insert transactions manually during and after recording. For more information on manually inserting transactions, see Chapter 5, "Enhancing Vuser Scripts."

Note: If you require the Vusers to generate breakdown data for diagnostics (J2EE) during the scenario run, do not use automatic transactions. Instead, manually define the beginning and end of each transaction.

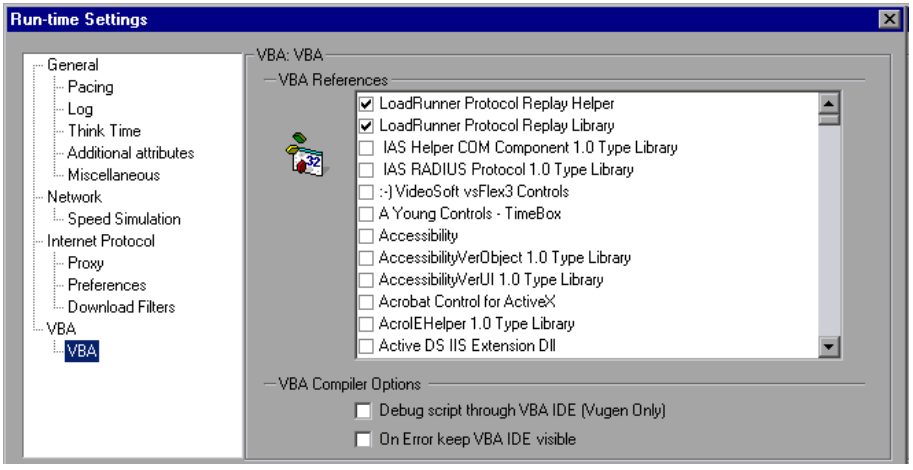
Setting the VB Run-Time Settings

Before running your Visual Basic script, you indicate which libraries to reference during replay. VuGen displays a list of all of the libraries stored on the machine.

You use the Run-Time Settings dialog box to display and configure the run-time settings. To display the Run-Time Settings dialog box, click the **Run-Time Settings** button on the VuGen toolbar.

To set the VBA Run-Time settings:

1 Open the **Run-Time Settings** dialog box and select the **VBA:VBA** node.



2 In the **VBA References** section, select the reference library that you want to use while running the script. Select a library to display its description and version in the bottom of the dialog box.

3 Select the appropriate compiler options:

Select **Debug script through VBA IDE** to enable debugging through the Visual Basic IDE (Integrated Development Environment).

Select **On Error keep VBA IDE visible** to keep the Visual Basic IDE visible during script execution.

4 Select **OK** to apply the run-time settings.

3

Running Vuser Scripts in Standalone Mode

After you develop a Vuser script and set its run-time settings, you test the Vuser script by running it in stand-alone mode.

This chapter includes:

About Running Vuser Scripts in Standalone Mode

You run a script as a standalone test to check its basic functionality.

Once you run a script in standalone mode, you can enhance and customize it:

- with Vuser functions (see the *Online Function Reference*).
- with parameters (see Chapter 20, "Working with VuGen Parameters")

The above steps are optional and may not apply to all scripts.

Running a Vuser Script in VuGen

After developing a Vuser script, run it using VuGen to verify that it executes correctly. You can set several options for replay.

Configuring Replay Options

You can run a Vuser script in animated mode or non-animated mode. When you run in animated mode, VuGen highlights the line of the Vuser script being executed at the current time. You can set a delay for this mode, allowing you to better view the effects of each step. When you run in non-animated mode, VuGen executes the Vuser script, but does not indicate the line being executed.

- **Animated run delay.** The time delay in milliseconds between commands. The default delay value is 0.
- **Only animate functions in Actions sections.** Only animates the content of the Action sections, but not the *init* or *end* sections.
- **Prompt for results directory.** Prompts you for a results directory before running a script from VuGen. If this option is not selected, VuGen automatically names the directory *result1*. Subsequent script executions will automatically overwrite previous ones unless you specify a different result file. Note that results are stored in a subdirectory of the script.
- **After replay show.** Instructs VuGen how to proceed after the replay:
  - **View before replay.** Return to the view you had before replay.
  - **Replay summary.** Go directly to the Replay Summary window in the Workflow Wizard.
  - **Visual Test Results**. Open the Test Results Summary. (This is the same as choosing **View** > **Test Results** after replay.)

Use Defaults

When you click **Use Defaults**, VuGen resets the original values:

Animated run delay to 0 msec.

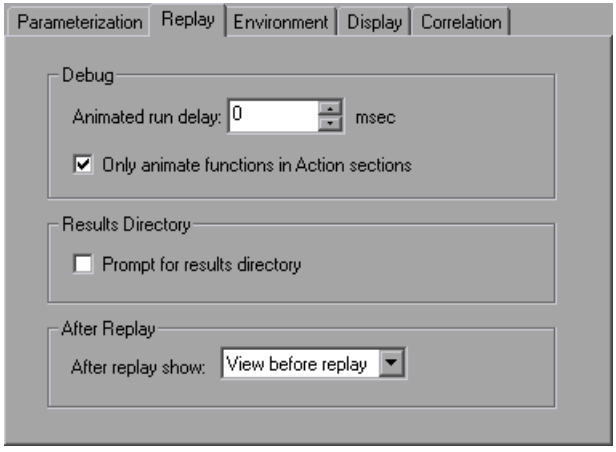
Only animate functions in action sections becomes enabled.

Prompt for results directory becomes disabled.

After replay show is set to View before replay.

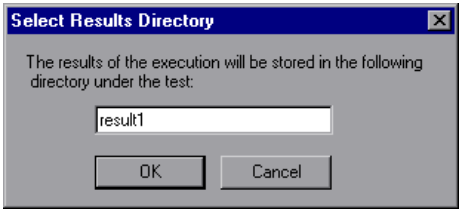
To enable animation and set its properties:

1. Select **View** > **Animated Run** to run in animated mode. VuGen places a check mark beside the **Animated Run** menu option to enable animated mode.
2. To set the delay for the animated run, select **Tools** > **General Options**. The General Options dialog box opens.



3. Select the **Replay** tab.
4. In the **Animated run delay** box, specify a delay in milliseconds and click **OK**.
5. Select **Only animate functions in Actions sections** to animate only the content of the Action sections.

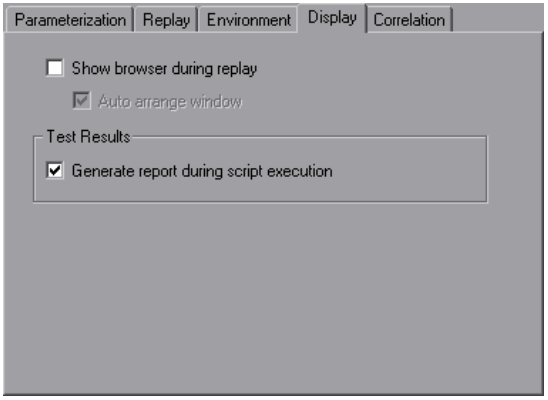
6 Select **Prompt for results directory** to be prompted for a results directory before running a script from VuGen. The Select Results Directory dialog box opens when you click the run command.



7 Type a directory name for the execution results, or accept the default name and click **OK**.

Setting the Display Options

If you are running a Web Vuser script, you can set the Display options (**Tools** > **General Options**). These options specify whether to display VuGen's run-time viewer, whether to generate a report during script execution, and so forth.



- **Show browser during replay.** Enables the run-time viewer. The **Auto arrange window** options instructs VuGen to minimize the run-time viewer when script execution is complete.
- **Generate report during script execution.** Instructs a Vuser to generate a Results Summary report. You can open the report after script execution by selecting **View** > **Test Results**.

By default, the **Show browser during replay** option is disabled, and the **Generate report during script execution** option is enabled. To restore these values, click **Use Defaults**.

For more information on how to use these options for debugging, see "Using VuGen's Debugging Features for Web Vuser Scripts" on page 85.

To set the Display options:

1. Select **Tools** > **General Options** from the VuGen menu. The General Options dialog box opens. Select the **Display** tab.
2. Select **Show browser during replay** to enable the run-time viewer. Select **Auto arrange window** to minimize the run-time viewer when script execution is complete.
3. Select **Generate report during script execution** to instruct a Vuser to generate a Results Summary report. You can open the report after script execution by selecting **View** > **Test Results**.
4. Click **OK** to accept the settings and close the General Options dialog box.

Replaying a Vuser Script

Before you integrate a script into a test or production environment, you run it from VuGen to make sure it is functional. VuGen provides several tools that allow you to monitor the replay and locate any existing and potential problems. These include:

- Viewing the Replay Log
- The Run-Time Data Tab
- The Run Step by Step Command
- Breakpoints
- Bookmarks
- Go To Commands

To replay a script in VuGen:

1 Select **Vuser** > **Run**.

The Output window opens at the bottom of the VuGen main window—or clears, if already open—and VuGen begins executing the Vuser script. In tree view, VuGen runs the Vuser script from the first icon in the script. In Script view, it runs the Vuser script from the first line of the script.

2 Click the Output window's **Replay Log** tab for a log of all of the actions of the Vuser, along with warnings and errors. For more information, see "Viewing the Replay Log" on page 78.

3 To view a summary of the run-time data and the parameters as they are being used, see the Output window's **RunTime Data** tab. For more information, see "The Run-Time Data Tab" on page 79.

4 To hide the Output window during or after a script run, select **View** > **Output Window**. VuGen closes the Output window and removes the check mark from next to **Output Window** on the **View** menu.

5 To interrupt a Vuser script that is running, select **Vuser** > **Pause**, to temporarily pause the script run, or **Vuser** > **Stop**, to end the script run.

Viewing the Replay Log

The Output window's Replay Log displays messages that describe the actions of the Vuser as it runs. This information tells you how the script will run when executed in a scenario or profile.

When script execution is complete, you examine the messages in the Replay Log to see whether your script ran without errors.

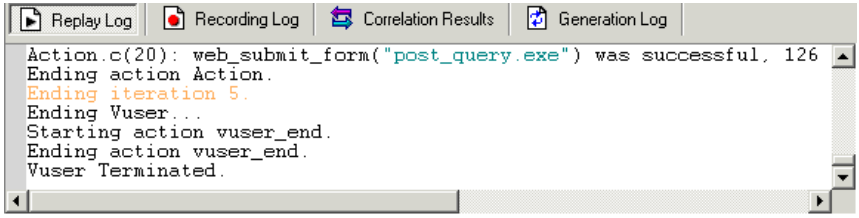
Various colors of text are used in the Replay Log.

- **Black.** Standard output messages.
- **Red.** Standard error messages.
- **Green.** Literal strings, such as URLs, that appear between quotation marks.
- **Blue.** Transaction Information (starting, ending, status and duration).
- **Orange.** The beginning and ending of iterations.

If you double-click on a line beginning with the Action name, the cursor jumps to the step within the script that generated.

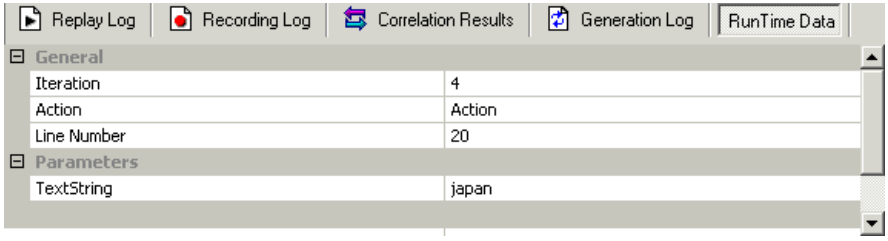
For more information on closing and opening the Output window, see "Replaying a Vuser Script" on page 77.

The following example shows Replay Log messages from a Web Vuser script run.



The Run-Time Data Tab

You can track the script information that becomes updates during replay, using the Run Time Data tab.



During replay, click the rightmost tab, **RunTime Data**. The tab contains two expandable/collapsible sections:

- **General.** The general section shows the current iteration number, the Action name of the currently replayed step, and the line number within the script (Script view).

➤ **Parameters.** The parameters section shows all parameters defined with the script and their substitution values based on the selected update method (sequential, unique, etc.). VuGen shows this information even if the parameter is not used in the script. For more information, see Chapter 20, "Working with VuGen Parameters."

Note that the RunTime Data tab is not accessible after the test run, since it only displays data that changes during replay.

Using VuGen's Debugging Features

VuGen contains two options to help debug Vuser scripts—the Run Step by Step command and breakpoints. These options are not available for VBscript and VB Application type Vusers.

VuGen contains additional features to help debug Web Vuser scripts. For details, see "Using VuGen's Debugging Features for Web Vuser Scripts" on page 85.

To view the Debug toolbar:

Right-click the toolbar area and select **Debug**. The Debug toolbar appears in the toolbar area.

The Run Step by Step Command

The Run Step by Step Command runs the script one line at a time. This enables you to follow the script execution.

To run the script step by step:

1 Select **Vuser** > **Run Step by Step**, or click the **Step** button on the Debug toolbar.

VuGen executes the first line of the script.

2 Continue script execution by clicking the **Step** button until the script run completes.

Breakpoints

Breakpoints pause execution at specific points in the script. This enables you to examine the effects of the script on your application at pre-determined points during execution. To manage the breakpoints, use the Breakpoint Manager.

To set breakpoints:

1 Place the cursor on the line in the script at which you want execution to stop.

2 Select **Insert** > **Toggle Breakpoint**, or click the **Breakpoint** button in the Debug toolbar. Alternatively, press F9 on the keyboard. The Breakpoint symbol () appears in the left margin of the script.

3 To disable a breakpoint, place the cursor on the line with the breakpoint symbol, and click the **Enable / Disable Breakpoint** button on the Debug toolbar. A white dot inside the Breakpoint symbol indicates a disabled breakpoint. When one breakpoint is disabled, script execution is paused at the following breakpoint. Click the button again to enable the breakpoint.

To remove the breakpoint, place the cursor on the line with the breakpoint symbol, and click the **Breakpoint** button or press F9.

To run the script with breakpoints:

1 Begin running the script as you normally would.

VuGen pauses script execution when it reaches a breakpoint. You can examine the effects of the script run up to the breakpoint, make any necessary changes, and then restart the script from the breakpoint.

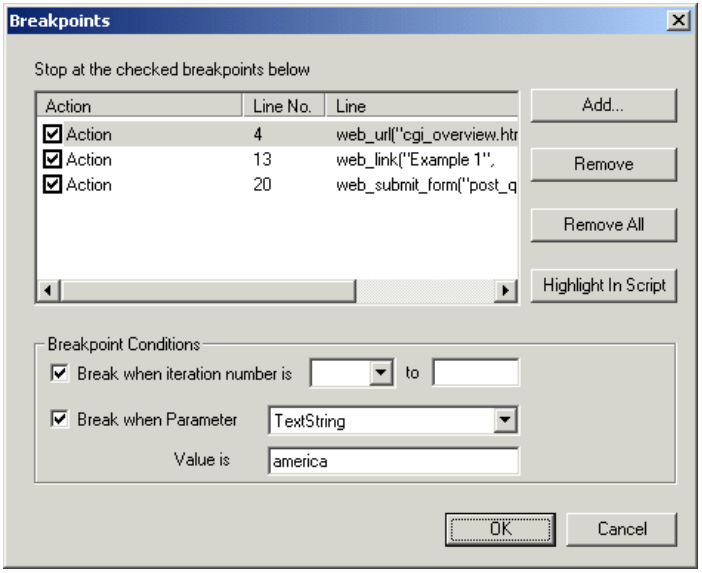
2 To resume execution, select **Vuser** > **Run**.

Once restarted, the script continues until the next breakpoint is encountered or until the script is completed.

The Breakpoint Manager

You can view and manage breakpoints using the Breakpoint Manager. From the Breakpoint Manager you can manipulate all of the breakpoints in your script.

To open the Breakpoint Manager, select **Edit** > **Breakpoints**.



To jump to the breakpoint location in the script:

1 Select a breakpoint from the list.

2 Click **Highlight in Script**. The line in the script becomes highlighted.

Note that you can only highlight one breakpoint at a time.

Managing Breakpoints

From the Breakpoint Manager, you can add, remove, disable, or conditionalize a breakpoint.

To add a breakpoint:

1. Click **Add**. The Add Breakpoint dialog box opens.
2. Select an **Action** and specify the **Line** number where you want add the breakpoint.
3. Click **OK.** The Breakpoint is added to the list of breakpoints.

To remove a breakpoint:

1. To remove a single breakpoint, select the breakpoint and click **Remove**.
2. To remove all the breakpoints at once, click **Remove All**.

To enable/disable a breakpoint:

1. To enable a breakpoint, in the Action column, select the action's check box.
2. To disable a breakpoint, in the Action column, clear the action's check box.

From the Breakpoint Manager, you can set breakpoints to pause execution under certain conditions.

To conditionalize a breakpoint:

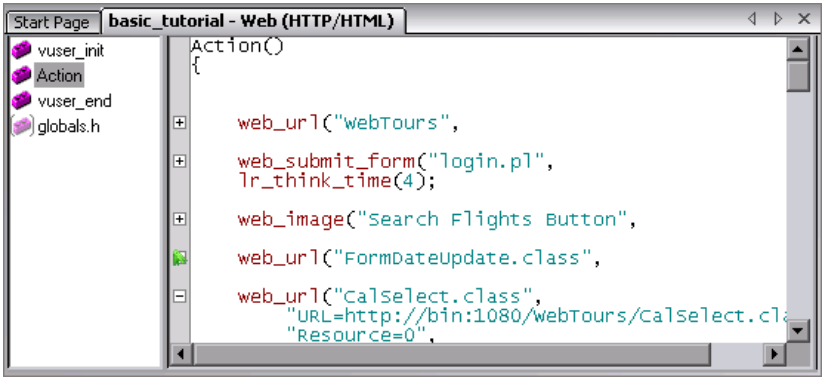
1. To pause the script after a specific number of iterations, select **Break when iteration number is** and enter the desired number.
2. To pause the script when parameter *X* has a specific value, select **Break when Parameter** *X* **Value is** and enter the desired value. For more information about parameters, see Chapter 20, "Working with VuGen Parameters."

Bookmarks

When working in Script view, VuGen lets you place bookmarks at various locations within your script. You can navigate between the bookmarks to analyze and debug your code.

To create a bookmark:

1 Place the cursor at the desired location and press Ctrl + F2. VuGen places an icon in the left margin of the script.



2 To remove a bookmark, click on the desired bookmark and press Ctrl + F2. VuGen removes the bookmark icon from the left margin.

3 To move between bookmarks:

To move to the next bookmark, click F2.

To navigate to the previous bookmark, click Shift + F2.

You can also create and navigate between bookmarks through the **Edit** > **Bookmarks** menu item.

Note: You can only navigate between bookmarks in the current action. To navigate to a bookmark in another action, select that action in the left pane and then press F2.

Go To Commands

To navigate around the script without using bookmarks, you can use the Go To command. Select **Edit** > **Go To Line** and specify the line number of the script. This navigation is also supported in Tree view.

If you want to examine the Replay log messages for a specific step or function, select the step in VuGen and select **Edit** > **Go To Step in Replay Log.** VuGen places the cursor at the corresponding step in the Output window's Replay Log tab.

Using VuGen's Debugging Features for Web Vuser Scripts

VuGen provides two additional tools to help you debug Web Vuser scripts—the run-time viewer (online browser) and the Results Summary report.

- You can instruct VuGen to display a run-time viewer when you run a Web Vuser script. The run-time viewer was developed specifically for use with VuGen—it is unrelated to the browser that you use to record your Vuser scripts. The run-time viewer shows each Web page as it is accessed by the Vuser. This is useful when you debug Web Vuser scripts because it allows you to check that the Vuser accesses the correct Web pages.
- You can specify whether or not a Web Vuser generates a Results Summary report during script execution. The Results Summary report summarizes the success or failure of each step in the Web Vuser scripts and allows you to view the Web page returned by each step. For additional details on working with the Results Summary report, select **View** > **Test Results** and click F1 to open the online help.

For information on setting the above Display options, see "Setting the Display Options" on page 76.

Note: Transaction times may be increased when a Vuser generates a Results Summary report.

Vusers can generate Results Summary reports only when run from VuGen. When you run a script from the Controller or Business Process Monitor, Vusers do not generate reports.

Working with VuGen Windows

You can show and rearrange VuGen's windows to view the relevant data for your script, using the following features:

- **Show/Hide the Output Window.** Select **View** > **Output Window** to show and hide the Output window below the VuGen script editor. The Output window has several tabs, depending on the protocol. The most common tabs are the Replay Log, Recording Log, Generation Log, and Correlation Results. For more information, see "Viewing the Replay Log" on page 78.
- **Display All Thumbnails.** Select **View** > **Show All Thumbnails** to show all of the script's steps as thumbnails. To show thumbnails for primary steps only, clear this option. For more information, see "Viewing Script Thumbnails" on page 32.
- **Display Grids.** Select **View** > **Enable Data Grids** to enable grid display of the data in the protocols that support data grids (Database, COM, and Microsoft . NET). The grids appear inside the script.
- **Window Manipulation.** Select **Window** > **Close All** to close all of the open scripts. If necessary, VuGen will prompt you to save the unsaved scripts.

Find In Files

You use Find In Files to search for any string or expression in all the files of the script you currently have open.

For example, if VuGen fails to replay a script due to an error in the replay log, you can search through all the files in the script simultaneously for a specific value that you think may be causing the failure.

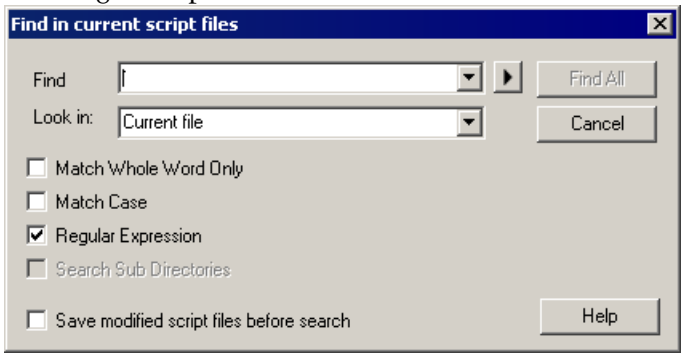
Vugen displays all matches in a separate results window. You double-click on any line in the window to open the relevant file.

Note: VuGen also includes a regular **Find** feature. With this feature, you can search for values in only one file at a time. The value is highlighted in the script itself, and you press F3 to move to the next match.

To search with Find In Files:

1 Open a script in VuGen.

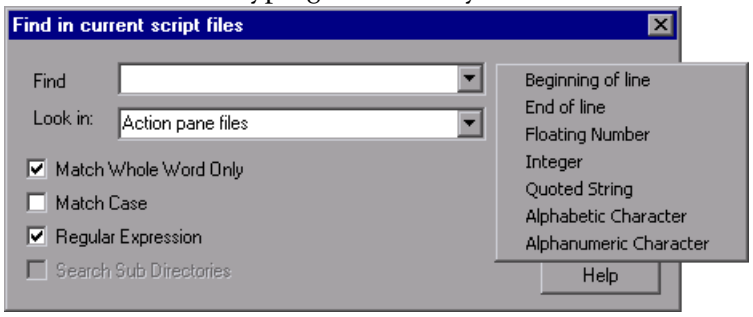
2 Select **Edit** > **Find in current script files.** The Find in current script files dialog box opens.



3 In the **Find** box, enter the string you want to search for. You can select any of the previous ten searches from the drop-down list.

You can also search with a regular expression. A regular expression is a search string made up of a single character, or a set of characters, that is interpreted by VuGen as a pattern in the script. VuGen searches the script for values that match that pattern.

If you select **Regular Expression**, the arrow button by the **Find** box is activated, and provides seven common regular expressions that you can select instead of typing in manually.



4 In the **Look in** box, specify the script directory in which to search. You can also select from the two options provided by VuGen in the drop-down list.

- **Current file.** Searches the file currently displayed in the right pane.
- **Action pane files.** Searches all files that appear in the left pane.

If you specify a script directory other than those provided, the **Search Sub Directories** option is activated. You select this option to expand the search to sub directories of the current Vuser folder.

5 Select the desired settings from: **Match Whole Word Only** and **Match Case**.

6 Click **Find All**. The Search Results tab opens.

Note: We recommend that you save the script before you search. If you make changes in the results, you might not be able to return to the original script. Note that Find In Files only searches the last saved version of the script. The **Save modified script files before search** option appears if you have already made changes to the script.

Running Web Service Scripts from the Command Prompt

You can run a Web Services script from a command prompt or from the Windows Run dialog box. This feature allows you to bypass the Service Test UI while allowing you to configure display parameters and log settings. This is beneficial for running Service Test scripts in automated batch files.

When running Vuser scripts from the command line, you can specify several types of arguments. The only mandatory one is the location of the Replay Log.

To run a Vuser script from the command line use the following command format:

```
ServiceTestRunner "<test path>" [arguments]
```

Where **test path** is the path to the .usr file which contains the Vuser script and **arguments** are any number of arguments from the table below.

Argument Name	Mandatory	Syntax
Log Target Path	yes	**/log "<path.log>"**
Parameter File	no	**/params or /p "<param_1.dat>; <param_2.dat>..."**
Silent Output	no	**/silent**
Delimiter	no	**/del "<delimiter>"**
Append	no	**/a**
Open Report	no	**/show**

- **Log Target Path.** The path to a .log file in which to store the Replay Log (mandatory). This file is automatically generated when you run a Vuser script.
- **Parameter File.** The path to one or more .dat files containing parameter values. If your script uses several parameter files, you can specify multiple parameter files, separated by semicolons. Service Test uses these values for the command line invocation, and restores the original values after the test run. The parameter file names must be identical to the original ones. You can only use parameter arguments to replace existing parameters, but not to create new ones.
- **Silent Output.** Turns off the console display during run-time. By default, Service Test automatically displays the Replay Log on the console, in addition to storing it in the log file.
- **Delimiter.** The EOL (End of Line) delimiter. This option allows advanced users to configure the results log file for use with external programs. If undefined, the default CR is used.
- **Append.** Appends the new log data to the existing log file instead of overwriting. By default, Service Test overwrites the old data.
- **Open Report.** Automatically displays the Results Summary report upon completion of the test.

In the following example, Service Test runs a Vuser script located in **C:/example_path/example.usr**. It stores the Replay Log to **C:/example_path/example.log** and appends the data to any existing data in that file. It uses the parameter files **param1.dat** and **param2.dat**. The **/silent** argument prevents the Replay Log from being displayed automatically. The end of line delimiter is set to **EOL**.

```
ServiceTestRunner “C:/example_path/example.usr” /log “C:/example_path/example.log
/p "C:/example_path/param1.dat;C:/example_path/param2.dat" /silent /a /del “EOL”
```

The command line invocation returns a 1 for success and 0 when it fails.

License Check

The machine upon which you run the command line string, must have a valid Service Test license. When you run a script from the command line, the console checks the license. If it is not current, the console issues an error message.

For information about the different types of licenses and how to install them, see the *HP Service Test Installation Guide*.

4

Viewing Test Results

To assist with debugging a Vuser script, you can view a report that summarizes the results of your script run. Service Test generates the report during the Vuser script execution and you view the report when script execution is complete.

This chapter includes:

About Viewing Test Results

After you run a script, the Test Results window displays all aspects of the test run and can include:

- a high-level results overview report (pass/fail status)
- the data used in all runs
- an expandable tree of the steps, specifying exactly where application failures occurred
- the exact locations in the script where failures occurred
- a still image of the state of your application at a particular step
- a movie clip of the state of your application at a particular step or of the entire test
- detailed explanations of each step and checkpoint pass or failure, at each stage of the test

Reports for Web Services Vusers contain several enhacements, such as breakdown by operations, checkpoint results, and a view of the HTTP traffic. For more information, see "Viewing Web Services Reports" on page 111.

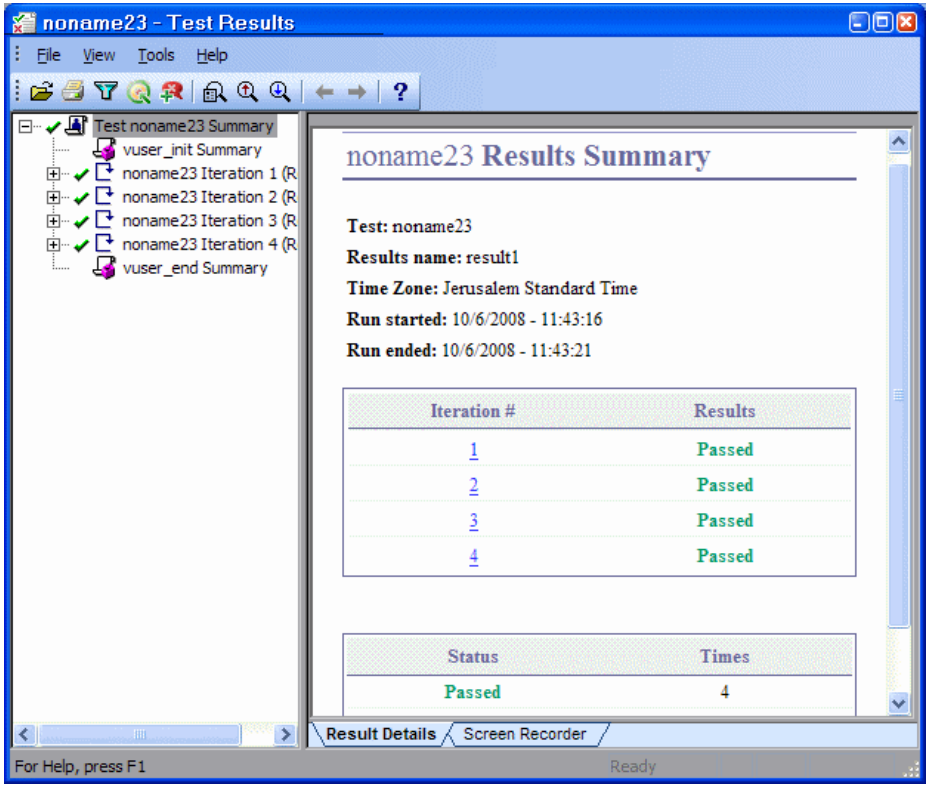
You can open the Test Results window as a standalone application from the **Start** menu. To open the Test Results window, choose **Start** > **All Programs** > **HP Service Test** > **Test Results Viewer**.

Note: The Test Results window can show results with up to 300 levels in the tree hierarchy. If you have results with more than 300 nested levels, you can view the entire report by manually opening the **results.xml** file.

The Test Results Window

After a run session, you view the results in the Test Results window.

- The left pane displays the report tree—a graphical representation of the results. In the report tree, a green check mark represents a successful step, and a red X represents a failed step.
- The right pane displays the report details—an overall summary of the script run, as well as additional information for a selected branch of the report tree.



The Test Results window contains the following key elements:

- **Test results title bar.** Displays the name of the test.
- **Menu bar.** Displays menus of available commands.
- **Run results toolbar.** Contains buttons for viewing test results (choose **View** > **Test Results Toolbar** to display the toolbar). For more information, see "Test Results Toolbar" on page 95.
- **Run results tree.** Contains a graphic representation of the test results in the run results tree. The run results tree is located in the left pane of the Test Results window. For more information, see "Run Results Tree" on page 94.
- **Result Details tab.** Contains details of the selected node in the run results tree. The Result Details tab is located in the right pane of the Test Results window. For more information, see "Run Results Details" on page 95.
- **Screen Recorder tab.** Contains the recorded movie associated with the test results. The screen recorder tab is located in the right pane of the Test Results window This is supported for scripts created with HP QuickTest Professional.
- **Status bar.** Displays the status of the currently selected command (choose **View** > **Status Bar** to view the status bar).

You can change the appearance of the Test Results window. For more information, see "Changing the Appearance of the Test Results Window" on page 96.

Run Results Tree

The left pane in the Test Results window displays the **run results tree**—a graphical representation of the test results:


- indicates a step that succeeded.


- indicates a step that failed.

- indicates a warning, meaning that the step did not succeed, but it did not cause the test to fail.


- indicates a step that failed unexpectedly.

You can collapse or expand a branch in the run results tree to change the level of detail that the tree displays.

Run Results Details

By default, when the Test Results window opens, it displays a summary of the script run in the **Result Details** tab in the right pane of the window.

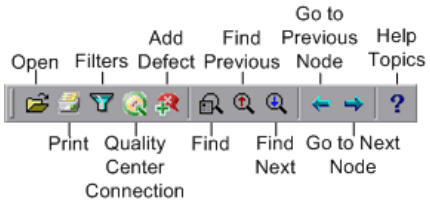
The right pane of the Test Results Window contains tabs labeled **Result Details** and **Screen Recorder**. When you select the top node of the run results tree, the Result Details tab shows a summary of the results for your test. When you select a branch or step in the tree, the Result Details tab contains the details for that step. The Result Details tab may also include a still image of your application for the highlighted step.

The Screen Recorder tab contains the movie associated with your test results. If there is no movie associated with your test results, the Screen Recorder tab contains the message: No movie is associated with the results.

When you select the top node of the results tree in the **Result Details** tab, it indicates the script name, results name, the start and end date and time of the run, the number of iterations, and whether an iteration passed or failed. For checkpoints in Web Services scripts, the possible results are **Passed** or **Failed**. If an iteration does not contain checkpoints, the possible results are **Done** or **Failed**.

Test Results Toolbar

The Test Results toolbar contains buttons for viewing test results.



Changing the Appearance of the Test Results Window

By default, the Test Results window has the same look and feel as the QuickTest window, using the Microsoft Office 2003 theme. You can change the look and feel of the Test Results window, as required.

To change the appearance of the Test Results window:

In the Tests Results window, choose **View > Window Theme**, and then select the way the window should appear from the list of available themes. For example, you can apply a Microsoft Office 2000 or Microsoft Windows XP theme.

Note: You can apply the Microsoft Windows XP theme to the Tests Results window only if your computer is set to use a Windows XP theme.

Viewing the Results

By default, VuGen generates test results and automatically opens them at the end of a run.

To prevent VuGen from generating the results, choose **Tools** > **General Options**, select the **Display** tab and clear the **Generate report during script execution option**.

To indicate whether or not to open the results after running the script, choose **Tools** > **General Options**, select the **Replay** tab, and select a view in the **After Replay** section.

For more information on setting the Display options, see "Running Vuser Scripts in Standalone Mode" in the *Virtual User Generator* User Guide.

In addition, you can view the results of previous runs of the current script, and results of other script. You can also preview results on screen and print them to your default Windows printer, as well as export them to an HTML file.

For more information, see:

➤ "About Viewing Test Results" on page 92

➤ "The Test Results Window" on page 93

➤ "Viewing the Results" on page 96

➤ "Finding Results Steps" on page 104

➤ "Printing Results" on page 105

➤ "Previewing Test Results" on page 106

➤ "Exporting Test Results" on page 108

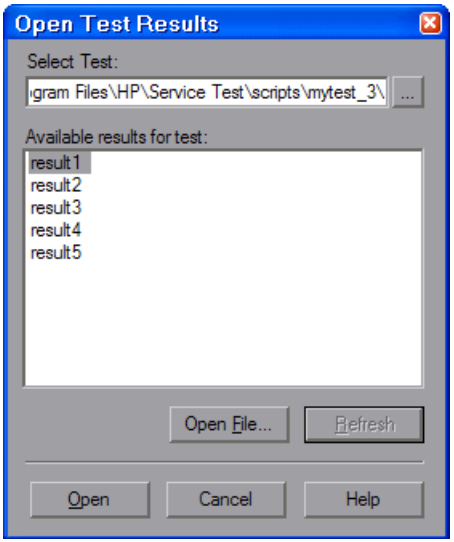
Opening Test Results to View a Selected Run

You can view the saved results for the current script, or you can view the saved results for other scripts.

To select a specific set of test results:

1 Choose **File** > **Open** from within the Test Results window.

2 To view all of the results, specify the parent script path.



3 Select a results set and click **Open**.

Tip: To update the results list after you specify a new path, click **Refresh**.

Searching for Results in the File System

By default, the results saved in the file system are stored in the script folder. If you are connected to Quality Center, the results are stored together with You can search for results in the file system by script or by result file.

To search for results in the file system by script:

1 In the Open Test Results dialog box, enter the path of the folder that contains the results file for your script, or click the browse button to open the Open Test dialog box.

2 Find and highlight the script whose results you want to view, and click **Open**.

3 In the Open Test Results dialog box, highlight the result set you want to view, and click **Open**. The Test Results window displays the selected results.

To search for results in the file system by result file:

1 In the Open Test Results dialog box, click the **Open File** button to open the Select Results File dialog box.

2 Browse to the folder where the results file is stored.

3 Highlight the results (**.xml**) file you want to view, and click **Open**. The Test Results window displays the selected results.

Notes:

➤ By default, result files for tests are stored in <**Script**>\<**ResultName**>.

➤ Results files for earlier versions were saved with a **.qtp** file extension. In the Select Results File dialog box, only results files with an **.xml** extension are shown by default. To view results files with a **.qtp** extension in the Select Results File dialog box, select **Test Results (*.qtp)** in the **Files of type** box.

Searching for Results Saved in Quality Center

If your script is stored in Quality Center, the results are also stored in Quality Center. You cannot change the location of the test results.

To search for test results saved in Quality Center:

1 In the Test Results window, choose **Tools** > **Quality Center Connection** or click the **Quality Center Connection** button and connect to your Quality Center project.

2 In the Open Test Results dialog box, enter the path of the folder that contains the results file for your QuickTest test, or click the browse button to open the Open Test from Quality Center Project dialog box.

3 Select **DB Vuser** in the **Test Type** list.

4 Find and highlight the script whose test results you want to view, and click **OK**.

5 In the Open Test Results dialog box, highlight the test result set you want to view, and click **Open**. The Test Results window displays the selected test results.

For more information on working with Quality Center, see Chapter 5, "Managing Scripts Using Quality Center".

For more information on working with Quality Center, see the Quality Center chapter in the *HP Virtual User Generator* User Guide.

Working in Test Results Tree

The following steps describe how to work within the nodes of the Test Results tree:

1 You can collapse or expand a branch in the run results Tree to select the level of detail that the tree displays.

- To collapse a branch, select it and click the collapse (–) sign to the left of the branch icon, or press the minus key (–) on your keyboard number pad. The details for the branch disappear in the results tree, and the collapse sign changes to expand (+).
- To collapse all of the branches in the run results tree, choose **View** > **Collapse All** or right click a branch and select **Collapse All**.
- To expand a branch, select it and click the expand (+) sign to the left of the branch icon, or press the plus key (+) on your keyboard number pad. The tree displays the details for the branch and the expand sign changes to collapse.

  If you just opened the Test Results window, the tree expands one level at a time. If the tree was previously expanded, it reverts to its former state.
- To expand a branch and all branches below it, select the branch and press the asterisk (*) key on your keyboard number pad.

- To expand all of the branches in the run results tree, choose **View** > **Expand All**; right click a branch and select **Expand All**; or select the top level of the tree and press the asterisk (*****) key on your keyboard number pad.

2 You can view the results of an individual iteration, an action, or a step. When you select a step in the run results tree, the right side of the Test Results window contains the details of the selected step. Depending on your settings in the Run tab of the Options dialog box, the right side of the Test Results window may be split into two panes, with the bottom pane containing a still image (or in selected cases, other data) of the selected step.

The results can be one of the following types:

- Iterations, actions, and steps that contain checkpoints are marked **Passed** or **Failed** in the right part of the Test Results window and are identified by icons in the tree pane.
- Iterations, actions, and steps that ran successfully, but do not contain checkpoints, are marked **Done** in the right part of the Test Results window.
- Steps that were not successful, but did not cause the script to stop running, are marked **Warning** in the right part of the Test Results window and are identified by the icon  or .

3 To filter the information displayed in the Test Results window, click the **Filters** button or choose **View** > **Filters**. The Filters dialog box opens.

The default filter options are displayed in the image above. The Filters dialog box contains the following options:

Iterations area:

- **All.** Displays test results from all iterations.
- **From iteration X to Y.** Displays the test results from a specified range of test iterations.

Status area:

- **Fail.** Displays the results for the steps that failed.
- **Warning.** Displays the results for the steps with the status **Warning** (steps that did not pass, but did not cause the script to fail).
- **Pass.** Displays the results for the steps that passed.
- **Done.** Displays the results for the steps with the status **Done** (steps that were performed successfully but did not receive a pass, fail, or warning status).

Content area:

- **All.** Displays all steps from all nodes in the test.
- **Show only actions.** Displays the action nodes in the test (not the specific steps in the action nodes).

4 To find specific steps within the Test Results, click the **Find** button or choose **Tools** > **Find**. For more information, see "Finding Results Steps."

5 To move between previously selected nodes within the run results tree, click the **Go to Previous Node** or **Go to Next Node** buttons.

6 To view the results of other run sessions, click the **Open** button or choose **File** > **Open.** For more information, see "Opening Test Results to View a Selected Run" on page 97.

7 To print results, click the **Print** button or choose **File** > **Print**. For more information, see "Printing Results" on page 105.
(You can preview the run results before you print them. For more information, see "Previewing Test Results" on page 106.)

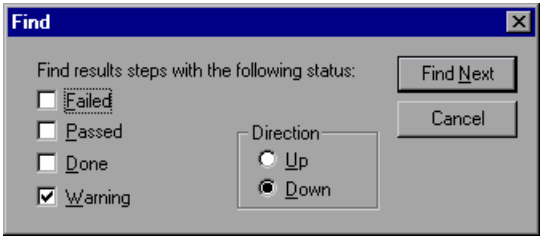
Note: If you have Quality Center installed, you can add a defect to a Quality Center project. For more information, see "Submitting Defects to Quality Center" on page 109.

8 To export the run results to an HTML file, choose **File** > **Export to HTML File**. For more information, see "Exporting Test Results" on page 108.

9 Choose **File** > **Exit** to close the Test Results window.

Finding Results Steps

The Find dialog box enables you to find specified steps such as errors or warnings from within the Test Results. You can select a combination of statuses to find, for example steps that are both **Passed** and **Done**.



The following options are available:

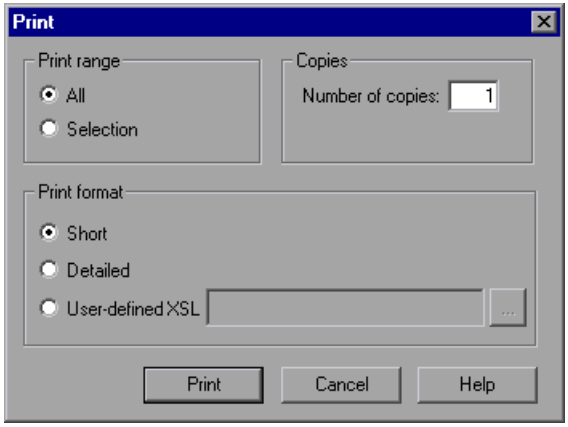
Option	Description
Failed	Finds a failed step in the Test Results.
Warning	Find a step where a warning was issued.
Passed	Finds a passed step in the Test Results.
Done	Finds a step that has finished its run.
Direction	Indicates whether to search up or down within the steps of the Test Results.

Printing Results

You can print results from the Test Results window. You can select the type of report you want to print, and you can also create and print a customized report.

To print the results:

1 Click the **Print** button or choose **File** > **Print**. The Print dialog box opens.



2 Select a **Print range** option:

- **All.** Prints the results for the entire script.
- **Selection.** Prints the results for the selected branch in the run results tree.

3 Specify the **Number of copies** of the results that you want to print.

4 Select a **Print format** option:

- **Short.** Prints a summary line (when available) for each item in the run results tree. This option is only available if you selected **All** in step 2.
- **Detailed.** Prints all available information for each item in the run results tree, or for the selected branch, according to your selection in step 2.
- **User-defined XSL.** Enables you to browse to and select a customized **.xsl** file. You can create a customized **.xsl** file that specifies the information to be included in the printed report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 111.

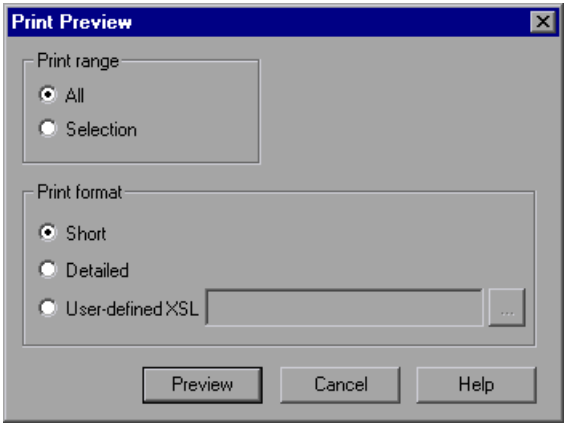
5 Click **Print** to print the selected results information to your default Windows printer.

Previewing Test Results

You can preview results on screen before you print them. You can select the type and quantity of information you want to view, and you can also display the information in a customized format.

To preview the results:

1 Choose **File** > **Print Preview**. The Print Preview dialog box opens.



2 Select a **Print range** option:

- **All.** Previews the results for the entire script.
- **Selection.** Previews results information for the selected branch in the results tree.

3 Select a **Print format** option:

- **Short.** Previews a summary line (when available) for each item in the results tree. This option is only available if you selected **All** in step 2.
- **Detailed.** Previews all available information for each item in the results tree, or for the selected branch, according to your selection in step 2.
- **User-defined XSL.** Enables you to browse to and select a customized **.xsl** file. You can create a customized **.xsl** file that specifies the information to be included in the preview, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 111.

4 Click **Preview** to preview the appearance of your results on screen.

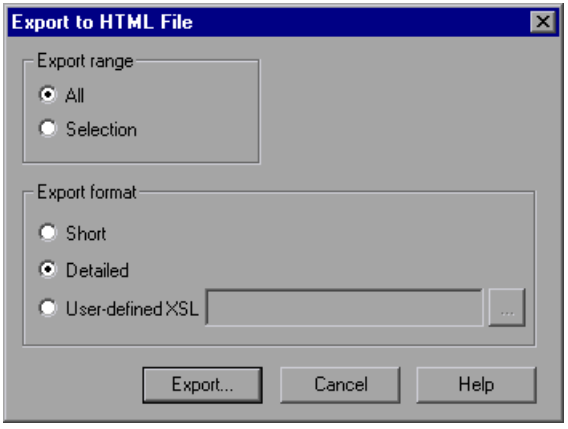
Tip: If some of the information is cut off in the preview, for example, if checkpoint names are too long to fit in the display, click the **Page Setup** button in the Print Preview window and change the page orientation from **Portrait** to **Landscape**.

Exporting Test Results

You can export the results to an HTML file. This enables you to view the results even if the Test Results Viewer environment is unavailable. For example, you can send the file containing the results in an e-mail to a third-party. You can select the type of report you want to export, and you can also create and export a customized report.

To export the results:

1 Choose **File** > **Export to HTML File**. The Export to HTML File dialog box opens.



2 Select an **Export range** option:

- **All.** Exports the results for the entire script.
- **Selection.** Exports result information for the selected branch in the results tree.

3 Select an **Export format** option:

- **Short.** Exports a summary line (when available) for each item in the results tree. This option is only available if you selected **All** in step 2.
- **Detailed.** Exports all available information for each item in the results tree, or for the selected branch, according to your selection in step 2.
- **User-defined XSL.** Enables you to browse to and select a customized **.xsl** file. You can create a customized **.xsl** file that specifies the information to be included in the exported report, and the way it should appear. For more information, see "Customizing the Test Results Display" on page 111.

4 Click **Export**. The Save As dialog box opens, enabling you to change the default destination folder and rename the file, if required. By default, the file is saved in the results folder.

5 Click **Save** to save the HTML file and close the dialog box.

Submitting Defects to Quality Center

When viewing the results, you can submit any defects detected to a Quality Center project directly from the Test Results window.

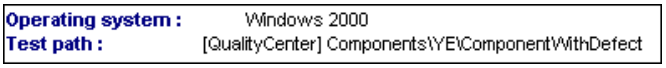
To manually submit a defect to Quality Center:

1 Ensure that the Quality Center client is installed on your computer. (Enter the Quality Center Server URL in a browser and ensure that the Login screen is displayed.)

2 Choose **Tools** > **Quality Center Connection** or click the **Quality Center Connection** button to connect to a Quality Center project. For more information on connecting to a project, see "Connecting to and Disconnecting from Quality Center" on page 118.

3 Choose **Tools** > **Add Defect** or click the **Add Defect** button to open the Add Defect dialog box in the specified Quality Center project. The Add Defect dialog box opens.

4 You can modify the defect information if required. Basic information on the is included in the description:



5 Click **Submit** to add the defect information to the Quality Center project.

6 Click **Close** to close the Add Defect dialog box.

Connecting to Quality Center from the Test Results Window

To manually submit defects to Quality Center from the Test Results window, you must be connected to Quality Center.

The connection process has two stages. First, you connect to a local or remote Quality Center server. This server handles the connections between the Test Results and the Quality Center project.

Next, you log in and choose the project you want to access. The project stores tests and run session information for the application you are testing. Note that Quality Center projects are password protected, so you must provide a user name and a password.

For more information on connecting to a Quality Center project, see "Connecting to and Disconnecting from Quality Center" on page 118.

Customizing the Test Results Display

Each result set is saved in a single **.xml** file (called **results.xml**). This **.xml** file stores information on each of the test result nodes in the display. The information in these nodes is used to dynamically create **.htm** files that are shown in the top-right pane of the Test Results window.

Each node in the run results tree is an element in the **results.xml** file. In addition, there are different elements that represent different types of information displayed in the test results. You can take test result information from the **.xml** file and use XSL to display the information you require in a customized format (either when printing from within the Test Results window, when displaying test results in your own customized results viewer, or when exporting the test results to an HTML file).

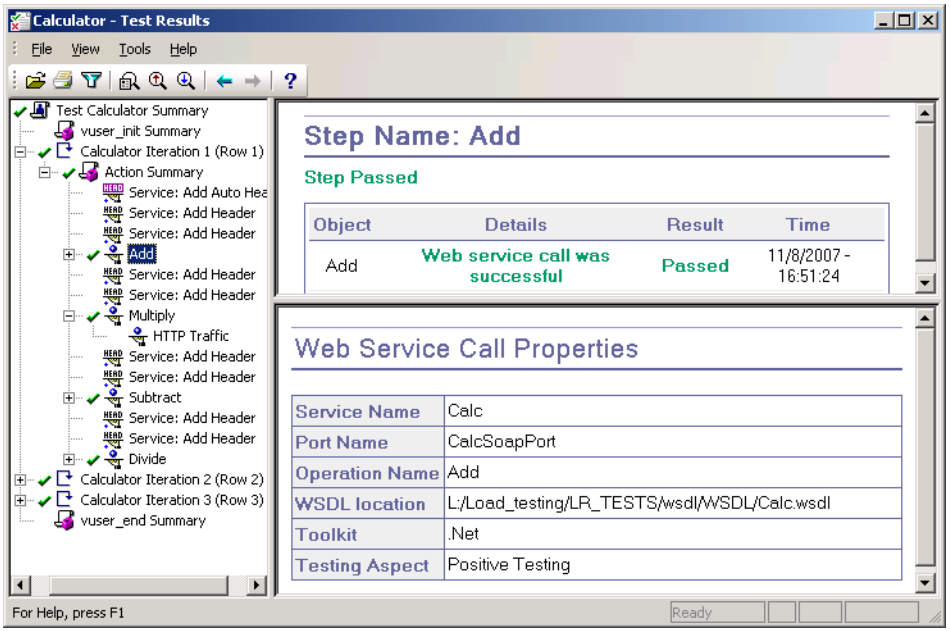
XSL provides you with the tools to describe exactly which test result information to display and exactly where and how to display, print or export it. Using a XSL editor, you can modify the **.css** and **.xsl** files in the results folder, to change the appearance of the report (for example, fonts, colors, and so forth).

For example, in the **results.xml** file, one element tag contains the name of an action, and another element tag contains information on the time at which the run was performed. Using XSL, you could tell your customized editor that the action name should be displayed in a specific place on the page and in a bold green font, and that the time information should not be displayed at all.

Viewing Web Services Reports

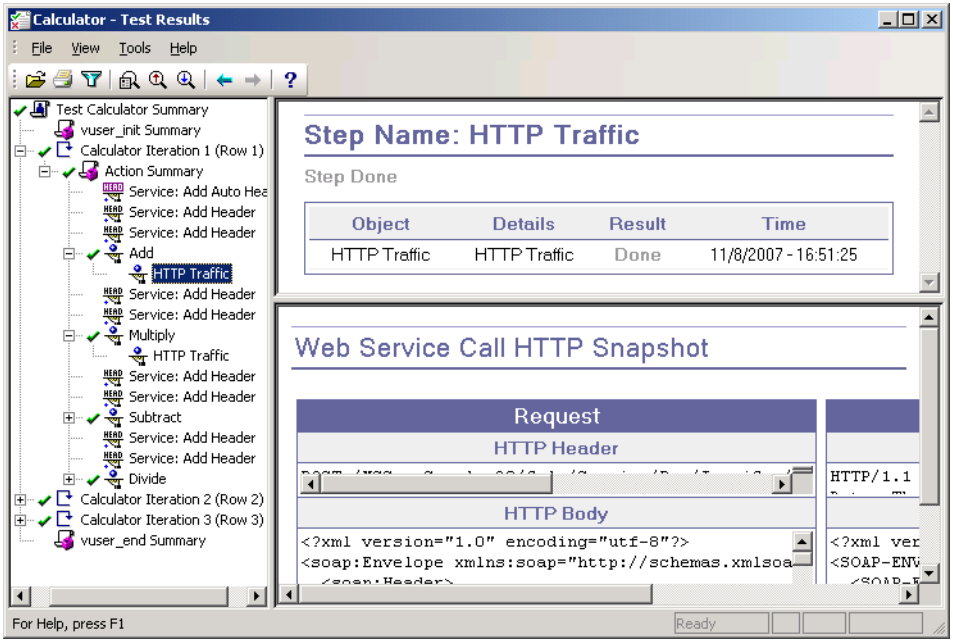
This section describes the report information specific to Web Services scripts.

The Web Services test results shows a list of the service's operations that were called during replay. When you select an operation, the report shows information about the service, operation, toolkit, testing aspect, and WSDL.



If Service Test cannot interpret the script or if it encounters another type of error, the report displays a message in the right pane stating the problem.

If you expand an operation's node further, the report shows the actual SOAP traffic for the Request and Response.

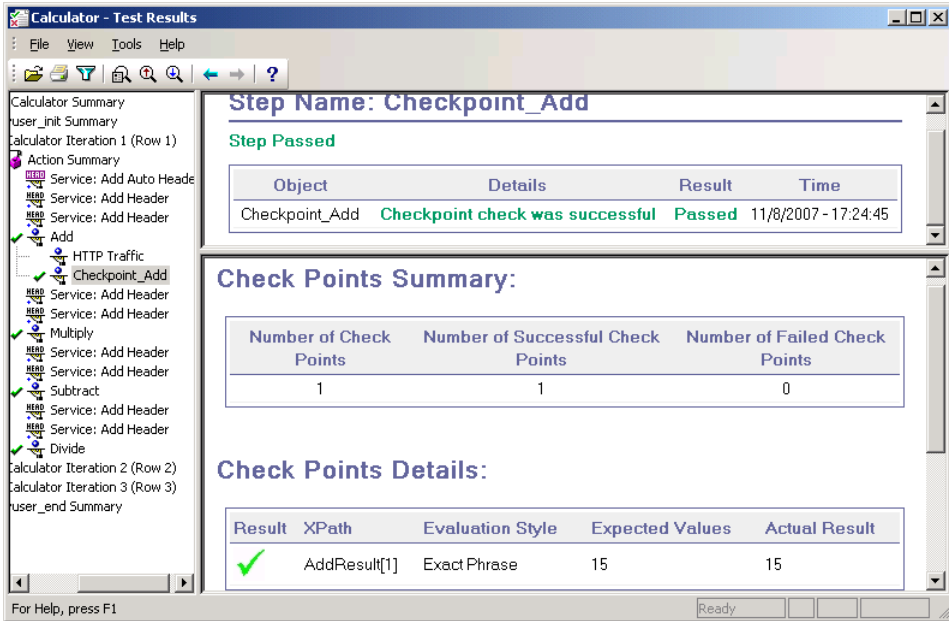


Checkpoint Results

The Results window also shows checkpoint results. Setting checkpoints is described in "Setting Checkpoints" on page 278

If your checkpoints fail, the report provides a summary with a reason for the failure. It also provides the Expected Value and Actual Results as well as the argument tree.

To view the checkpoint details, expand the appropriate step under the operation in the left pane and click the **Checkpoint** node.



Sending Custom Information to the Report

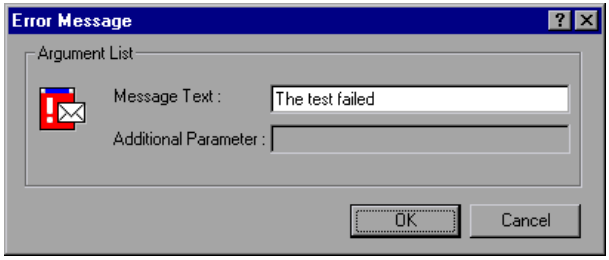
In addition to the information sent automatically to the report, for Web Service Vusers, you can send information to the report using the message functions **lr_output_message** or **lr_error_message**.

You can use the following message functions in your Vuser scripts:

To insert an error or output message function:

1 Select **Insert** > **New Step**. The Add Step dialog box opens.

2 Select the **Error Message** or **Output Message** step and click **OK**. The Error Message or Output Message dialog box opens.



3 Type the message into the **Message Text** box.

4 Click **OK** to insert the message and close the dialog box.

An **lr_error_message** or **lr_output_message** function is inserted at the current point in the script, and the messages will be sent to the Test Summary report.

For more information about the message functions, see the *Online Function Reference* (**Help** > **Function Reference**).

5

Managing Scripts Using Quality Center

Service Test's integration with Quality Center lets you manage Vuser scripts using Quality Center.

This chapter includes:

About Managing Scripts Using Quality Center

Service Test works together with Quality Center, HP's Web-based test management tool. HP Quality Center provides an efficient method for storing and retrieving Vuser scripts, scenarios, and results. You store scripts in a Quality Center project and organize them into unique groups.

In order for Service Test to access a Quality Center project, you must connect it to the Web server on which Quality Center is installed. You can connect to either a local or remote Web server.

For more information on working with Quality Center, see the *Quality Center User Guide*.

Connecting to and Disconnecting from Quality Center

To store and retrieve scripts from Quality Center, you need to connect to a Quality Center project. You can connect or disconnect from a Quality Center project at any time during the testing process.

Connecting to Quality Center

The connection process has two stages. First, you connect to a local or remote Quality Center Web server. This server handles the connections between Service Test and the Quality Center project.

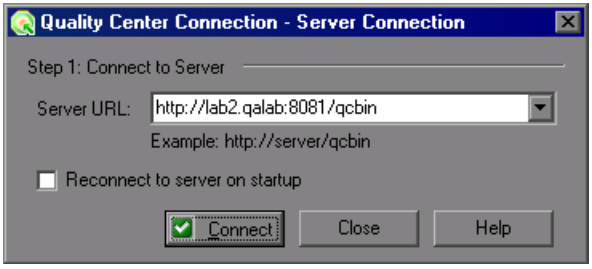
Next, you select the project you want to access. The project stores the scripts that you created in Service Test.

Note: Quality Center projects are password protected, so you must provide a user name and a password.

To connect to Quality Center:

1 Select **Tools** > **Quality Center Connection**.

2 The Quality Center Connection - Server Connection dialog box opens.

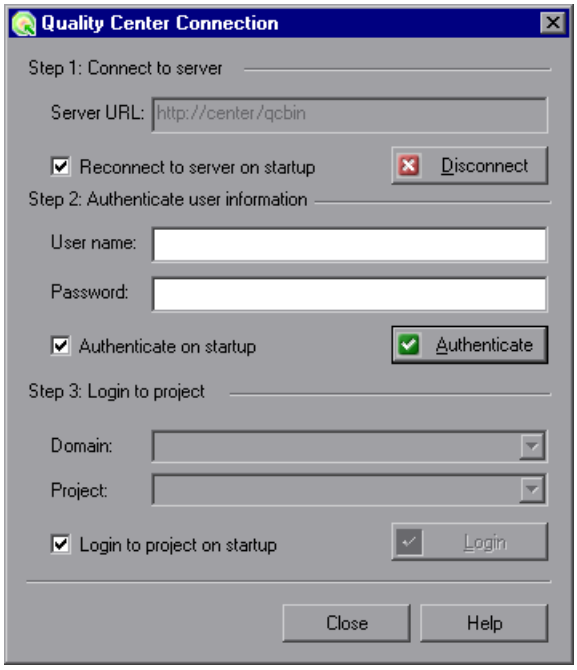


3 In the **Server URL** box, type the URL address of the Web server where Quality Center is installed.

Note: You can select a Web server accessible via a Local Area Network (LAN) or a Wide Area Network (WAN).

4 To automatically reconnect to the Quality Center server the next time you open Service Test, select the **Reconnect to server on startup** check box.

5 Click **Connect**. The Quality Center Connection dialog box opens.



After the connection to the server is established, the server's name is displayed in read-only format in the Server box.

6 Authenticate your user information:

a In the **User name** box, type your Quality Center user name.

b In the **Password** box, type your Quality Center password.

c Click **Authenticate** to authenticate your user information against the Quality Center server.

After your user information has been authenticated, the fields in the Authenticate user information area are displayed in read-only format. The **Authenticate** button changes to a **Change User** button.

You can log in to the same Quality Center server using a different user name by clicking **Change User**, entering a new user name and password, and then clicking **Authenticate** again.

7 If you selected **Reconnect to server on startup** above, the **Authenticate on startup** option is enabled. To authenticate your user information automatically, the next time you open Service Test, select **Authenticate on startup**.

8 Enter the details for logging in to the project:

a In the **Domain** box, select the domain that contains the Quality Center project. Only those domains containing projects to which you have permission to connect to are displayed. (If you are working with a project in versions of TestDirector earlier than version 7.5, the **Domain** box is not relevant. Proceed to the next step.)

b In the **Project** box, enter the Quality Center project name or select a project from the list. Only those projects that you have permission to connect to are displayed.

c Click **Login**.

9 To log in to the selected project on startup, select **Login to project on startup**. This option is only enabled when the **Authenticate on startup** check box is selected.

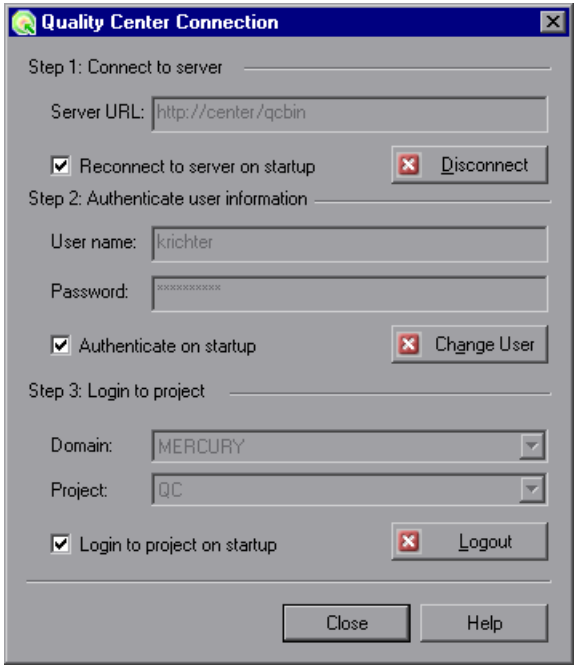
10 Click **Close** to close the Quality Center Connection dialog box.

Disconnecting from Quality Center

You can disconnect Service Test from a selected Quality Center project and Web server.

To disconnect from Quality Center:

1 Select **Tools** > **Quality Center Connection**. The Quality Center Connection dialog box opens.



2 To disconnect from the selected project, under **Login to project**, click **Logout**. If you want to open a different project while using the same server, enter the Quality Center project name in the **Project** box or select a project from the list.

3 To disconnect from the Quality Center server, under **Connect to server**, click **Disconnect**.

4 Click **Close** to close the Quality Center Connection dialog box.

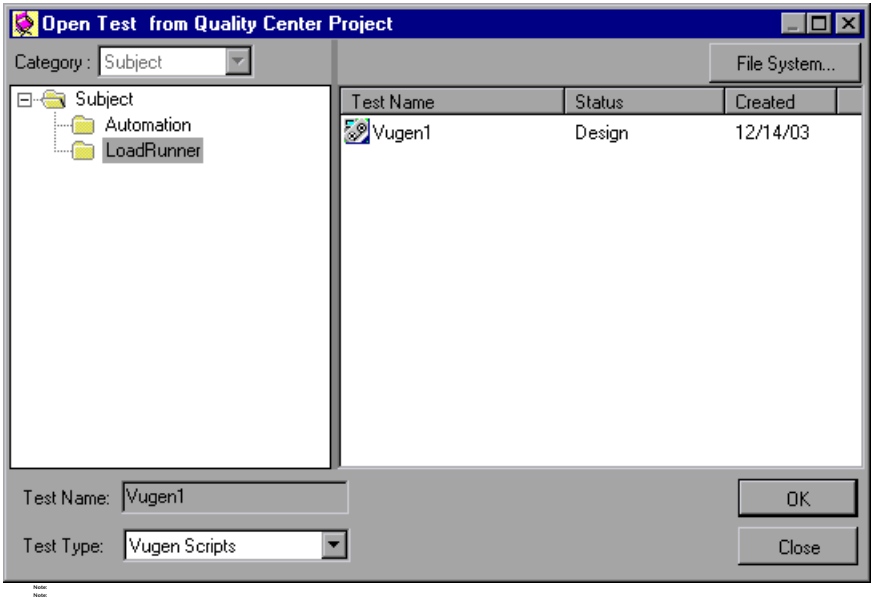
Opening Scripts from a Quality Center Project

When connected to a Quality Center project, you can open your scripts from Quality Center. You locate tests according to their position in the test plan tree, rather than a location in the file system.

To open a script from a Quality Center project:

1 Connect to the Quality Center server. (**Tools** > **Quality Center Connection**)

2 Select **File** > **Open**. The Open Test from Quality Center Project dialog box opens and displays the test plan tree.



Note: To open a script directly from the file system, click the **File System** button. (To return to the Open from Quality Center Project dialog box, click the **Quality Center** button.)

3 Click the relevant subject in the test plan tree. To expand the tree and view sublevels, double-click closed folders. To collapse the tree, double-click open folders.

Note that when you select a subject, the scripts that belong to the subject appear in the Test Name list.

4 Select a script from the Test Name list. The script appears in the read-only Test Name box.

5 Click **OK** to open the script. Service Test loads the script and displays its name in the title bar.

You can also open scripts from the recent file list in the **File** menu. If you select a recent script from a Quality Center project, but you are not connected to that project, the Quality Center Connection dialog box opens.

Opening Tests from the Recent Files List

You can open scripts from the recent files list in the File menu. If you select a file located in a Quality Center project, but you are not connected to Quality Center or to the correct project for that file, Service Test prompts you to connect to Quality Center.

Log in to the project to continue working with the script.

The Connect to Quality Center Project dialog box also opens if you choose to open a script that was last edited on your computer using a different user name. You can either log in using the displayed name or you can click **Cancel** to stay logged in with your current user name.

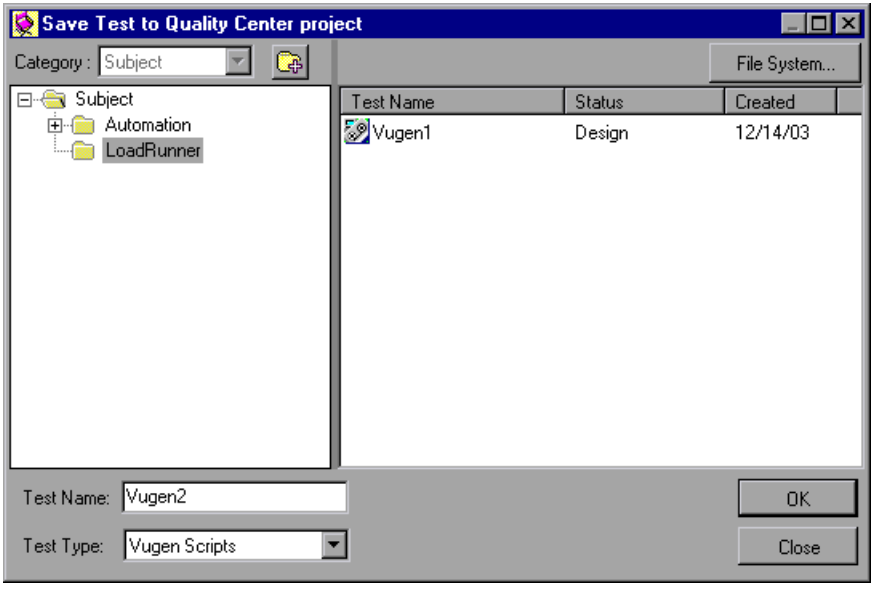
Saving Scripts to a Quality Center Project

When Service Test is connected to a Quality Center project, you can create new scripts in Service Test and save them directly to your project. To save a script, you give it a descriptive name and associate it with the relevant subject in the test plan tree. This helps you to keep track of the scripts created for each subject and lets you view the progress of test planning and creation.

To save a script to a Quality Center project:

1 Connect to the Quality Center server. (**Tools** > **Quality Center Connection**)

2 Select **File** > **Save As**. The Save Test to Quality Center Project dialog box opens and displays the test plan tree.



To save a script directly in the file system, click the **File System** button. The Save Test dialog box opens. (From the Save Test dialog box, you may return to the Save Test to Quality Center Project dialog box by clicking the **Quality Center** button.)

3 Select the relevant subject in the test plan tree. To expand the tree and view a sublevel, double-click a closed folder. To collapse a sublevel, double-click an open folder.

4 In the Test Name box, enter a name for the script. Use a descriptive name that will allow you to identify the script easily.

5 Click **OK** to save the script and close the dialog box.

The next time you start Quality Center, the new script will appear in Quality Center's test plan tree.

Managing Script Versions

When connected to a Quality Center project with version control support, you can update and revise your automated scripts while maintaining old versions. This helps you keep track of the changes made to each script, see what was modified from one version of a script to another, or return to a previous version of the script.

You add a script to the version control data base by saving it in a project with version control support. You manage versions by checking scripts in and out of the version control database.

The script with the latest version is the script that is located in the Quality Center repository and is used by Quality Center for all test runs.

Note: The **Quality Center Version Control** options in the **File** menu are available only when you are connected to a Quality Center project database with version control support.

This section includes:

Adding Scripts to the Version Control Database

When you use **Save As** to save a new script in a Quality Center project with version control support, Service Test automatically saves the script in the project, checks the script into the version control database with version number 1.1.1 and then checks it out so that you can continue working.

The status bar indicates each of these operations as they occur. Note, however, that saving your changes to an existing script does not check them in. Even if you save and close the script, it remains checked out until you choose to check it in. For more information, see "Checking Scripts Out of the Version Control Database" on page 126.

Checking Scripts Out of the Version Control Database

When you select **File** > **Open** to open a script that is currently checked in to the version control database, it is opened in read-only mode.

Note: The Open Test from Quality Center Project dialog box displays icons that indicate the version control status of each script in your project. For more information, see "Opening Scripts from a Quality Center Project" on page 122.

You can review the checked-in script. You can also run the script and view the results.

To modify the script, you must check it out. When you check out a script, Quality Center copies the script to your unique check-out directory (automatically created the first time you check out a script), and locks the script in the project database. This prevents other users of the Quality Center project from overwriting any changes you make to the script. However, other users can still run the version that was last checked in to the database.

You can save and close the script, but it remains locked until you return the script to the Quality Center database. You can release the script by either checking the script in, or undoing the check out operation. For more information on checking script in, see "Checking Scripts into the Version Control Database" on page 127. For more information on undoing the check-out, see "Canceling a Check-Out Operation" on page 133.

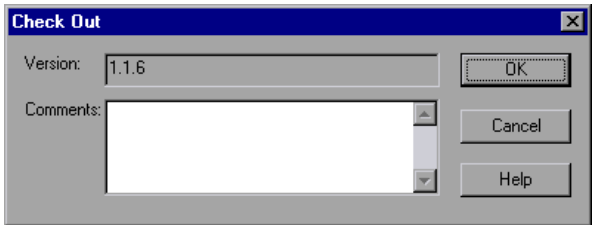
By default, the check out option accesses the latest version of the script. You can also check out older versions of the script. For more information, see "Using the Version History Dialog Box" on page 129.

To check out the latest version of a script:

1 Open the script you want to check out. For more information, see "Opening Scripts from a Quality Center Project" on page 122.

Note: Make sure the script you open is currently checked in. If you open a script that is checked out to you, the **Check Out** option is disabled. If you open a script that is checked out to another user, all **Quality Center Version Control** options, except the **Version History** option, are disabled.

2 Select **File** > **Quality Center Version Control** > **Check Out**. The Check Out dialog box opens and displays the script version to be checked out.



3 You can enter a description of the changes you plan to make in the **Comments** box.

4 Click **OK**. The read-only script closes and automatically reopens as a writable script.

Checking Scripts into the Version Control Database

While a script is checked out, Quality Center users can run the previously checked-in version of your script. For example, suppose you check out version 1.2.3 of a script and make a number of changes to it and save the script. Until you check the script back in to the version control database as version 1.2.4 (or another number that you assign), Quality Center users can continue to run version 1.2.3.

When you have finished making changes to a script and you are ready for Quality Center users to use your new version, you check it in to the version control database.

Note: If you do not want to check your changes into the Quality Center database, you can undo the check-out operation. For more information, see "Canceling a Check-Out Operation" on page 133.

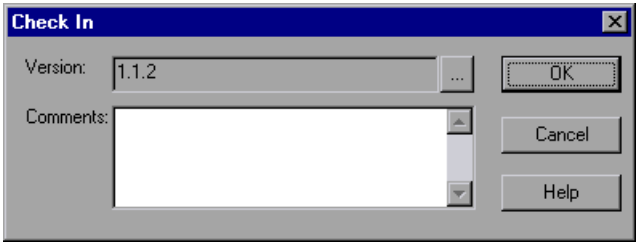
When you check a script back into the version control database, Quality Center deletes the script copy from your checkout directory and unlocks the script in the database so that the script version will be available to other users of the Quality Center project.

To check in the currently open script:

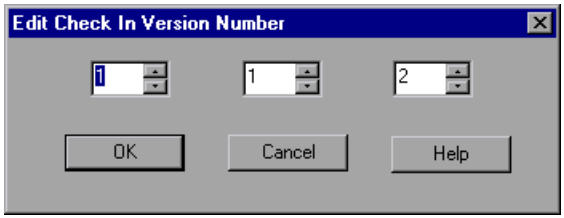
1 Confirm that the currently open script is checked out to you. For more information, see "Viewing Version Information For a Script" on page 129.

Note: If the open script is currently checked in, the **Check In** option is disabled. If you open a script that is checked out to another user, all **Quality Center Version Control** options, except the **Version History** option, are disabled.

2 Select **File** > **Quality Center Version Control** > **Check In**. The Check In dialog box opens.



3 Accept the default new version number and proceed to step 7, or click the browse button to specify a custom version number. If you click the browse button, The Edit Check In Version Number dialog box opens.



4 Modify the version number manually or using the up and down arrows next to each element of the version number. You can enter numbers 1-900 in the first element. You can enter numbers 1-999 in the second and third elements. You cannot enter a version number lower than the most recent version of this script in the version control database.

5 Click **OK** to save the version number and close the Edit Check In Version Number dialog box.

6 If you entered a description of your change when you checked out the script, the description is displayed in the **Comments** box. You can enter or modify the comments in the box.

7 Click **OK** to check in the script. The script closes and automatically reopens as a read-only file.

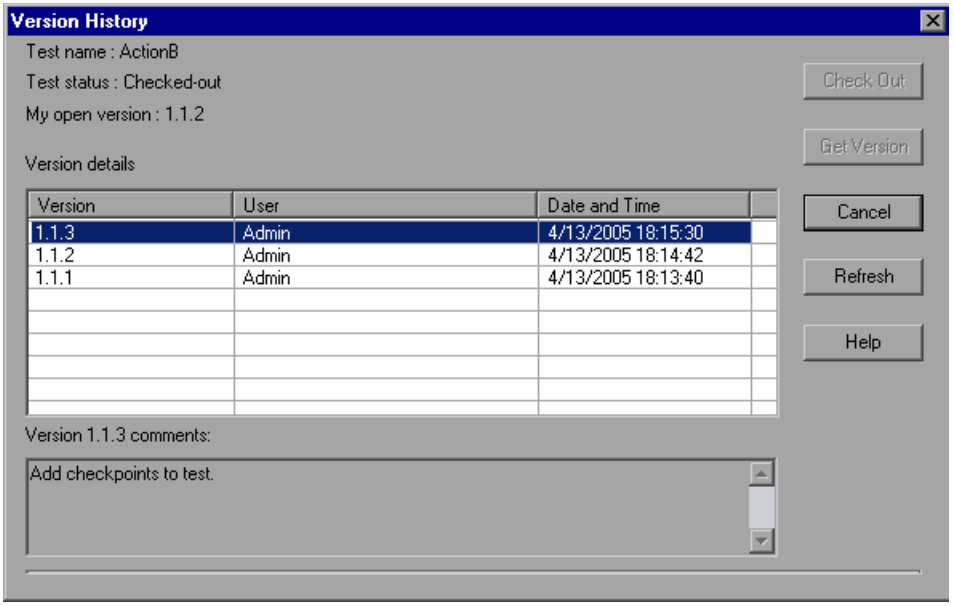
Using the Version History Dialog Box

You can use the Version History dialog box to view version information about the currently open script and to view or retrieve an older version of the script.

Viewing Version Information For a Script

You can view version information for any open script that has been stored in the Quality Center version control database, regardless of its current status.

To open the Version History dialog box for a script, open the script and select **File** > **Quality Center Version Control** > **Version History**.



The Version History dialog box provides the following information:

- **Test name**. The name of the currently open script.
- **Test status**. The status of the script. The script can be:
  - **Checked-in**. The script is currently checked in to the version control database. It is currently open in read-only format. You can check out the script to edit it.
  - **Checked-out**. The script is checked out by you. It is currently open in read-write format.
  - **Checked-out by <another user>**. The script is currently checked out by another user. It is currently open in read-only format. You cannot check out or edit the script until the specified user checks in the script.
- **My open version**. The script version that is currently open on your QuickTest computer.

- **Version details.** The version details for the script.
  - **Version**. A list of all versions of the script.
  - **User**. The user who checked in each listed version.
  - **Date and Time**. The date and time that each version was checked in.
- **Version comments**. The comments that were entered when the selected version was checked in.

Working with Previous Script Versions

You can view an old version of a script in read-only mode or you can check out an old version and then check it in as the latest version of the script.

To view an old version of a script:

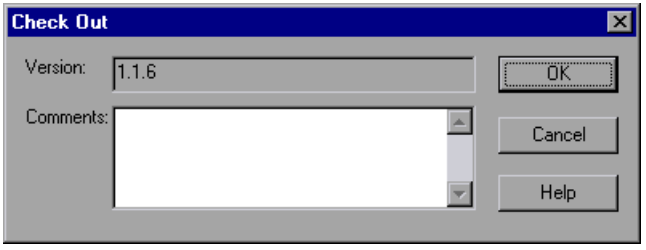
1. Open the Quality Center script. The latest version of the script opens. For more information, see "Opening Scripts from a Quality Center Project" on page 122.
2. Select **File** > **Quality Center Version Control** > **Version History**. The Version History dialog box opens.
3. Select the version you want to view in the **Version details** list.
4. Click the **Get Version** button. QuickTest reminds you that the script will open in read-only mode because it is not checked out.
5. Click **OK** to close the QuickTest message. The selected version opens in read-only mode.

Tips: To confirm the version number that you now have open in QuickTest, look at the **My open version** value in the Version History dialog box.

After using the **Get Version** option to open an old version in read-only mode, you can check-out the open script by choosing **File** > **Quality Center Version Control** > **Check Out**. This is equivalent to using the **Check Out** button in the Version History dialog box.

To check out an old version of a script:

1. Open the Quality Center script. The latest version of the script opens. For more information, see "Opening Scripts from a Quality Center Project" on page 122.
2. Select **File** > **Quality Center Version Control** > **Version History**. The Version History dialog box opens.
3. Select the version you want to view in the **Version details** list.
4. Click the **Check Out** button. A confirmation message opens.
5. Confirm that you want to check out an older version of the script. The Check Out dialog box opens and displays the version to be checked out.



6. You can enter a description of the changes you plan to make in the **Comments** box.
7. Click **OK**. The open script closes and the selected version opens as a writable script.
8. View or edit the script as necessary.
9. If you want to check in your script as the new, latest version in the Quality Center database, select **File** > **Quality Center Version Control** > **Check In**. If you do not want to upload the modified script to Quality Center, select **File** > **Quality Center Version Control** > **Undo Check out**.

   For more information on checking in script, see "Checking Scripts into the Version Control Database" on page 127. For more information on undoing the check-out, see "Canceling a Check-Out Operation" on page 133.

Canceling a Check-Out Operation

If you check out a script and then decide that you do not want to upload the modified script to Quality Center you should cancel the check-out operation so that the script will be available for check out by other Quality Center users.

To cancel a check-out operation:

1. If it is not already open, open the checked-out script.
2. Select **File** > **Quality Center Version Control** > **Undo Check out**.
3. Click **Yes** to confirm the cancellation of your check-out operation. The check-out operation is cancelled. The checked-out script closes and the previously checked-in version reopens in read-only mode.

Part II

Preparing Web Services Scripts

6

Web Services and SOA Tests

You use HP Service Test to create tests for your Web Services. After creating a Web Services test script, you can view or modify it in either Script view or Tree view.

This chapter includes:

About Web Services Scripts and SOA Tests

SOA systems are based on Web Services, self-contained applications that can run across the Internet on a variety of platforms. The services are built using Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP). They serve as building blocks enabling the rapid development and deployment of new applications.

Using Service Test, you create test scripts for testing your SOA environment. You can use a test generation wizard to automatically generate scripts, or create the scripts manually.

To automatically generate test scripts, you use the SOA Test Generator. A wizard guides you through the process of selecting testing aspects such as interoperability with different toolkits, boundary testing, and standard compliance. For more information, see Chapter 9, "Web Services - Automatic Test Generator."

To manually create scripts, you begin by creating an empty script. Then you add content to the script either by recording a session, analyzing network traffic, or manually inserting calls to the Web service as described in Chapter 8, "Web Services - Adding Script Content."

An additional test type is BPT (Business Process Testing). BPT is a methodology in which several tasks are combined to create a complete business process. For more information, see "BPT (Business Process Testing)" on page 214.

For manual scripts, you use Service Test to create one of the following scripts.

- **Single Protocol Script.** A script that emulates SOAP traffic by sending SOAP requests to the Web service.
- **Multi Protocol Script.** A script that emulates several protocols in a single script. For example, if your environment contains a client that accesses a Web Services and Web pages, select both the Web Services and Web (Click and Script) protocols.

Getting Started with Web Services Vuser Scripts

This section provides an overview of the process of developing a Web Services / SOA Vuser script.

To develop a test script:

1 Create a new Web Services script.

Create a new script using the SOA Test Generator, or manually create a new single or multiple protocol script, or a Business Process Testing component.

2 Validate your service's XML.

Service Test lets you insert a validation check for your XML, XML schema, and their expected value during runtime. For more information, see "Validating XML" on page 145.

3 Add content to the script.

Add content to the script (excluding the SOA Test Generator). For details, see Chapter 8, "Web Services - Adding Script Content."

4 Set properties, values, and checkpoints.

Enhance the script by customizing the step properties, inserting argument values, and setting checkpoints. For details, see Chapter 11, "Web Services Call Properties."

5 Parameterize your script.

Parameterization lets you replace constant values with a variable to substitute new values for each iteration. To parameterize a value, double-click on a step to open its properties and click the **ABC** icon adjacent to the value box. For complex type elements, use the XML parameter type as described in "Setting Properties for XML Parameters" on page 455.

6 Enhance your script with transactions.

Define a group of actions as a transaction to check the applications's performance. For example, if you want to check the time it took for a service to update an address, you mark those actions as a transaction. For more information, see "Inserting Transactions into a Vuser Script" on page 114.

7 Validate the service.

Test your service for compliancy. For more information, see "Performing WS-I Validation" on page 190.

8 Configure the Run-Time settings.

The Run-Time settings control the script's behavior during execution. These settings include Web Service-specific settings (client emulation) and General settings—run logic, pacing, logging, and think time.

For information about the Web Service-specific settings, see "Web Services JMS Run-Time Settings" on page 286, and Chapter 2, "Configuring Run-Time Settings."

9 Verify that the script is functional.

Replay the script in Service Test to verify that it runs correctly.

For details about replaying the script, see Chapter 3, "Running Vuser Scripts in Standalone Mode."

10 Save the script.

Save the script in the file system or in a Quality Center repository. If you save the scripts in Quality Center, you can associate them to a test set and perform functional and regression testing directly from Quality Center. For more information about Quality Center and its integration with scripts, see "Working with Service Test Management" on page 220.

After you prepare a script, you are ready to use it for your testing. For more information, see "Using Your Script" on page 213.

Use Quality Center to manage all of your tests while tracking defects and requirements. For more information, see www.hp.com or contact your sales representative.

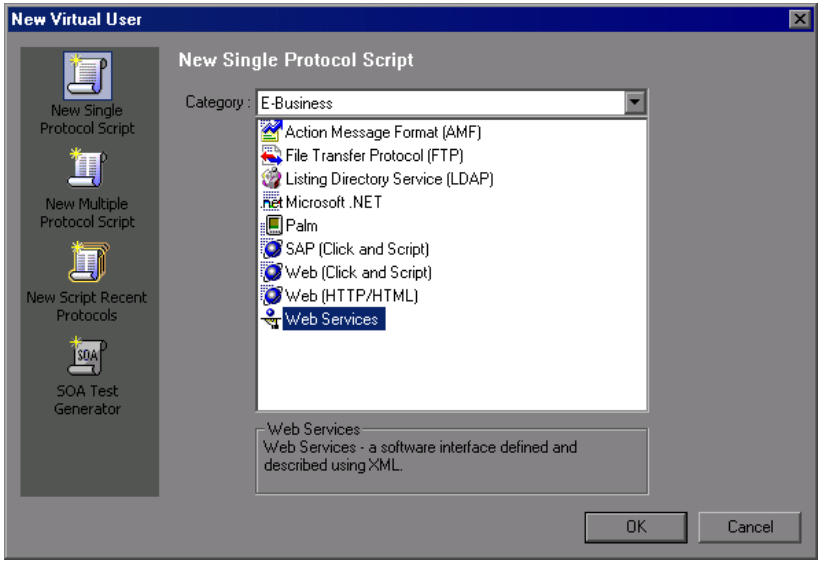
Creating an Empty Web Services Script

To create a script through manual Web Service calls, you must first create an empty script. The empty script serves as a base from which you begin to add Web Service calls and other SOA related steps.

To create an empty script:

1 Create an empty script. Click the New Script button or select **File** > **New** to open the New Virtual User dialog box.

2 Click **New Single Protocol Script** in the left pane. Select **Web Services** protocol from the **E-Business** category. Click **OK**.



If you need to record several different protocols, such as Web Services and Web, click **New Multiple Protocol Script** in the left pane and specify the desired protocols. Click **OK**.

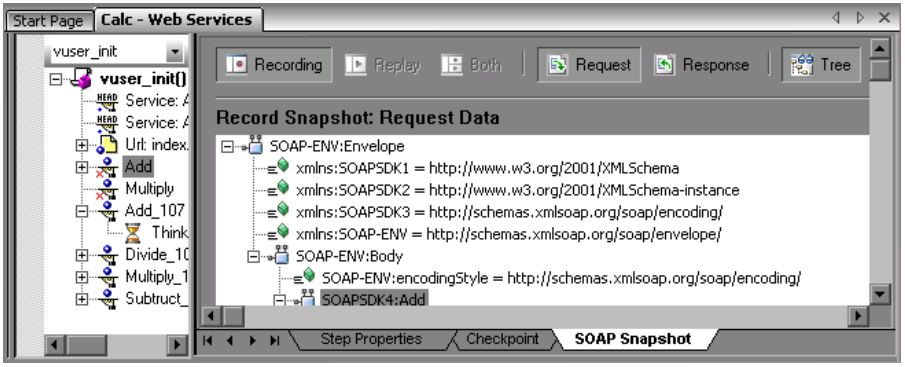
Viewing and Editing Scripts

You can view and edit all of the scripts that you created both manually and automatically in the Service Test window.

You can view a script in either Tree View or Script View. Tree view displays the steps of the script in a graphical interface, while the Script view shows all steps, including the actual **web_service_call** functions that emulate your service. Script view is ideal for advanced users that require more flexibility within the script.

Tree View

The Tree view shows a graphical representation of each one of the script's steps.



When you select a step, Service Test displays information about the step in several tabs:

- **Step Properties.** The properties and argument values of the Web service call. This tab allows you to modify the properties of an existing step. See "Understanding Web Service Call Properties" on page 245.
- **Checkpoint.** A list of checkpoints defined for the step. See "Setting Checkpoints" on page 278.
- **SOAP Snapshot.** A snapshot of the SOAP request and response for both record and replay. See "Viewing Web Services SOAP Snapshots" on page 242.

For more information about these tabs, see Chapter 11, "Web Services Call Properties."

To view a script in Tree view:

1. Click the **Tree** button or select **View** > **Tree View**.
2. In the upper left box, select the section containing the steps of the script that you want to view: **vuser_init**, **Action**, or **vuser_end**. To specify a new action select **Actions** > **Create New Action**.
3. In the left pane, select the step or sub-node that you want to view or modify.

4 Select the **Step Properties** tab in the right pane to view or modify the properties.

5 Select the **Snapshot** tab to view the step's SOAP header and body. To display a specific replay iteration, select **View** > **Snapshot** > **Select Iteration**.

6 To add additional Web Service steps, click the **Add Service Call** button. For more information, see "Adding New Web Service Calls" on page 207.

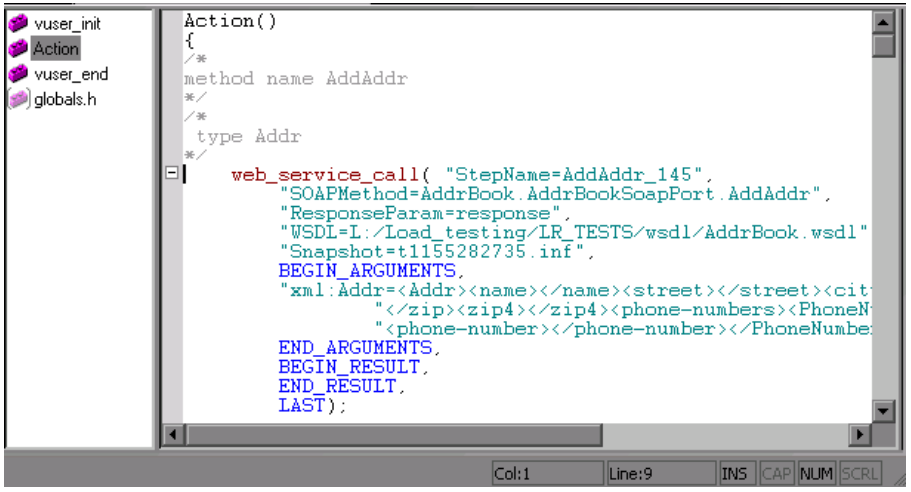
7 To insert advanced functionality, such as JMS queue retrieval and SAML security, select **Insert** > **Add Step** and select the appropriate step. For more information, see Chapter 14, "Web Services - Security."

8 To replace argument values with parameters, go to the **Step Properties** tab. Select the node whose value you want to replace in the script, and click the **ABC** icon to the right of the **Value** box.

9 To set a checkpoint, click the **Checkpoint** tab. For more Information, see "Setting Checkpoints" on page 278.

Script View

The Script view shows the actual functions that were generated in the script. You can expand or collapse each of the **web_service_call** functions to view only the functions that interest you.



To view a script in Script view:

1. Click the **Script** button or select **View** > **Script View**.
2. In the left pane, select the section containing the steps of the script that you want to view: **vuser_init**, **Action**, or **vuser_end**. To specify a new section select **Actions** > **Create New Action**.
3. To add additional Web Service steps at the location of the cursor, click the **Add Service Call** button. For more information, see Chapter 11, "Web Services Call Properties."
4. To insert advanced functionality, such as JMS queue retrieval and SAML security, select **Insert** > **Add Step** and select the appropriate step. For more information, see Chapter 14, "Web Services - Security."
5. To replace argument values with parameters, select the value you want to replace in the script, and select **Replace with Parameter** from the right-click menu.

For more information about the functions, see the *Online Function Reference* (**Help** > **Function Reference**) or select a function and click F1.

Parameterizing Scripts

Service Test supports parameterization for all of the argument values. Parametrization lets you substitute the original values with external values. This is useful for testing your service with different values, or passing information from one step to another. For an overview on parameterization, see Chapter 20, "Working with VuGen Parameters."

If your arguments are the simple, non-array type, you can replace them with a simple parameter. For example, if you want to test a service that does addition, you can substitute each of the input arguments with a parameter, and store the values in a file or a table.

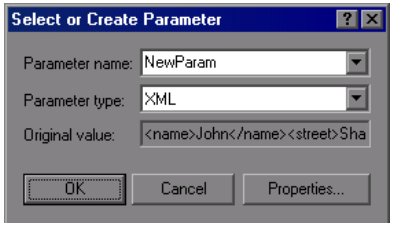
If, however, your arguments are a complex structure with many values, you can use an XML type parameter to replace the entire structure with a single parameter. You can create several value sets for the XML type parameter and assign a different value set for each iteration. For more information, see "XML Parameter Types" on page 427.

Using parameters, you can pass the output value from one operation, as input for a later operation. For more information, see "Using Web Service Output Parameters" on page 289.

To replace a constant value with a parameter:

1 Switch to the **Step Properties** tab and select the parent or child element whose value you want to parameterize.

2 Under the **Input Arguments** node, select the argument you want to parameterize. In the right pane, click the **ABC** icon in the **Value** box. The Select or Create Parameter dialog box opens.



3 Specify a parameter name and type.

4 Click **Properties** to set the type of parameter—File, XML, and so on—and to assign values.

For more information, see Chapter 20, "Working with VuGen Parameters."

Validating XML

When you run a Web Service, it transfers data requests and responses to and from a server. In Web Services scripts, it is common to use parameters for the data instead of hard-coded values. Each parameter contains the actual XML data being transferred. By using parameters, you can test your Web Service with many different values.

Service Test provides an interface for validating the XML data in the parameters by checking for:

- well-formed XML
- correctness of checkpoint values

➤ compliance with an XML schema (manual loaded schemas only)

To be well-formed, an XML document must follow more than 100 different rules as defined by the World Wide Web Consortium (W3C). For example, one important rule is that each element must have start and end tags.

Checkpoint validation compares argument values within the XML file with expected values. You manually specify the expected values or load them from an earlier session.

An XML schema represents the interrelationship between the attributes and elements of the XML objects. To check that an XML file complies with its schema, you specify the XML file together with its XSD (XML Schema Definition) file. This check only applies to XSDs that you load manually—not ones that Service Test detects automatically.

The following sections describe how to create XML validation steps:

➤ Validation Comparison Operators

➤ Checking for the Expected Response

➤ Adding an XML Validation Step

➤ Validating an XSD Schema

➤ Viewing Validation Results

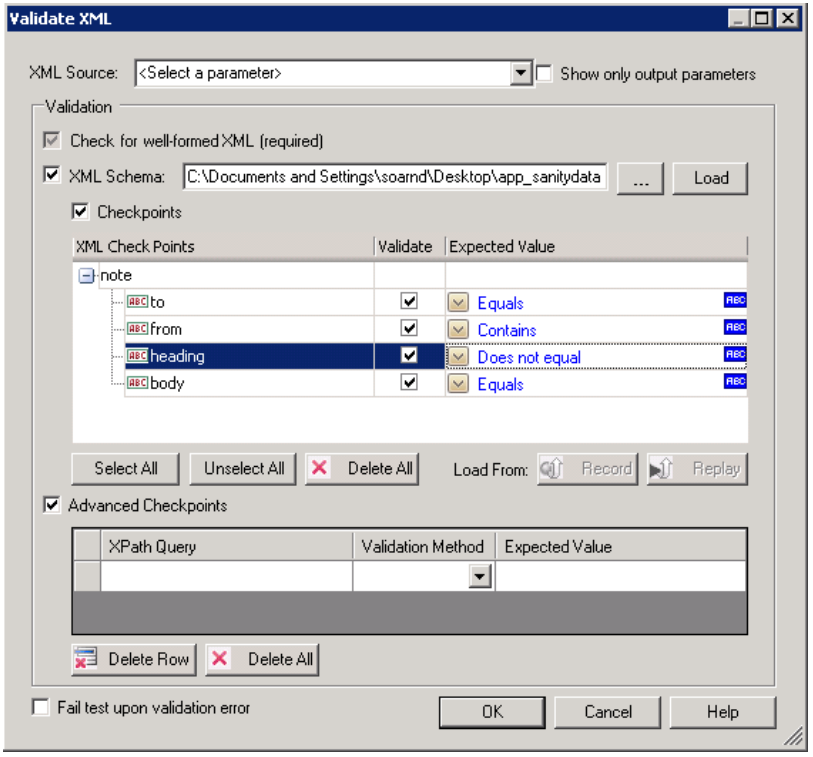
➤ Working with REST Web Services

Validation Comparison Operators

You use checkpoints in XML validation to check for expected values in the XSD using the following comparison operators:

- **Equals.** match the exact text.
- **Does not equal.** Does not match the text.
- **Contains**. match part of the text.
- **Starts With.** a match if the returned text begins with the following phrase.
- **Ends With**. a match if the returned text ends with the following phrase.

➤ **Reg. expression.** matches the string pattern using regular expression syntax



For numerical elements, you indicate one of the standard numerical relationships, such as equal, not equal, greater than, less than, and so forth.

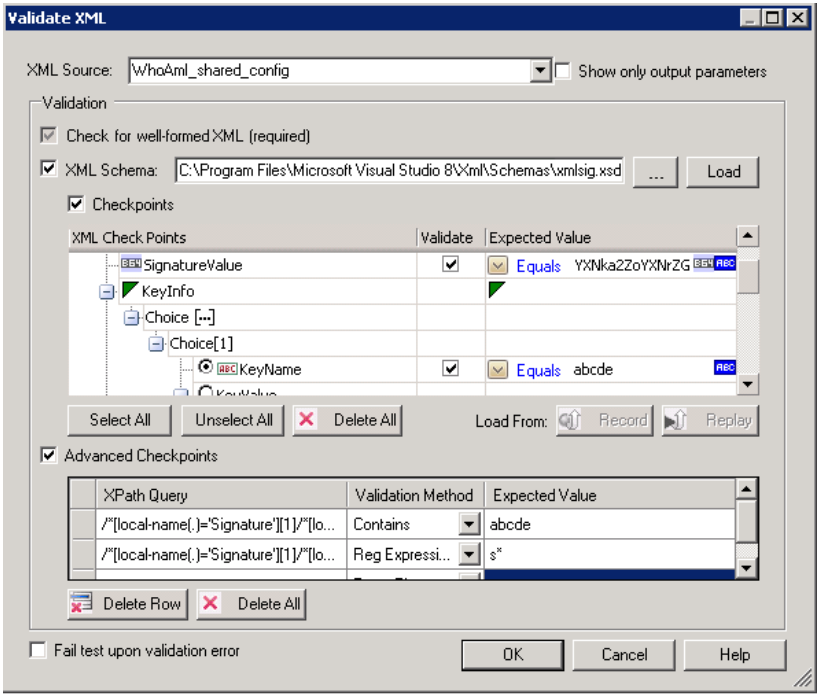
For information on defining advanced queries using regular expressions, see "XPATH Expressions" on page 149.

Basic and Advanced Checkpoints

You can perform both basic or advanced validations for your checkpoints.

In basic validation, Service Test looks for exact matches of the value in the **Expected value** column. You can load expected values from a Recorded or Replay snapshot.

Use **Advanced Checkpoints** to validate a checkpoint on non-leaf nodes or to define expected values in terms of a regular expression.



You define the advanced validation values by entering an XPATH query in the **Advanced Checkpoint** section. To obtain the initial XPATH expression, copy it from a row in the **Basic Validation**. Choose **Copy Row XPATH** from the right-click menu and paste the contents in the **Advanced Validation** section.

You can define both basic and advanced validations for the same step.

Note that when you select a non-leaf node, you need to supply all of the XML beneath the node.

In the Vuser script, Service Test indicates an exact match by **Value=** and a regular expression with **Expression=**:

```
BEGIN_CHECKPOINTS,
        CHECKPOINT, "XPATH=/AddResult[1]", "Value=50"
        CHECKPOINT, "XPATH=AddResult", "Expression=Hel*?",
END_CHECKPOINTS,
```

XPATH Expressions

Checkpoint expressions support both Nodeset and extended XPath syntax.

NodeSet expressions are XPath expressions evaluated to a single XML node or to a set of XML nodes. For example:

```
/a/b/c
/x/y[1]
//a/b
```

Service Test also supports full XPath expressions, such as **count(/a/b/c)** which is evaluated to int, or **count(/a/b/c) < 3** which is evaluated to a boolean value. The XPATH support allows you to do numeric comparisons. For example, to verify whether a result is between 8 and 16, you can use the following expression:

```
number(/a/b/c) > 8 and number(/a/b/c) < 16
```

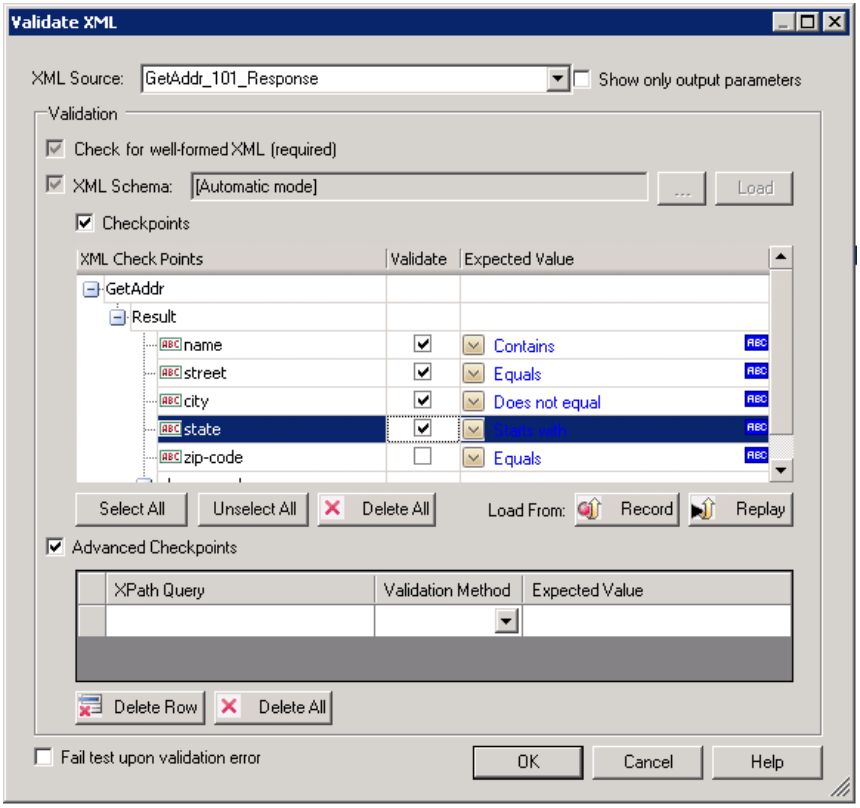
Checking for the Expected Response

Using XML Validation you can check the entire response of your Web Service call. This capability is similar to the standard checkpoints, but it has the added capability of saving the entire response to a single parameter, instead of saving each argument to its own parameter.

To validate your responses, you add an XML validation step after the Web Service call you want to validate (it need not be immediately after the call).

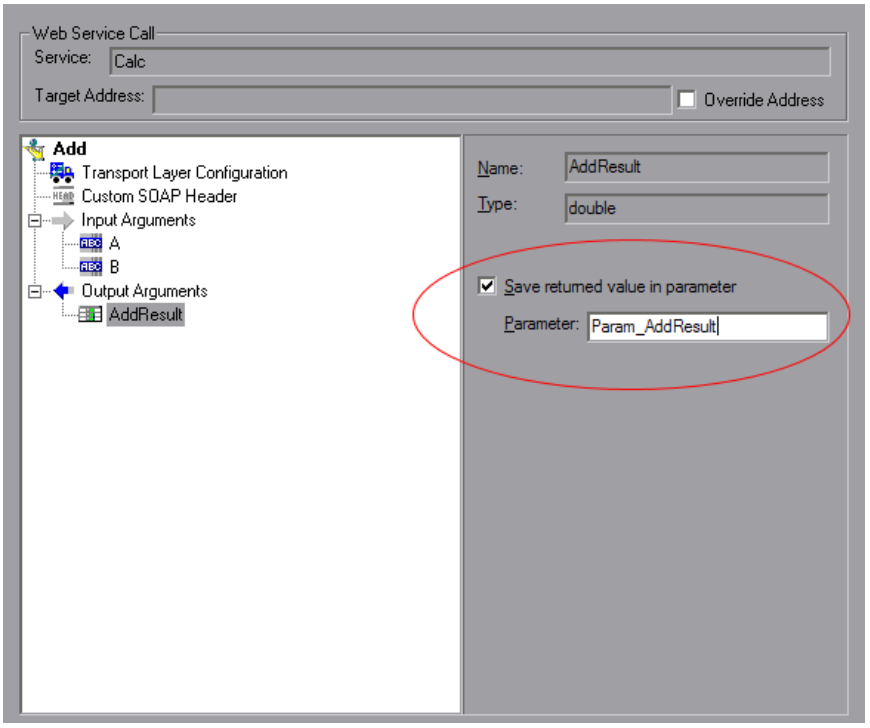
For each Web Service call, Service Test saves the entire XML response to a parameter. This lets you validate the entire response in a single step. Note that this is not the actual SOAP response. The SOAP response contains the response values and the entire SOAP envelope. This response while containing the XML values, does not contain the SOAP header or envelope.

The name of the parameter is the Web Service call step name with a **_Response** suffix. For example, if the step name is **GetAddr_101**, the response parameter is **GetAddr_101**_Response.



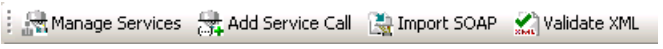
Since the response is unique for each Web Service call, you must add separate validation steps for each saved parameter. When you select a response parameter, Service Test automatically loads the XSD, in order to perform a schema validation.

You can also validate an individual argument, whose value you manually saved to a parameter in design time.



To validate the response:

1 Open the XML Validation dialog box by clicking the **Validate XML** button in the upper toolbar.



2 Place the cursor after (in Script view) or on (in Tree view) the Web Service call whose response you want to validate.

3 Click **Validate XML** to open the validation tool.

4 In the XML Source box, choose a response parameter from the drop down list. For the complete response of the Web Service call, choose the response parameter—the name of the Web Service call step with a **_Response** suffix. If you manually saved a response argument for an individual parameter, you can select it too.

5 Select the **Checkpoints** option if it is not already selected.

6 Provide the expected values. Type in values for the desired rows. To load a record or replay snapshot, click the **Record** or **Replay** buttons. To define a parameter for the expected value, click the blue ABC button to the right of the row.

7 In the **Validate** column, select the arguments whose values you want to validate.

8 To perform an XPATH query, select the **Advanced Checkpoints** option to specify a query. For more information, see "XPATH Expressions" on page 149.

Adding an XML Validation Step

This section describes how to add XML validation steps to a script, without the intention of checking the complete response.

To add an XML validation step:

1 Open the XML Validation dialog box.

Click the **Validate XML** button in the upper toolbar.

2 Select an XML parameter.

Select a parameter containing the XML, saved in a previous step. If you already added a Web Service call, choose the **<Step_Name>_Response** parameter to validate the entire XML structure of the response. For more information, see "Checking for the Expected Response" on page 149. If there are no parameters, go back to the **Properties** tab and save an output parameter, or choose **Create/Select parameter** from the XML Source dropdown list.

3 Enable XSD validation.

To enable XSD validation, select **XML Schema**. If you disable this option, Service Test will only check for well-formed XML—not XSD compliance or expected values.

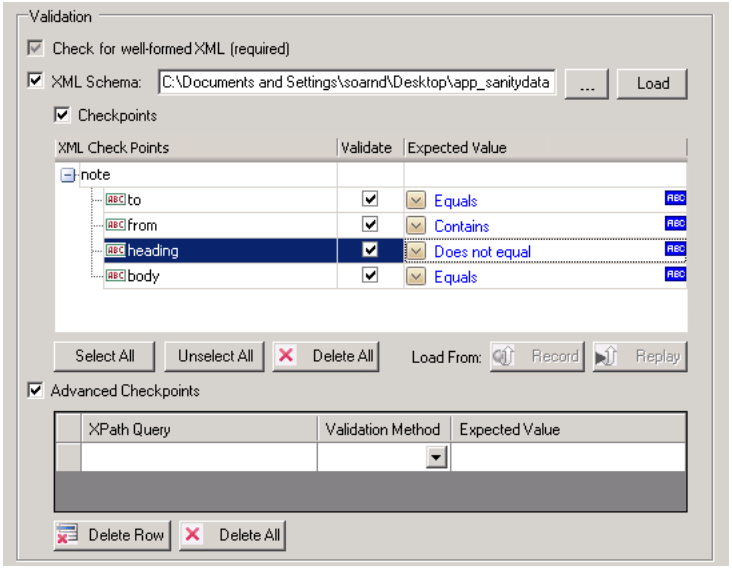
4 Select an XSD.

Specify the path or browse for the corresponding XSD file. Service Test loads the XSD grid into the dialog box. If you modify the XSD at its original location and you want Service Test to take the updated version, click **Load**.

5 Select the values to validate.

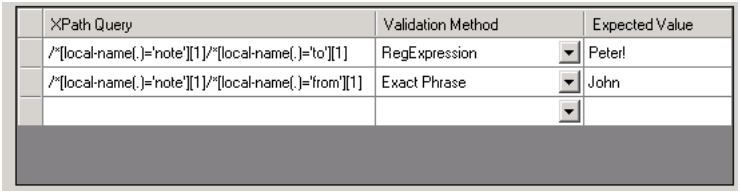
To check the values generated during replay against expected values, select the **Checkpoints** box. For each value you want to validate, select the checkbox in the **Validate** column, or click **Select All to** choose them all. If you don't want to use checkpoints, clear the **Checkpoints** box.

Specify a comparison operator for the selected elements, such as **Equals**, **Contains**, and so forth, and an expected value for each of the elements.



6 Specify values for Advanced Checkpoints (optional).

Click **Advanced Checkpoints**. In the Advanced Validation grid, specify an **XPath Query**, a **Validation Method** and the **Expected Value**. To copy an Xpath expression, right-click a value in the **XML Checkpoints** schema and copy it. Paste it in the **XPath Query** column.



7 Add the Validation step.

Click **OK** to add the XML Validation step. It appears in your script as **soa_xml_validate**.

8 Set the Extended logging options.

Click F4 to open the Run-Time settings. Select the **General:Log** node and select the **Extended log** option. You do not need to enable any of the sub options. Click **OK**.

9 Run the script.

Click F5 to run the script. View the Replay log to check for errors.

10 View the Test Results.

Select **View** > **Test Results** and expand the XML Validation step. You can view the errors, the source XML, and the checkpoint results.

Validating an XSD Schema

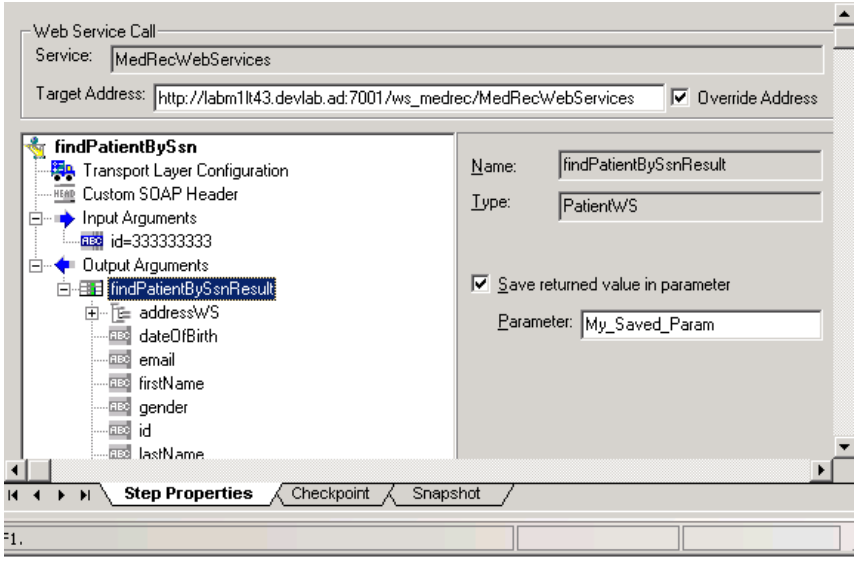
To perform complete XML validation, you select a parameter and indicate its corresponding XSD schema. When specifying a parameter for XML validation, you can use either an input or output parameter.

For input parameters, you use a parameter that you saved earlier. You can also select **Create/Select Parameter** from the **XML Source** dropdown list to create a new one. The parameter type should be XML parameter or a file type parameter where the file's content is XML.

Ideally, this parameter is XML with an ANY type element. By parsing the XSD, Service Test determines whether or not the WSDL is compliant with the schema.

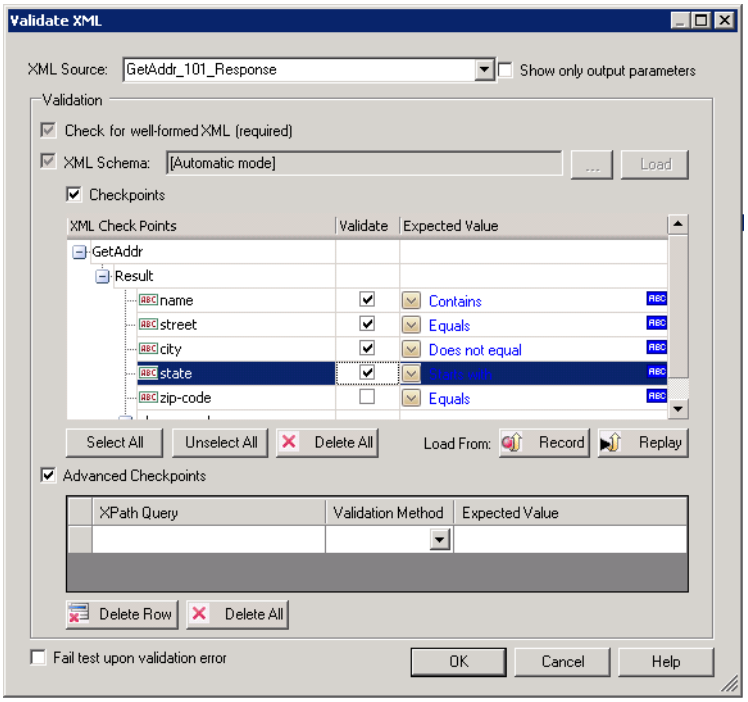
You create output parameters in the Properties tab, by enabling **Save returned value in parameter** while selecting the output value. If you select the top node of the output parameter, Service Test saves the value as an XML parameter.

The following example shows an output array saved to a parameter, **My_Saved_Param**.



For information about defining parameters, see "Parameterizing Scripts" on page 144.

In the XML Validation dialog box, you provide the saved parameter as the **XML Source**.



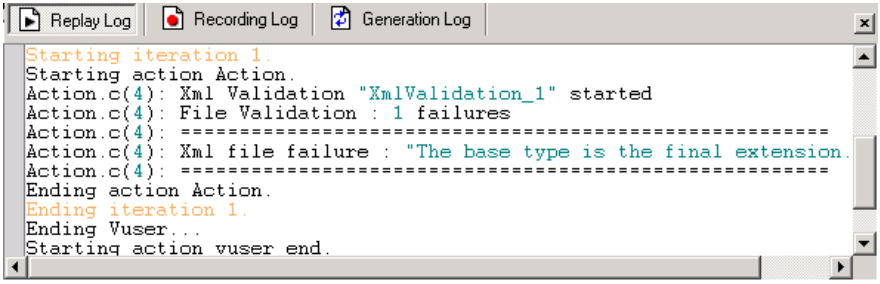
Full XML validation checks for well-formed XML, XSD compliance and checkpoint values. If you disable the **Validate XML** option, Service Test only checks for well-formed XML in the parameter—not the schema or checkpoints.

Viewing Validation Results

By adding an XML Validation step to your script, Service Test performs the validations in runtime. You can determine the validation results in several ways:

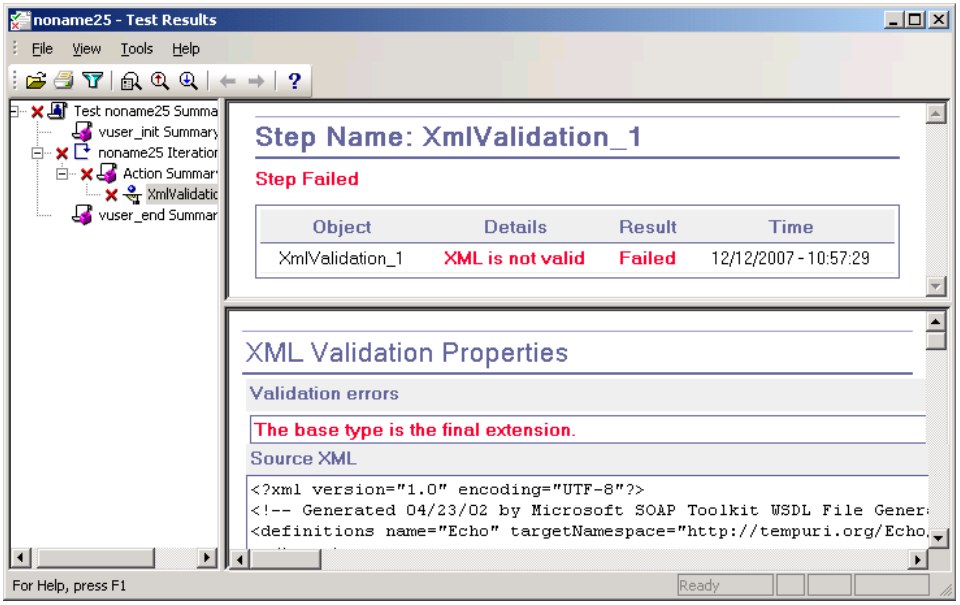
Replay Log

During replay, Service Test displays the results in the Replay log. We recommend that you enable the Extended Log Run-Time settings to obtain more details about the validation.



Test Results Report

The Test Results report shows the results of the validation step, along with the XML source code. To open the report, select **View** > **Test Results**.



Return Values

In addition to viewing the Replay log, you can evaluate the return values for **soa_xml_validate** to determine the outcome of the validation. The following table lists the possible return values:

Value	Description
0	Validation succeeded. XML is well-formed, compliant, and checkpoints are in order.
1	The parameter is empty or does not exist.
2	The XML is not well-formed.
3	The XML is well-formed, but not the XSD.
4	The XML does not comply with the XSD.

Value	Description
5	The values in the XML string do not match the values specified in the checkpoints' XPATH queries.
6	The XSD file could not be found.
7	An error, other than the ones listed in this table, occurred, such as an internal or unknown error.
8	The XSD file is not well-formed and the values in the XML string do not match the values specified in the checkpoints' XPATH queries.
9	XML failed validation against the XSD file and the values in the XML string do not match the values specified in the checkpoints' XPATH queries.

For more information about the function and its return values, see the *Online Function Reference* (**Help > Function Reference**).

Checking for Optional Element Values

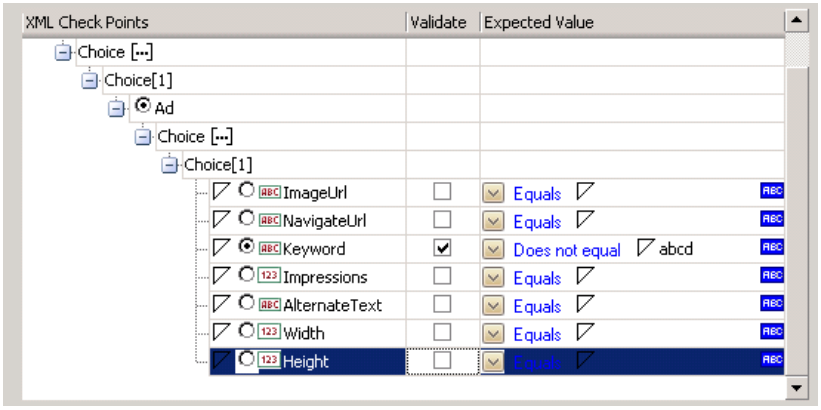
The XML Validation tool lets you check for the presence of an optional value. In some cases, you need to check that a value is present, while in others you need to verify that a value is not present.

When you include the optional element, by filling in the icon, replay checks that the service returned a value. When you exclude the optional element, by clearing the optional icon, replay verifies that the service did not return a value.

In the following example, the optional element was included and given the value of **abcd**.

During replay, Service Test checks for a value for the element. If it does not detect any value during replay, Service Test issues an error message indicating that it did not find a value for an element that was included. You can see detailed information in the Test Results report. For more information, see "Viewing Validation Results" on page 157.

In the following example, Service Test checks that the server did not return a value. If it finds a value for the element, an error message indicates that Service Test detected a value for an element that should have been excluded.



For both types of testing—inclusion and exclusion—you need to select the checkbox in the **Validate** column. Otherwise, the Validation tool will ignore the element.

Working with REST Web Services

You can use the XML validation for non-SOAP Web Services, such as REST. The following example illustrates the collaboration between a REST Web Service and XML Validation.

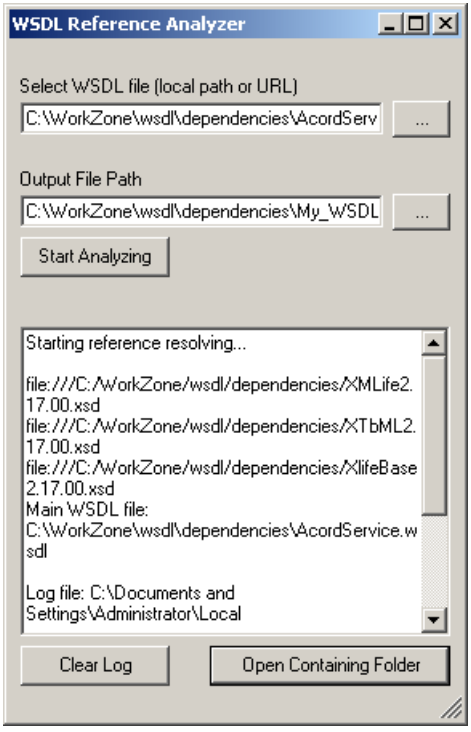
```
// Save the content between the commas to a Design Run type parameter
web_reg_save_param("WCSParam_Text1",
          "LB=",
          "RB=",
          "RelFrameId=1",
          "Search=Body",
          "IgnoreRedirections=Yes",
          LAST);

//The service request submitted as a URL
       web_url("xml",
"URL=http://ecs.amazonaws.com/onca/xml?Service=AWSECommerceService&Operat
ion=ItemSearch&AWSAccessKeyId=123456789ABCD&SearchIndex=Books&Keyword
s=Cortisol&Author=Tolstoy",
          "Resource=0",
          "RecContentType=text/xml",
          "Referer=",
          "Snapshot=t1.inf",
          "Mode=HTML",
          LAST);

//Use the parameter's value in the validation statement.
          soa_xml_validate ("StepName=XmlValidation_1",
          "Snapshot=t3e9876543214.inf",
          "XML={WCSParam_Text1}",
          "StopOnValidationError=0",
          "XSD=C:\\Bugs\\67571_Rest\\AWSECommerceService.xsd",
           BEGIN_CHECKPOINTS,
     …
          END_CHECKPOINTS,
          LAST);
```

To use a POST, PUT, or DELETE action, use **web_custom_request** as described in the *Online Function Reference*.

WSDL Referencing

Many WSDL files reference other files such as XSD and other XML files. Before running a script, you may want to determine what these files are and if they are available.

Service Test's WSDL Reference Analyzer checks the WSDL for dependencies, and lists them in the WSDL Reference Analyzer window.



The Analyzer places the WSDL and its dependent files in a a zip archive file. It saves the dependency information to a log file, listing its path in the Analyzer window.

To analyze a WSDL's dependencies:

1. Select **SOA Tools** > **WSDL Reference Analyzer**.
2. In the **Select WSDL file** box, indicate the location of the WSDL you want to analyze.

3 In the **Output file path** box, indicate a location for the zip file.

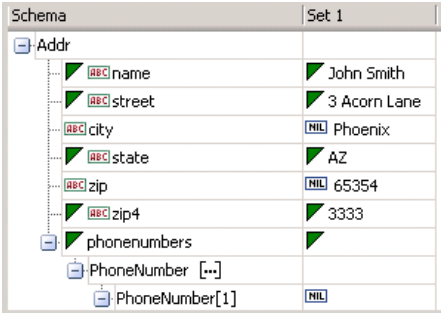
4 Click **Start Analyzing**. The Analyzer lists all of the dependencies in the output window along with their paths.

5 To empty the log, click **Clear Log**.

6 To view the folder containing the zip file, click **Open Containing Folder**.

XML Editing

Service Test provides an editor that displays the XML structure according to a WSDL or XML schema. The grid-like display lets you view the XML in its proper hierarchy and set values for each of the elements. The left column represents the schema, while the other columns show the XML that is generated and its properties.



While you can use the XML Editor as a standalone editor to format your XML code, it is also integrated with many of the Service Test features, such as parameterization and XML validation.

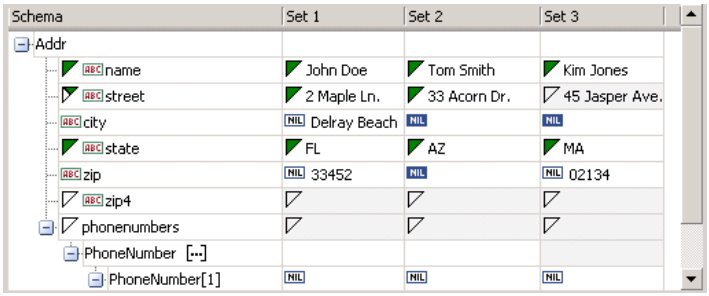
To set values for the elements, you can manually edit the XML or import XML files with the values.

The editor denotes optional elements with a small triangle in the upper left corner of the cell. A filled-in triangle indicates an included element. To exclude an optional element, click the small triangle to clear it.

When you exclude an element, the editor works dynamically and removes the entire element from the XML code. When you re-include the element, the editor adds it back into the XML.

Multiple Value Sets

Value sets are arrays that contain a set of values. You can create multiple value sets for your parameter and use them for different iterations or test runs.



You can use optional elements that will appear in one value set, but not in another. This allows you to vary the values you send for each of the iterations—some iterations can include specific array elements, while other iterations exclude them.

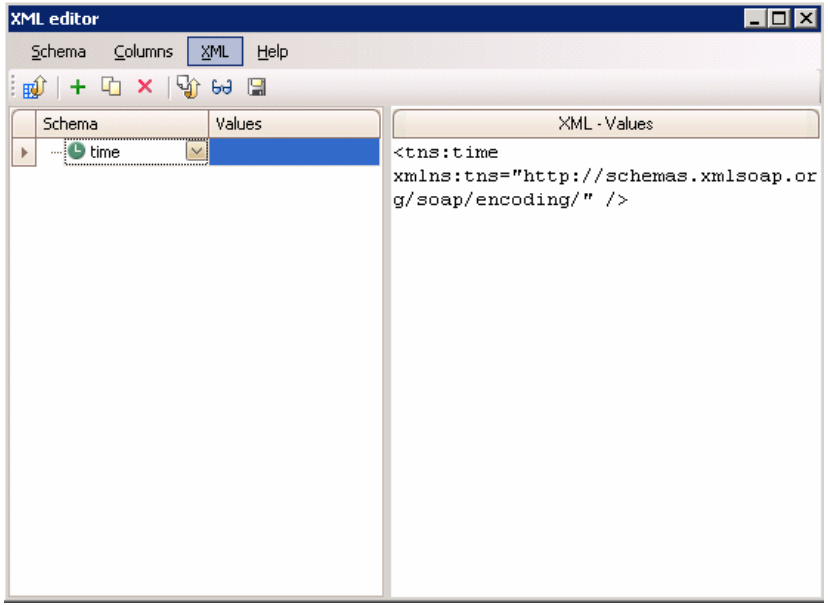
When using value sets, the number of array elements per parameter does not have to be constant. The exception to this is Choice, Derived, and <any> types, where the number of elements is fixed for all value sets.

To resize the columns of the value sets, click the divider in the column's title and drag your mouse to the desired width.

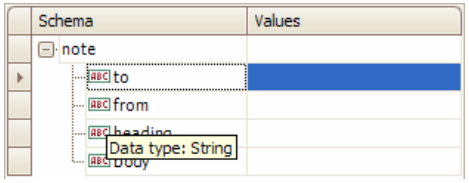
To open the XML editor:

1 Choose **SOA Tools** > **XML Editor**. The editor opens.

2 To load a schema, select **Schema** > **Load**. To load an XML file, choose **XML** > **Load.**

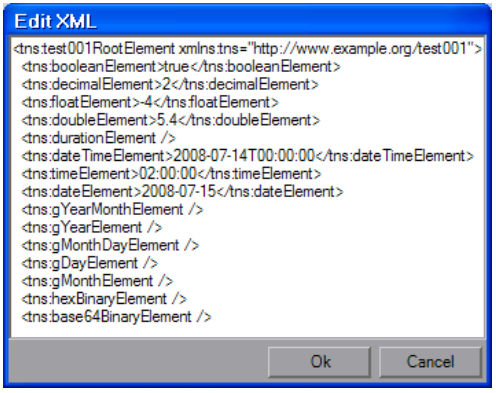


3 Fill in the **Values** column manually, or load values from an XML file. Click the **Load XML from file into the selected column** button. Place your mouse over the element's icon to discover its data type.



4 Choose **Columns** > **Add** or click the Add Column button to add another value set. Add the values manually or by loading XML as described in the previous step.

5 To edit the actual XML code, select a column and click the **Edit XML from the selected column** button. After the editing, click **OK**.



6 To duplicate a column, select it and choose **Columns** > **Duplicate** or click the **Duplicate Column** button.

7 To remove a column, select it and choose **Columns** > **Remove** or click the **Remove Column** button.

8 To save a column to an XML file, select it and choose **XML** > **Save** or click the **Save XML from the selected column** button.

Arrays

You can manipulate array elements and duplicate their whole structure.

Service Test also allows you to include or exclude individual array elements. When you exclude an array element, Service Test excludes it from the SOAP request.

This feature lets you dynamically control the content of the XML by excluding elements in certain value sets, while including them in others. When you exclude an array element, it automatically excludes all of its descendants.

The following example excludes an array element in several value sets.

Schema	Set 1	Set 2	Set 3
phone-numbers			
PhoneNumber [...]			
PhoneNumber[1]	◆	◆	◆
description	Home	Home	Home
phone-number	888-8888	111-1111	444-4444
PhoneNumber[2]	◆	[ ]	◆
description	Office	Office	Office
phone-number	666-6666	222-2222	999-9999
PhoneNumber[3]	◆	◆	[ ]
description	Mobile	Mobile	Mobile
phone-number	555-5555	333-3333	123-4567

To include or exclude array elements, click on the green diamond in the square brackets.

- If the green diamond is visible, it includes the element.
- If the green diamond is removed, it excludes the element and all of its descendants.

Duplicating Arrays

You can duplicate arrays within the grid. Service Test adds an array with an identical structure to the schema column and its value sets.

To duplicate an array, right-click the parent node and select **Duplicate Array Element**.

Schema	Set 1	Set 2	Set 3
phone-numbers			
PhoneNumber [...]			
PhoneNumber[1]	◆	◆	◆
description	Home	Home	Home
phone-number	888-8888	111-1111	444-4444
PhoneNumber[2]	◆	[ ]	◆
description	Office	Office	Office
phone-number	666-6666	222-2222	999-9999
PhoneN...	◆	◆	[ ]
...	...	bile	Mobile
...	...	3-3333	123-4567

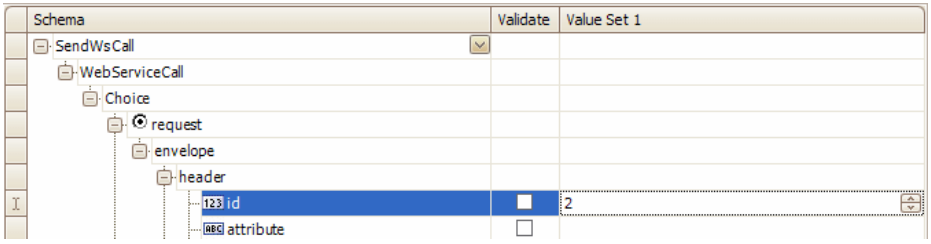
Duplicate Array Element
Remove Array Element

Data Types

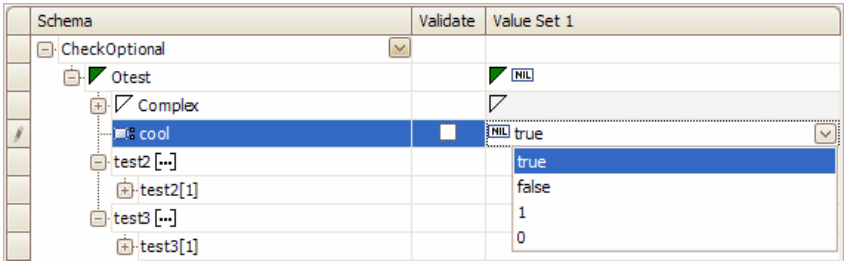
The XML editor lets you manipulate arrays of elements and sub-elements and assign them values.

To determine the actual data type, place your mouse over the icon adjacent to the element name, such as **123**, **ABC**, **B64**, and so forth. The mouseover popup indicates the data type.

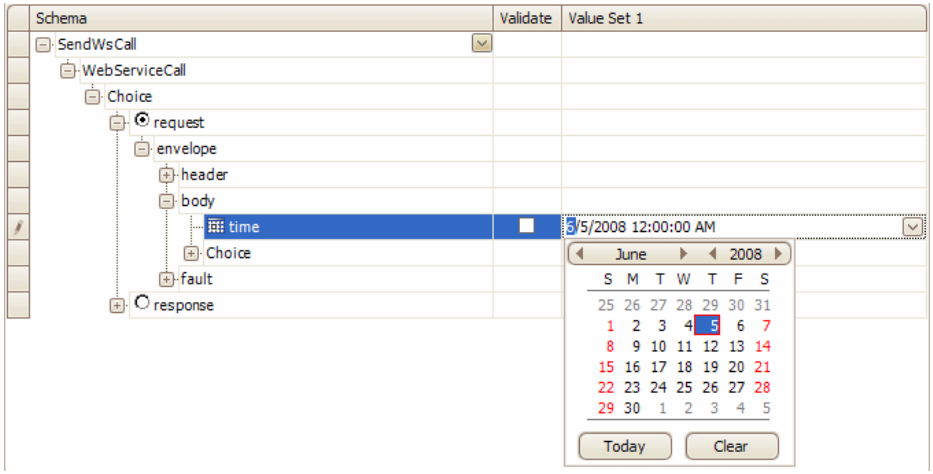
The grid recognizes element data types and provides an interface for setting the values. For example, for an **Int** type, the value cell contains a number scrolling control.



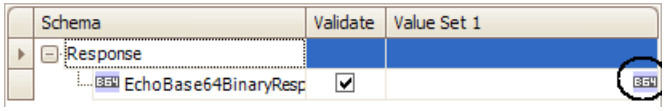
For boolean, it contains a list box with the values **true**, **false**, **1**, or **0**.



For a **Date** element, you can open a calendar to select a date.



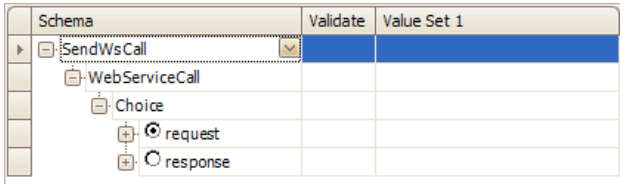
The schema indicates **Base64** elements with a **B64** button to the right of the value box. Click the button to open the Process Base 64 Data dialog box. For more information, see "Base 64 Encoding" on page 260.



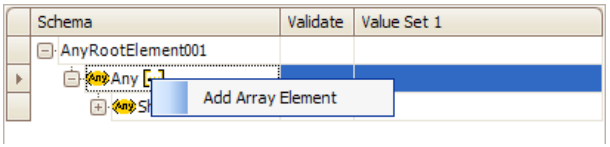
Any Type Elements

Using the XML editor, you can manipulate <**any**> type elements.

For simple, non-array, Any elements, the grid display shows the elements.

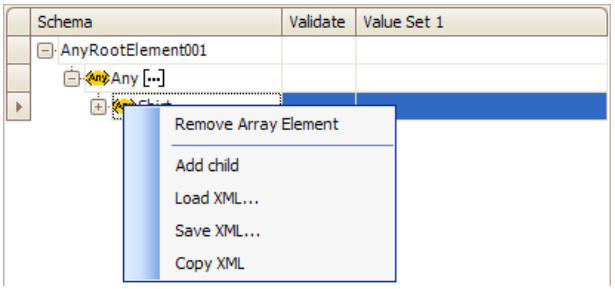


For an array of Any elements, you can manipulate the elements, their names, and their values. Use the right-click menu to add elements.



Manipulating Any Array Elements

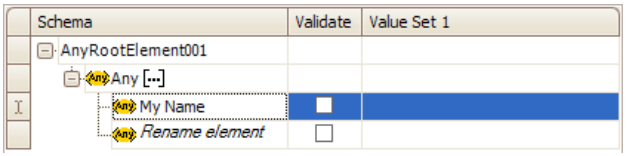
To manipulate Any array elements, open the right-click menu.



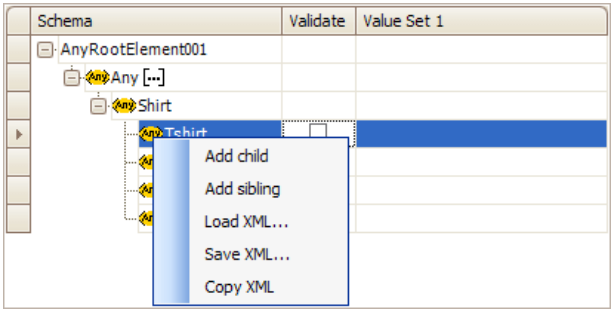
The right click menu provides some or all of the following options, depending on the location of the cursor:

- **Remove Array Element.** Removes the select array element.
- **Add child.** Adds a sub-element to the selected element.
- **Load XML.** Loads the element values from an XML file.
- **Save XML.** Saves the array as an XML file.
- **Copy XML.** Copies the full XML of the selected element to the clipboard.

To provide a name for the Any element, click the *Rename Element* text, and type in a name.

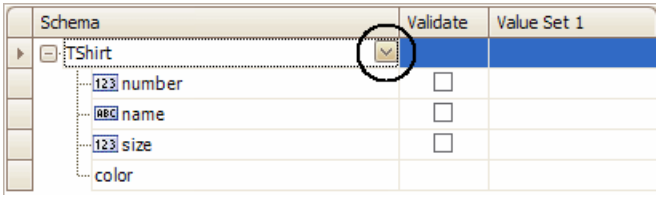


When manipulating a child element, you can use the right-click menu to add a sibling element.



Multiple Roots

If the schema for your Web Service has multiple root elements, you can select one of the elements. Click the small button adjacent to the root name to open the root drop down box.



XML Editor Integration

Besides serving as a standalone editor, the XML Editor is integrated into several of the Service Test components: Parameterization, XML Validation, and Service Emulation.

Parameterization uses the editor to display parameter elements and values as described in "Setting Properties for XML Parameters" on page 455.

XML validation uses the grid to display the XSD schemas. For details, see "Validating XML" on page 145.

The Service Emulation console uses the grid to display rules. For more information, see "Setting Rules" on page 404.

7

Web Services - Managing

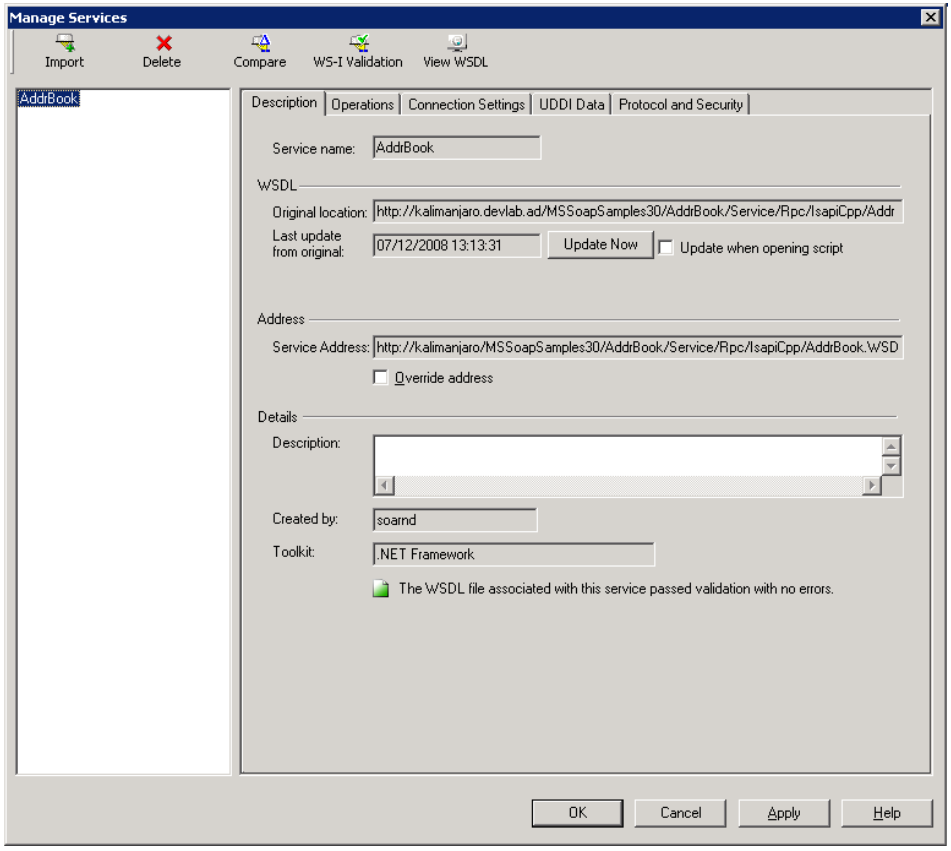
Service Test provides utilities that let you validate and manage the WSDL files associated with your service entries.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Managing Web Services Vuser Scripts

The Service Management window lets you manage a list of service entries for the current script. You can view and set the properties of each service entry.



You add service entries to the list by importing WSDL files. When you add a WSDL to the list, Service Test creates a working copy that it saves with the script—it is not global. Therefore, for each script that you create, you must import the desired WSDL files.

To view the copy of the locally saved WSDL in Internet Explorer, click the **View WSDL** button.

Note: All validations and modifications to WSDL files are done on the working copy. If you want to replace the imported WSDL file with a newer version, use the **Update Now** option described in "Description" on page 175.

To open the Service Management window, select **SOA Tools** > **Manage Services** or click the Manage Services toolbar button.

The Service Management window provides an interface for:

- Viewing and Setting Service Properties
- Importing Services
- Deleting Services
- Comparing WSDL Files
- Performing WS-I Validation
- Viewing WSDL Files

Viewing and Setting Service Properties

The Service Management window lets you view and modify information about the imported WSDLs. It shows details about the selected service entry in the following tabs:

- Description
- Operations
- Connection Settings
- UDDI Data

Description

The Service Management window's **Description** tab displays information about the service: WSDL, Endpoint Address, and Details. You can add annotations or notes about your service, in the Details area.

WSDL

The WSDL area provides information about the location of the WSDL, and the date that it was last imported.

- **Original location.** The original source of the WSDL file (read-only).
- **Service name.** The name of the Web Service.
- **Last update from original.** (For services **not** imported from Quality Center) The last date that the local copy was updated with a WSDL file from the original source.
  - To manually update the working copy of the WSDL, click **Update Now**. Service Test backs up the existing WSDL and updates it from the location indicated above.
  - To instruct Service Test to update the WSDL every time you open the script, select **Update when opening script**.
  - date of the last of the service from QC
- **Last updated from QC.** (For services imported from Quality Center) The last date that the service was updated from Quality Center.

Address

- **Service address.** An endpoint address to which the request is sent.

  If you want to override the endpoint specified in the WSDL file, select **Override address** and specify a different address in the **Service address** box.

  This is useful for implementing emulated services. Service Test uses the override address as targetAddress for the Web Service call. This overriding affects all Web Service calls. To use a different target address for a particular Web Service call, you specify it in that step's properties. For more information, see "Using Emulated Services in Scripts" on page 410.

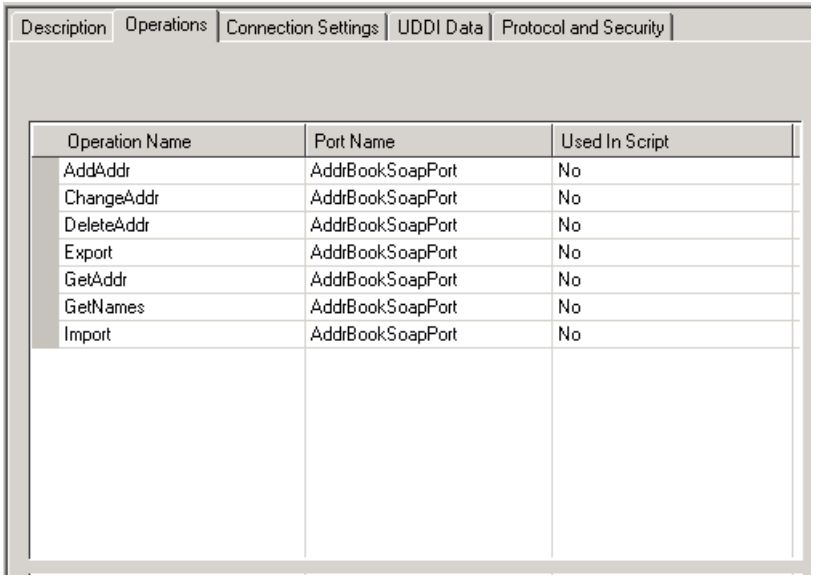
Details

- **Description.** A description of the Web service, taken by default from the WSDL file. This text area is editable.
- **Created by**. The name of the user who originally imported the service (read-only).

➤ **Toolkit.** The toolkit associated with the script. You set this before importing the first WSDL file.

Operations

Each of the imported services may define multiple operations. The **Operations** tab indicates which operations are being used for the service you selected in the left pane.



You can sort the list of operations by clicking on the relevant column. For example, to list the operations by name, click the **Operation Name** column. To list them in descending order, click the column name again. A small arrow indicates the sorted column. An upward arrow indicates ascending order, while a downward arrow indicates descending order.

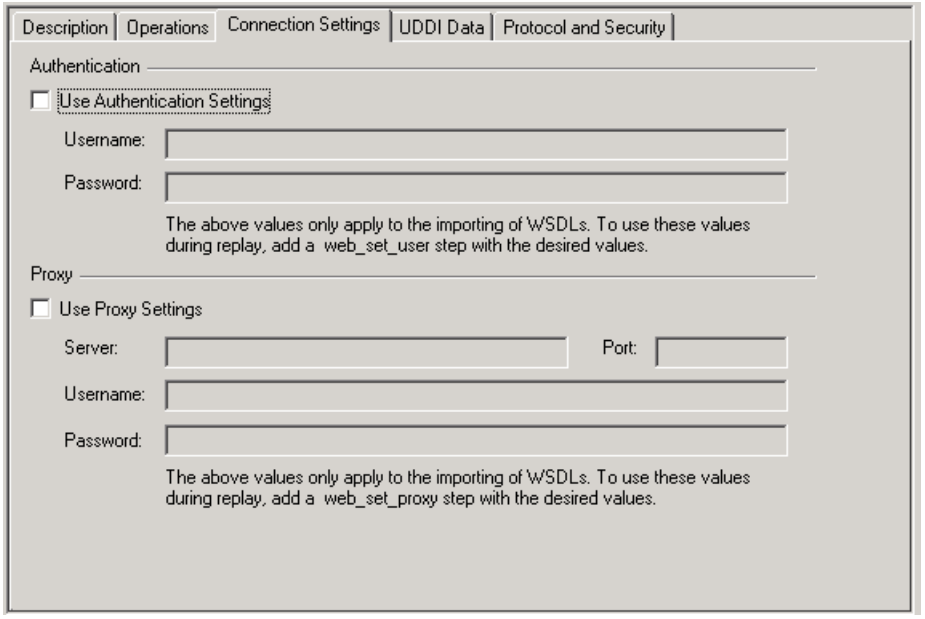
Connection Settings

In some cases WSDLs reside on secure sites requiring authentication. In certain instances, the WSDL is accessed through a proxy server.

Service Test supports the importing of WSDLs using security and WSDLs accessed through proxy servers. The following security and authentication methods are supported:

- SSL
- Basic and NTLM authentication
- Kerberos for the .NET and Generic toolkits

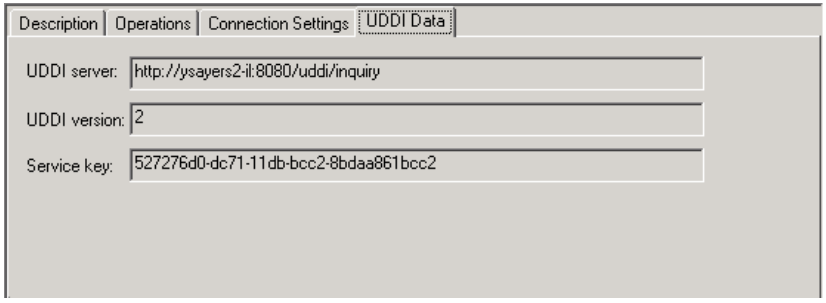
We recommend that you enter the authentication or proxy information while importing the WSDL. If however, the settings changed, you can modify them through the Service Manager's **Connection Settings** tab.



For more information about setting the connection information while importing the WSDL, see "Connection Settings" on page 182.

UDDI Data

You can view the details of the UDDI server for each service that you imported from a UDDI registry.



The read-only information indicates the URL of the UDDI server, the UDDI version, and the Service key.

For information about importing from a UDDI, see "Specifying a Service on a UDDI Server" on page 182.

Protocol and Security Settings

The Protocol and Security Settings tab shows the details of the security scenario applied to the script. If you did not choose a scenario, it uses the default **<no scenario>**.

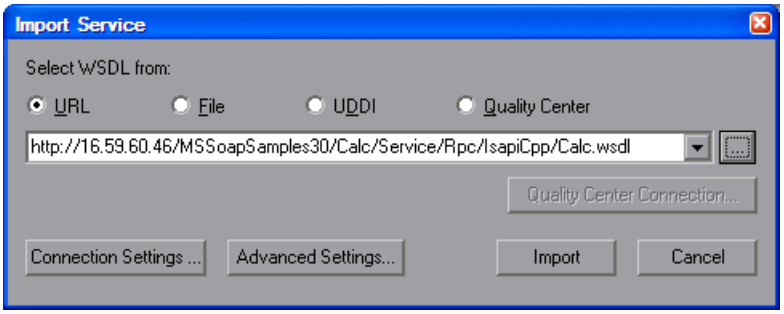
While this tab shows the information as Read-Only, click the **Edit Data** button to modify the security settings. For more information, see "Setting the Security Scenario" on page 330

Importing Services

Service Test lets you import services for the purpose of creating a high-level tests with Web Service Call steps. Typically, you begin creating a script by importing a WSDL file.

When importing a file, you specify the following information:

- **Source.** the source of the WSDL: URL, File, UDDI, or Quality Center
- **Location.** the path or URL of the WSDL, entered manually or by browsing
- **Toolkit.** the toolkit to permanently associate with all services in the script (only available for the first service added to the script)
- **Connection Settings.** authentication or proxy server information (optional)



If Service Test detects a problem with your WSDL when attempting to do an import, it issues an alert and prompts you to open the report. The report lists the errors and provides details about them.

Source

When specifying a WSDL, you can indicate the source:

- **URL.** The complete URL of the service.
- **File.** The complete path and name of the WSDL file.
- **UDDI.** Universal Description, Discovery, and Integration—a universal repository for services. For more information, see "Specifying a Service on a UDDI Server" on page 182.
- **Quality Center.** A service stored in the Quality Center repository. For more information, see "Choosing a Service from Quality Center" on page 183.

Service Test supports URL or UDDI paths that are secure, requiring authentication or accessed through proxy servers. For more information, see "Specifying a Service on a UDDI Server" on page 182.

Location

In the Location box, you specify the path or URL of the WSDL.

For the URL or UDDI options, make sure to insert a complete URL—not a shortened version. Click the **Browse** button to the right of the text box to open the default browser.

For a file, click the **Browse** button to the right of the text box to locate the WSDL on the file system.

For Quality Center, click the **Quality Center Connection** button to specify a server URL and to initiate a connection. For more information, see "Choosing a Service from Quality Center" on page 183.

Toolkit Selection

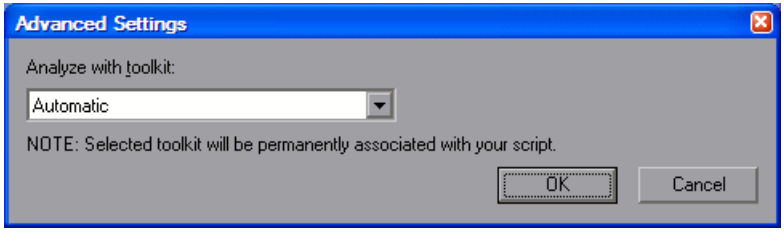
Choosing a toolkit instructs Service Test to send real client traffic using an actual toolkit—not an emulation. Once you select a toolkit, it becomes permanently associated with the script for all subsequent recordings, imports, and replays.

Service Test supports the .NET Framework with WSE 2 version SP3 and Axis/Java based Web Services Framework toolkits. Service Test imports, records, and replays the script using the actual .NET or Axis toolkit.

By default, Service Test uses automatic detection to determine the most appropriate toolkit.

To select a toolkit (for a new script only):

1 In the Import Service dialog box, click **Advanced Settings**.



2 Select a toolkit, or use the default **Automatic** detection.

Connection Settings

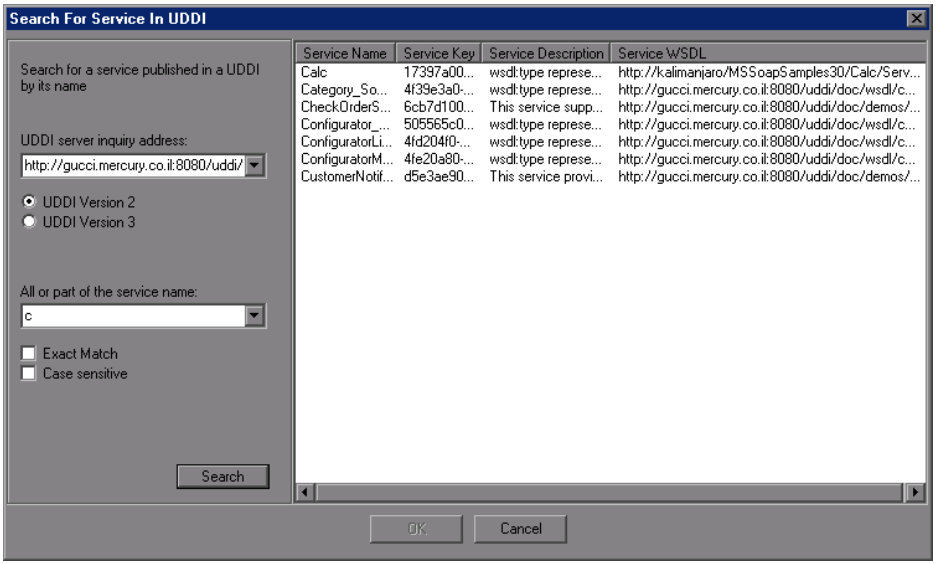
When importing WSDL files from a URL or UDDI, the WSDL may require authentication if it resides in a secure location. In certain cases, the access to the WSDL may be through a proxy server. Using the Connection Settings button, you can specify this information. For more information, see "Specifying WSDL Connection Settings" on page 184.

Specifying a Service on a UDDI Server

Service brokers register and categorize published Web Services and provide search capabilities. The UDDI business registry is an example of a service broker for WSDL-described Web Services.

Your Web Service client can use broker services such as the UDDI, to search for a required WSDL-based service. Once located, you bind to the server and call the service provider.

Click the **Browse** button to open the Search for Service in UDDI dialog box.



To search for a service on a UDDI:

1 In the **UDDI server inquiry address** box, enter the URL of the UDDI server.

2 Specify the UDDI version.

3 Specify the name or part of the name of the service. Select **Exact Match** or **Case Sensitive** to refine your search, if they are applicable. To perform a wildcard search, use the percent (%) character.

4 Click **Search**. Service Test displays all of the matching results.

5 Double-click on a service in the list to import it.

Choosing a Service from Quality Center

HP Quality Center with the Service Test Management add-on, integrates with Service Test. This integration allows you to store service entries and tests in Quality Center. You can also create and organize services according to your test requirements and test plan.

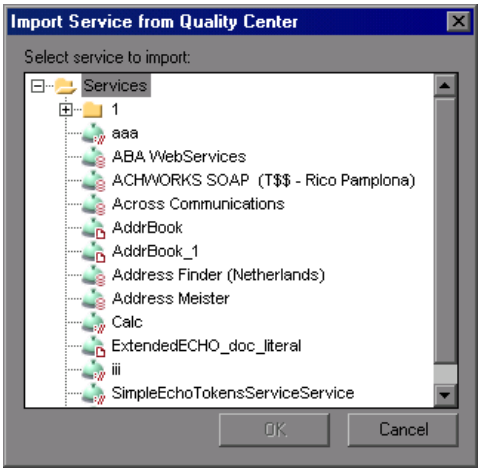
To specify a service from Quality Center:

1 In the Import Service dialog box, select **Quality Center**.

2 If you have not yet connected to your Quality Center project, click Quality Center Connection to open the Connection dialog box. For information on opening a Quality Center connection, see "Connecting to and Disconnecting from Quality Center" on page 118.

3 Click the **Browse** button to view the list of service entries saved in Quality Center.

4 Double-click on a service in the list to import it.



Specifying WSDL Connection Settings

Service Test supports the importing of WSDLs using authentication and WSDLs accessed through proxy servers.

Once you enter the security or proxy information, it remains with the WSDL, visible through the **Connection Settings** tab in the Service Management dialog box. If you enable the **Keep up to date** option to allow automatic synchronization, Service Test accesses the WSDL at its source using the authentication or proxy server settings.

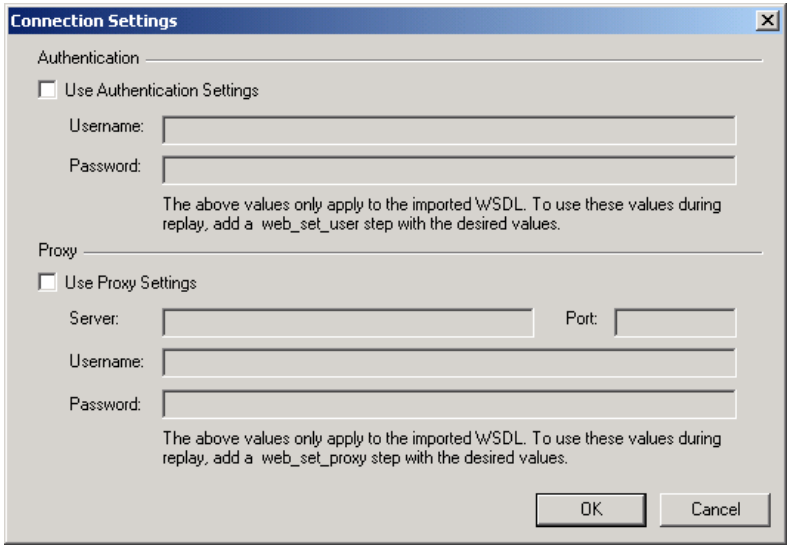
These connection settings only apply to the importing of a WSDL. To use authentication during replay for access to a server, use the **web_set_user** or **web_set_proxy** steps. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

To specify authentication or proxy information for importing:

1 Open the Import Service dialog box as you normally would, either with a new Web Service call, recording, or Traffic Analysis.

2 Select either the **URL** or **UDDI** option and specify a URL of the service to be imported.

3 In the Import Service dialog box, click the **Connection Settings** button to open the box.



4 In the Connections Settings dialog box, select the desired option: **Use Authentication Settings**, **Use Proxy Settings,** or both.

5 Specify the authentication details, and, for a proxy server, the name and port of the server. If you attempt to import the secure service before specifying the necessary credentials, Service Test prompts you to enter the information.

6 To update or modify the security settings, open the Service Manager and select the appropriate service in the left pane. Click the **Connection Settings** tab. Edit the required fields and click **OK**.

Deleting Services

You can delete service entries from the Service Management dialog box, when they are no longer required. If a service was updated, you can synchronize the WSDL from the source—you don't need to delete it and reimport the service.

Before deleting a service, make sure that it is not required for your script. If you created a script based on a specific service and you then attempt to delete it, Service Test warns you that the deletion may affect your script. Deletions cannot be undone.

To delete a service, select it from the list of services and click the **Delete** button.

Comparing WSDL Files

When you import a WSDL file, Service Test makes a working copy and saves it along with the script. This saves resources and enables a more scalable and stable environment.

It is possible, however, that by the time you run the script, the original WSDL file will have changed. If you run the script, the test results may be inaccurate and the script may no longer be functional. Therefore, before replaying a Web Services script that was created at an earlier date, you should run a comparison test on the WSDL file.

Service Test provides a comparison tool which compares the local working copy of the WSDL file with the original file on the file system or Web server.

If the differences are significant, you can update the WSDL from the original copy using the **Synchronize** option in the Service Management dialog box.

Service Test also has a general utility that allows you to compare any two XML files. For more information, see "Comparing XML Files" on page 190.

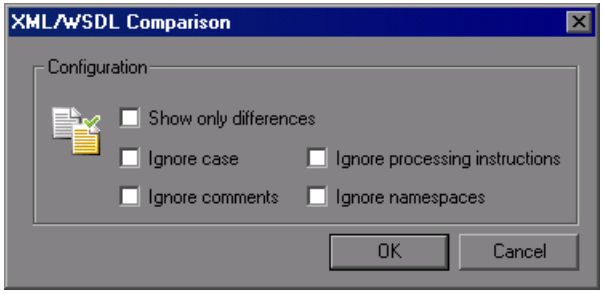
Setting WSDL/XML Comparison Options

Service Test offers the following options when comparing the local and global copies of the WSDL documents, or the revisions of an XML file:

- **Show only differences.** Show only lines with differences. Do not show the entire document.
- **Ignore case.** Ignore case differences between the texts.
- **Ignore comments.** Ignore all comments in the texts.
- **Ignore processing Instructions.** Ignore all texts with processing instructions.
- **Ignore namespaces.** Ignore all namespace differences.

To configure the comparison options:

1 Configure the comparison settings. Select **SOA Tools** > **SOA Settings** > **XML/WSDL Compare.** The WSDL Operations Options dialog box opens. Select the desired options.



2 Click **OK**.

Note: The comparison option settings apply to both WSDL comparisons from within the Service Management window, and for XML comparisons accessed from the **Tools** menu.

Comparison Reports

Service Test lists the differences between the files in a Comparison report.

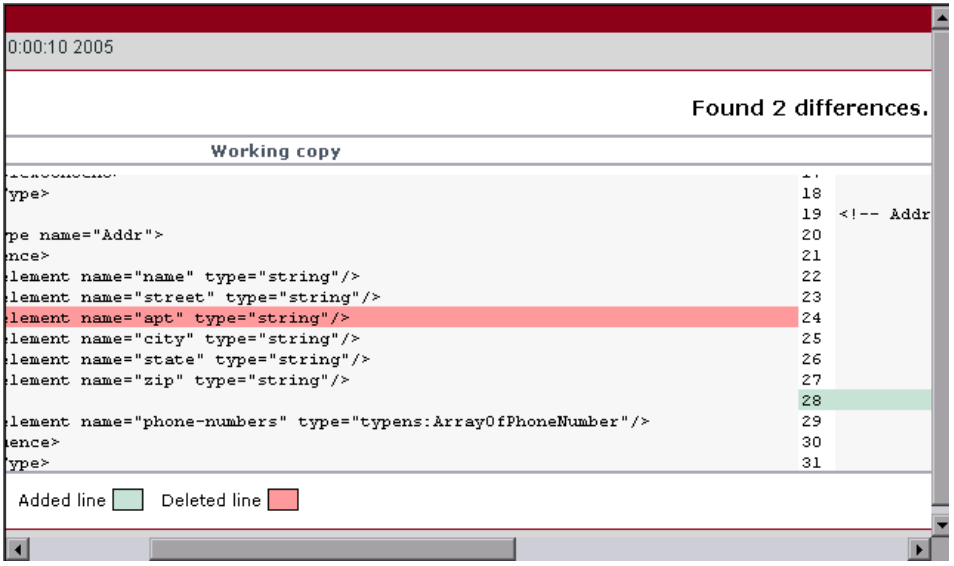
In WSDL Comparison reports, there are two columns— **Working Copy** and **Original File.** The Working Copy is the WSDL stored with the script, while the Original File is the WSDL at its original location—a network file path or a URL.

In XML Comparison reports, each column displays the path of an XML file.

The Comparison report uses the following legend to mark the differences between the two files:

- **Yellow.** Changes to an existing element (shown in both versions).
- **Green.** A new element added (shown in the original file copy).
- **Pink.** A deleted element (shown in the working copy).

In the following example, line 24 was deleted from the original copy and and line 28 was added.

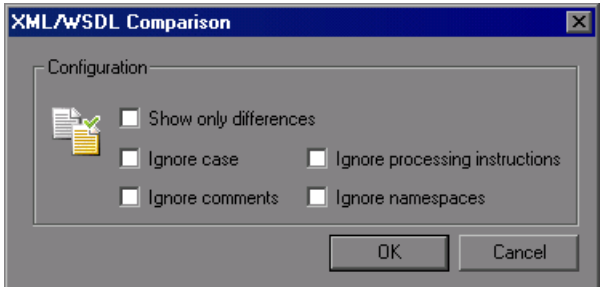


Running a WSDL Comparison

After running a file comparison, you can decide whether to ignore the changes, if they exist, or reload the WSDL file.

To compare WSDL files:

1 Configure the comparison settings. Select **SOA Tools** > **SOA Settings** > **XML/WSDL Compare.** The XML/WSDL Comparison dialog box opens. Select the desired options.



2 Open the Service Management window. Select **SOA Tools** > **Service Management** or click the **Manage Services** toolbar button

3 Select the service upon which you want to perform a comparison. You can only run the comparison on one service at a time.

4 Click **Compare**. The WSDL Comparison Report opens.

5 Scroll down through the file to locate the differences.

If you find differences between the two files and you want to update Service Test's working copy of the WSDL file, click on the WSDL file in the tree in the left pane. Select **Refresh file from global copy** from the right-click menu. This copies the current version of the WSDL into the script's WSDL directory.

6 To close the WSDL Comparison Report window, select **File** > **Exit**.

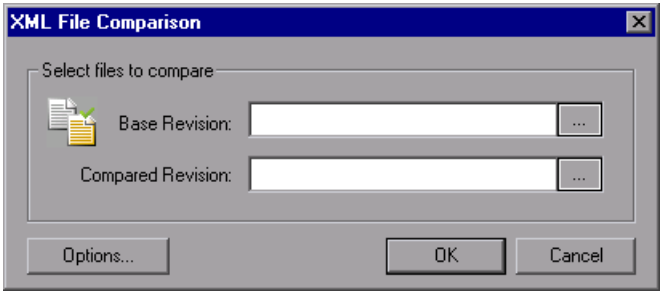
Comparing XML Files

Service Test provides a utility that lets you compare two XML files.

You can specify what differences to ignore, such as case or comments. For additional information about the comparison options, see "Setting WSDL/XML Comparison Options" on page 186.

To compare two XML files:

1 Select **Tools** > **Compare XML Files**. The XML File Comparison dialog box opens.



2 Click the Browse button to the right of the **Base Revision** box to locate the original XML file.

3 Click the Browse button to the right of the **Compared Revision** box to locate the newer XML file.

4 Click **OK**. Service Test opens the XML Comparison Report window.

For information about the Comparison report, see "Comparison Reports" on page 188.

Performing WS-I Validation

WS-I (Web Services Interoperability) is an organization that created standards for Web Services, to promote compatibility across platforms, operating systems, and programming languages. For more information about WS-I, see http://ws-i.org.

If your Web Service is WS-I compliant, it means that it meets the WS-I standards and it should be suitable for multiple platforms and languages.

To claim that your Web Service is WS-I compliant, you need to validate it using a WS-I testing tool, distributed by the WS-I organization. Service Test integrates with the testing tool after you install it.

Download the tool that matches your platform: C# with .NET or Sun Java. The recommended tool for use with Service Test is Java. After you download the zip file, extract the files to a local drive, maintaining the original directory structure, **wsi-test-tools**.

You can download the tools from WS-I's Web site:

- **C#.** http://www.ws-i.org/Testing/Tools/2005/06/WSI_Test_CS_Final_1.1.zip
- **Java.** http://www.ws-i.org/Testing/Tools/2005/06/WSI_Test_Java_Final_1.1.zip

If you define an environment variable called **WSI_HOME** with a value of the path of **wsi-test-tools**, Service Test will recognize the path and automatically activate WS-I validation for WSDL. If you do not define an environment variable, you can manually enable WS-I validation and browse to the location of the WS-I testing tool.

Service Test supports WS-I Validation in several areas:

- **Service Management**. Manually validate one or more WSDLs from the Service Management window. For more information, see "Validating WSDLs" on page 193.
- **WSDL WS-I Validation Calls.** Insert validation steps into your script to automatically validate the WSDL every time your run the script. For more information, see "Adding Validation Calls" on page 193.
- **SOAP validation per step.** Validates the SOAP message for a specific step. For more information, see "Validating SOAP Messages" on page 198.
- **SOAP validation per iteration.** Validates the SOAP message for the entire iteration. For more information, see "Validating SOAP Messages" on page 198.

Note: Service Test performs WS-I validation against the original WSDL document—not against the local copy.

Configuring WS-I Validation

Service Test lets you validate a WSDL or the SOAP message sent by the service during record or replay for compliance with the WS-I standard.

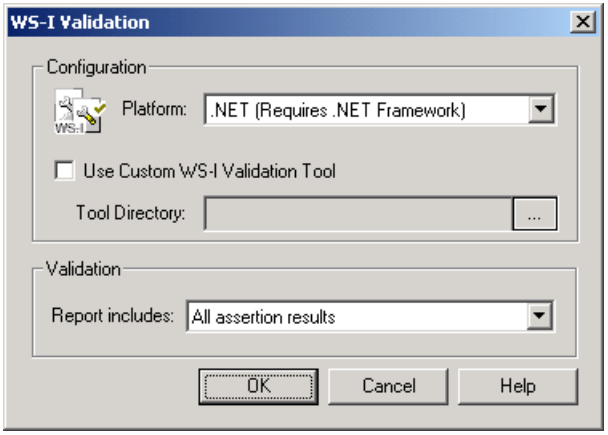
Service Test contains a built-in tool for performing WS-I validation. If you prefer a custom testing tool, you can override the built-in tool.

You can select the type of messages to include in the report:

- **All assertion results**. Show all results.
- **Assertions with the result "Failed"**. Show only the assertions with a "Failed" result status.
- **Assertions with a Result different than "Passed"**. Show the assertions that did not have a "Passed" result status.

To set up WS-I validation:

1 Select **SOA Tools** > **SOA Settings** > **WS-I validation**. Alternatively, click the **WS-I Validation** button in the Service Management window. If this is the first time you are doing a validation, the WSI-validation box opens.



2 Select the **platform** of your Web Service: Microsoft .NET, which requires the .NET Framework, or Java, which requires JRE.

3 Accept the built-in WS-I validation tool, or select the **Use custom WS-I Validation tool** option and specify the tool directory for example, **wsi-testing-tool**.

4 In the **Report includes** box, indicate which messages to include in the report.

5 Click **OK**.

Validating WSDLs

You can validate your WSDL file before or after creating a script.

To validate a WSDL file:

1 Open the Service Management window. Select **SOA Tools** > **Service Management** or click the Manage Services toolbar button

2 Select the services you want to validate. Use the **Ctrl** button to select multiple services.

3 Click **WS-I Validation**. If this is the first time you are doing a validation, you need to configure the WS-I settings as described in "Configuring WS-I Validation" on page 192.

4 Click **OK**. The WS-I Validation Report opens.

Adding Validation Calls

You can manually add validation call to check a specific service every time you replay the script. We recommend that you make a separate test script to perform the validations since if the validation fails, the script execution stops.

When you run your script, Service Test runs the validation and submits a message to the Output log indicating whether the WSDL is WS-I compliant. It also provides the location of the validation report.

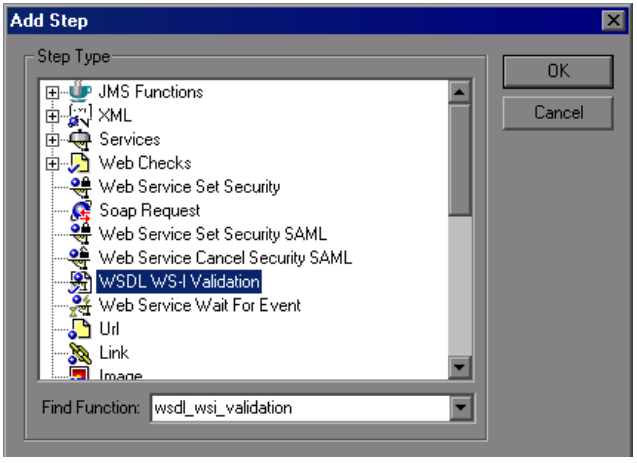
```
Action.c(33): WSDL WS-I validation for service "Calc" started
Action.c(33): Error: The service is not WS-I compliant
Action.c(33): For more information, see report in "C:\Program
    Files\Mercury\Mercury SOA Tester\scripts\My Test
   3\WSDL\WSIValidationReport_Calc_CalcSoapBinding.xml"
```

If you are running several iterations of the script, we recommend that you place the validation step in the **init** section. This will allow Service Test to validate the service only once, instead of upon every iteration.

To manually add a validation step to your script:

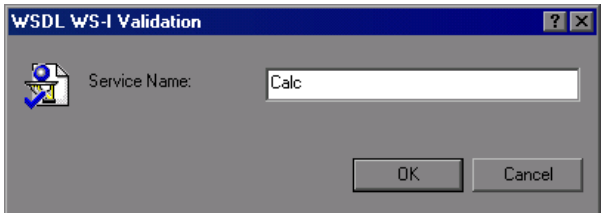
1 In Tree view, click at the appropriate location in your script.

2 Select **Insert** > **New Step** to open the Add Step dialog box.



3 Select **WSDL WS-I Validation** and click **OK**.

4 Provide the name of the service you want to validate.



In Script view, you can add the **wsdl_wsi_validation** function. For additional information about these functions, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

During replay, you can see the results in the Replay log or Test Results report.

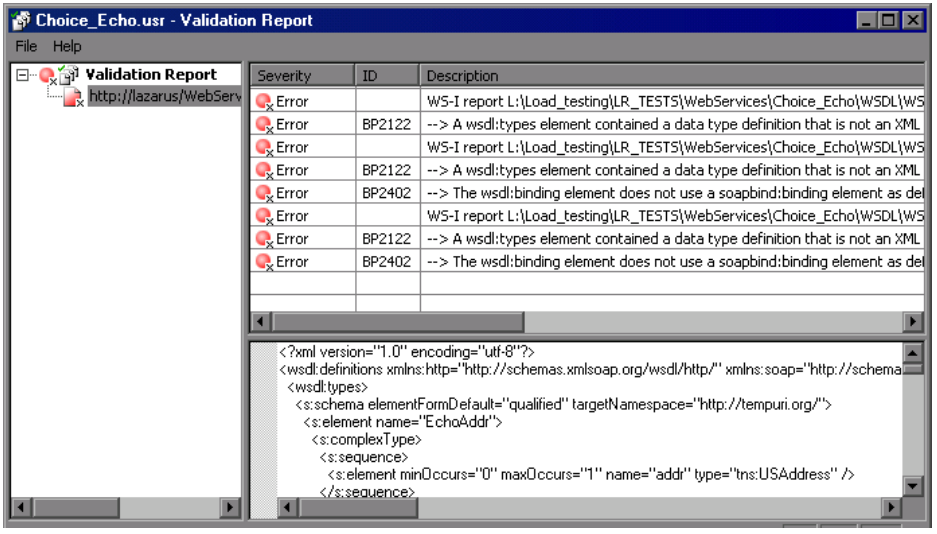
Validation Reports

WSDL Validation Reports provide information about your WSDL file. A green icon in the **Severity** column implies that there are no problems, yellow indicates warnings, and red marks an error.

You can view several types of reports to help you understand compliancy issues: WSDL Validation Report and the native WS-I report.

WSDL Validation Report

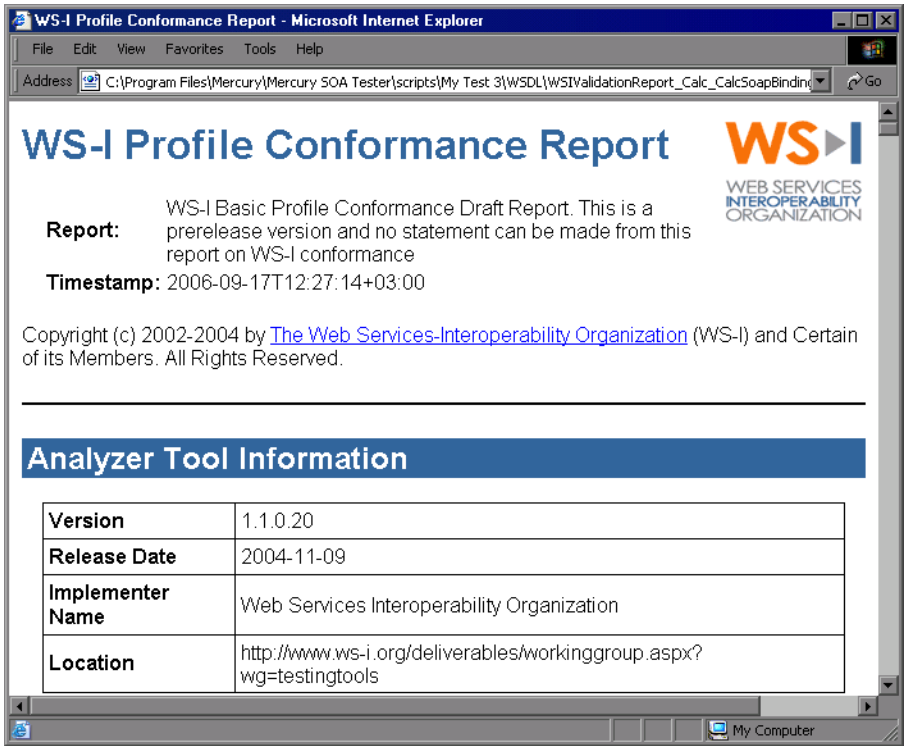
If there is an error in an WSDL file, Service Test displays its details in the right pane.



To save the report to an HTML file, select **File > Export as HTML**.

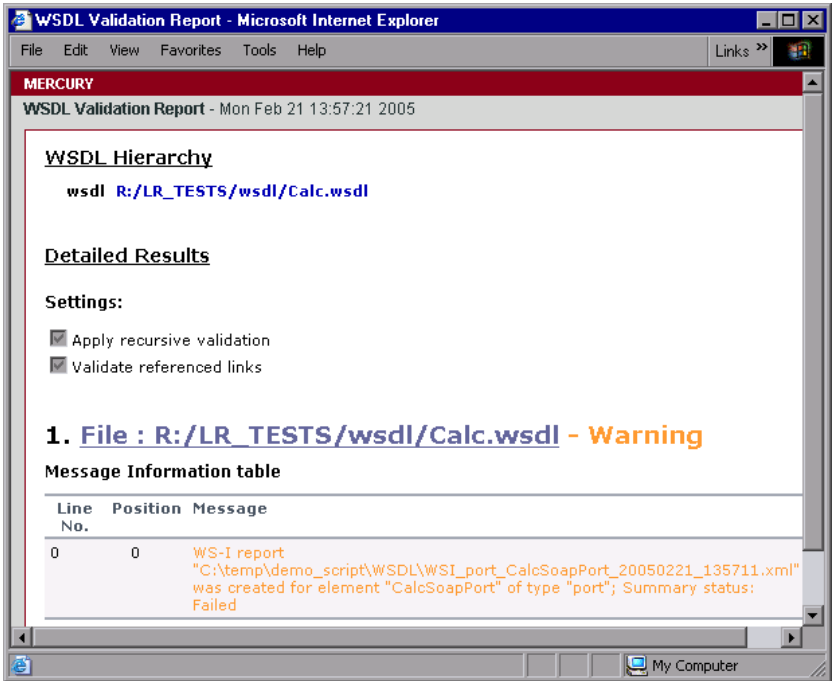
Native WS-I Reports

Native WS-I reports are WS-I basic profile conformance reports that provide details as to why the WSDL complies or does not comply with the WS-I standards.



To view the native WS-I report, select the WS-I error in the Validation report and select **Open WS-I Report** from the right-click menu. A browser opens with the native WS-I report describing the error in detail.

You can also save the report to an HTML file for later viewing.



To create and save an HTML summary report of the validation:

1 Select **File** > **Export to HTML**. Service Test opens a browser with an HTML report of the Validation results.

2 Select **File** > **Save As** from the browser window to save the HTML file.

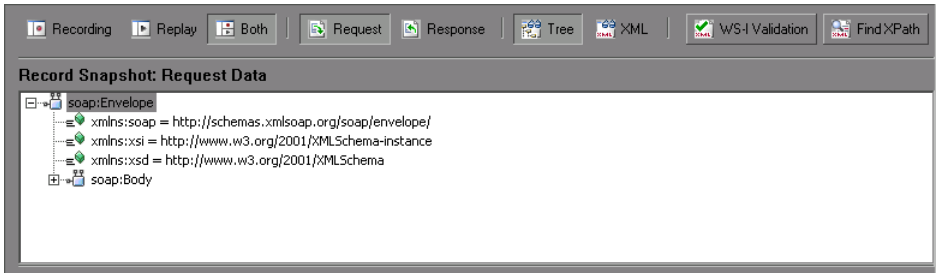
Validating SOAP Messages

Service Test lets you validate the SOAP messages of the various responses.

To validate the SOAP message for a specific step:

1 In Tree view, select the step you want to validate.

2 Select the snapshot tab.



3 Click the **WS-I Validation** button to begin the validation.

You can also validate the SOAP messages of an entire iteration—either Record or Replay.

- To validate the recorded snapshot, select **SOA Tools** > **SOAP Validation** > **Record Validation**.
- To validate the current replay snapshot, select **SOA Tools** > **SOAP Validation** > **Replay Validation**.

Service Test opens the Validation report with the validation data. For more information, see "Validating WSDLs" on page 193.

Viewing WSDL Files

The Service Management window lets you view the WSDL in your default browser.

To view a WSDL:

1 Select it in the left pane.

2 Click the **View WSDL** button in the Service Management window.

8

Web Services - Adding Script Content

You use Service Test to create a script to test your Web Services through recording, manually adding calls, or analyzing server traffic.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Adding Content to Web Services Scripts

Web Services scripts let you test your environment by emulating Web Service clients.

After creating an empty Web Services script, as described in "Creating an Empty Web Services Script" on page 140, you add content through one of the following methods: recording, manually inserting Web Service calls, importing SOAP, or by analyzing server traffic.

To create scripts automatically, run the SOA Test Generator using the wizard to add content. For more information, see Chapter 9, "Web Services - Automatic Test Generator."

For information on creating scripts by	See
recording	"Recording a Web Services Script" below
manually inserting Web Service calls	"Adding New Web Service Calls" on page 207
importing SOAP requests	"Importing SOAP Requests" on page 210
analyzing server traffic	Chapter 10, "Web Services - Server Traffic Scripts"
running the SOA Test Generator	Chapter 9, "Web Services - Automatic Test Generator"

Recording a Web Services Script

By recording a Web Services session, you capture the events of a typical business process. If you have already built a client that interacts with the Web Service, you can record all of the actions that the client performs. The resulting script emulates the operations of your Web Service client. After recording, you can add more Web Service calls and make other enhancements.

Specify Services for Recording

When you record an application, you can record it with or without a Web Service WSDL file. We recommend that you record with a WSDL when possible.

If you include a WSDL file, Service Test allows you to create a script by selecting the desired methods and entering values for their arguments. Service Test creates a descriptive script that can easily be updated when there are changes in the WSDL.

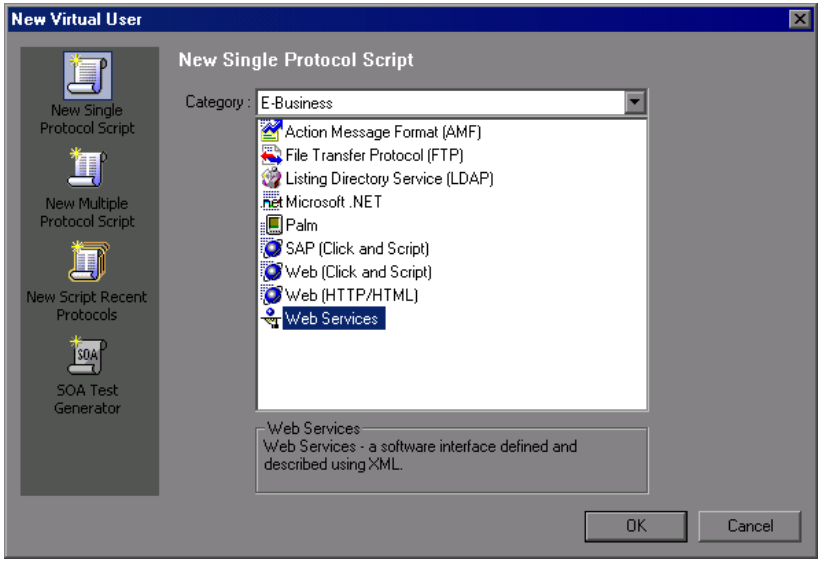
If you do not specify a WSDL file (not recommended), Service Test creates a test with SOAP requests instead of Web Service call steps. If you create a script without importing a service, Service Test creates a **soap_request** step whose arguments are difficult to maintain.

To create Web Services script through recording:

1 Create an empty script.

Select **File** > **New** to open the New Virtual User dialog box.

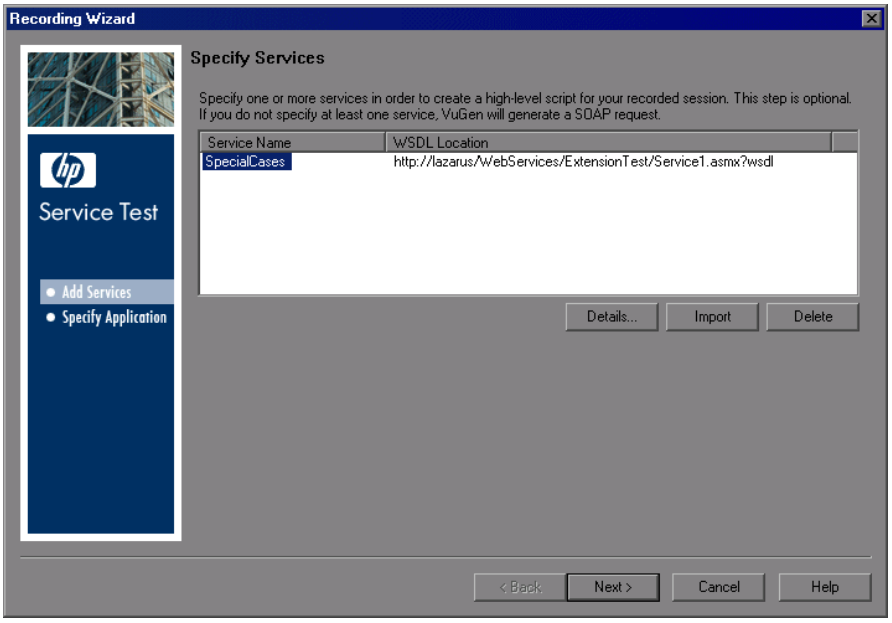
For a single protocol script, click **New Single Protocol Script** in the left pane. Select **Web Services** protocol from the E-Business category. Click **OK**.



If you need to record several different protocols, such as Web Services and Web, click **New Multiple Protocol Script** in the left pane and specify the desired protocols. Click **OK**.

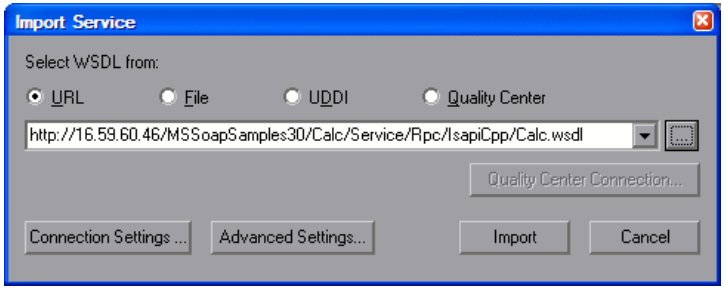
2 Begin the recording process.

Click the **Start Record** button or Ctrl+R to open the Specify Services screen.



3 Add a service to the list.

To produce a high-level Web Service script, add one or more services using the **Import** button. If the WSDL of the recorded Web Service is available, we recommend to import it here. If you create a script without importing a service, Service Test creates a **soap_request** step. The Import Service dialog box opens.



a Select a source and location for the WSDL.

b Select a toolkit. The toolkit you select is permanently associated with the script. For more information, see "Importing Services" on page 179.

c Click **Import**.

Repeat the above step for each WSDL you want to import.

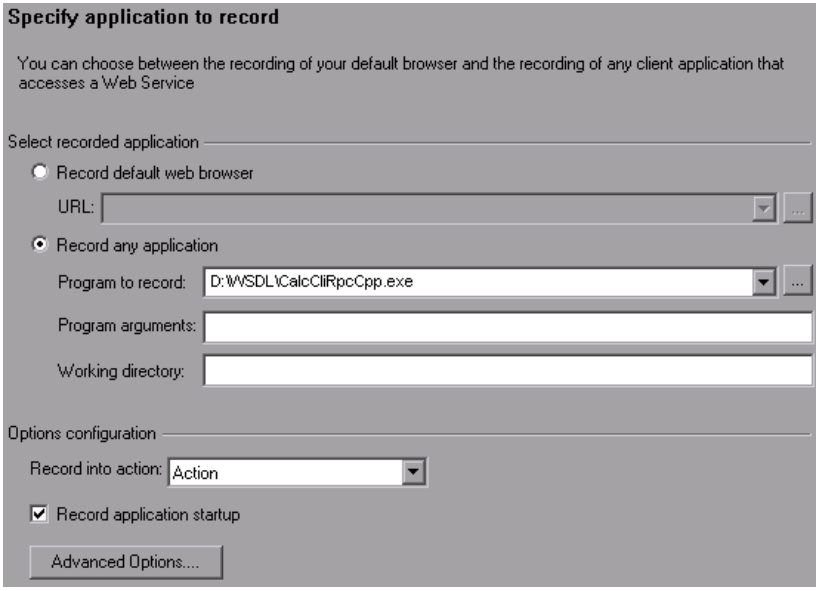
4 Enter details for the Web Service.

Click **Details** to open the Service Manager window and view the details of the Web Services that you added. For more information, see Chapter 11, "Web Services Call Properties."

In the next step, you select an application to record.

Selecting an Application to Record

In this screen you specify the application to record. You can record a browser session or client application.



1 Specify the recording details.

- **Record default Web Browser.** Records the actions of the default Web browser, beginning with the specified URL. Select this option where you access the Web Service through a Web-based UI.
- **Record any application.** Records the actions of a your client application. Specify or browse for the path in the **Program to record** box. Specify any relevant arguments and working directories.

2 Configure options.

- **Record into action.** The action in which to generate the code. If there are startup procedures that you do not need to repeat, place them in the **vuser_init** section. During recording you can switch to another section, such as **Action**.

- **Record application startup.** Records the application startup as part of the script. If you want to begin recording at a specific point, not including the startup, clear this check box.
- **Advanced Options.** Opens the Recording Options dialog box, allowing you to customize the way in which the script is generated. For more information, see the section on setting Script Generation preferences.

3 Click Finish.

Service Test opens the application and begins recording. Perform the desired actions within your application and then press the **Stop** button on the floating toolbar. Service Test generates **web_service_call** functions, or **soap_request** functions if you did not import a WSDL.

After recording, you can enhance your script with additional service calls and parameterization.

For more information, see "Getting Started with Web Services Vuser Scripts" on page 138.

Adding New Web Service Calls

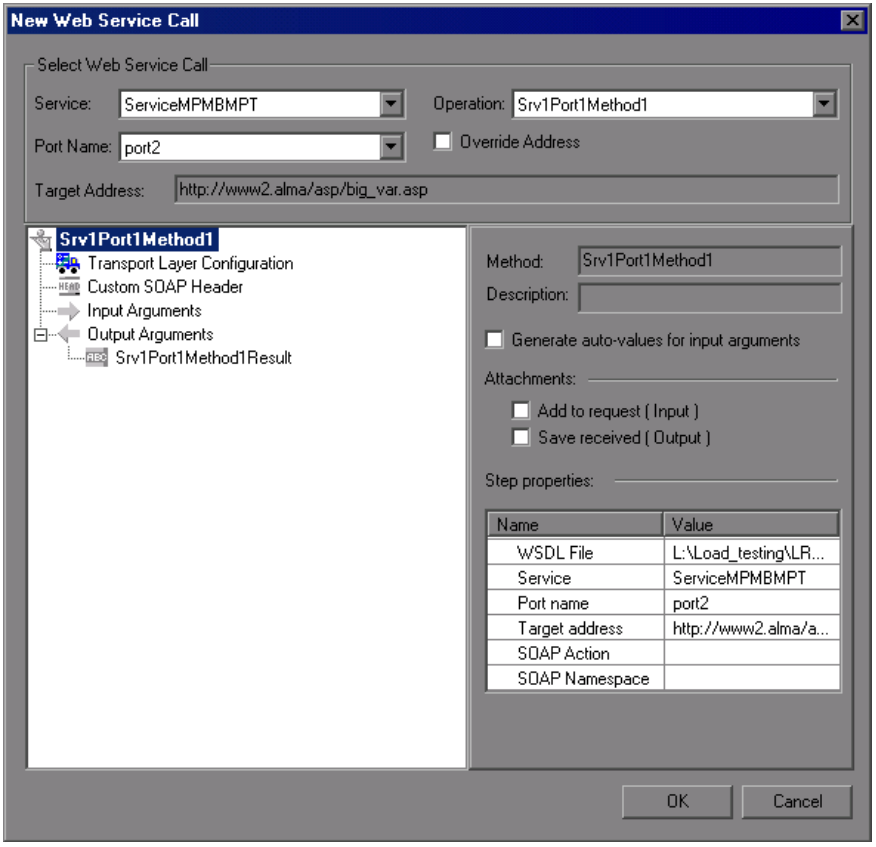
You can add new Web Service call functions in both Tree and Script views.

To add a Web Service call:

1 Click the cursor at the desired location in your script.

2 Click the **Add Service Call** button. The New Web Service Call dialog box opens. Select the desired **Service**. If this is a new script and you did not yet import a WSDL, do so at this point. For more information, see "Importing Services" on page 179.

3 Select an **Operation**. For services using multiple ports, select a **Port Name** for the operation. This lets you differentiate between identical operations performed on separate ports.



4 To specify a target address other than the default for the active port, select **Override address** and enter the address.

5 To provide sample input values for the service, click on the highest level node (the service name) and select **Generate auto-values for input arguments**. Service Test adds sample values and places an arrow before each of the arguments, to indicate that it is using auto-values.

To provide sample values for all Input arguments, select the **Input Arguments** node and click **Generate**.

6 To parameterize the input arguments of the operation, see "Setting Properties for XML Parameters" on page 455.

7 Select the **Transport Layer Configuration** node to specify advanced options, such as JMS transport for SOAP messages (Axis toolkit only), asynchronous messaging, or WS Addressing. For more information, see Chapter 16, "Web Services - Transport Layers and Customizations.".

8 Click on each of the nodes and specify your preferences for argument values. For more information, see Chapter 11, "Web Services Call Properties."

9 To add an attachment to an input argument, or to specify a parameter to store output arguments, select the operation's main node and make the appropriate selection. For more information, see "Attachments" on page 265.

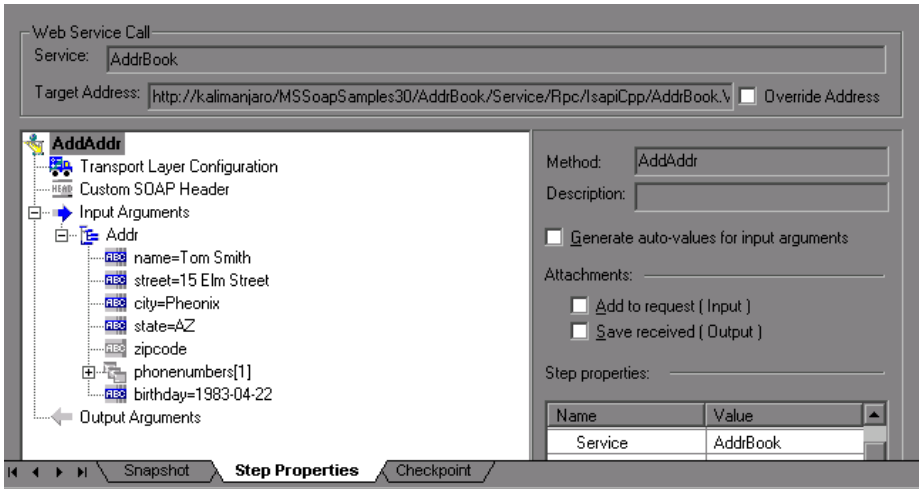
Importing SOAP Requests

Service Test lets you create Web service calls from SOAP files. If you have a SOAP request file, you can load it directly into your script. Service Test imports the entire SOAP request (excluding the security headers) with the argument values as they were defined in the XML elements. By importing the SOAP, you do not need to set argument values manually as in standard Web Service calls.

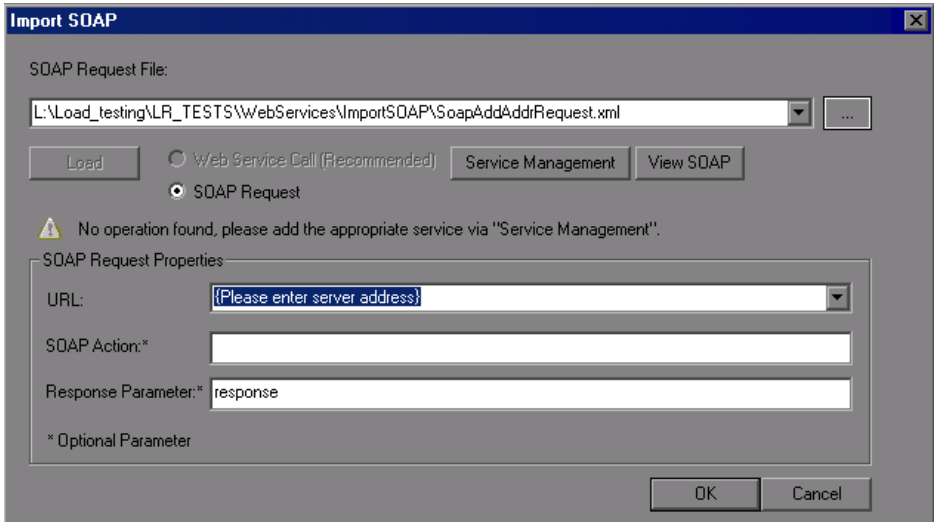
For example, suppose you have a SOAP request with the following elements:

```
- <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  - <q1:AddAddr xmlns:q1="http://tempuri.org/AddrBook/message/">
              <Addr href="#id1" />
    </q1:AddAddr>
   - <q2:Addr id="id1" xsi:type="q2:Addr" xmlns:q2="http://tempuri.org/AddrBook/type/">
          <name xsi:type="xsd:string">Tom Smith</name>
               <street xsi:type="xsd:string">15 Elm Street</street>
               <city xsi:type="xsd:string">Pheonix</city>
              <state xsi:type="xsd:string">AZ</state>
              <zip-code xsi:type="xsd:string">97432</zip-code>
              <phone-numbers href="#id2" />
              <birthday xsi:type="xsd:date">1983-04-22</birthday>
    </q2:Addr>
…
```

When you import the SOAP request, Service Test imports all of the values to the Web Service call:



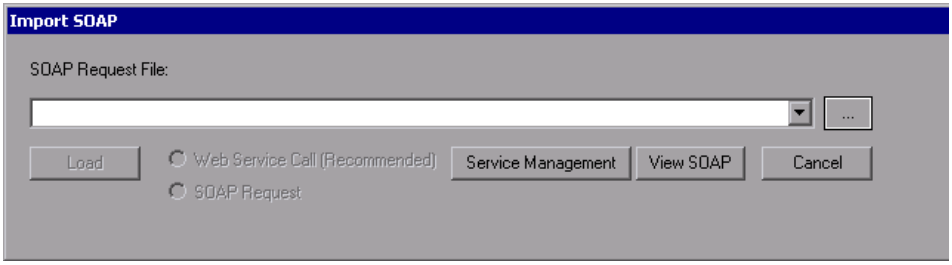
To create a new Web Service call based on a SOAP request, you must first import a WSDL file. If a WSDL is not available, or if you want to send the SOAP traffic directly, you can create a SOAP Request step. You specify the URL of the server, the SOAP action, and the response parameter.



In Script view, the SOAP Request step appears as a **soap_request** function, described in the *Online Function Reference* (**Help** > **Function Reference**).

To import a SOAP request:

1 Click the **Import SOAP** button or select **SOA Tools** > **SOAP Import**.



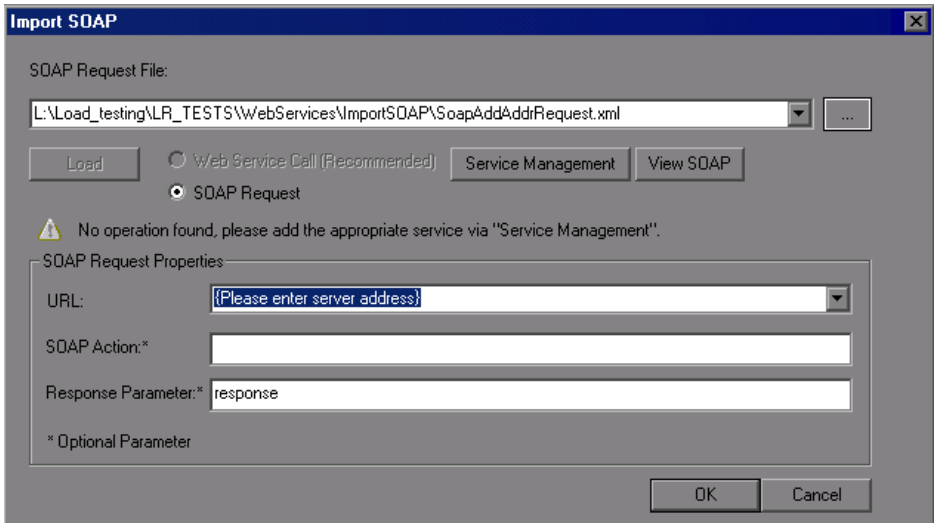
2 Browse for the XML file that represents your SOAP request.

3 Select the type of step you would like to generate: **Create Web Service Call** or **Create SOAP Request**. In order to create a Web Service Call, you must first import at least one WSDL that describes the operation in the SOAP request file. To import a WSDL, click **Service Management** and then click the **Import** button. To view the SOAP before loading it, click **View SOAP**.

4 Click **Load**. Service Test loads the XML element values.

For a Web Service Call, set the properties for the Service call as described in "Understanding Web Service Call Properties" on page 245.

For a SOAP request, provide the URL and the other relevant parameters.



5 For a Web Service Call, if there are multiple services with same operation (method) names, you need to select the service whose SOAP traffic you want to import. For information about additional properties, see "Understanding Web Service Call Properties" on page 245.

6 Click **OK** to generate the new step within your script.

7 Set checkpoints and replay the step. For more information, see "Setting Checkpoints" on page 278.

Using Your Script

After you create scripts, you can manage them in one of the following ways:

- **Service Test Management.** An add-on for HP Quality Center that lets you manage SOA testing by allowing you to import, store and define services in Quality Center. Its sections include Requirements Management, Test Plan, Test Lab, and Defects Management. For more information, see "Working with Service Test Management" on page 220.
- **BPT.** Business Process Testing lets you combine several scripts to form a complete business process. For more information, see "BPT (Business Process Testing)".

You can use the completed script to test your system in several ways:

- **Functional Testing.** Run the script to see if your Web services are functional. You can also check to see if the Web service generated the expected values For more information, see Chapter 12, "Web Services - Preparing for Replay."
- **Load Testing.** Integrate the script into a LoadRunner Controller scenario to test its performance under load. For more information, see the *HP LoadRunner Controller* or *Performance Center* documentation.
- **Production Testing.** Check your Web service's performance over time through a Business Process Monitor profile. For more information, see the *HP Business Availability Center* documentation.

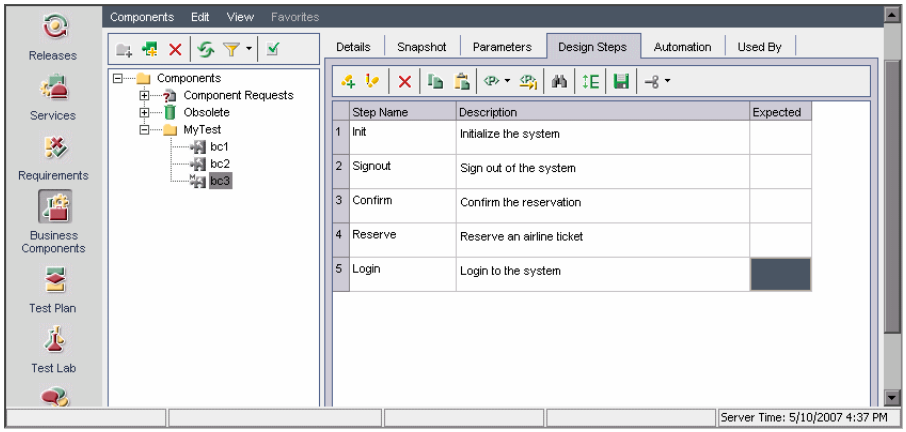
BPT (Business Process Testing)

BPT (Business Process Testing) is a methodology in which several tests are combined to create a complete business process. The BPT user composes a complete test by combining a series of test components with data flow between them.

Components are comprised of steps. For example, a login component's first step may be to open the application. Its second step could be entering a user name. Its third step could be entering a password, and its fourth step could be clicking the **Enter** button.

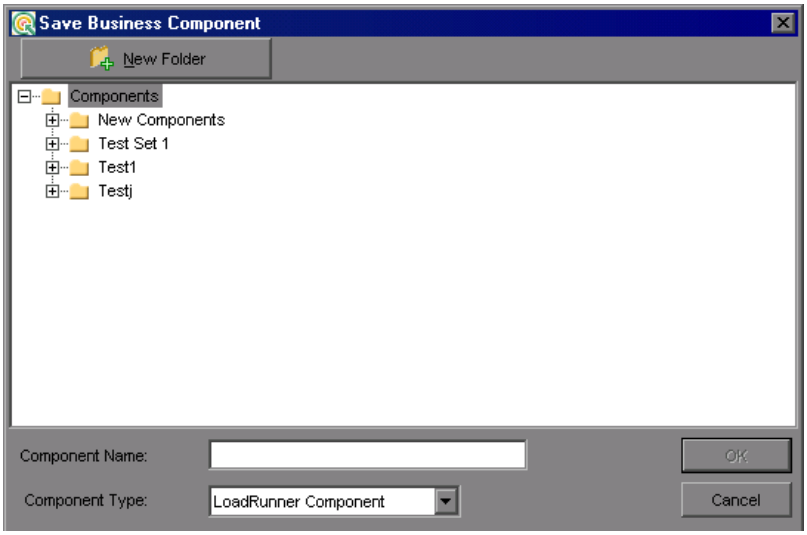
You can create business process test components from within Quality Center or through HP Service Test.

In Quality Center, non-technical SMEs (Subject Matter Experts) define design steps that are required for the test.

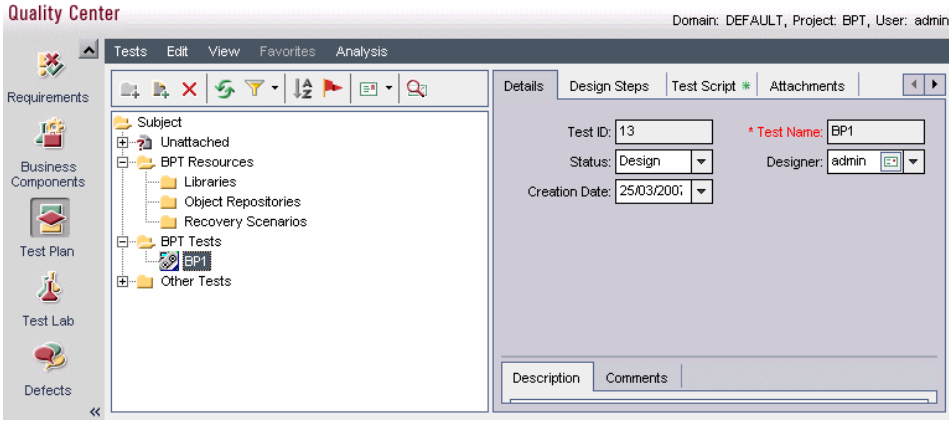


After these are defined, the technical engineer creates a script through Service Test that performs the desired actions. For more information about creating Business Components from within Quality Center, see the *Business Process Testing user guide*.

You create components in Service Test by recording a session with your application or manually editing a script. You can add checkpoints, parameterize selected items, and enhance the component with flow statements and other testing functions. You then save the component to a project in Quality Center.



A Subject Matter Expert using Business Process Testing in Quality Center combines your saved components into one or more business process tests, which are used to check that the application behaves as expected.



When you create parameters for your business component in Quality Center, they will be available in Service Test's parameter list.

Some of the advantages of working with a BPT module over individual scripts are:

- Enables less-technical subject matter experts to create tests
- Enables structured automated testing
- Reduces the duplication of effort when combining manual tests with automatic scripts
- Allows component reusability to speed-up the automation process
- Provides the ability to pass parameters from one step to another within your business process. You can save the output of a step to a parameter and use it as an input value for subsequent steps.
- Simplifies on-going test maintenance
- Minimizes time-to-test

For more information about creating a BPT components in Quality Center, see the *Business Process Testing User Guide*.

Creating Business Components in Service Test

You can create a new business component in Service Test and add content through recording or any other means as described in this section. You can also save existing scripts as business components.

To create a Business Component in Service Test:

1. Open Service Test and connect to the Quality Center server. Select **Tools** > **Quality Center Connection** and specify the connection details.
2. Select **File** > **Business Component** > **New**. The New Business Component dialog box opens.
3. Create a new Web Services script and add script content by recording, importing SOAP, or by manually adding Web Service calls. For more information, see "About Adding Content to Web Services Scripts" on page 201

4 Parameterize the desired arguments. If you want to share the parameter between business components in Quality Center, make sure to create a **BPT** type parameter. For more information, see the section on Setting Properties for BPT Type Parameters.

5 Save the script to the desired location in Quality Center. When you create a business component in Service Test, it is regarded as an automated test in Quality Center.

6 To convert an existing Vuser script to a Business Component, select **File** > **Business Component** > **Save as Business Component** and save the script to the desired location in Quality Center.

Creating Components in Quality Center

The following section describes briefly how to create a business component in Quality Center and automate it to be compatible with Service Test. For complete information about working with Business Components in Quality Center, see the *Business Process Testing User Guide*.

To create a Service-Test type automated business component in Quality Center:

1 Select the **Business Components** module in the left pane and select **Component** > **New Component** to create a new business component.

2 Select the **Design Steps** tab, create new manual steps, and define parameters (optional).

3 Automate the business component. In the **Design Steps** tab, select **Automate Component** (rightmost button) > **Service Test.**

Editing Automated Components in Service Test

You can also use Service Test to edit a SERVICE-TEST Automated component, that was created and automated in Quality Center. You can open the business component in Service Test or directly from Quality Center.

To open and edit an automated business component in Service Test:

1 Open Service Test and connect to the Quality Center server. Select **Tools** > **Quality Center Connection** and specify the connection details.

2. Select **File** > **Business Component** > **Open**. The Open Business Component dialog box opens.
3. Specify a script to open.
4. Edit the script, set parameters and add content as you would with any other script. For more information, see "About Adding Content to Web Services Scripts" on page 201.

To open and edit an automated business component from Quality Center:

1. Select the **Business Components** module in Quality Center's left pane and select a business component in the tree.
2. Select the **Automation** tab.
3. Click **Launch**. Service Test opens with the script.
4. Edit the script, set parameters and add content as you would with any other script. For more information, see "About Adding Content to Web Services Scripts" on page 201.

For more information about working with Business Components in Quality Center, see the *Business Process Testing User Guide*.

Managing Data in Quality Center

When working with Quality Center, you can assign different parameter values for each instance of the Quality Center test set. This enables you to run the same test with different data.

When you modify the parameter values for one test instance, it does not affect other test instances. At any point, you can restore the original parameter values.

For more information, see the *HP Quality Center User Guide*.

Working with Service Test Management

HP Quality Center is a Web-based application for test management. Its sections include Requirements Management, Test Plan, Test Lab, and Defects Management.

The **Service Test Management** add-on for Quality Center, lets you manage SOA testing by allowing you to import, store and define services in Quality Center.

The Service Test Management integration lets you:

- **Store Web Services.** You can store and organize Web Services in Quality Center for use within Service Test.
- **Write Service Test scripts.** You can create scripts based on the services stored in Quality Center, while maintaining up-to-date WSDLs throughout the life-cycle of the service and the script.
- **Compose a Business Process Test.** You can create a BPT (Business Process Test) in Quality Center based on services defined through Service Test Management.

Service Test Management also integrates with HP's Systinet Registry, to create test requirements and plans. Once the services are imported, Service Test Management identifies any changes to the services and automatically generates the necessary test cases that need to be run.

Using the **Service Test Management** add-in for Quality Center, groups throughout your organization can contribute to the quality process in the following ways:

- Business analysts define application requirements and testing objectives.
- Test managers and project leads design test plans and develop test cases.
- Test managers automatically create QA testing requirements and test assets for SOA services and environments.
- Test automation engineers create automated scripts and store them in the repository.
- QA testers run manual and automated tests, report execution results, and enter defects.

- Developers review and fix defects logged into the database.
- Project managers can export test and resource data in various reports, or in native Microsoft Excel for analysis.
- Product managers decide whether an application is ready to be released.
- QA analysts can auto-generate test asset documentation in Microsoft Word format.

For more information about the integration, see the *HP Service Test Management User Guide*.

9

Web Services - Automatic Test Generator

The SOA Test Generator automatically generates test scripts based on your requirements.

This chapter includes:

About Using the SOA Test Generator

To test your SOA environment, you can create tests manually, or use the SOA Test Generator to automatically generate scripts.

This section describes how to use the SOA Test Generator. For information on creating tests manually, see Chapter 8, "Web Services - Adding Script Content."

The SOA Test Generator guides you through the process of creating scripts to test your services. Through the wizard, you indicate which aspects of the service you want to test. These aspects include interoperability with different toolkits, boundary testing, and standard compliance.

After you select the testing aspects, Service Test automatically generates one or more scripts for each of the aspects.

Selecting Test Aspects

The SOA Test generation wizard helps you create scripts that test different aspects of your service. It creates a separate script for each aspect. If an aspect has sub-aspects, Service Test creates a separate script for each one of the sub-aspects.

Testing Aspects

The SOA Test Generator supports the following testing aspects:

- **Positive Testing.** Generates a full positive test for the selected services. It tests the service operations both separately and with one another.
- **Standard Compliance.** Checks the service's compliancy with industry standards such as WS-I and SOAP.
- **Service Interoperability.** Checks that the service's operations are interoperable with all supported Web Services toolkits. Note that several tests are created for this aspect.
- **Security testing.**
  - **SQL Injection.** Attempts to hack the service by injecting SQL statements that will result in an unauthorized extraction of data.
  - **Cross-site Scripting.** Attempts to hack the service by injecting code into a Web site that will disrupt its functionality.
- **Boundary Testing.** Tests the services using the negative testing technique. Based on the service's description, Service Test creates tests to manipulate data, types, parameters, and the actual SOAP message in order to test the service to its limits.
  - **Extreme Values.** Tests for extreme values of simple data types.
  - **Null values.** Provides NULL parameters to the services to verify that they are not accepted.

Tip: When working with Quality Center, it assigns a Pass or Fail status to each of the scripts. For more information, refer to the documentation for the Service Test Management module.

Generating Tests Automatically

The Test Generator's wizard guides you through the steps of creating automatic tests:

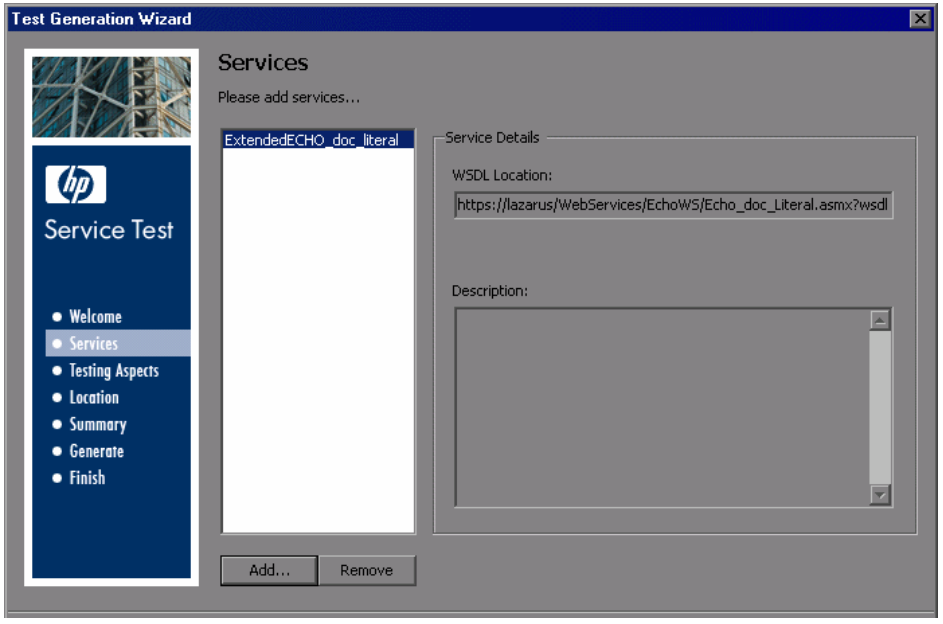
To create automatic tests:

1 Choose **File** > **New** to open the New Virtual User dialog box.

2 Select **SOA Test Generator** in the left pane.

3 Make sure **Web Services** is listed in the Selected Protocols box. Click **OK**.

4 In Test Generation Wizard welcome screen, click **Next**.

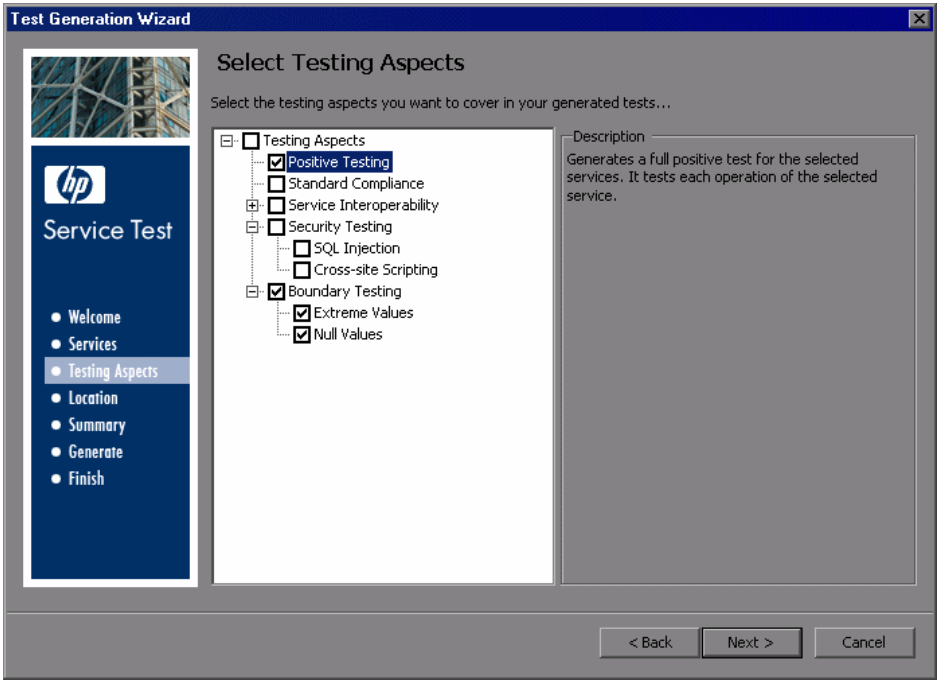


5 Add at least one service. Click **Add** to open the Import Service dialog box.

6 Specify the WSDL information. For more information, see "Importing Services" on page 179.

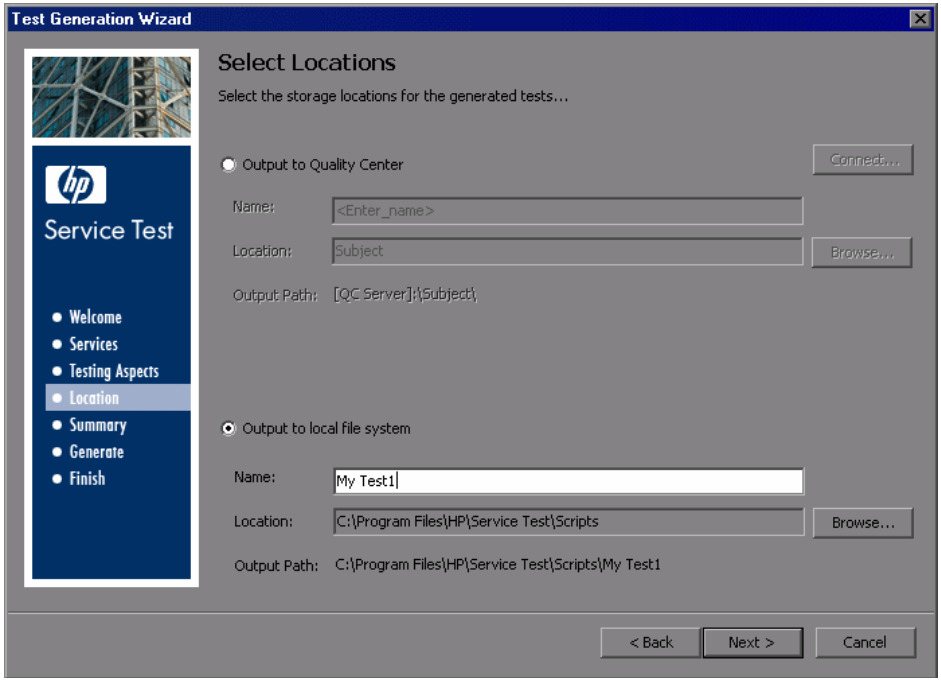
7 Repeat this step for all of the services you want to import. Click **Next** to select testing aspects.

8 Select the desired testing aspects and sub-aspects. For more information, see "Selecting Test Aspects" on page 224.



Click **Next** to select the tests' output location.

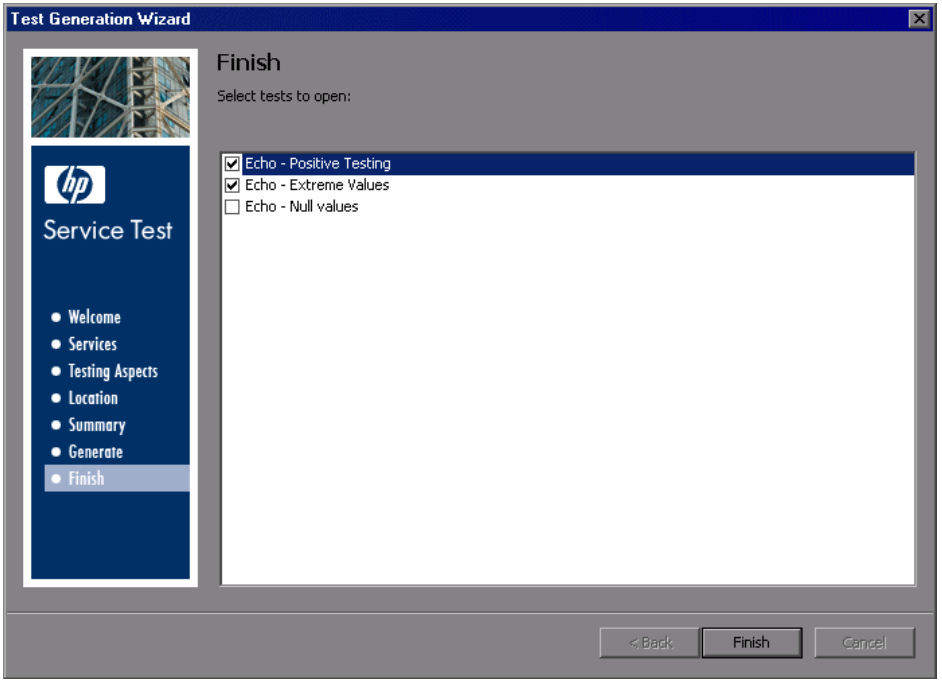
9 Specify a test name and a location for the test scripts: **Quality Center** or a **local file system**.



If you specified Quality Center, click **Connect** to log on to the server and **Browse** to locate the test node.

10 Click **Next** to display a summary of your selections: selected services, testing aspects, and output location.

11 Click **Generate** to begin creating the scripts. After Service Test completes generating the tests, it lists them in the **Finish** screen.



12 Select the scripts that you want Service Test to open and click **Finish**. The Script Generator opens the selected scripts.

After you create scripts, you can enhance them with additional service calls and parameterization. You then run the completed script in Service Test to check its functionality.

For more information, see "Getting Started with Web Services Vuser Scripts" on page 138.

10

Web Services - Server Traffic Scripts

Using Service Test, you can create scripts to test your Web Service by analyzing server traffic capture files.

This chapter includes:

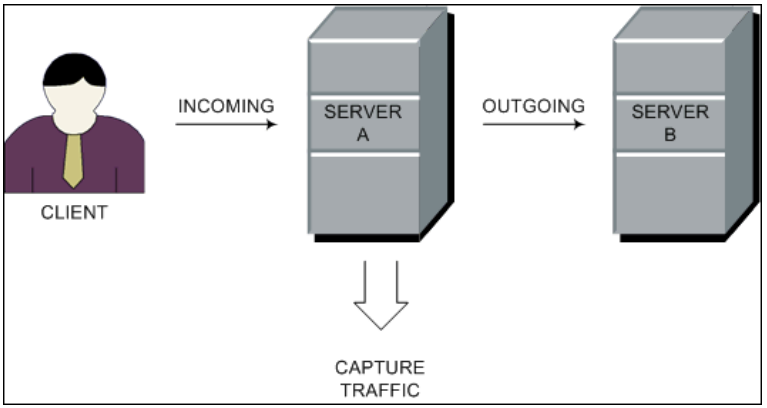
The following information only applies to Web Services/SOA Vuser scripts.

About Creating Server Traffic Scripts

The main focus when testing enterprises and complex systems, is to measure the performance from the client end. Ordinarily, Service Test records the actions you perform in the application or browser, and generates a script emulating the client actions and requests to the server.

In certain test environments, you may be unable to record the client application to retrieve the requests to the server. This may be a result of the server acting as a client, or because you do not have access to the client application. In these cases, you can create a script using Service Test's **Analyze Traffic** feature.

The **Analyze Traffic** feature examines a capture file containing the server network traffic, and creates a script that emulates requests sent to or from the server. The steps in creating a script by analyzing server traffic are described below in Getting Started with Server Traffic Scripts.



There are two types of emulations: **Incoming traffic** and **Outgoing traffic**.

Incoming traffic scripts emulate situations in which you want to send requests to the server, but you do not have access to the client application, for example, due to security constraints. The most accurate solution in this case is to generate a script from the traffic going **into** the server, from the side of the client.

When you specify an Incoming server network traffic, you indicate the IP address of the server and the port number upon which the application is running. Service Test examines all of the traffic going into the server, extracts the relevant messages, and creates a script. In the above diagram, if the client is unavailable, you could create an Incoming script to emulate the requests coming into **Server A**.

Outgoing Traffic scripts emulate the server acting as a client for another server. In an application server that contains several internal servers, you may want to emulate communication between server machines, for example between **Server A** and **Server B** in the above diagram. The solution in this case is to generate a script from the traffic sent as output **from** a particular server.

When you create an Outgoing traffic script, you indicate the IP address of the server whose outgoing traffic you want to emulate, and Service Test extracts the traffic going out of that server. In the above diagram, an Outgoing script could emulate the requests that **Server A** submits to the **Server B**.

Getting Started with Server Traffic Scripts

The following section outlines the process of creating a script that analyzes server traffic.

1 Create a capture file.

Service Test uses the capture file to analyze the server traffic and emulate it. For more information, see "Generating a Capture File" on page 232.

2 Create a new Web Services script.

Using Service Test's main interface, you create a new Web Services script. For more information see Chapter 8, "Web Services - Adding Script Content."

3 Specify the Services (optional, but recommended).

To create a high-level script, import a WSDL which describes the Web Service you want to test.

4 Specify the traffic information.

Click the **Analyze Traffic** button. Specify the location of the traffic file and whether your script will be for Incoming or Outgoing traffic. For more information, see "Specifying Traffic Information" on page 235.

5 Specify the traffic filter Recording options.

Filter options let you determine which hosts to include or exclude in your script. For more information, see "Choosing an Incoming or Outgoing Filter" on page 236.

6 Specify the SSL certificate information.

The SSL configuration lets you analyze secure traffic over HTTPS in order to generate the script. For more information, see "Providing an SSL Certificate" on page 238.

Generating a Capture File

A capture file is a trace file containing a log of all TCP traffic over the network. Using a sniffer application, you obtain a dump of all of the network traffic. The sniffer captures all of the events on the network and saves them to a capture file.

To generate a smaller, more manageable script, try to capture the network traffic only for the time that you perform actions in your application.

Note: Capture files do not contain loopback network traffic.

You can obtain a capture file using the command line utility or any existing capture tool.

The Capture File Command Line Utility

The Service Test command line utility, **lrtcpdump**, is located in the product's **bin** directory. There is a separate utility for each of the platforms: **lrtcpdump.exe** (Windows), **lrtcpdump.hp9**, **lrtcpdump.ibm**, **lrtcpdump.linux**, and **lrtcpdump.solv4**.

To invoke the capture tool, type:

```
lrtcpdump -i<interface> -f<file>
```

where **interface** is the name of the network card whose traffic you want to capture, and **file** is the name of the capture file in which to store the information. Do not leave a space between the command line option (**i** or **f**) and the value.

To create a capture file on a Windows platform:

1. Select **Start** > **Run**, type **cmd** and click **OK** to open a command window.
2. Drag in or enter the full path of the **lrtcpdump.exe** program located in the product's **bin** directory.
3. Provide a file name for the capture file using the following syntax:
   `lrtcpdump -f <file>`

for example **lrtcpdump -fmydump.cap**.

4 **lrtcpdump** prompts you to select a network card. If there are multiple interface cards, it lists all of them. Type in the number of the interface card (1, 2, 3 etc.) and click **Enter**.

5 Perform typical actions within your application.

6 Return to the command window and click **Enter** to end the capture session.

To create a capture file on a UNIX platform:

1 Locate the appropriate **lrtcpdump** utility for your platform in the product's **bin** directory. Copy it to a folder that is accessible to your UNIX machine. For example, for an HP platform, copy **lrtcpdump.hp9**. If using FTP, make sure to use the binary transfer mode.

2 Switch to the root user to run the utility.

3 Provide execution permissions. **chmod 755 lrtcpdump.<platform>**

4 On UNIX platforms, if there are multiple interface cards, **lrtcpdump** uses the first one in alphabetical order. To get a complete list of the interfaces, use the **ifconfig** command.

5 Run the utility with its complete syntax, specifying the interface and file name. For example, **lrtcpdump.hp9 -ietho -fmydump.cap**. The capturing of the network traffic begins.

6 Perform typical actions within your application.

7 Return to the window running **lrtcpdump** and follow the instructions on the screen to end the capture session.

8 Place the capture file on the network in a location accessible to the machine running Service Test.

An Existing Capture Tool

Most UNIX operating systems have a built-in version of a capture tool. In addition, there are many downloadable capture tools such as Ethereal/tcpdump.

When using external tools, make sure that all packet data is being captured and none of it is being truncated.

Note: Certain utilities require additional arguments. For example, **tcpdump** requires the **-s 0** argument in order to capture the packets without truncating their data.

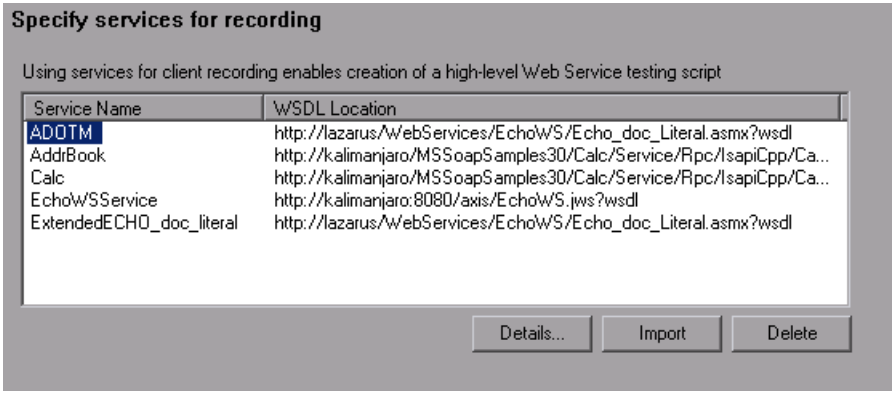
Creating a Basic Script from Server Traffic

You create a script from server traffic just as you would create a recorded script.

You can optionally specify a Web Service for your script. If you specify a service, Service Test will create a script with **web_service_call** functions. If you do not specify a service, Service Test creates a script with **soap_request** functions.

To create a server traffic script:

1. Select **File** > **New** and click **New Single Protocol Script** in the left pane.
2. Select the **Web Services** protocol and click **OK**.
3. Click the **Analyze Traffic** button or select **Vuser** > **Analyze Traffic**. The wizard opens.
4. Add one or more services to the list. This step is optional.



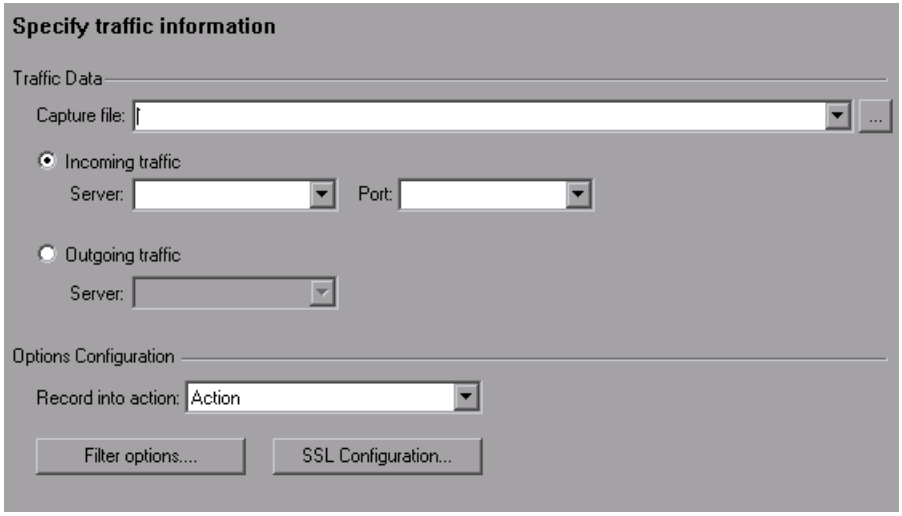
- ➤ To add a new service, click **Import**. In the Import Service dialog box, specify the location of the WSDL. You can specify a URL, File, UDDI server (such as Systinet), or a location in Quality Center. In the Import Service dialog box, you also select a toolkit for analyzing the service. The selected toolkit will be permanently associated with the script—it cannot be changed. For more information, see "Importing Services" on page 179.
- ➤ To set or view details about the services, click **Details** to open the Service Management window. For more information about Service Management, see Chapter 7, "Web Services - Managing."
- ➤ To remove a listed service, select it and click **Delete**

5 On the bottom of the wizard screen, click **Next** to specify the traffic file information. For more information, see below.

6 After providing traffic information, click **Finish** to generate a script.

Specifying Traffic Information

The traffic file contains a dump of all the network traffic. Using the wizard, you specify the location of the traffic file and whether you want your script to emulate incoming or outgoing traffic.



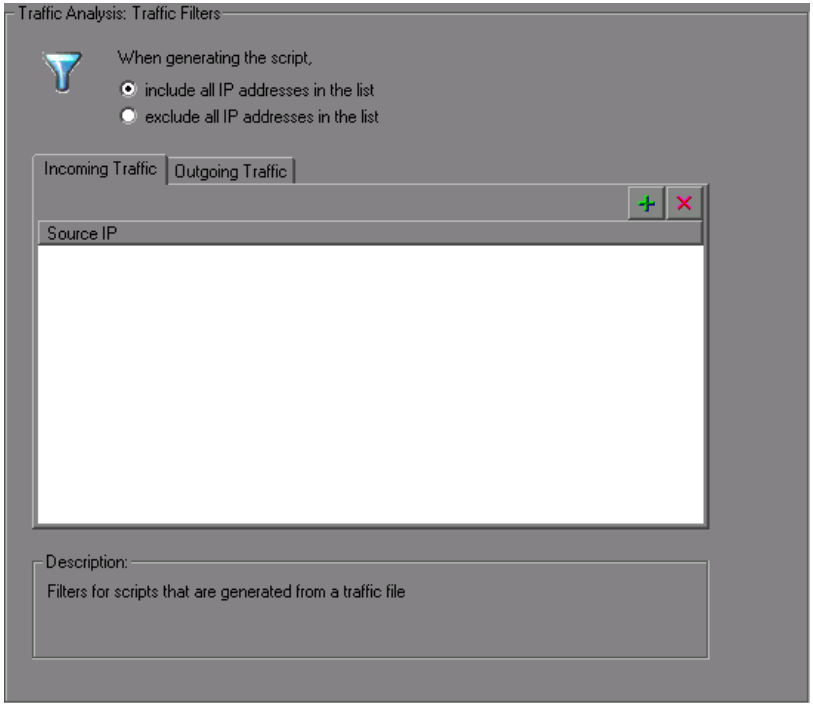
- **Capture file.** The name and path of the traffic file, usually with a cap extension.
- **Incoming traffic:Server/Port.** The IP address and port of the server whose incoming traffic you want to examine.
- **Outgoing traffic:Server.** The IP address of the server whose outgoing traffic you want to examine.
- **Record into action.** The section into which to create the script. If you want to use iterations, specify the **Actions** section.
- **Filter options.** Opens a filter interface allowing you to specify which IP addresses to include or exclude from the script. For more information, see below.
- **SSL Configuration.** Allows you to add SSL certificates to analyze traffic from a secure server with the required credentials. For more information, see "Providing an SSL Certificate" on page 238.

Choosing an Incoming or Outgoing Filter

You can provide a filter to drill down on specific requests going to or from a server, by specifying its IP address and port.

It is also possible to filter your capture file with an external tool before loading it into Service Test. In that case, you may not require additional filtering.

You filter the requests by choosing the relevant host IP addresses. The filter can be inclusive or exclusive—you can include only those IPs in the list, or include everything except for those IPs that appear in the list.



To filter the traffic file:

1 Open the Traffic Filters recording options. Click **Filter Options** in the Specify Traffic Information step, or select **Tools** > **Recording Options**. Select the **Traffic Analysis:Traffic Filters** node.

2 Select one of the filtering options: **include all IP addresses in the list** or **exclude all IP addresses in the list**.

3 Select the tab that corresponds to your script type: **Incoming Traffic** or **Outgoing Traffic**.

4 Add hosts to the list.

To add a host to the list, click the **Add** button. Specify the IP address of the server you want to add to the list. For incoming traffic, specify the port of the server to include or exclude. Click **OK** to accept the settings.

Click **Delete** to remove an entry.

After the script is created you can change the filters and regenerate the script—there is no need to re-analyze the capture file.

Providing an SSL Certificate

To analyze traffic from a secure server, you must provide a certificate containing the private key of the server.

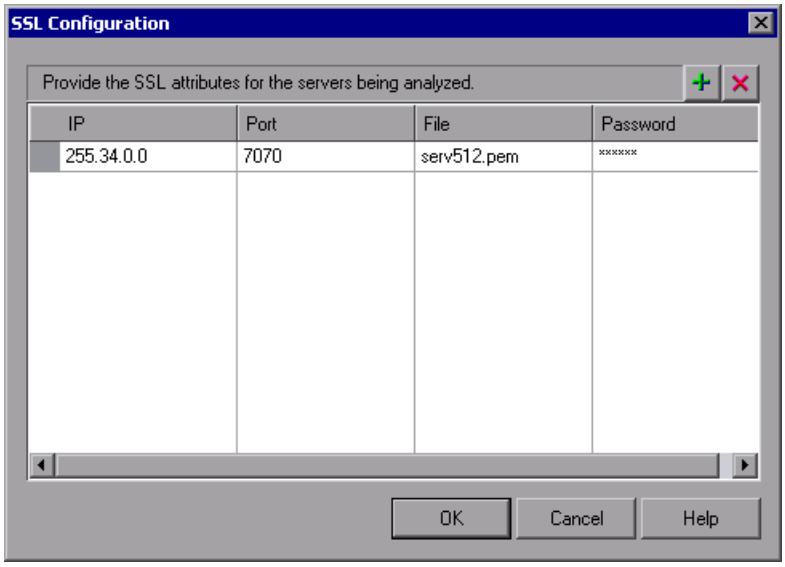
If the traffic is SSL encrypted, you must supply a certificate file and password for decryption. If you want traffic from multiple servers to be reflected in the script, you must supply a separate certificate and password for each IP address that uses SSL.

To specify an SSL certificate:

1 In the Specify Traffic Information screen, click **SSL Configuration**.

2 Add certificates to the list.

To add a certificate to the list, click the **Add** button. Specify the IP address, port, path of the certificate file (with a **pem** extension), and the password for the certificate. Make sure the **pem** file contains the private key. If you are unsure of how to obtain the certificate, contact your system administrator.



Click the **Delete** button to remove an entry from the list.

3 Repeat the above steps for every certificate you want to add.

4 Click **OK** to close the dialog box.

11

Web Services Call Properties

You use the Web Service Call view to display snapshots, set properties, and add checkpoints to Web Service calls.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About the Web Service Call View

Using the Web Service Call view, you can view snapshots, set properties, and define checkpoints for each of the Web Service calls in your script.

To open the Web Service Call view, you must be in Tree view. Select **View > Tree View** in the Service Test window to open Tree view.

The Snapshot lets you view the SOAP structure of each step. You can view snapshots of the recording, replay, request, and response. For more information, see below.

The Properties tab lets you configure the settings for each Web Service call. You can set properties in the area of argument values, parameterization, transport layer, and security. For more information, see "Understanding Web Service Call Properties" on page 245.

Checkpoints let you verify your Web service results. You can check for expected values and view the output to see if they were matched. For more information, see "Setting Checkpoints" on page 278.

Viewing Web Services SOAP Snapshots

You can use Service Test's snapshot viewer to examine the SOAP requests and responses that occurred during record and replay. Note that you must replay the session at least once in order to view a replay snapshot.

There are several ways to view the SOAP snapshots:

- Record and/or Replay
- Request or Response Data
- Tree or XML View

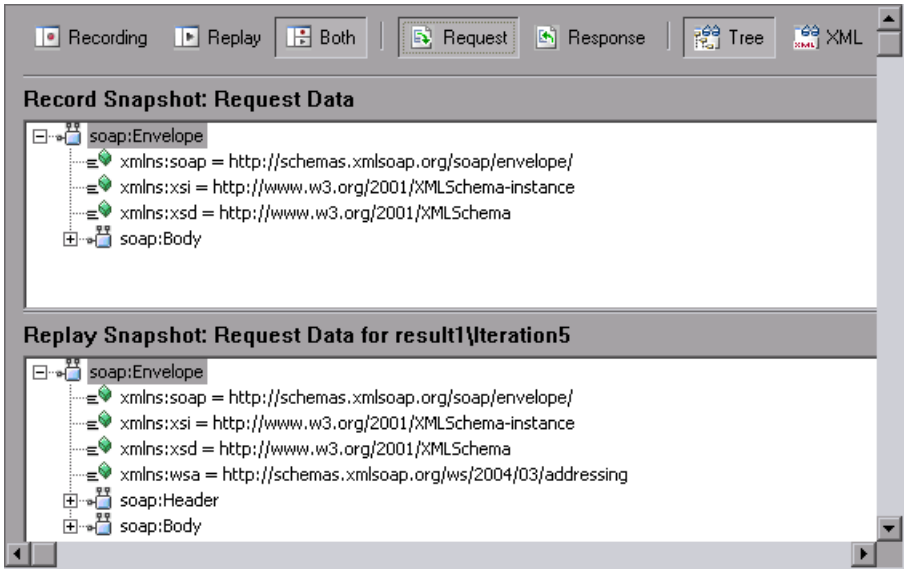
The SOAP Snapshot view also provides you with several utilities that allow you to work with the XML code: **FInd an XML Element** and **Validate XML**.

Using the buttons in the Snapshot window, you can control the view:

In **Tree** view, you can expand the nodes to view the values of the arguments, if they were assigned.

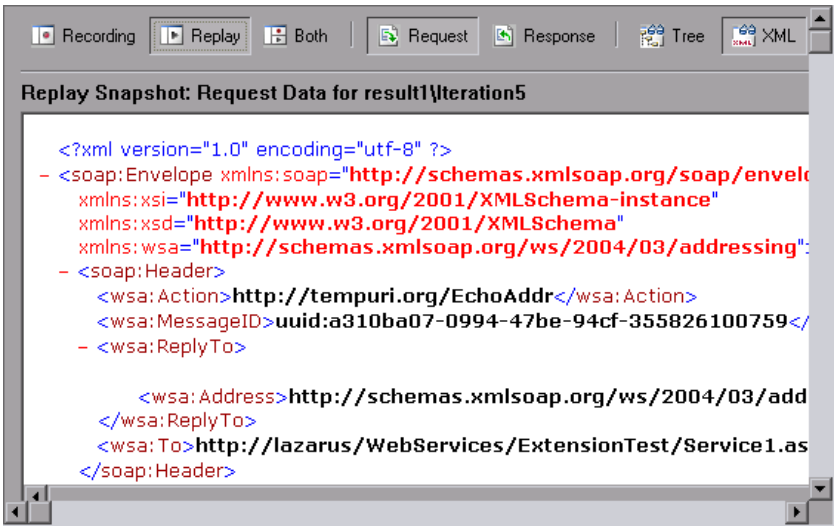
In **Request** view, the displayed values are those that were sent to the server during the Web Services session. In **Response** view, the displayed values are the results returned by the server.

In the following example, the Snapshot window shows the **Record** and **Replay** snapshots of the **Request** data in **Tree** view.



To learn more about a node, select it and select **Node Properties** from the right-click menu.

In the **XML view**, you can view the whole SOAP message in XML format.



Specifying a Replay Iteration

If you replayed the script with multiple iterations, you can specify which iteration to display in the snapshot. In addition, you can display a snapshot from test results that you saved in a location other than the script's folder. By default, the Snapshot view shows the last iteration.

To specify an iteration to display:

1. Select **View** > **Snapshot** > **Choose Iteration**.
2. Select the desired iteration and click **OK**.
3. To display results from another folder, select **Select Folder** and browse to the location of the test results.

Understanding Web Service Call Properties

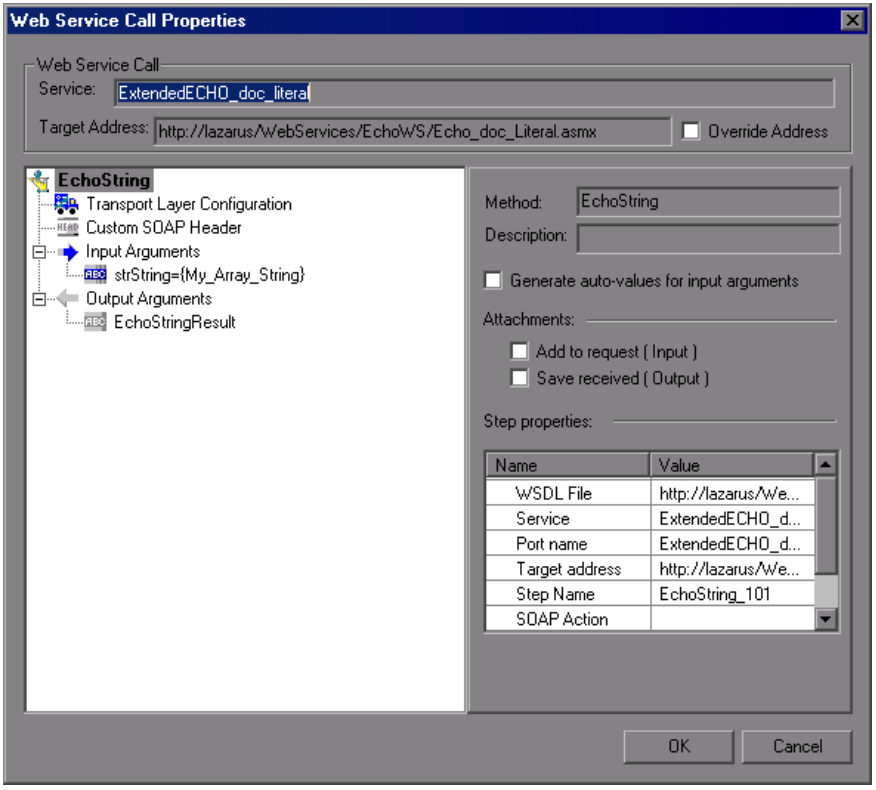
Service Test provides an interface for you to view and modify the properties of each one of the Web Service calls.

Properties describe the behavior of each method within your service. You can set a target address, argument values, parameterization, and transport layer preferences for each of your service's methods.

You can view a step's properties from Tree view (**View** > **Tree view**) in one of the following ways:

- Double-click on a step in the left pane to open the Web Service Call Properties dialog box.
- Select the **Step Properties** tab in the right pane.

The Properties view displays each of the service's operations in a tree hierarchy. The nodes of the tree represent the Transport Layer Configuration, the SOAP header, input arguments, and output arguments.



By default, the script takes the target address from the WSDL. You can override this address for each operation. Select the **Override Address** option and specify the desired **Target Address**.

The contents of the right pane changes, depending on the level of the selected tree node. The following table describes the content for each node:

If you select...	The right pane shows...
A method or operation name	➤ A checkbox to include input/output attachments ➤ The step properties

Transport Layer Configuration	Advanced transport options: ➤ **HTTP/S Transport** with **Async** or **WSA** support. ➤ **JMS Transport** support with response and request queue information.
SOAP Header	An edit box to indicate the value of the SOAP header for the current element. For more information, see "SOAP Headers" on page 269.
Input Arguments node	➤ The **Name** of each method and its **Value.** ➤ **Include All**, **Reset** and **Generate** buttons.
An individual argument	➤ **Name.** The name of the argument (read-only) ➤ **Type.** The argument type as defined in the WSDL. When the WSDL contains derived types, this box becomes a drop down list. For more information, see "Derived Types" on page 255. ➤ **Include argument in call.** Includes the Optional parameters in the Web Service call. See "Working with Optional Parameters" on page 256. ➤ **Nil.** Sets the Nillable attribute to True. ➤ **Value.** The value of the argument. For base64 binary type arguments, **Get from file** or **Base64 Encoded text**. ➤ **Generate auto-value for this argument**. Insert automatic values for this node.
A complex input argument	➤ **XML.** Enables the **Edit**, **Import,** and **Export** buttons. By editing the XML, you can manually insert argument values. Click on the **ABC** icon to replace the entire XML structure with a single XML type parameter. **Note**: This import operation handles XML files that were previously exported—not standard SOAP files. To import SOAP, see "Importing SOAP Requests" on page 210. ➤ **Generate auto-value for this argument.** Inserts automatic values for all arguments of this complex type node. ➤ **Add**/**Delete**. Adds or removes elements from the array.

Output Arguments	➤ The output argument list ➤ Negative Testing options, as described in Chapter 18, "Web Services - Negative Testing"
Input Attachments	The **Attachment Format** for encoding the soap request: DIME or MIME for the Service Test toolkit, DIME only for .NET (see "Including MIME Attachments" on page 386), and MIME only for Axis.
Attachment	➤ **Take Data from.** The name of the file to attach or the name of the parameter containing the data. ➤ **Content Type.** The attachment's content type. You can instruct Service Test to detect it automatically or select a type from the dropdown list or enter a value manually. ➤ **Content ID.** The attachment's ID attribute. You can instruct Service Test to automatically generate a value or specify your own ID. ➤ **Delete Attachment.** A button to remove the attachment.
Output Attachments	➤ **Save All Attachments**. Saves all attachments to parameters: ➤ **Content.** The name of the parameter prefix for storing the attachment. ➤ **Content Type**. The attachments' content type attribute (read-only). ➤ **Content ID**. The attachment's ID attribute (read-only). ➤ **Save Attachments by Index**. Save the attachments by number, beginning with 1. Click **Add** to insert additional attachment indexes.

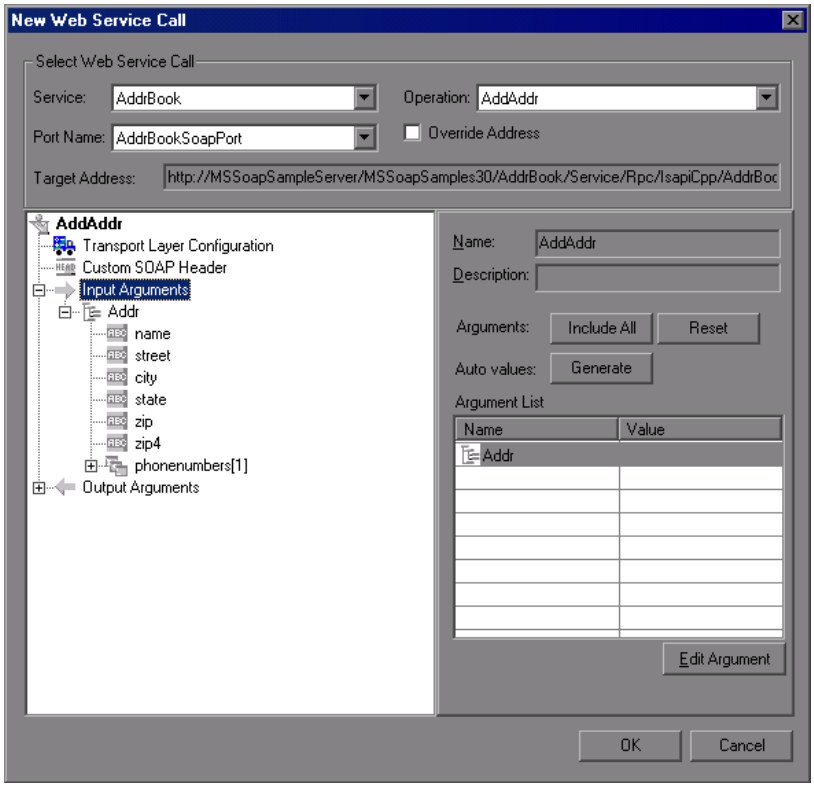
You can set argument values for the following elements: (manually edited arguments are displayed in blue)

- Input Argument Values
- Output Arguments
- Arrays

- Attachments
- SOAP Headers

Input Argument Values

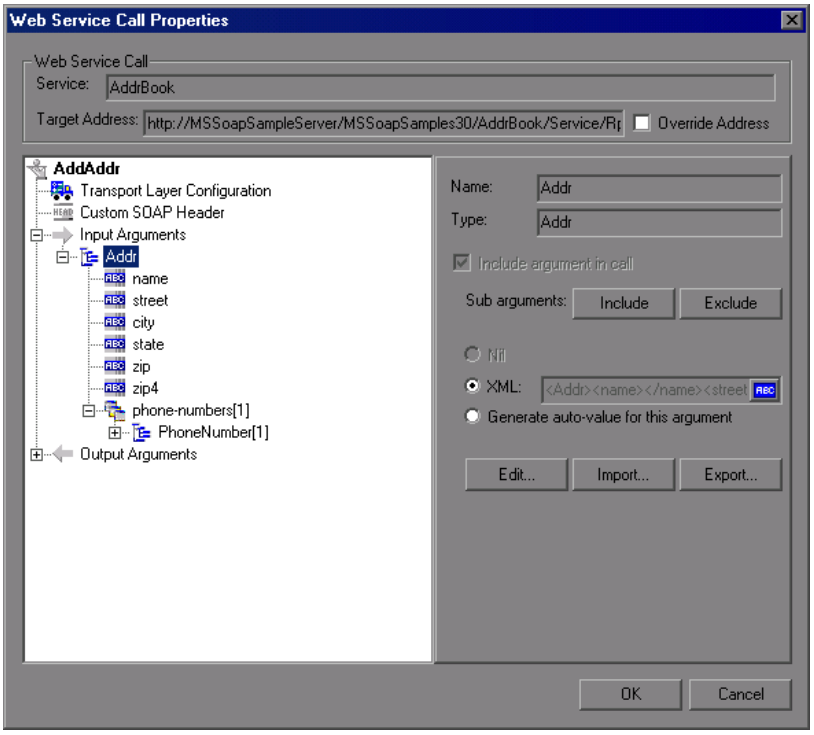
The Input Arguments node lets you define values for all of the input arguments and lets you control the Optional elements. For more information about Optional elements, see "Working with Optional Parameters" on page 256.



- **Include All.** Includes all Optional elements—all of the operation's elements are included.
- **Reset.** Excludes all Optional elements and only includes the mandatory ones.

- **Generate.** Includes all Optional elements and generates automatic values for all of the operation's elements.
- **Edit Argument.** Opens the node of the selected argument and lets you set its values.

The individual argument nodes lets you define values for each of the input arguments.

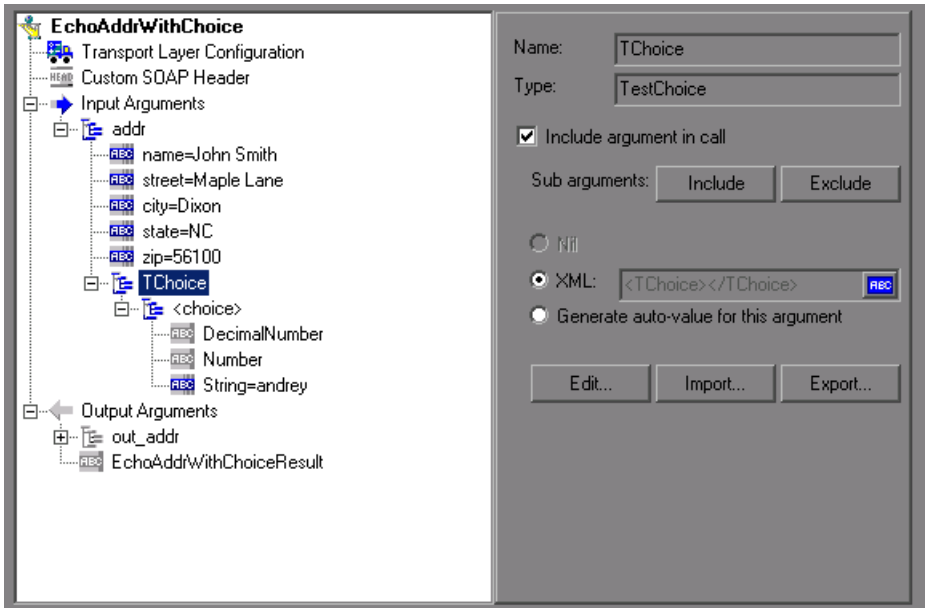


- **XML/Value.** A manually specified value for the node. If your argument is an array, you can specify an XML structure. Otherwise, specify an ordinary value. To create a parameter for the argument, click the **ABC** icon in the right corner of the **XML/Value** box to open the Select or Create Parameter dialog box.

➤ **Generate auto-value for this argument.** If you want Service Test to automatically generate a value for this argument, select this option or select the argument in the tree hierarchy and select **Generate Auto-values** from the right-click menu.

Choice Elements

If your WSDL defines Choice elements, you can view them and set their values in the Properties view.



To set a value for a choice element, select the parent element, enable the **Include argument in call** option in the right pane, and provide a value.

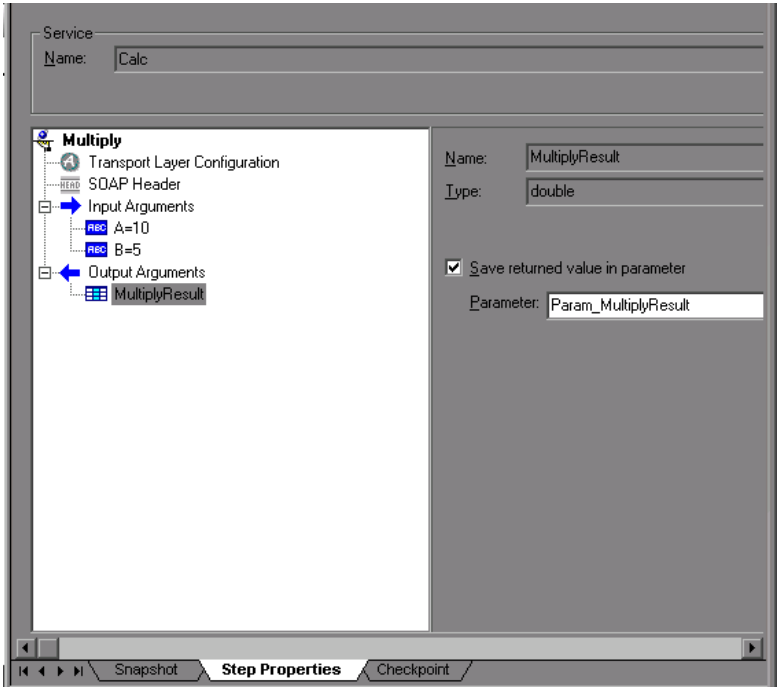
To parameterize the argument, click the **ABC** icon. In the Parameter Properties dialog box, provide values for the choice argument. You only need to provide values for one of the choice elements. When running multiple iterations, the script uses the values for the same choice element, according to the assignment method (sequential, unique or random). For example, if your choice elements are **Decimal Number**, **String**, and **Number**, and you provided values for **Number**, the Vuser will always use the **Number** element.

Choice support is provided for both Input and Output arguments, Parameterization, Checkpoints, and Service Emulation.

For information about working with optional arguments, see "Working with Optional Parameters" on page 256.

Output Arguments

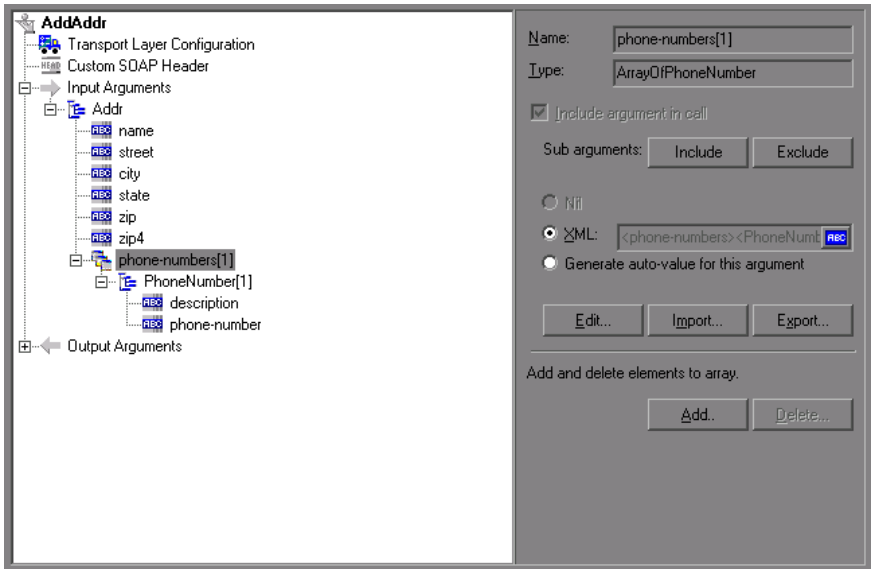
You can view the output argument values and save them to parameters or in an array.



➤ **Save returned value in parameter.** Saves the returned value to a parameter whose name you specify in the text box.

Arrays

To work with an array—for either input or output arguments, select it in the left pane.



➤ **XML.** The path of the XML file containing the values of the array elements. Click the **ABC** icon to replace the XML with an XML type parameter. XML parameterization supports arrays as input arguments. In the XML parameter, you define the number of array elements as required.

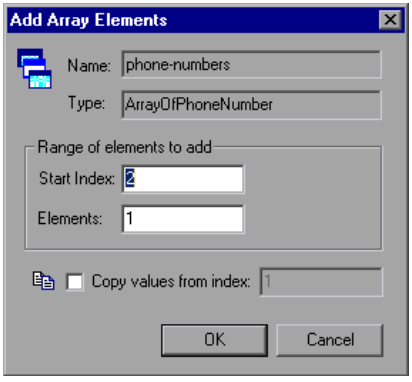
When saving an array to a parameter, the number of array elements per parameter is constant. If you want to run multiple iterations, with each iteration using a different number of array elements, you need to define separate parameters, each containing the desired number of array elements.

For more information about XML parameters, see "Setting Properties for XML Parameters" on page 455.

- **Edit/Import/Export.** To modify complex types and arrays, select the elements and arguments and click **Edit**. Click **Export** to export the selected entry to a separate XML file, or **Import** to load a previously exported XML file. **Note:** This **Import** operation handles XML files that were previously exported—not standard SOAP files. To import SOAP, see "Importing SOAP Requests" on page 210.
- **Add.** Opens the Add Array Elements dialog box, allowing you to add array elements, either simple or complex.
- **Delete.** Deletes array elements. Specify the starting index and the number of elements to remove.

Adding Array Elements

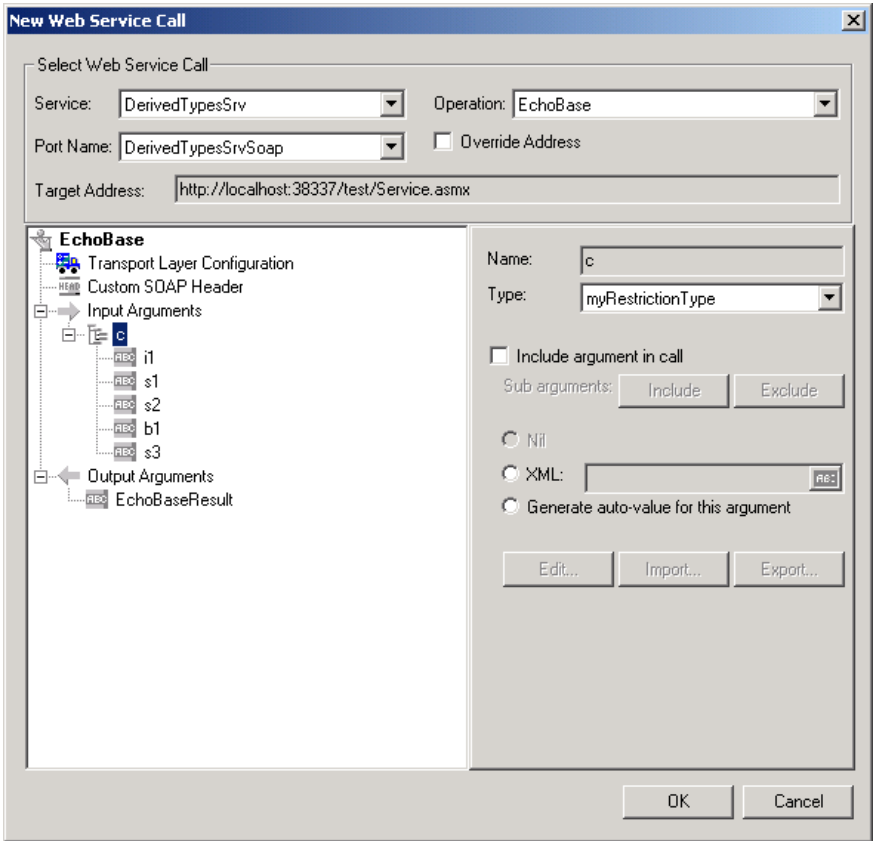
When you click **Add** in the Array Elements section, the Add Array Elements dialog box opens as described below.



- **Start Index.** The index of the first element that will be added.
- **Elements.** The number of elements to add.
- **Copy values from index.** Assigns values of an existing array element to the new elements. Specify the array index of the element whose value you want to use.

Derived Types

Service Test supports WSDLs with derived types. When setting the properties for a Web Service Call, you can set the arguments to use the base type or derived type.



After you select the desired type, Service Test updates the argument tree node to reflect the new type.

Abstract Types

Abstract is a declaration type declared by the programmer. When an element or type is declared to be **abstract**, it cannot be used in an instance document. Instead, a member of the element's substitution group, provided by the XML schema, must appear in the instance document. In such a case, all instances of that element must use the **xsi:type** to indicate a derived type that is not abstract.

When Service Test encounters an Abstract type, it cannot create an abstract class and replay will fail. In this case, Service Test displays a warning message beneath the **Type** box, instructing you to replace the Abstract type with a derived type.

Working with Optional Parameters

If your WSDL file contains optional parameters, you can indicate whether or not to include them in the SOAP request.

In WSDL files, optional parameters are defined by one of the following attributes:

minoccurs='0'

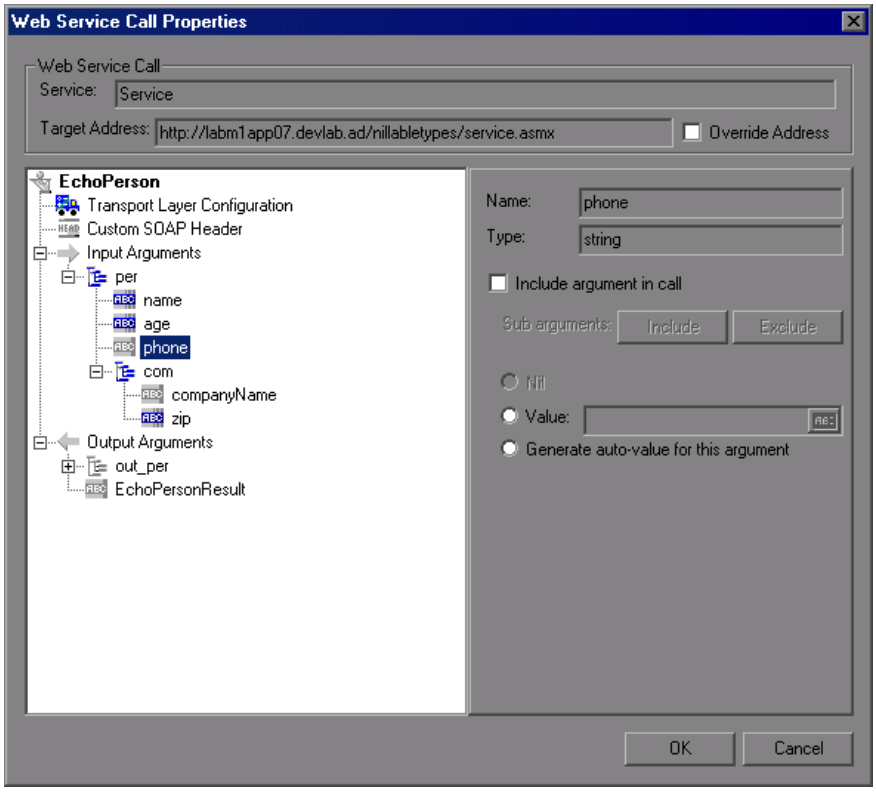
nillable='true'

minoccurs = 0 indicates a truly optional element, that can be omitted. Nillable means that the element can be present without its normal content, provided that the nillable attribute is set to true or 1. By default, the **minoccurs** and **maxoccurs** attributes are set to 1.

In the following example, **name** is mandatory, **age** is optional, and **phone** is nillable.

```
<s:element minOccurs="1" name="name" type="s:string" />
<s:element minOccurs="0" name="age" type="s:int" />
<s:element minOccurs="1" name="phone" nillable="true" type="s:string" />
```

When setting argument values for your service call, Service Test indicates the type of element by enabling or disabling the options:



The following table indicates the availability of the options:

Parameter type	Nil radio button	Include arguments in call
Mandatory	disabled	disabled
MinOccurs=0	disabled	enabled
Nillable	enabled	disabled

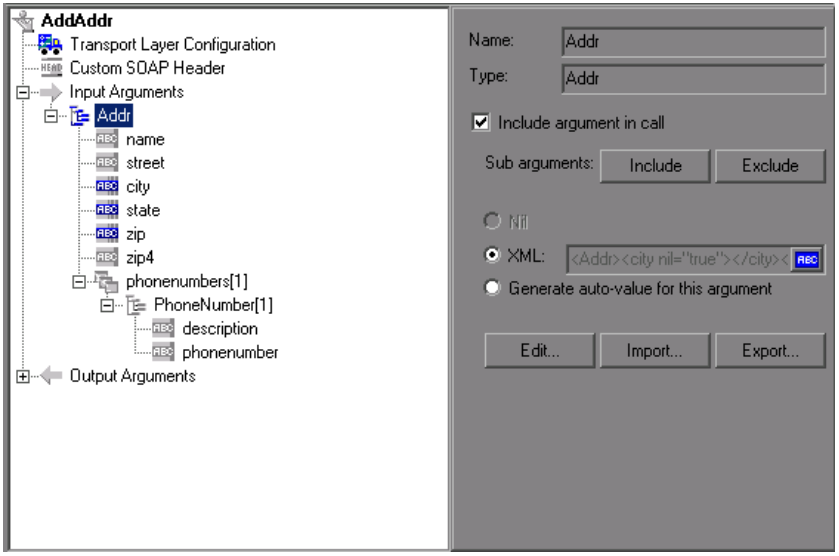
To include a specific optional argument in the service call, click the node and select **Include Argument in Call**. The nodes for all included arguments are colored in blue. Arguments that are not included are colored in gray.

If you include an element on a parent level, it automatically includes all mandatory and nillable children elements beneath it. If it is a child element, then it automatically includes the parent element and all other mandatory or nillable elements on that level. If you specify **Generate auto-value** to a parent element, Service Test provides values for those child elements that are included beneath the parent.

Note: Service Test interprets whether elements are mandatory or optional through the toolkit implementation. This may not always be consistent with the element's attributes in the WSDL file.

To include a sub elements:

1. To include a specific sub element, select it in the left pane and select the **Include Argument in Call** option.
2. To include all sub elements of a parent element, apply **Include Argument in Call** to the parent element and click the **Include** button beneath it.
3. To exclude all sub elements, select the parent element and click the **Exclude** button. It will exclude all omittable arguments.

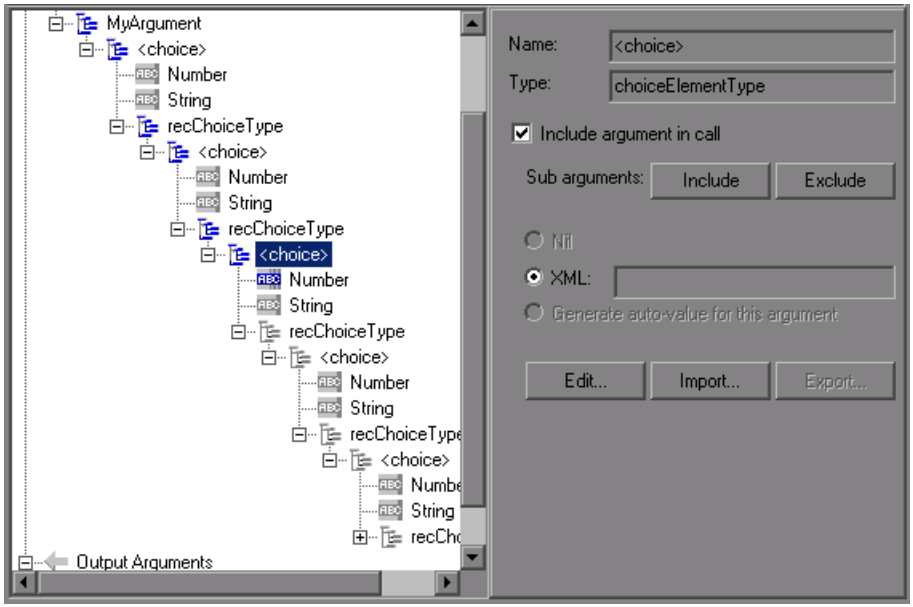


Recursive Elements

Using the Properties dialog box, you can control the level of recursive elements to include in the Web Service call.

To exclude a certain level and exclude those below, highlight the lowest parent node that you want to include and select **Include Argument in Call**. Service Test includes the selected nodes, its mandatory children, and all of its parent nodes.

In the following example, three levels of the Choice argument are included—the rest are not. A non-included node is grayed out.



Choice Optional Elements

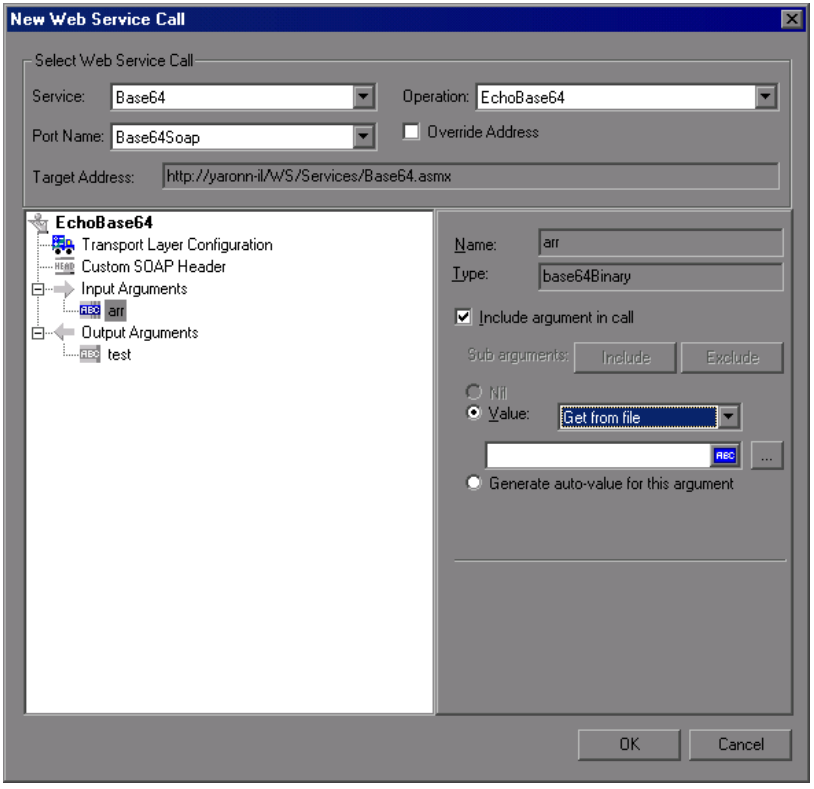
A Choice element in a WSDL defines a set of elements where only one of them appears in the SOAP message. In some cases, one of the Choice elements is optional, while the others are not. In Service Test, you can select the Choice element and still prevent its optional element from appearing in the SOAP envelope. In Tree view, select the Choice element, and clear the **Include argument in call** option. In Script view, delete the line that defines the Choice argument.

Base 64 Encoding

Base 64 encoding is an encoding method used to represent binary data as ASCII text. Since SOAP envelopes are plain text, you can use this encoding to represent binary data as text within SOAP envelopes.

When Service Test detects a WSDL element of **base64Binary** type, it lets you provide an encoded value. You can specify a value in two ways:

- **Get from file.** Reference a file name.
- **Embed encoded text.** Specify the text to encode.

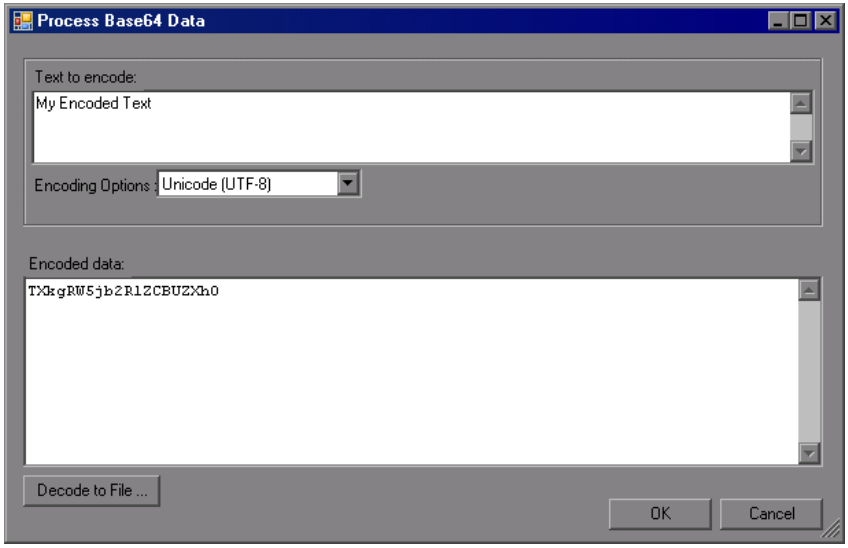


To specify a base64Binary value:

1 Select the **Value** option.

2 To specify a file, select **Get from file** and locate the file using the Browse button below.

3 To specify text, select the **Embed encoded text** option and click the Browse button below. The Process Base64 Data dialog box opens.



Enter text in the **Text to encode** box.

To use an encoding other than the default UTF-8 method, select it from the **Encoding Options** list.

Click **Encode**.

4 Click **OK**. Service Test adds the Web Service call to the script. You can now view the step and its properties in Tree view.

If you referenced the value from a file, the Web Service call will contain the file name:

```
"xml:arr="
    "<arr base64Mode=\"file\">C:\\Load_testing\\TEcho.xml</arr>",
```

If you inserted the actual text using the **Base 64 Encoding Text** option, then the Web Service call in the script will contain the encoded text.

```
"xml:arr="
    "<arr base64Mode=\"encoded\">YWJjZGVmZw==</arr>",
```

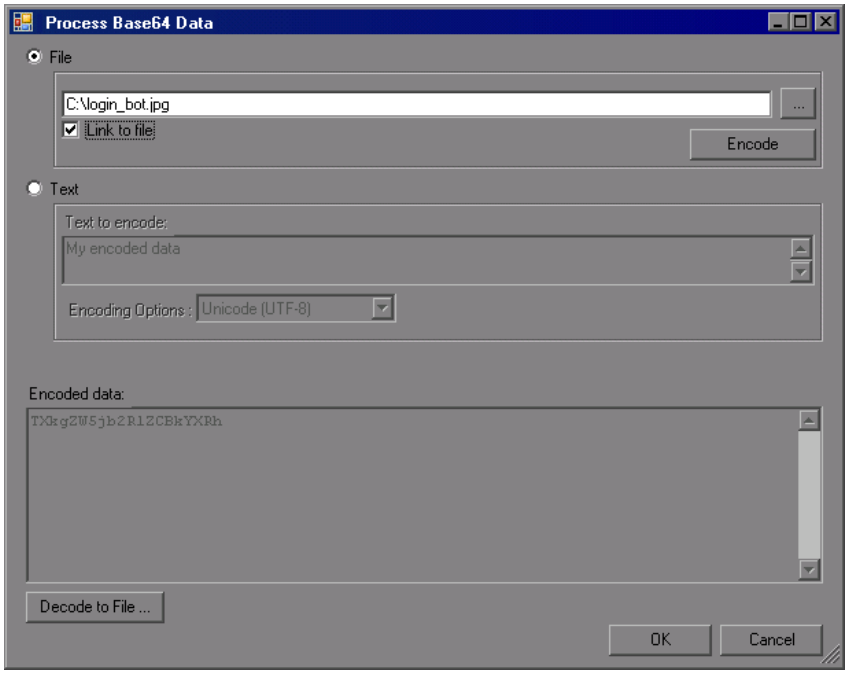
Setting a Parameter Value for Base64Binary Data

To set the Base64 argument value to a parameter, create a new parameter of of File type, or XML type for Complex arguments. For more information, see "Setting Properties for XML Parameters" on page 455.

For complex type arguments containing base64Binary values, Service Test lets you process the base64Binary for setting parameter, checkpoint, or emulation values. When specifying the values, you can get values from a **File** or specify the **Text** manually and apply encoding.

If you choose to get the value from a file, specify one of the following options:

- **Link to file.** Reference the file containing the values.
- **Do not Link to a file.** Use the content of the specified file. Service Test copies the content to the script folder. To use this option, clear the **Link to file** check box.



Tip: It is generally recommended to link to a file since this improves the script's performance. If your text exceeds 10KB, you must link to a file.

To open the Process Base64 dialog box:

1. Create a new parameter for a complex argument.
2. In the grid view (XML parameter, checkpoints, or Service Emulation), click the Browse button to the right of the value box. The Process Base64 dialog box opens.

3 Specify **File** or **Text** as a source for the value.

4 If you chose **File** as the Source, specify whether or not to link to a file.

5 To decode an encrypted value, for example, a value obtained during replay, click **Decode to File**. For more information, see below.

This section applies to all of the places within Service Test that use the grid view of argument values: the parameter list, checkpoints, and Service Emulation.

Decoding to a File

Service Test lets you decode the encoded text to a file. This is especially useful for checking the correctness of base64 encoded values returned from the server, such as images.

The following procedure describes how to check if the Record and Replay images match one another.

To validate images using decoding:

1 Create a New Web Service call.

2 Set a value for the Base64 argument, using the **Get from file** option. Specify an image file. Continue creating the script.

3 Save the script and replay it.

4 Switch to the Checkpoint tab and load the Replay values.

5 Click on the Replay value of the Base 64 argument and open its properties.

6 Click **Decode to file**. Specify a file name to which to save the file. Use the same extension as the original file.

7 Compare the decoded image to your original one to verify a match.

Attachments

When transferring binary files such as images over SOAP, the data must be serialized into XML. Serialization and deserialization can cause a significant amount of overhead. Therefore, it is common to send large binary files using an attachments mechanism. This keeps the binary data intact, reducing the parsing overhead.

Using attachments, the original data is sent outside the SOAP envelope, eliminating the need to serialize the data into XML and making the transfer of the data more efficient.

The formats used for passing a SOAP message together with binary data are MIME (Multipurpose Internet Mail Extensions) and the newer, more efficient DIME (Direct Internet Message Encapsulation) specifications. Service Test supports DIME for all toolkits, but MIME only for the Axis toolkit. To use MIME attachments for the .NET toolkit, see "Including MIME Attachments" on page 386.

Service Test supports the sending and receiving of attachments with SOAP messages. You can send Input (Request) or save Output (Response) attachments.

To add or save attachments, select an operation or a method in the left pane to which the attachments will be associated. You can add both input or output attachments

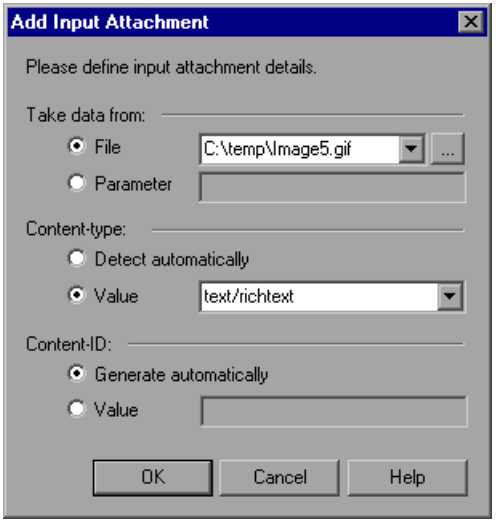
Input Attachments

Input attachments are added to the request message.

To add an attachment to the request:

1. Select the operation in the left pane, to which you want to add the attachment.
2. In the right pane, select **Add to request (Input)**. Service Test prompts you to enter information about the attachment and adds it to the method's tree structure.

The Add Input dialog box opens.



Specify the following information:

- **Take data from.** The location of the data. This can be a file or a parameter that contains the binary data.
  - **File.** You can specify the file location in two ways:
    - Absolute Path: The full path of the file. Note that this file must be accessible from all machines running the script.
    - Relative Path: (recommended) A file name. Using this method, during replay, Service Test searches for the attachment file in the script's folder. To add it to the script's folder, select **File** > **Add Files to Script** and specify the file name.
  - **Parameter.** You specify the name of a parameter containing the data.
- **Content-type.** The content type of the file containing the data. The **Detect Automatically** option instructs Service Test to automatically determine the content type. You can also select from a list of the common content types in the **Value** box, such as text/html, and image/gif or type in another content type.

➤ **Content-ID.** The ID of the content. By default, Service Test generates this automatically by Service Test and serves as a unique identifier for the attachment. Optionally, you can specify another ID in the **Value** box.

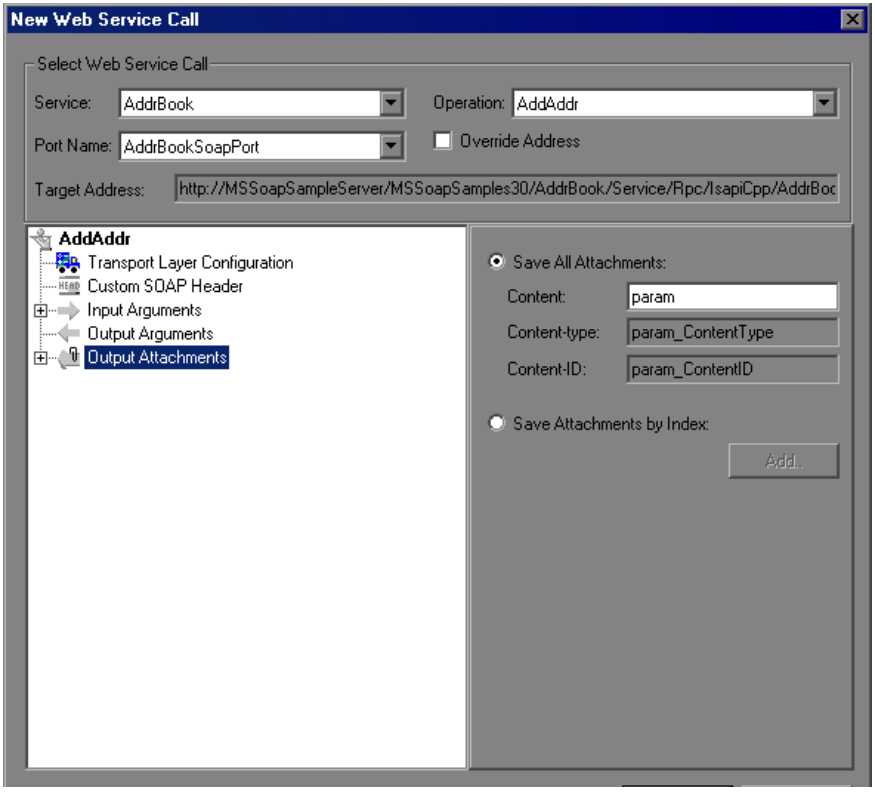
Output Attachments

Output attachments are added to the response message.

To save the response as an attachment:

1. Select the operation in the left pane, for which you want to save the response.
2. In the right pane, select **Save received (Output)**. Service Test adds an Output Attachment node to the method's tree structure in the left pane.

3 Select the desired option: **Save All Attachments** or **Save Attachment by Index** based on their index number—beginning with 1.



When you specify **Save All Attachments,** Service Test creates three parameters for each attachment based on the parameter name that your specify: a parameter containing the attachment data, the content type of the attachment, and a unique ID for the attachment.

For example, if you specify the name MyParam in the Content field, the parameter names for the first attachment would be:

```
MyParam_1
MyParam_1_ContentType
MyParam_1_ContentID
```

When you specify **Save Attachments by Index**, you specify the index number and name of the parameter in which to store the attachment. The parameter name that you specify for **Content**, is used as a prefix for the Content type and Content ID parameters.

To edit the properties of either an Input or Output attachment, click the attachment in the left pane, and enter the required information in the right pane.

SOAP Headers

This view is available when you select **SOAP header** in the method's tree view. The right pane lets you indicate whether or not to use SOAP headers. To use them, select **Use SOAP header**. Note that you must individually specify SOAP headers for each element. You can import XML code for the SOAP header, or compose your own using the Edit XML option. For more information, see "Editing an XML Tree" on page 272.

Querying an XML Tree

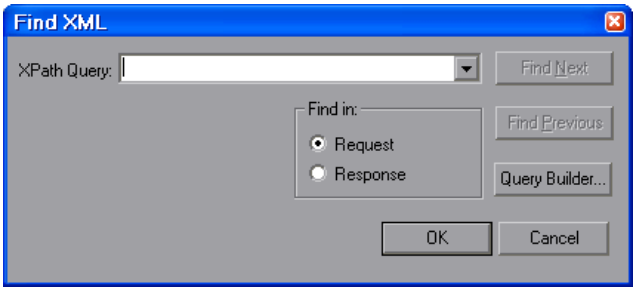
You can query an XML tree in order to locate and examine a specific element and value.

To search an XML, you use a query in the standard XPATH syntax. To build a valid XPATH query, use the built-in Query Builder. You can perform a query on your XML document, and search for a specific Namespace URI, value, or attribute. Note that all queries are case-sensitive.

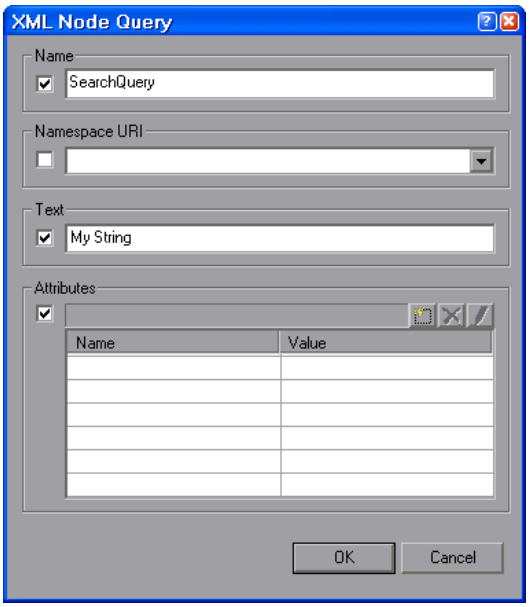
To perform a query:

1 In the snapshot's Tree view, select the Request or Response snapshot and click the **Find XPath** button.

2 Enter an XPATH query.



3 Click **Query Builder** for help in building an XPATH query. The XML Node Query dialog box opens. Enable one or more items for searching.

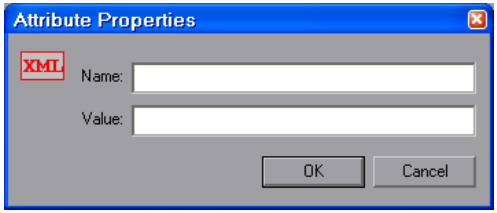


Enter the search text in the appropriate boxes.

- Enable the **Name** section to search for the name of a node or element.
- Enable the **Namespace URI** section to search for a namespace.
- Enable the **Text** section to search for the value of the element indicated in the Name box.

➤ Enable the **Attributes** section to search for an attribute.

➤ To add an attribute, click the **Add** button. The Attribute Properties box opens. Enter an attribute name and value. Click **OK**.



4 Click **OK** in the XML Node Query dialog box. Service Test places the text of the query in the XPath query box.

5 Click **Find Next** to begin the search.

Working with the XML

Web Services allow you to view and edit your XML.

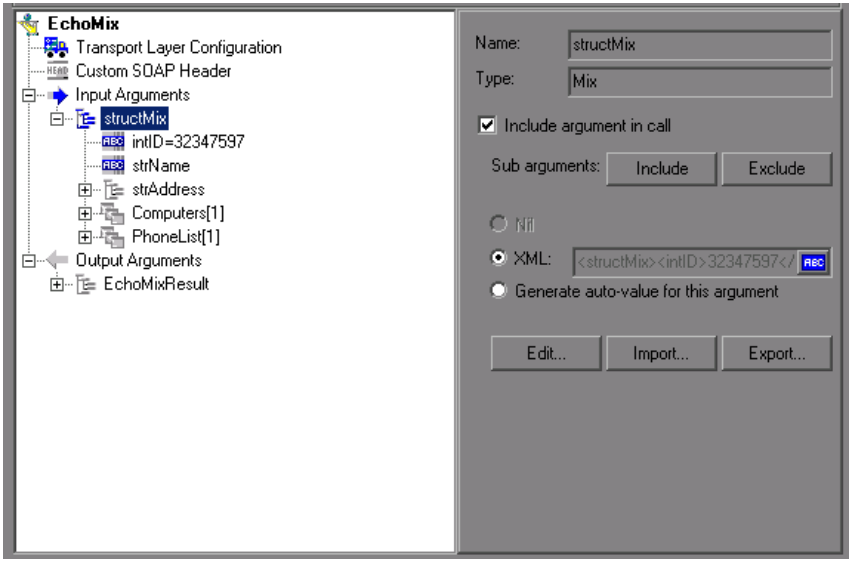
The following sections describe:

➤ Editing an XML Tree

➤ Saving and Copying the SOAP Response

Editing an XML Tree

You can use Service Test's XML Editor to view and edit the XML representation of complex types (structures, objects, etc.) and arrays.

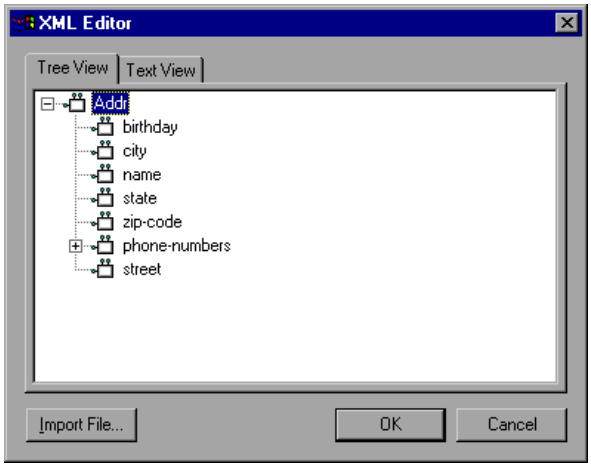


Entering the values for the XML elements is a tedious and error-prone task. Service Test provides you with an interface that simplifies the task of entering, saving, and restoring the information. Once you enter the data manually, you can save it to an XML file using the **Export** option. For subsequent tests, you can import this file without needing to reenter the values a second time.

To edit XML strings:

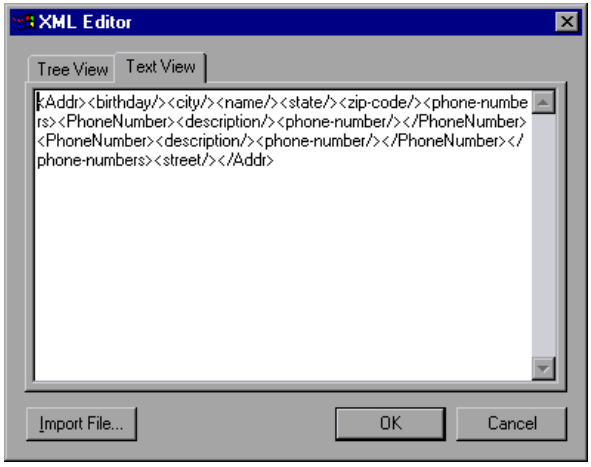
1. Select the Web Service call whose element you want to modify and click the **Step Properties** tab.
2. In the method's tree hierarchy, click on a complex type or array argument. The right-most pane shows the XML code as a single string. Select the **XML** option.
3. To edit the XML code for that entry, click **Edit**. Service Test may issue a warning indicating that only changes to element values and the number of array elements will be saved—not changes in the XML elements themselves.

4 Click **OK**. The XML Editor dialog box opens.



5 In the XML Tree view, double-click on a node to open its property dialog box. Edit the value as required. Click **OK** to save the new values.

6 To edit the code in text mode, click the **Text View** tab. Edit the XML code manually. Click **OK** to save the changes.



7 To import a previously saved XML file, click **Import** and specify the file's location. Edit the file in the XML Editor dialog box.

8 To save your XML data to a file so it can be used for other tests, click **Export** and specify a location.

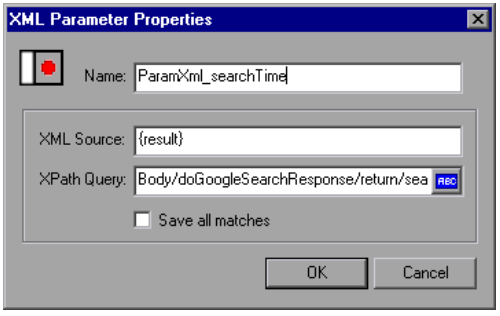
Saving and Copying the SOAP Response

In addition to saving the input argument values as an XML type parameter, you can also save the SOAP response to a parameter or copy it for use within an editor.

To save the SOAP response to a parameter:

1 Switch to the **SOAP Snapshot** tab and select the parent or child element whose value you want to parameterize.

2 Select **Save XML in parameter** from the right-click menu. The XML Parameter Properties dialog box displays the properties of the selected XML element.



3 Specify a name for the XML parameter, and click **OK**.

To copy the XML structure for use within another editor, select **Copy XML** from the right-click menu.

Part III

Running Web Service Scripts

12

Web Services - Preparing for Replay

After you create a Web Services script, you prepare it for replay so that it can accurately emulate your environment. After replay, you view the test results to see whether the services performed as expected.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Preparing Web Services Scripts for Replay

After you create a script with Web Service calls, you prepare it for replay.

You can enhance it with custom error and log messages or with transactions. See Chapter 5, "Enhancing Vuser Scripts" for more information.

In addition, you can enhance your script with JMS functions, **jms_<suffix>** or XML functions, **lr_xml_<suffix>**. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

You can configure run-time settings to help you emulate real users more accurately. These settings include general run-time settings (iteration, log, think time, and general information). Web Services specific settings relate to the JMS transport method, and are described in "Web Services JMS Run-Time Settings" on page 286.

For more information about the general run-time settings, see Chapter 2, "Configuring Run-Time Settings."

Before you replay the script, you can set up checkpoints to make sure that you are getting the correct response from the server. For more information, see below.

In certain cases, you may need to use the result of one Web Service call as input for another. To do this, you save the result to a parameter and reference it later. For more information, see "Using Web Service Output Parameters" on page 289.

Setting Checkpoints

In functional testing, one of the most important tasks is to check the response from the server to confirm that your test performed the actions correctly. In Web Services, the response can contain several arguments, each containing several data items. The **Checkpoint** tab is a central point for defining the required response values for your test.

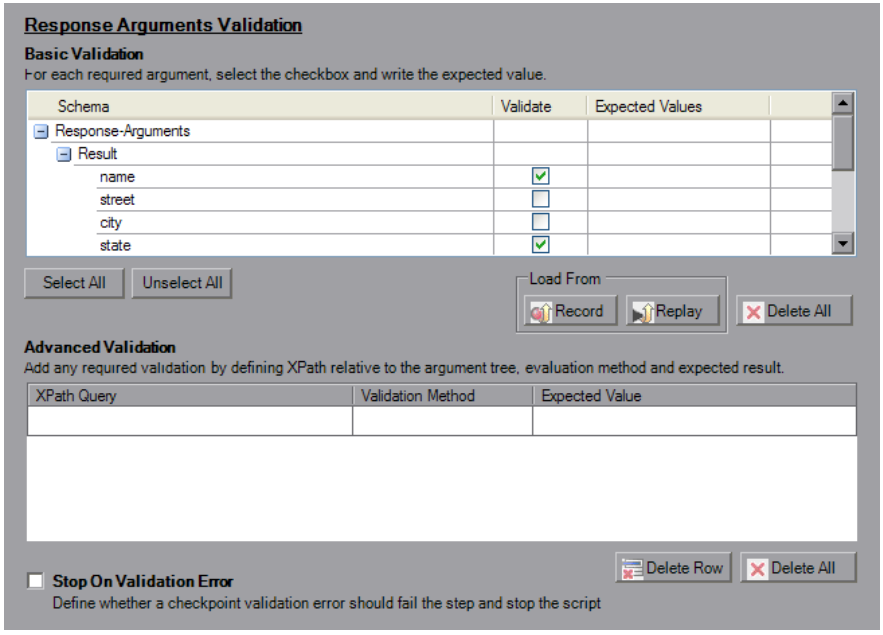
Tip: For a more comprehensive validation of the response, use the XML Validation tool. This tool lets you insert independent validation steps into your script and load a complete XML tree as an expected value. For more information, see "Checking for the Expected Response" on page 149.

About Validating Results Using the Checkpoint Tab

Before replaying a test, you set expected values for the arguments. You can load a set of expected values as they were captured either during recording or during replay. This is useful when you have many argument values—instead of manually entering values, you automatically load them.

After the test run, you can view the Replay log or the test results and determine if the results were as expected.

Service Test automatically displays all of the method's arguments with a check box. To include a checkpoint in the test, you select its check box. You can load the recorded or replay values (if they exist) and then select only those that you want to check.



An optional **Stop on Validation Error** flag indicates whether or not the step fails in case of a checkpoint failure.

The script generates a checkpoint argument for each row that you enable in the **Checkpoint** tab.

```
web_service_call("StepName=Add_2",
        "SOAPMethod=Calc.CalcSoapPort.Add",
…
        BEGIN_CHECKPOINTS,
            StopOnValidationError=1,
            CHECKPOINT, "XPATH=Result[1]", "Value=13",
            CHECKPOINT, "XPATH=AddResult", "Expression=Hel*?",
        END_CHECKPOINTS,
        LAST);
```

The Checkpoint validation also provides support for standard XPATH validations using **lr_xml_find**.

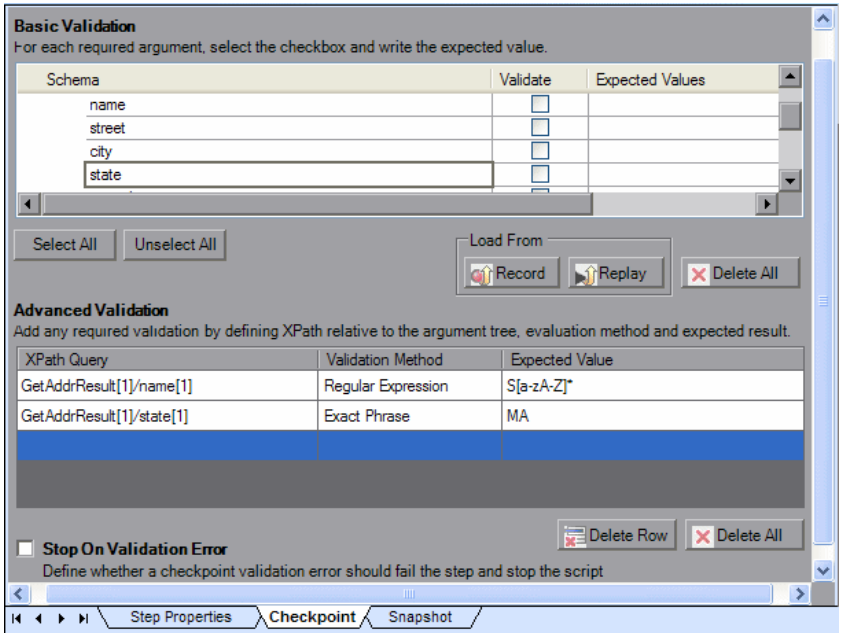
For verification of the SOAP body (or SOAP headers with the .NET toolkit), you can use checkpoints. However, when using a toolkit other than .NET, checkpoints are not supported for SOAP headers—instead use **lr_xml_find**. For more information, see the *Online Function Reference* (**Help > Function Reference**).

Basic and Advanced Validation

You can perform both **Basic** or **Advanced** validations.

In **Basic Validation**, Service Test looks for exact matches of the value in the **Expected value** column. You can load expected values from a Recorded or Replay snapshot.

Use **Advanced Validation** to validate a checkpoint on non-leaf nodes or to define expected values in terms of a regular expression.



You define the advanced validation values by entering an XPATH query in the **Advanced Validation** section. To obtain the initial XPATH expression, copy it from a row in the **Basic Validation**. Choose **Copy Row XPATH** from the right-click menu and paste the contents in the **Advanced Validation** section.

You can define both basic and advanced validations for the same step.

Note that when you select a non-leaf node, you need to supply all of the XML beneath the node.

In the Vuser script, Service Test indicates an exact match by **Value=** and a regular expression with **Expression=**:

```
BEGIN_CHECKPOINTS,
        CHECKPOINT, "XPATH=/AddResult[1]", "Value=50"
        CHECKPOINT, "XPATH=AddResult", "Expression=Hel*?",
END_CHECKPOINTS,
```

XPATH Expressions

Checkpoint expressions support both Nodeset and extended XPath syntax.

NodeSet expressions are XPath expressions evaluated to a single XML node or to a set of XML nodes. For example:

```
/a/b/c
/x/y[1]
//a/b
```

Service Test also supports full XPath expressions, such as **count(/a/b/c)** which is evaluated to int, or **count(/a/b/c) < 3** which is evaluated to a boolean value. The XPATH support allows you to do numeric comparisons. For example, to verify whether a result is between 8 and 16, you can use the following expression:

```
number(/a/b/c) > 8 and number(/a/b/c) < 16
```

Setting Basic Checkpoints

The following section describes how to set a Basic checkpoint.

For a more comprehensive validation of the response, use the **XML Validation** tool. This tool lets you insert independent validation steps into your script and load a complete XML tree as an expected value. For more information, see "Checking for the Expected Response" on page 149.

To set a Basic checkpoint:

1 In Tree view (**View** > **Tree view**), select a step in the left pane.

2 Select the **Checkpoint** tab.

3 To check for exact matches, specify expected values in the upper **Basic Validation** section:

- ➤ To manually specify expected values, enter the values in the **Expected Values** column.
- ➤ To load data from a recording or replay session, click the **Record** or **Replay** buttons in the **Load From** section. Service Test fills in the data as it was captured during record or replay.

4 Select the check boxes in the **Validate** column for all the results you want to check. To select all Basic validation checkpoints, click **Select All**. To clear all of the selections, click **Unselect All**.

5 Click **Delete All** in the upper section to clear all of the expected values.

6 Select **Stop On Validation Error** to instruct the Vusers to fail the step when the replay did not generate the expected values.

7 Run the script and view the Replay log to determine if the service returned the expected values. We recommend that you enable the Extended log in the run-time settings. If there is no match, the Replay log issues an appropriate message:

```
Action.c(14): Failure: checkpoint "/AddResult[1]" expected value="3" actual result="15"
Action.c(14): Error: Web service call "Add" 1/1 checks failed
```

8 Open the Test Results (**View > Test Results**) to see a detailed report of the validation. For more information about viewing test results, see below.

Setting Advanced Checkpoints

The following section describes how to set an Advanced checkpoint.

To set an Advanced checkpoint:

1 In Tree view (**View > Tree view**), select a step in the left pane.

2 Select the **Checkpoint** tab.

3 Provide expected values in the bottom section, **Advanced Validation**:

- An **XPATH Query** expression describing the criteria of the search. You can copy XPATH expressions from the **Basic Validation** section in the upper window. To copy an XPATH expression, select the text in the **Schema** column, and select **Copy Row XPath** from the right-click menu. In the **Advanced Validation** section, double-click in the next available row and select **Paste** from the right-click menu. Modify the expression as required.
- A **Validation Method**: Select **Exact Phrase** or **Regular Expression** from the drop-down list.
- The **Expected Value**, either in the form of an exact value or a regular expression.

4 To delete an advanced checkpoint, select it and click **Delete Row** in the **Advanced Validation** section.

5 To delete all of the advanced checkpoints, click the **Delete All** button in the **Advanced Validation** section.

6 Select **Stop On Validation Error** to instruct the Vusers to stop when the replay did not generate the expected values.

7 Run the script and view the Replay log, which contains information on whether or not the match was found.

8 Open the Test Results (**View > Test Results**) to see a detailed report of the validation. For more information about viewing test results, see "Viewing Test Results" on page 91.

Viewing Checkpoint Results

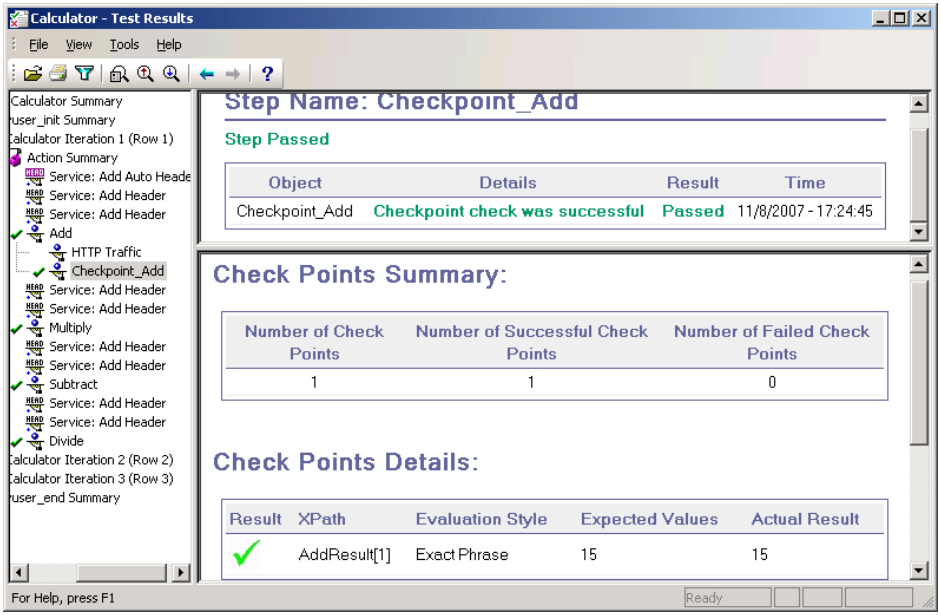
After running a script, you can view the checkpoint results to see their status—Passed or Failed, and the reason for the failure.

To view the test results for the checkpoints:

1 Select **View > Test Results** to open the Test Results window.

2 In the left pane, expand the step whose checkpoint you want to view.

3 Click on the Checkpoint step in the left pane. The right pane shows the details about the test run.



The lower pane provides detailed information about the checkpoint:

- The number of successful and failed checkpoints (for multiple iterations)
- The expected values and actual results
- The type of evaluation (exact phrase or regular expression)
- The response argument tree

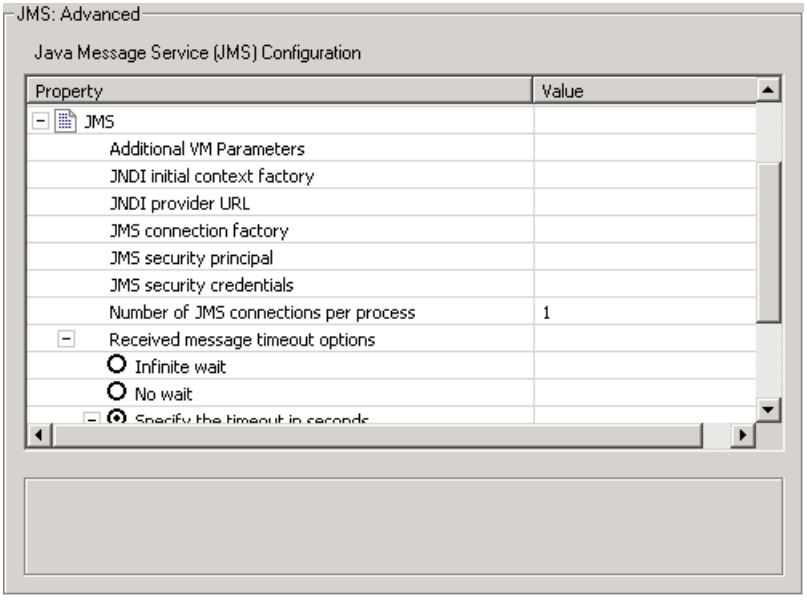
For more information about viewing test results, see Chapter 4, "Viewing Test Results."

Web Services JMS Run-Time Settings

To use JMS as a transport for Web Service calls, there are several resources that need to be allocated and configured. Those resources include the JVM, JNDI initialization parameters, JMS resources, and timeout values. For information on setting the transport level, see Chapter 16, "Web Services - Transport Layers and Customizations."

Service Test lets you configure some of those resources through the run-time settings.

You can set options in the area of VM (Virtual Machine), the JMS connections, and message timeouts.



VM

- **Use external VM.** Enables you to select a VM (Virtual Machine) other than the standard one. If you disable this option, Vusers use the JVM provided with Service Test.
- **JVM Home.** The location of the external JVM. This should point to the JDK home directory, defined by JDK_HOME. Service Test supports JDK 1.4 and above.

- **Classpath.** The vendor implementation of JMS classes together with any other required supporting classes, as determined by the JMS implementation vendor

JMS

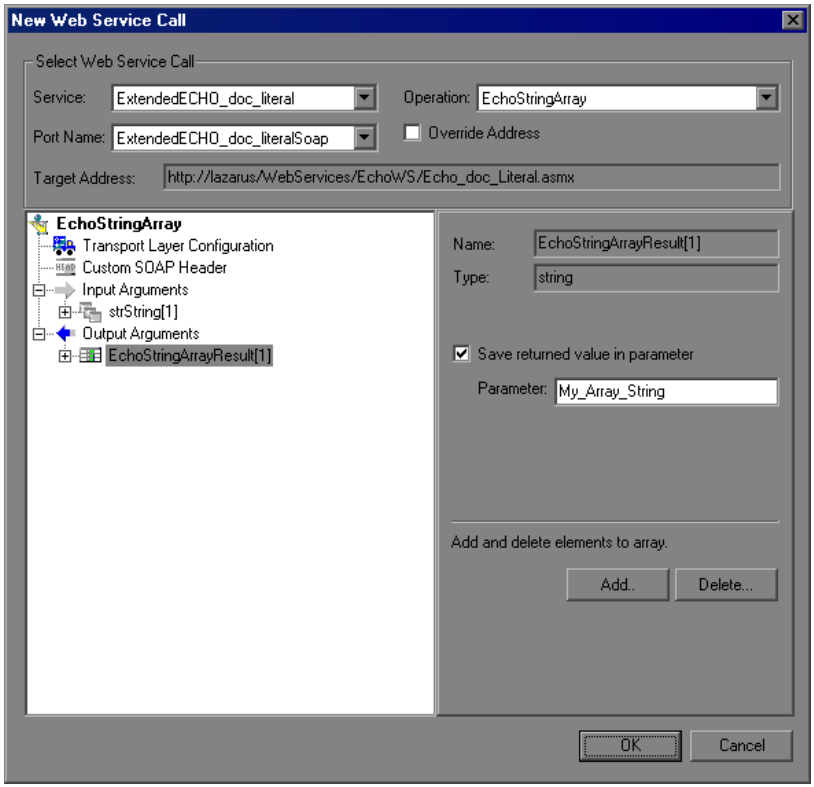
- **Additional VM Parameters.** Extra parameters to send to the JVM such as Xbootclasspath, and any parameters specified by the JVM documentation.
- **JNDI initial context factory.** The fully qualified class name of the factory class that will create an initial context. Select a context factory from the list or provide your own.
- **JNDI provider.** The URL string of the service provider. For example:
  Weblogic - t3://myserver:myport
  Websphere - iiop://myserver:myport
- **JMS connection factory.** The JNDI name of the JMS connection factory. You can only specify one connection factory per script.
- **JMS security principal.** Identity of the principal (for example the user) for the authentication scheme.
- **JMS security credentials.** The principal's credentials for the authentication scheme.
- **Number of JMS connections per process.** The number of JMS connections per **mdrv** process, or Vuser. All Vusers sharing a connection will receive the same messages. The default is 1, and the maximum is 50 Vusers. The fewer connections you have per process, the better your performance.

- **Receive message timeout options.** The timeout for received messages. The default is **No wait**.
  - **Indefinite wait.** Wait as long as required for the message before continuing.
  - **No wait.** Do not wait for the Receive message, and return control to the script immediately. If there was no message in the queue, the operation fails.
  - **Specify the timeout in seconds.** Manually specify a timeout value for the message. If the timeout expired and no message has arrived, the operation fails. (default)
  - **User defined timeout.** Specify the amount of seconds to wait for the message before timing out. The default is twenty seconds.
- **Automatically generate selector.** Generates a selector for the response message with the correlation ID of the request (**No** by default). Each JMS message sent to the server has a specific ID. Enable this option if you want Service Test to automatically create a selector that includes the message ID.

Using Web Service Output Parameters

In certain cases, you may need to use the result of one Web Service call as input for another. To do this, you save the result to an output parameter and reference it at the required point.

In the following example, the output argument is saved to a parameter, **My_Array_String**.

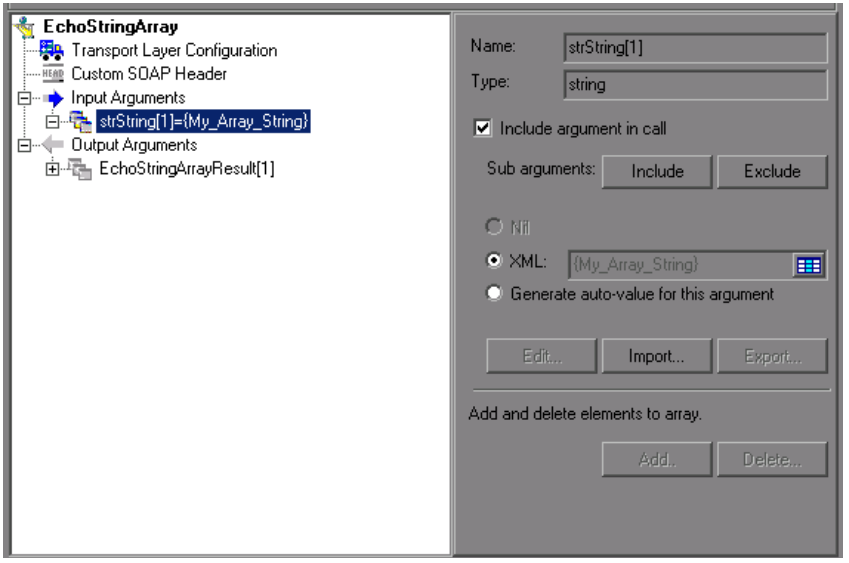


The script shows the saved output parameter as a result argument:

```
web_service_call( "StepName=EchoStringArray_101",
"SOAPMethod=ExtendedECHO_doc_literal.ExtendedECHO_doc_literalSoap.EchoStri
ngArray",
        "ResponseParam=response",
        "Service=ExtendedECHO_doc_literal",
        "Snapshot=t1169994766.inf",
        BEGIN_ARGUMENTS,
        "xml:strString=<strString><string></string></strString>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        "EchoStringArrayResult/*[1]=My_Array_String",
        END_RESULT,
        LAST);
```

For information on saving results to parameters, see "Saving Output Parameters" on page 291.

After you save an output parameter, it becomes available for parameter substitution or for other referencing, such as evaluating it and printing its value. In the following example, the saved output parameter, **My_Array_String**, is used as an input argument for a subsequent Web Service call.



For information on using saved output parameters, see "Using Saved Parameters for Input" on page 292.

Saving Output Parameters

You can save multiple result arguments to parameters through the Properties dialog box, or by manually editing them in the script code.

You can also save result parameters from XML actions, and use them as input arguments. For example, if you save the result parameter for **lr_xml_insert**, you can reference the saved parameter in a subsequent Web Service call. For more information, see the *Online Function Reference*.

To save an output parameter:

1. **View the script in Tree view.**

   Make sure you are in Tree view. Otherwise, select **View > Tree view**.

2. **Check for a Service.**

   Click the **Manage Services** button to verify that you have imported at least one service. To import a new service, click **Import** in the Service Management dialog box.

3. **View the step's properties.**

   For a new Web Service call, click **Add Service Call**.

   For an existing Web Service call, double-click on the step or click the **Properties** tab in the right pane.

4. **Select the output argument.**

   In the left pane, select the output argument whose value you want to save to a parameter.

5. **Enable the saving of the output parameter.**

   In the right pane, select **Save returned value in parameter**. Accept the default name or specify a custom name.

Using Saved Parameters for Input

After saving output parameters, you can use them in subsequent Web Service calls.

You can also use saved result parameters from XML actions as input. For example, if you saved the result parameter for **lr_xml_insert**, you can reference it in a subsequent Web Service call. For more information, see the *Online Function Reference*.

To use a saved parameter for input:

1 View the step properties in Tree view.

For a new Web Service call, click **Add Service Call**.

For an existing Web Service call, double-click on the step or click the **Properties** tab in the right pane.

2 Select the input argument.

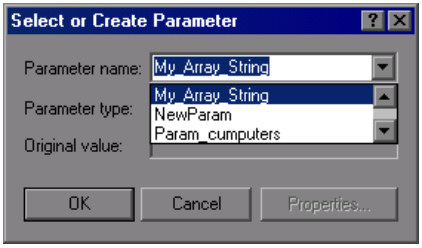
In the left pane, select the input argument whose value you want to replace with a previously saved output parameter.

3 Open the Select Parameter dialog box.

In the right pane, select **Value**, and click on the ABC icon adjacent to the **Value** box. The Select or Create Parameter box opens.

4 Select an output parameter.

Select the desired output parameter from the drop-down list and click **OK**.



To specify an input parameter in Script view, select the value you want to replace and select **Use Existing Parameters** from the right-click menu. Select one of the available parameters.

Note: If you modify an output parameter name in Script view, it will not be updated in the parameter list until you switch to Tree view.

Handling Special Cases

This section provides guidelines for running scripts in special cases.

Any Type XSD

For Web Services that have an XSD schema with an **Any** type element, <xsd:element name="<*Any_element*>" type="xsd:anyType" />, make sure your script conforms with the following model:

```
BEGIN_ARGUMENTS,
                "xml:Any_element="
                        "<Any_element>"
                                "<string>the string to send</string>"
                        "</Any_element>",
END_ARGUMENTS,
```

The actual SOAP may differ slightly, but as long as your script conforms to the above model, it will run properly.

You can also send complex type elements for the <any> type. For example:

```
"xml:Any_element="
                        "<Any_element>"
                                "<myComplexTypeName>"
                                        "<property1>123</property1>"
                                        "<property2>456</property2>"
                                "</myComplexTypeName>"
                        "</Any_element>",
```

13

Web Services - Database Integration

Using Service Test, you can integrate with a live database service to retrieve data for your test.

This chapter includes:

About Database Integration

When testing your Web Service, it is vital that you use data that is accurate and up to date. If you use a snapshot of data from a past date, it may no longer be valid or relevant.

The database integration allows you to access values in a database during your test, ensuring that the data is up to date.

The database integration is useful in the following scenarios:

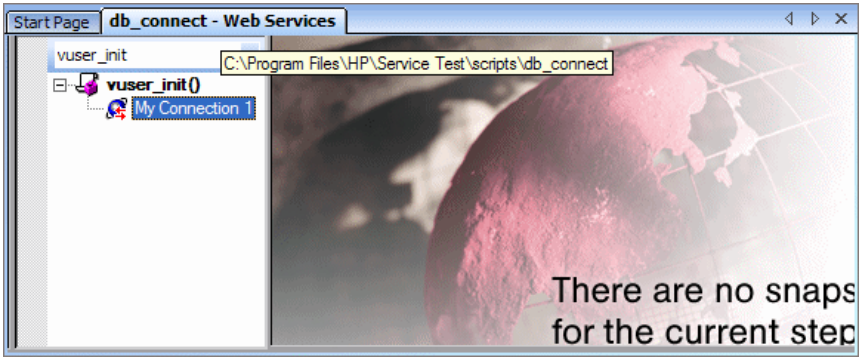
- Using Data Retrieved from SQL Queries
- Validating Database Values after a Web Service Call
- Checking Returned Values Through a Database

Service Test saves all of the database interaction information and displays it in the Test Results report. See Chapter 4, "Viewing Test Results" for more information.

Connecting to a Database

To connect to a database, you add a connection step to your script. Use the Add Step dialog box (**Insert > New Step**) in Tree view, to add a Database Connection step. A built-in Connection String Generator guides you in creating a database connection string specific to your database and credentials. You can also test your connection before inserting the step.

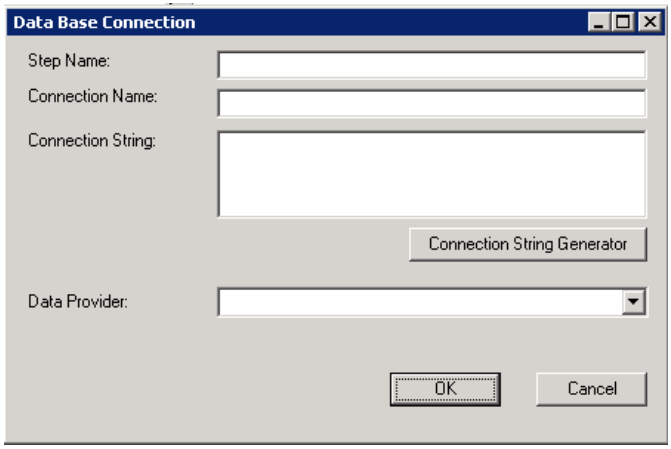
When running your script with iterations, virtual users only repeat the **Action** section of the script. If you include the database connection step in the **Action** section, the test will repeat it for each iteration. Virtual Users only repeat the **Action** section of the script, but not the **vuser_init** or **vuser_end** sections. Therefore, we recommend that you place the database connection step in the **vuser_init** section, and the disconnect step, **lr_db_disconnect** in the **vuser_end** section.



In cases where you only need to do one query and scroll through the data, you should also place the **Database: Execute SQL Query** step in the **vuser_init** section.

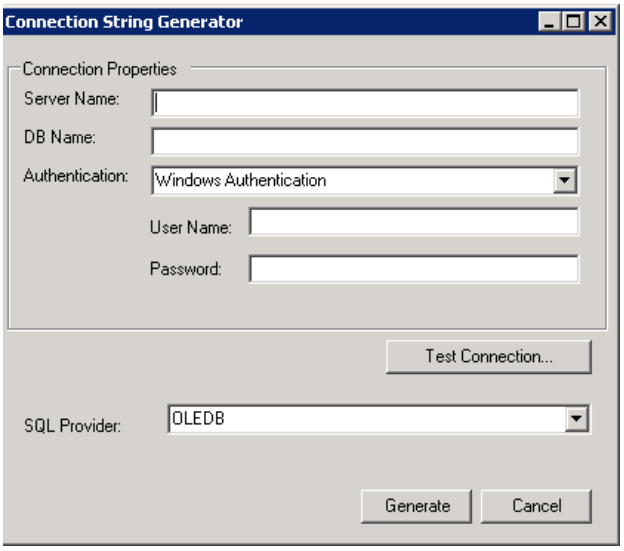
To add a database connection step through Tree view:

1. Select **View** > **Tree View** to enter Tree view (if it is not already visible).
2. Select the desired section: **vuser_init** or **Action**. We recommend placing the connection step in the **vuser_init** section. For more information, see below.
3. Select **Insert** > **New Step**. Choose the **Database: Connect** step. The Database Connection dialog box opens.



4. Specify a **Step Name**, **Connection Name**, and **Data Provider**, OLEDB or SQL.

5 Click **Connection String Generator** to generate a database connection string specific to your environment.



6 Indicate the connection properties:

- **Server Name**
- **Database Name**
- **Authentication** method: Windows Authentication or User/password.
- **Username** and **Password**

7 Click **Test Connection** to verify that the information you provided is correct.

8 Select an **SQL Provider**, OLEDB or SQL, and click **Generate**.

For more information about the required syntax of **lr_db_connect**, see the Online Reference (**Help** > **Function Reference**).

Using Data Retrieved from SQL Queries

In this scenario, the test fetches data from the database and uses it at a later point in the script, such as calls to the Web Service. Since the script retrieves the data during each test run, the data is up to date and relevant.

The following table shows a typical flow of the script:

Step	API function
Connect to database	lr_db_connect
Execute an SQL query	lr_db_executeSQLStatement
Retrieve and save the data	lr_db_getvalue to <param_name>
Web Service call	web_service_call with {<param_name>}
Disconnect from database	lr_db_disconnect

You can iterate through the results in two ways:

- save them to a simple parameter during each iteration
- use Service Test's built-in iterations to scroll through the data

For more information, see the Online Reference (**Help > Function Reference**).

In the following example, the **vuser_init** section connects to the database and performs a database query.

```
vuser_init()
{
lr_db_connect("StepName=myStep",
     "ConnectionString=Initial Catalog=MyDB;Data Source=mylab.net;user id =sa
;password = 12345;" ,
     "ConnectionName=MyConnection",
     "ConnectionType=SQL",
     LAST);

lr_db_executeSQLStatement("StepName=MyStep",
          "ConnectionName=MyConnection",
          "SQLQuery=SELECT * FROM Addresses",
          "DatasetName=ds1",
          LAST);

     return 0;
}
```

At the end of your test, disconnect from the database in the **vuser_end** section.

```
vuser_end()
{

     lr_db_connect("StepName=myStep",
          "ConnectionString=Initial Catalog=MyDB;Data Source=LAB1.devlab.net;user id
=sa ;password = soarnd1314;" ,
          "ConnectionName=MyConnection",
          "ConnectionType=SQL",
          LAST);

     return 0;
}
```

In the Action section, you include the steps to repeat. Note the use of the **Row** argument. In the first call to the database, you specify the first row with **Row=next**. To retrieve another value in the same row, use **current**.

```
Action()
{
    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=Name",
        "Row=next",
        "OutParam=nameParam",
        LAST);

    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=city",
        "Row=current",
        "OutParam=cityParam",
        LAST);

/* Use the values that you retrieved from the database in your Web Service call */
    web_service_call( "StepName=EchoAddr_101",
        "SOAPMethod=SanityService|SanityServiceSoap|EchoAddr",
        "ResponseParam=response",
        "Service=SanityService",
        "ExpectedResponse=SoapResult",
        "Snapshot=t1227168459.inf",
        BEGIN_ARGUMENTS,
        "xml:addr="
            "<addr>"
                "<name>{nameParam}</name>"
                "<street></street>"
                "<city>{cityParam}</city>"
                "<state></state>"
                "<zip></zip>"
            "</addr>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);

return 0;
}
```

Validating Database Values after a Web Service Call

In this scenario, the test executes a Web Service call that modifies a database on the backend. The goal of this scenario is to validate that the resulting values in the database are correct.

The following table shows a typical flow of the script:

Step	API function
Connect to database	lr_db_connect (in **vuser_init** section)
Web Service call	web_service_call
Execute an SQL query	lr_db_executeSQLStatement
Retrieve and save the data	lr_db_getvalue to <param_name>
Check the data	lr_checkpoint
Disconnect from database	lr_db_disconnect (in **vuser_end** section)

For more information, see the Online Reference (**Help > Function Reference**).

The following example illustrates this process of checking the data:

```
Action()
{
/* A Web Service call that modifies a database on the back end. */
    web_service_call( "StepName=addAddr_102",
        "SOAPMethod=Axis2AddrBookService|Axis2AddrBookPort|addAddr",
        "ResponseParam=response",
        "Service=Axis2AddrBookService",
        "ExpectedResponse=SoapResult",
        "Snapshot=t1227169681.inf",
        BEGIN_ARGUMENTS,
        "xml:arg0="
            "<arg0>"
                "<name>{Customers}</name>"
                "<city>{City}</city>"
            "</arg0>",
        END_ARGUMENTS,
        LAST);

/* Query the database by the cusotmer name that was modified by the Web Service*/
    lr_db_executeSQLStatement("StepName=MyStep",
        "ConnectionName=MyConnection",
        "SQLQuery=SELECT * FROM Addresses WHERE name = '{Customers}' ",
        "DatasetName=ds1",
        LAST);

/* Get the values retrieved by the database query. */
    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=Name",
        "Row=current",
        "OutParam=CustomerName",
        LAST);

/* Compare the actual value with the expected value stored in the database. */
    lr_checkpoint("StepName=validateCustomer",
        "ActualValue={Customers}",
        "ExpectedValue={CustomerName}",
        "Compare=Equals",
        "StopOnValidationError=false",
        LAST);

return 0;
}
```

Checking Returned Values Through a Database

In this scenario, the user executes a Web Service call which returns an XML response. The goal of this scenario is to validate the response of the Web Service call against expected values. The expected values are stored in a database. The script fetches the expected results from a database and then compares them with the actual response.

The following table shows a typical flow of the script:

Step	API function
Connect to database	lr_db_connect (in **vuser_init** section)
Web Service call	web_service_call with **Result=<result_param>**
Execute an SQL query	lr_db_executeSQLStatement
Retrieve the expected data	lr_db_getvalue to <param_name>
Validate the data	soa_xml_validate with an XPATH checkpoints.
Disconnect from database	lr_db_disconnect (in **vuser_end** section)

You can use the XML validation tool to create a checkpoint for the response data. When creating the validation step, use the database parameter that you retrieved through **lr_db_getvalue**. For information on the XML Validation tool, see "Validating XML" on page 145.

The following example illustrates a typical validation of data returned by a Web Service call. The Validation step compares the actual expected results:

```
Action()
{
    web_service_call( "StepName=GetAddr_102",
        "SOAPMethod=AddrBook|AddrBookSoapPort|GetAddr",
        "ResponseParam=response",
        "Service=AddrBook",
        "ExpectedResponse=SoapResult",
        "Snapshot=t1227172583.inf",
        BEGIN_ARGUMENTS,
        "Name=abcde",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);

    lr_db_executeSQLStatement("StepName=MyStep",
        "ConnectionName=MyConnection",
        "SQLQuery=SELECT * FROM Addresses WHERE name = 'abcde' ",
        "DatasetName=ds1",
        LAST);

    lr_db_getvalue("StepName=MyStep",
        "DatasetName=ds1",
        "Column=Name",
        "Row=current",
        "OutParam=CustomerName",
        LAST);

    soa_xml_validate ("StepName=XmlValidation_1146894916",
        "Snapshot=t623713af7a594db2b5fef43da68ad59d.inf",
        "XML={GetAddrAllArgsParam}",
        "StopOnValidationError=0",
        BEGIN_CHECKPOINTS,
            CHECKPOINT,"XPATH=/*[local-name(.)='GetAddr'][1]/*[local-
name(.)='Result'][1]/*[local-name(.)='name'][1]","Value_Equals={CustomerName}",
        END_CHECKPOINTS,
        LAST);
    return 0;
}
```

For more information, see the Online Reference (**Help** > **Function Reference**).

Performing Actions on Datasets

Service Test lets you perform actions on datasets returned by SQL queries.

The **lr_db_dataset_action** function performs the following actions on datasets:

- **Reset.** Set the cursor to the first record of the dataset.
- **Remove.** Releases the memory allocated for the dataset.
- **Print.** Prints the contents of the entire dataset to the Replay Log and other test report summaries.

Note that when you retrieve binary data through **lr_db_getvalue**, you cannot print its contents using the **Print** action.

For information about the syntax and usage of this function, see the Online Reference (**Help** > **Function Reference**).

14

Web Services - Security

Advanced users can customize Web Service calls by setting the transport layer properties and security policies, and by writing user handlers to define the behavior of the Web Service calls.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Adding Security for Web Service Testing

When building Web Service applications, there is a challenge in building scalable applications that are secure. You can secure Web Services by having the message sent over a secure transport, such as Secure Sockets Layer (SSL), but this is limited to point-to-point communication.

To allow you to send your messages securely, Service Test supports several security mechanisms, Security Tokens (WS-Security), and SAML.

For more information on tokens, see below. For more information on SAML, see "Setting SAML Options" on page 316.

To learn more about customizing WS-Security and testing with WCF, see Chapter 15, "Web Services - Advanced Security and WS Specifications."

Note: If your WSDL is located in a secure location, you must provide the security information through the Manage Services dialog box. For more information, see "Specifying WSDL Connection Settings" on page 184.

Service Test supports two models for configuring security for your Web Service calls: **Legacy** and **Scenario**. This chapter describes the Legacy security model using **web_service_set_security**. For information on the Scenario model, see Chapter 15, "Web Services - Advanced Security and WS Specifications."

The following table lists the considerations for using each of the models.

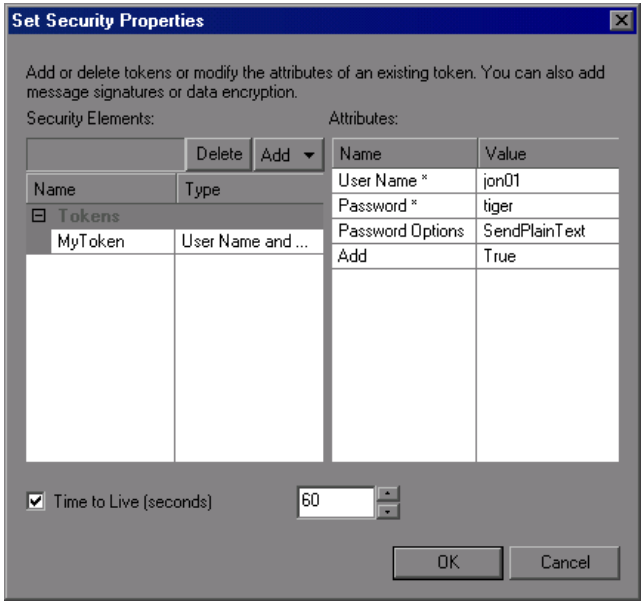
Legacy Model	Scenario Based Model
You are working with a script that already uses the legacy model	You are testing a WCF Service
You are testing a service written in frameworks such as .NET 2.0, Axis, or other older toolkits	You are testing a service written in a new framework, such as Axis2 or Metro (WSIT)
You require a low-level control over WS-Security tokens	Your service uses advanced specifications such as WS-SecureConversation or WS-Trust
You are having trouble using the new model or find the capabilities of the legacy more adequate for your needs	You are having trouble using the legacy model or you find the capabilities of the new model more adequate

Adding Security to a Web Service Call - a General Workflow

The following section describes the general workflow for adding security to a Web Service call:

To add Web Service security:

1. Place the cursor at the point at which you want to add the security settings. In most cases, we recommend that you place it in the vuser_init section so that the security scope will be applied to the whole script. If you only want the security for specific calls, place it at the desired location.
2. Select **Insert** > **New Step** to open the Add Step dialog box.
3. Select **Web Services Set Security** and click **OK**. The Set Security Properties box opens.



4 Click **Add** to add a new token. The Add Token dialog box opens.

5 Select a token type. Common tokens are Username and X.509 certificate. For information about the token types, see "Security Tokens and Encryption" on page 311.

In the **Logical Name** box, assign an arbitrary name for the token to be used by Service Test in identifying the token.

Add any relevant information, such as **User Name** and **Password** for the User Name and Password type token.

To send the token explicitly in the SOAP envelope header, select **True**. To exclude the token from the SOAP envelope header, select **False**.

6 To specify a time for which the message packet is considered valid, select **Time To Live** and specify the time in seconds.

7 Click **Add** to add a message signature and encryption if they are required. Both signatures and encryptions require you to specify a token previously defined as the encrypting/signing token. See the "Examples Using web_service_set_security" on page 319 to learn how to encrypt or sign a specific XPath in the SOAP.

8 To cancel the security settings at a specific point within the script, add a **Web Service Cancel Security** step at the desired point.

9 Click **OK**. Service Test inserts a Web Services Set Security step at the location of the cursor.

"Examples Using web_service_set_security" on page 319 illustrates some of the common security settings.

Security Tokens and Encryption

The WS-Security specification lets you place security credentials in the actual SOAP message. You accomplish this by instructing a client to obtain security credentials from a source that is trusted by both the sender and receiver. When a SOAP message sender sends a request, those security credentials, known as security **tokens**, are placed in the SOAP message. When the Web server receives the SOAP request, it does not need to send additional requests to verify the integrity of the sender. The server verifies that the credentials are authentic before letting the Web Service execute the application. By not having to go back to the source of the credentials, this significantly improves the application's scalability.

To further secure Web Services, it is common to use digital signatures or encryption for the SOAP messages. Digitally signing a SOAP message verifies that the message has not been altered during transmission. Encrypting a SOAP message helps secure a Web Service by making it difficult for anyone other than the intended recipient to read the contents of the message.

The Web Services security mechanism associates security tokens with messages. This mechanism supports several security token formats to accommodate a variety of authentication requirements. For example, a client might need to provide a proof of identity or a security certificate.

To support WS-Security, Service Test allows you to create security tokens for your script. You can create multiple tokens and set their properties. After creating a token, you use it to sign or encrypt a SOAP message.

In certain instances, you do not send the token explicitly—you use the token for the purpose of signatures or encryption, without including the actual token in the SOAP envelope header. Using the **Add** option, you can indicate whether to send the actual token explicitly.

The available tokens are **Username and Password**, **X.509 Certificate**, **Kerberos Ticket**, **Kerberos2 Ticket**, **Security Context Token**, and **Derived Token**. The information you need to provide differs for each token.

- **User Name and Password.** The **User Name and Password** token contains user identification information for the purpose of authentication: **User Name** and **Password**.

You can also specify Password Options, indicating how to send the password to the server for authentication: **SendPlainText**, **SendNone**, or **SendHashed**.

- **X.509 Certificate.** This security token is a token based on an X.509 certificate. To obtain a certificate, you can either purchase it from a certificate authority, such as VeriSign, Inc. or set up your own certificate service to issue a certificate. Most Windows servers support the public key infrastructure (PKI) which enable you to create certificates. You can then have it signed by a certificate authority or use an unsigned certificate.

When you add an X.509 token to the Vuser script, you specify the **Logical Name**, **Store Name**, **Key identifier type**, **Key identifier value**, and **Store Location** arguments.

- **Kerberos Ticket/Kerberos2 Ticket.** (for Windows 2003 or XP SP1 and later) The Kerberos protocol is used to mutually authenticate users and services on an open and unsecured network. Using shared secret keys, it encrypts and signs user credentials. A third party, known as a KDC (Kerberos Key Distribution Center), authenticates the credentials. After authentication, the user may request a service ticket to access one or more services on the network. The ticket includes the encrypted, authenticated identity of the user. The tickets are obtained using the current user's credentials.

Service Test supports tokens based on both Kerberos and Kerberos2 security tokens. The primary difference between the Kereberos and Kerberos2 tokens is that Kerberos2 uses the Security Support Provider Interface (SSPI), so it does not require elevated privileges to impersonate the client's identity. In addition, the Kerberos2 security token can be used to secure SOAP messages sent to a Web Service running in a Web farm.

When you add a Kerberos token to the Vuser script, you specify a **Logical Name** for the token along with the **Host** and **Domain** names of the Web Services machine.

- **Security Context Token.** These tokens are security tokens that can be used repeatedly until they expire. SOAP message senders can use security context tokens to sign and/or encrypt a series of SOAP messages, known as a conversation, between a SOAP message sender and the target Web Service. The main benefits of this type of token are:
  - As long as the security context token has not expired, the SOAP message sender can use the same security context token to sign and/or encrypt the SOAP messages sent to the target Web Service.
  - Security context tokens are based on a symmetric key, making them more efficient at digitally signing or encrypting a SOAP message than an asymmetric key.
  - Security context tokens can be requested from one security token service by sending a SOAP message to another security token service.

When you add a **Security Context** token to the Vuser script, you specify values for the **Logical Name**, **Base Token**, **Issuer Token**, **End Point URI**, and **Add applies to** arguments.

- **Derived Token.** The Derived token is a token based on another existing token, excluding X.509 for which derivation is not supported. You need to specify a **Logical Name** and the **Derived From** token. If you remove the original token, then the derived token will no longer be available. Note that you cannot use a Derived type of token in a recursive manner.

For more information about configuring tokens, see the *Online Function Reference* (**Help** > **Function Reference**).

Adding the Security Policy

To add a security policy to a section of your script, you enclose the relevant steps with **Web Service Set Security** and **Web Service Cancel Security** steps.

When you add a **Web Services Set Security** step to your script, Service Test adds a **web_service_set_security** function that contains arguments with the tokens, message signatures, and encryption that you defined in the security properties.

```
web_service_set_security(
   SECURITY_TOKEN, "Type=USERNAME", "TokenName=mytoekn1",
   "UserName=bob", "Password=123", "PasswordOptions=SendNone",    "Add=True",
LAST);
```

Parameterization is not supported for the following arguments: **Token Type, Logical Name, Base Token, Issuer Token** or **Derive From** arguments.

Working with Message Signatures and Encrypted Data

When you add a security token to a SOAP message, it is added to the SOAP message in the form of an XML element in the WS-Security SOAP header.

The message, however, is exposed and therefore requires additional security. This is especially true when the credentials, including the password, are sent in plain text as it is with role-based security.

The two methods used to secure the data are digital signatures and encryption.

- **Digital Signatures.** Digital Signatures are used by message recipients to verify that messages were not altered since their signing. The digital signature is usually in the form of XML within the SOAP message. The recipient checks the signature to make sure it is valid. Certain environments, such as WSE, automatically verify the signature on the SOAP recipient's computer.

➤ **Encryption.** Although the XML digital signature offers a mechanism for verifying that the message has not been altered since it was signed, it does not encrypt the SOAP message—the message is still plain text in XML format. To secure the message in order that it should not be exposed, you encrypt it, making it difficult for an intruder to view and obtain a user's password.

Service Test allows you to supply information about the encryption and message signatures.

Note that parameterization is not supported for message signatures and encryption arguments. For more information on adding message signatures and encryption to your script, see below.

Setting SAML Options

Service Test supports SAML (Security Assertion Markup Language) for Web Services. SAML is an XML standard for exchanging security-related information, called **assertions**, between business partners over the Internet. The assertions can include attribute statements, authentication, decision statements, and authorization decision statements.

SAML uses brokered authentication with a security token issued by STS (Security Token Service). The STS is trusted by the client and the Web Service to provide interoperable security tokens. SAML tokens are important for Web Service security because they provide cross-platform interoperability and a means of exchanging information between clients and services that do not reside within a single security domain.

You can set the SAML settings for an entire script or part of the script. To set SAML security, add a **Web Services Set Security SAML** step. To remove the security, insert a **Web Services Cancel Security SAML** step.

Note: You cannot apply SAML security and the standard Web Service (a **Web Service Set Security** step) security to the same step. To cancel Web Service security, insert a **Web Service Cancel Security** step.

Signing an SAML Assertion

VuGen provides a method for signing an unsigned SAML assertion. As input, you provide the unsigned assertion, a certificate file, and the optional password. VuGen provides a signed SAML assertion as output.

To add this method to your script, use the Add Step dialog box (**Insert** > **New Step**). VuGen adds a **ws_sign_saml_assertion** to the script. For syntax information, see the *Online Function Reference* (**Help** > **Function Reference**).

Policy Files

SAML policy files follow the WSE 3.0 standard and define the attribute values for the SAML security. By default, Service Test uses the **samlPolicy.config** file located in the installation's **dat** folder.

When entering SAML security information, you can enter it manually in the properties dialog box, or you can refer to a policy file containing all of the security information. You can create your own policy file based on samlPolicy.config.

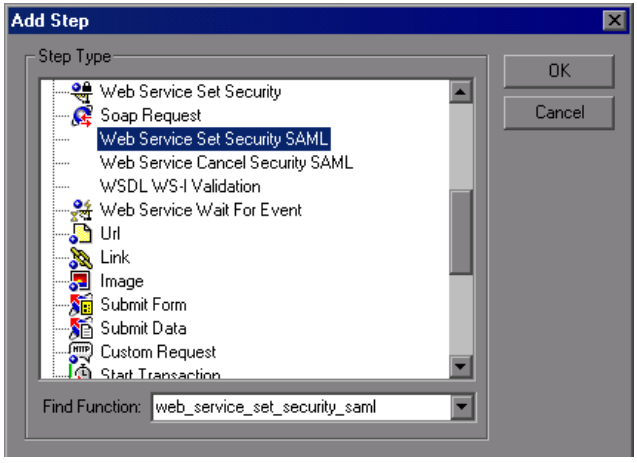
You can modify the policy file to include values for the security parameters, such as username and certificate information. When adding a SAML security step to your script, if you explicitly specify values for the security arguments, they override the values in the policy file.

If you make changes to the default policy file, we recommend that you copy the new policy file to your script's folder. Make sure to save custom policy files with a **.config** extension to insure that they remain with the script, even when running it on other machines or calling it from the LoadRunner Controller.

To learn more about the SAML policy files, see the SAML STS example on the MSDN Web site. If you want to emulate SAML Federation behavior, copy the **samlFederationPolicy.config** file from the data folder to your script's folder, and specify it as the policy file.

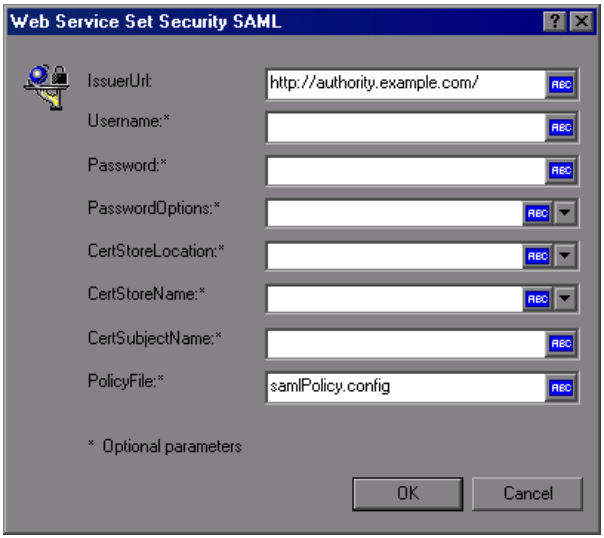
To add SAML security:

1. Click at the appropriate location in your script. To apply the security to the entire script, place the cursor at the beginning of the script.
2. Select **Insert** > **New Step** to open the Add Step dialog box.



3 To add SAML security, select **Web Services Set Security SAML**.

Enter the desired information. If you enter values into this dialog box, they override any values in the policy file. You must provide an Issuer URL, also known as the **STS URL**.



To use a different policy file, specify it in the **Policy File** box. Specify a full path, or a file location relative to the script's path.

4 To remove the security, select **Web Services Cancel Security SAML.** The security is cancelled from that point onward.

For additional information about these functions, see the *Online Function Reference* (**Help** > **Function Reference**, or click **F1** on the function).

Examples Using web_service_set_security

This section illustrates several common security scenarios.

Authenticating with a Username Token

The following example illustrates the sending of a message level username/password token (a username token), where the user name is John and the password is 1234.

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myToken",
"UserName=John", "Password=1234", "PasswordOptions=SendPlainText",
"Add=True",
        LAST);
```

Signing a Specific Element with an X.509 Certificate

It is possible to sign only a specific element in a message. The following example signs a specific element using an XPATH expression:

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=myCert", "StoreName=My",
"IDType=SubjectName", "IDValue=CN=myCert", "StoreLocation=CurrentUser",
"Add=True",
        MESSAGE_SIGNATURE, "UseToken=myCert", "TargetPath=//*[local-
name(.)='someElement' and namespace-uri(.)='http://myNamespace']",
        LAST);
```

Signing with an X.509 Certificate

The following example shows a script using an X.509 certificate for a digital signature.

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=myCert", "StoreName=My",
"IDType=SubjectName", "IDValue=CN=myCert", "StoreLocation=CurrentUser",
"Add=True",
        MESSAGE_SIGNATURE, "UseToken=myCert",
        LAST);
```

Note that your certificate needs to be installed in the Windows certificate store. In the example above, you need to set the actual store name, store location, and subject name of your certificate.

Encrypting with a Certificate

The following sample encrypts a message with the service's X.509 certificate.

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=serviceCert",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=serviceCert",
"StoreLocation=CurrentUser", "Add=False",
        ENCRYPTED_DATA, "UseToken=serviceCert",
        LAST);
```

After you specify the details of your X.509 certificate, you can encrypt a specific XPATH in the message.

Since we want to generate a Subject Key Identifier, we set the Add value to **False**. For more information, see "Using SubjectKeyIdentifier" on page 323.

Authenticating with a Username Token and Encrypting with an X.509 Certificate

The following example sends a username token to the service and encrypts it with the server's X.509 certificate:

```
web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=serviceCert",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=serviceCert",
"StoreLocation=CurrentUser", "Add=True",
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myUser",
"UserName=John", "Password=1234", "PasswordOptions=SendPlainText",
"Add=True",
        ENCRYPTED_DATA, "UseToken=serviceCert", "TargetToken=myUser",
        LAST);
```

The **UseToken** and **TargetToken** properties indicate which token to use and which to encrypt. Their values reference the **LogicalName** property of the tokens.

Encrypting and Signing a Message

This example shows how to sign a message using a private key and then encrypt it using the service's public key.

```
web_service_set_security(
        SECURITY_TOKEN, "Type=X509","LogicalName=myCert", "StoreName=My",
"IDType=SubjectName", "IDValue=CN=myCert", "StoreLocation=CurrentUser",
"Add=True",
        SECURITY_TOKEN, "Type=X509","LogicalName=serverToken",
"StoreName=My", "IDType=SubjectName", "IDValue=CN=serverCert",
"StoreLocation=CurrentUser", "Add=False",
        MESSAGE_SIGNATURE, "UseToken=myCert",
        ENCRYPTED_DATA, "UseToken=serverCert",
    LAST);
```

Referencing an X.509 Certificate Using a Hash

In certain cases, you may be unable to reference a certificate with a subject name. This example shows how to reference the certificate using its unique hash.

```
web_service_set_security(
   SECURITY_TOKEN, "Type=X509","LogicalName=serviceCert", "StoreName=My",
   "IDType=Base64KeyID", "IDValue=pOI0+1iuotKLlO91nhjDg5reEw0=",
   "StoreLocation=CurrentUser", "Add=False",
   ENCRYPTED_DATA, "UseToken=serviceCert",
 LAST);
```

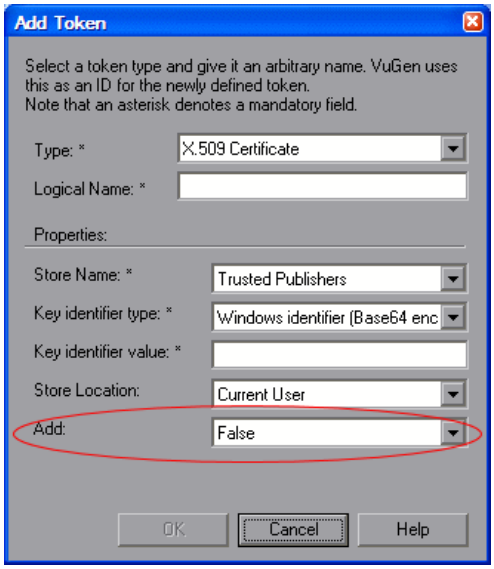
Customizing Your Security

The following sections describe how to configure special cases common to Web Service security.

Using SubjectKeyIdentifier

By default, Service Test adds all of the defined X.509 tokens to the SOAP envelope and references them as binary tokens. It is also possible to exclude the tokens from the message and reference them with a **SubjectKeyIdentifier**. This is common with tokens that are used for encryption.

In order to achieve this, when you add the token through the UI, choose **False** for the Add option.



Alternatively, you can configure this setting in the script:

```
SECURITY_TOKEN, "Type=X509","LogicalName=myToken", "StoreName=My",
"IDType=SubjectName", "IDValue=CN=myCert", "StoreLocation=CurrentUser",
"Add=False",
```

If you are using a SKI (Subject Key Identifier) you may also need to modify the **useRFC3280** settings as described in "Customizing WS-Security" on page 326.

Username Customization

When you add a new username token, the editor shows the **web_service_set_security** as follows:

```
    web_service_set_security(
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myToken",
"UserName=john", "Password=1234", "PasswordOptions=SendPlainText", "Add=True",
        LAST);
```

There are two additional settings that you can use for customization, that are not available in the user interface.

Name	Meaning	Possible values
IsNonceIncluded	Should the username token contain a nonce	True (default) or False
TimestampFormat	Should the username token contain a timestamp. If so, in what format	➤ **None**. no timestamp ➤ **Full**. a <timestamp> element with <created> and <expired> inner elements ➤ **Created**. (default) only a <created> element

Add these options to **web_service_set_security** in the following way:

```
web_service_set_security(
        SECURITY_TOKEN, "Type=USERNAME","LogicalName=myToken",
"UserName=John", "Password=1234",  "PasswordOptions=SendPlainText",
"IsNonceIncluded=true", "TimestampFormat=Full", "Add=True",
        LAST);
```

Customizing Encryption

You can customize encryption by indicating how to encrypt the element—encrypt the whole element or only its content. This is common when encrypting tokens such as a user name.

You can use the following setting to determine the exact encryption type:

Name	Meaning	Possible values
EncryptionType	What to encrypt	➤ Element ➤ Content (default)

Add this option to **web_service_set_security** in the following way:

```
web_service_set_security(
…
ENCRYPTED_DATA, "UseToken=myToken", "TargetToken=myOtherToken",
"EncryptionType=Element",
LAST);
```

Customizing WS-Security

It is sometimes necessary to change the algorithm Service Test uses for encryption or to modify some other low-level security details.

To change either of these items, open the **%Service Test%/bin/mmdrv.exe.config** file in a text editor. If this file does not contain the <**microsoft.web.services2**> element, add it as shown below.

```
<configuration>
…
  <microsoft.web.services2>
    <security>
    <x509 storeLocation="CurrentUser" allowTestRoot="true" useRFC3280="true" />
      <binarySecurityTokenManager valueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3">
        <sessionKeyAlgorithm name="TripleDES" />
        <keyAlgorithm name="RSA15" />
      </binarySecurityTokenManager>
      </security>
  </microsoft.web.services2>
…
<configuration>
```

Set the element values as required:

Name	Meaning	Possible values
verifyTrusy	Whether to check sent/received x.509 certificate's validity	➤ True (default) ➤ False
sessionKeyAlgorithm	The algorithm that the session symmetric key will use to encrypt the message	➤ AES128 ➤ AES192 ➤ AES256 ➤ TripleDES
keyAlgorithm	The algorithm used by the public key to encrypt the session key	➤ RSA15 ➤ RSAOAEP
useRFC3280	Whether to generate subject key identifiers that are interoperable and not windows specific	➤ True ➤ False (default)

15

Web Services - Advanced Security and WS Specifications

Service Test lets you customize your script to support additional WS standards. These include WS standards implemented in WCF and other WS - <*spec_name*> specifications.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Advanced Security and WS Specifications

Service Test allows you to test Web Services that utilize advanced security and WS-Specifications. Such services can be written in various platforms such as WCF (Windows Communication Foundation), Metro (WSIT), and Axis2. For WCF services, Service Test also supports proprietary standards and transports.

You enable this support by setting up a security scenario. Each scenario represents a typical environment used in conjunction with Web Service calls. Service Test provides several built-in security scenarios that are commonly used. It applies the scenario's settings individually to each service.

For the built-in scenarios, the user interface lets you provide identity information where required. You can customize security, transport, proxy, and other advanced settings.

If you cannot find a scenario that corresponds to your environment, you can use the generic custom scenario.

For a "How To" guide on selecting a scenario, see "Tips and Guidelines" on page 354.

Choosing a Security Model

Service Test supports two models for configuring security for your Web Service calls: **Legacy** and **Scenario**. This chapter describes the Scenario security model. The Legacy model refers to the manual addition of **Web Service Set Security** steps, or the **web_service_set_security** function. For information on the Legacy model, see Chapter 14, "Web Services - Security."

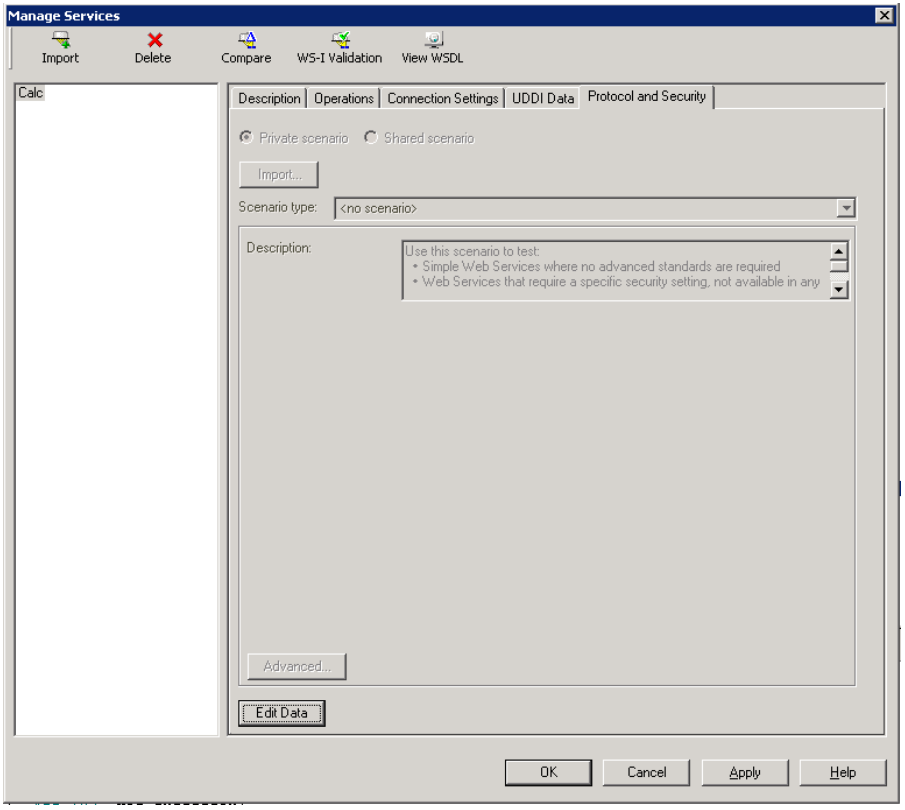
The following table lists the considerations for using each of the models.

Legacy Model	Scenario Based Model
You are working with a script that already uses the legacy model	You are testing a WCF Service
You are testing a service written in frameworks such as .NET 2.0, Axis, or other older toolkits	You are testing a service written in a new framework such as Axis2 or Metro (WSIT).
You require a low-level control over WS-Security tokens	Your service uses advanced specifications such as WS-SecureConversation or WS-Trust
You are having trouble using the new model or find the capabilities of the legacy functions adequate	You are having trouble using the legacy model or you find the capabilities of the new model more adequate

Setting the Security Scenario

You assign security scenarios on a service level per script—you assign a different security scenario for each service in your script.

The Manage Services window contains an interface to create and edit security scenarios for individual services. You access this interface from the **Protocols and Security** tab.



Using the Security Scenario editor, you can also create scenario files independent of the script. You can save them to a shared location for personal use or for collaboration with others.

Setting the Security Scenario for a Service

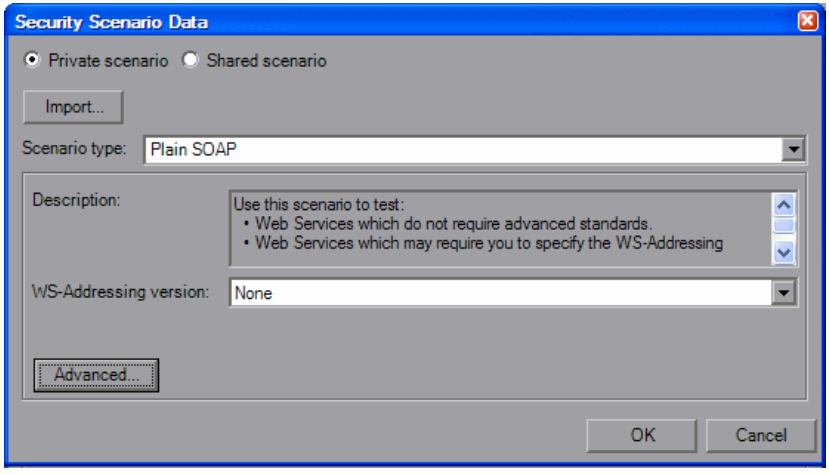
To assign a security scenario to a specific service, use the Manage Services window. The **Protocol and Security** tab contains the interface to create and view security scenarios for individual services.

You can select a scenario in three ways:

- **Private scenario.** Create a new scenario by selecting one of the built-in ones and customizing it for your Web Service.
- **Imported scenario.** Use a scenario created at an earlier time. The scenario will be editable, and if someone modifies the original scenario, it will not affect you.
- **Shared scenario.** Load a security scenario already configured by another user from a remote location or the file system. You cannot edit this scenario's settings from the Manage Services window. If someone edits the scenario, it will affect your environment. You usually use this option after working with the product for some time and saving the scenario files.

To create a security scenario for a specific service:

1. Click **Manage Services**. In the left pane, select the service for which you want to set the security scenario. If necessary, import a service, as described in "Importing Services" on page 179.
2. Select the **Protocol and Security** tab and click the **Edit Data** button. The Security Scenario Data dialog box opens.



3 Load a scenario. When you begin working with security scenarios, use the first method. After you have created several scenarios, you can use the other methods.

a To select a built-on security scenario for the current service, choose **Private scenario**.

In the Scenario type box, choose a scenario. The scenario types are described in "Scenario Types" on page 334.

Specify the required values for your scenario. For details, see "Specifying Scenario Information" on page 336.

b To use an existing scenario with the ability to modify it, choose **Private scenario.** Click **Import**. In the Shared Scenario dialog box, select a stored scenario. If required, modify the settings as described in "Specifying Scenario Information" on page 336.

c To use an existing scenario without the option of changing it, choose **Shared Scenario**. Use the Browse button to open the Shared Scenario dialog box and select a stored scenario.

Click **OK**. Your service is now loaded with a security scenario. If someone modifies the scenario file at its source, it will affect your script.

4 Click **Advanced** to configure the Proxy, Encoding, and other advanced setting (optional). For most scenarios, the default settings are ideal. For information about these settings, see "Advanced Settings" on page 343.

5 Click **OK** to close the dialog box and save your script.

Saving Security Scenarios for Collaboration

One user can customize a security scenario and make it available to others. Using the Security Scenario Editor, you customize your settings and save them to a scenario file.

To create a new security scenario:

1. Choose **SOA Tools** > **Security Scenario Editor**.
2. Select a **Scenario type** and enter the relevant information.
3. When enabled, click **Advanced** to configure the Proxy, Encoding, and other setting as described in "Advanced Settings" on page 343. Click **OK** to close the dialog box.
4. For a new scenario, click **Save as**, and specify a file name and path for the scenario file.

To edit an existing stored scenario:

1. Choose **SOA Tools** > **Security Scenario Editor**.
2. Click the **Open** button and browse for an existing scenario file.
3. Modify the scenario settings as required.
4. Click the **Save** or **Save as** button.
5. When enabled, click **Advanced** to configure the Proxy, Encoding, and other advanced settings. For more information, see "Advanced Settings" on page 343.
6. Click **OK** to close the Security Scenario editor.

Scenario Types

The scenario describes the configuration of your Web Service. It contains information such as encoding, identities, proxy, and so forth. Service Test provides a Security Scenario editor that allows you to configure the settings for each scenario.

To determine the scenario that best fits your service, refer to the table below. If you are unsure which scenario to choose, we recommend that you use the **Custom Binding** scenario which allows you to test any service. For more information, see "Custom Binding" on page 340.

The following table lists the built-in scenarios and describes when to use them.

Scenario Name	When to use
<no scenario>	➤ Default setting: if you have no need for a special security/protocols configuration, leave this value as is. Simple Web Services where no advanced standards are required. ➤ Scripts that use the legacy security model described in Chapter 14, "Web Services - Security." ➤ Web Services that require a specific security setting, not available in any of the existing scenarios ➤ If you select a built-in scenario and experience problems in replay, it is possible that no scenario is required and the problem is elsewhere. Reset the value to **<no scenario>.**
Plain SOAP	➤ Web services which do not require advanced standards ➤ Web services which may require you to specify the WS-Addressing version
MTOM	➤ MTOM enabled Web services ➤ Web Services which may require you to specify the WS-Addressing version

The following table shows the scenarios for Web Services that utilize WCF. The WSHttpBinding-based scenarios are divided according to the way the client authenticates itself to the server. For example, if your client presents a user name and a password to the server, choose the **Username (message protection)** scenario. The user interface lets you provide the identity information in the form of a user name or a certificate as required.

WCF Scenario Name	When to use
WSHttpBinding - No Authentication	➤ Client uses the server's X.509 certificate for encryption ➤ Client is not authenticated ➤ Communication may utilize advanced standards such as secure conversation and MTOM
WSHttpBinding - Windows authentication	➤ Client and server use Windows authentication ➤ Security is based on Kerberos or SPNEGO negotiations ➤ Communication may utilize advanced standards such as secure conversation and MTOM
wsHttpBInding - Certificate authentication	➤ Client uses the server's X.509 certificate for encryption ➤ Client uses its own X.509 certificate for signature ➤ Communication may utilize advanced standards such as secure conversation and MTOM
WSHttpBinding - username (message protection) authentication	➤ Client uses the server's X.509 certificate for encryption ➤ Client is authenticated with a username and password ➤ Communication may utilize advanced standards such as secure conversation and MTOM
WSHttpBinding - username (transport protection) authentication	➤ SSL is enabled ➤ Client is authenticated with a username and password ➤ Communication may utilize advanced standards such as secure conversation and MTOM

WCF Scenario Name	When to use
WSFederationHttpBinding	➤ Client authenticates against the STS using a predefined scenario ➤ Client uses the token given from the STS to authenticate against the server
Custom Binding	➤ Web Service that uses WS-* standards ➤ WCF services of any configuration

Specifying Scenario Information

This section describes the values required for each of the security scenarios.

<no scenario>

Since you are not using a secure scenario, you do not have to specify any values.

Plain SOAP

For this type of scenario, if your service uses WS-Addressing, specify the version.

MTOM

For MTOM type scenarios, if your service uses WS-Addressing, specify the version.

No Authentication

In this scenario, the client uses the server's certificate to encrypt a message; there is no client authentication.

Specify only one of the following settings:

➤ **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.

➤ **Specify service certificate.** Browse for a service certificate. For more information, see "Selecting a Certificate" on page 341. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information.

➤ **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Windows Authentication

This WCF scenario uses Windows Authentication.

You declare the expected identity of the server in terms of its **SPN** or **UPN** identities. If you are testing a WCF service that has not been customized and uses the default configuration, use this type of scenario.

Certificate Authentication

In this WCF WSHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message and its own certificate for a signature.

Specify only one of the following settings:

➤ **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.

➤ **Specify service certificate.** Browse for a service certificate. For more information, see "Selecting a Certificate" on page 341. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

➤ **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username Authentication (Message Protection)

In this WCF WSHttpBinding scenario, the client uses the server's X.509 certificate to encrypt the message, and sends a user name and password to authenticate itself.

Specify the following settings:

➤ **Username. Password.** The client's user name and password credentials.

Specify only one of the following settings:

➤ **Negotiate service credentials.** Negotiate the Web Service's certificate with the server.

➤ **Specify service certificate.** Browse for a service certificate. For more information, see "Selecting a Certificate" on page 341. If you select this option, the **Negotiate service credentials** option is not available.

Provide the DNS information:

➤ **Expected server DNS.** The expected identity of the server in terms of its DNS. This can be **localhost**, an IP address, or a server name. It can also be the common name by which the certificate was issued.

Username (Transport Protection) Authentication

This WCF WSHttpBinding scenario enables SSL and authenticates the client with a user name and password on the message level.

Specify the following settings:

➤ **Username. Password.** The client's user name and password credentials.

Federation

In the WSFederationHttpBinding scenario, the client authenticates against the STS (Security Token Service) to obtain a token. The client uses the token to authenticate against the application server.

Therefore, two bindings are needed, one against the STS and another against the application server.

First, use the Security Scenario editor to define an STS binding. For more information, see "Saving Security Scenarios for Collaboration" on page 333. When setting the binding against the application server, specify this file in the **Referenced file** box.

For the Federation scenario, specify the following server information:

- **Transport.** HTTP or HTTPS
- **Encoding.** Text or MTOM

For the Federation scenario, specify the following security information:

- **Authentication mode.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, IssuedTokenOverTransport, or SecureConversation
- **Bootstrap policy.** IssuedToken, IssuedTokenForCertificate, IssuedTokenForSslNegotiated, or IssuedTokenOverTransport

For the Federation scenario, specify the following identity information:

- **Server certificate.** Browse for a server certificate. For more information, see "Selecting a Certificate" on page 341.
- **Expected server DNS.** the expected identity of the server in terms of its DNS. This can be **localhost** or an IP address or server name.

For the Federation scenario, specify the following STS (Security Token Service) information:

- **Issuer address.** The address of the issuer of the STS. This can be **localhost,** an IP address, or a server name.

- **Referenced file.** The file that references the binding that contacts the STS (Security Token Service)

Custom Binding

The **Custom Binding** scenario enables the highest degree of customization. Since it is based upon WCF **customBinding**, it allows you to test most WCF services, along with services on other platforms such as Java that use WS - <*spec_name*> specifications.

Use the **Custom Binding** scenario to configure a custom scenario that does not comply with any of the predefined security scenarios.

For the Custom Binding scenario, specify the following server information:

- **Transport.** HTTP, HTTPS, TCP, or NamedPipe
- **Encoding.** Text, MTOM, or WCF Binary

Specify the following security information:

- **Authentication mode.** None, AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SecureConversation, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- **Bootstrap policy.** For SecureConversation type authentication, specify a bootstrap policy: AnonymousForCertificate, AnonymousForSslNegotiated, CertificateOverTransport, Kerberos, KerberosOverTransport, MutualCertificate, MutualSslNegotiated, SspiNegotiated, UserNameForCertificate, UserNameForSslNegotiated, UserNameOverTransport, or SspiNegotiatedOverTransport
- **Net security.** the network security. Select None, Windows stream security, or SSL stream security. For services with HTTP transport, leave the default value, **None**. To enable SSL for HTTP, choose the HTTPS transport.

If your Web Service uses Reliable messaging, enable the option, and select **Ordered** or **Not Ordered**.

Identities

Your security settings may require you to provide identity details for either the client and server, or both of them.

An example of identity details for the client, are user name/password or an **X.509** certificate.

For identity information, provide one or more authentication details as required by the service:

Username, **Password**, **Server certificate** or **Client certificate**. For information about choosing a certificate, see "Selecting a Certificate" on page 341.

Some scenarios require you to declare the expected identity of the server in terms of its DNS, SPN, or UPN identity.

- **DNS.** Provide the name of a server or use localhost.
- **SPN.** Provide the SPN identity in the domain\machine format.
- **UPN.** Provide the UPN identity in the user@domain format.

After setting the basic values, you can set advanced attributes as described in "Advanced Settings" on page 343.

Selecting a Certificate

This section describes how to select a certificate.

You can select a certificate from either a file or a Windows store.

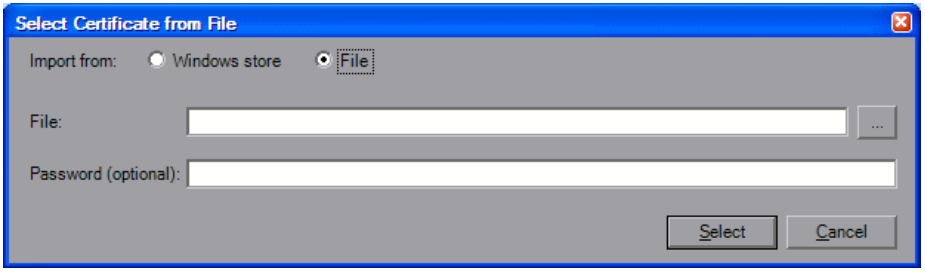
Certificates Stored in Files

In this case, the certificate is stored in a file.

To select a certificate from a file:

1 Click the Browse button adjacent to the Client Certificate or Server certificate box.

2 Choose **File**. Click the Browse button to the right of the File box and locate the certificate file.



3 If required, specify a password for the private key.

4 Click **Select**. The application now references the certificate file.

Certificates From a Windows Store

In this case, the location of the certificate is in a Windows store. The dialog box provides you with a mechanism to search for the certificate.

To select a certificate from a Windows Store:

1 Click the Browse button adjacent to the Client Certificate or Server Certificate box.

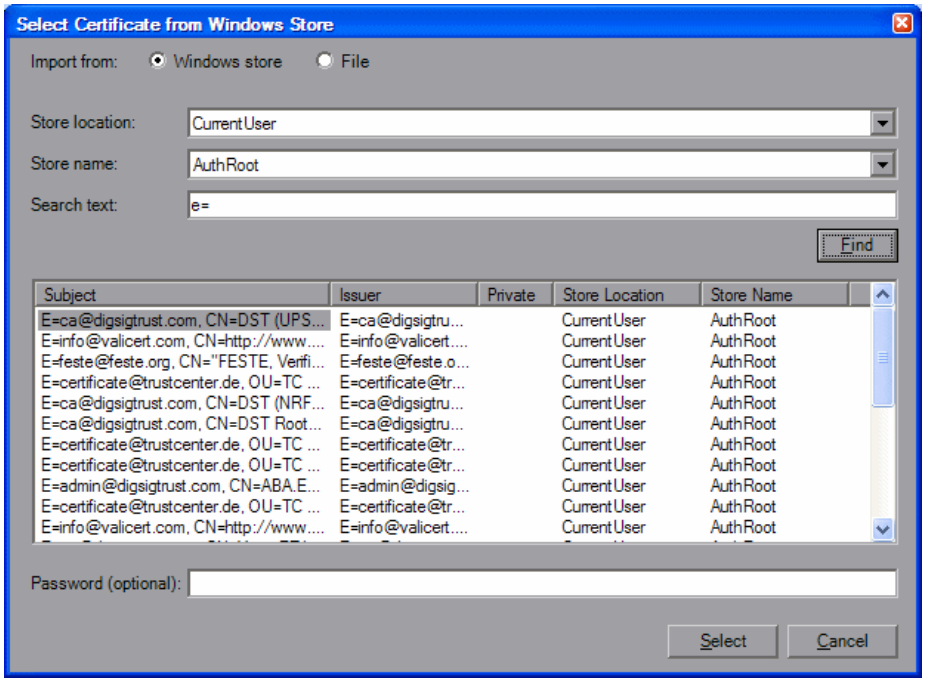
2 Choose **Windows Store**. Click the Browse button to the right of the File box and locate the certificate file.

3 Select a Store location: **All**, **CurrentUser**, or **LocalMachine** (optional).

4 Select a **Store name** from the drop-down list (optional).

5 To search for all certificates, leave the **Search text** box empty. To search for a specific certificate, specify a substring of the certificate name.

6 Click **Find** to generate the list of certificates found in the store.



7 If required, specify a password for the private key.

8 Select the certificate in the list and click **Select**. The application now references the chosen certificate.

Advanced Settings

This section describes the Advanced scenario settings. Using these settings, you can customize a security scenario in the following areas: Encoding, Advanced Standards, Security, or HTTP and Proxy.

Note that not all settings are relevant for all scenarios, so some of them might be disabled or hidden depending on the scenario.

Encoding

The Encoding tab lets you indicate the type of encoding to use for the messages: **Text**, **MTOM**, or **Binary**. The default is **Text** encoding.

For each of these encoding methods, you can choose a version of WS-Addressing:

- None
- WSA 1.0
- WSA 04/08

Advanced Standards

This tab lets you configure advanced WS- standards, such as Reliable Messaging.

If your service implements the **WS-ReliableMessaging** specification, enable the **Reliable Messaging** option and set the following options:

- **Reliable messaging ordered.** indicates whether the reliable session should be ordered
- **Reliable messaging version.** WSReliableMessagingFebruary2005 or WSReliableMessaging11

Security

The Advanced security settings correspond to the **WS-Security** specifications.

For security scenarios that are based upon WCF WSHttpBinding, you can indicate the following settings:

- **Enable secure session.** Establish a security context using the WS-SecureConversation standard.
- **Negotiate service credentials.** Allow WCF proprietary negotiations to negotiate the service's security.

For **WSHttpBinding, Custom Binding**, or **WSFederationHttpBinding** WCF type scenarios, you can set the default algorithm suite and protection level:

Attribute	Meaning	Possible Values
Default Algorithm Suite	the algorithm to use for symmetric/ asymmetric encryption. These are the values from the SecurityAlgorithmSuite configuration in WCF:	➤ Basic128 ➤ Basic128Rsa15 ➤ Basic128Sha256 ➤ Basic128Sha256Rsa15 ➤ Basic192 ➤ Basic192Rsa15 ➤ Basic192Sha256 ➤ Basic192Sha256Rsa15 ➤ Basic256 ➤ Basic256Rsa15 ➤ Basic256Sha256 ➤ Basic256Sha256Rsa15 ➤ TripleDes ➤ TripleDesRsa15 ➤ TripleDesSha256 ➤ TripleDesSha256Rsa15
Protection Level	Should the SOAP Body be encrypted/signed	None, Sign, and EncryptAndSign (default)

For **Custom Binding** or **WSFederationHttpBinding** WCF type scenarios, you can customize the security settings in greater detail. The following table describes the options and their values:

Attribute	Meaning	Possible Values
Message Protection Order	The order for signing and encrypting	➤ SignBeforeEncrypt ➤ SignBeforeEncrypt-AndEncryptSignature ➤ EncryptBeforeSign
Message Security Version	The WS-Security security version	A list of the current versions
Security Header Layout	The layout for the message header	➤ Strict ➤ Lax ➤ LaxTimeStampFirst ➤ LaxTimeStampLast
Key Entropy Mode	The entropy mode for the security key.	➤ Client Entropy ➤ Security Entropy ➤ Combined Entropy

You can enable or disable the following options:

- **Require derived keys.** Indicates whether or not to require derived keys.
- **Require security context cancellation.** Disabling this option implies that stateful security tokens will be used in the **WS-SecureConversation** session (if enabled).
- **Include timestamp.** Includes a timestamp in the header.
- **Allow serialized token on reply.** Enables the reply to send a serialized token.
- **Require signature confirmation.** Instructs the server to send a signature confirmation in the response.

For X.509 certificates, you can specify values for the following items:

Attribute	Meaning	Possible Values
X509 Inclusion Mode	When to include the X509 certificate	➤ Always to Recipient ➤ Never ➤ Once ➤ AlwaysToInitiator
X509 Reference Style	How to reference the certificate	➤ Internal ➤ External
X509 require derived keys	Should X509 certificates require derived keys	➤ Enable - Yes ➤ Disable - No
X509 key identifier clause type	The type of clause used to identify the X509 key.	➤ Any ➤ Thumbprint ➤ IssuerSerial ➤ SubjectKeyIdentifier ➤ RawDataKeyIdentifier

HTTP and Proxy

This tab lets you set the HTTP and Proxy information for your test.

HTTP (S) Transport

The following table describes the HTTP(S) Transport options:

Option	Meaning	Possible Values
Transfer Mode	The transfer method for requests/responses	Buffered, Streamed, StreamedRequest, StreamedResponse
Allow Cookies	Enable cookies	Enabled/Disabled
Keep-Alive Enabled	Enable keep-alive connections	Enabled/Disabled
Authentication Scheme	HTTP authentication method	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous
Realm	The realm of the authentication scheme	Any URL
Require Client Certificate	For SSL transport, require a certificate	Enabled/Disabled

Proxy Information

If the Web service's transport uses a proxy server, you can specify its details in the **Security** tab. The following table describes the proxy options:

Option	Meaning	Possible Values
Use Default Web Proxy	Use machine's default proxy settings	Enabled/Disabled
Bypass Proxy on Local	Ignore proxy when the service is on the local machine	Enabled/Disabled

Option	Meaning	Possible Values
Proxy Address	the proxy server	Any URL
Proxy Authentication Scheme	HTTP authentication method on Proxy	None, Digest, Negotiate, NTLM, IntegratedWindows Authentication, Basic, Anonymous

Customizing Your Security Model

The built-in scenarios with the scenario editor allow you to test most Web Services that use advanced standards.

However, you may still need to manually change a configuration file.

To modify a configuration file:

1. Select or set up a scenario as described in "Setting the Security Scenario for a Service" on page 331.
2. Open the script root directory. In Script view, click inside the script and choose **Open Script Directory** from the right-click menu.
3. Navigate to the inner folder **%Script Root%/WSDL/@config**. This folder contains one or more .stss files.
4. Open the relevant .stss file. If the directory only contains one .stss file, open it with a text editor. If it contains more then one, open the **configurationIndex.xml** to determine which .stss file matches the service you currently try to configure.
5. Modify the configuration file as described in the sections below. After you modify the file, do not update the configuration again from the Manage Services **Protocol and Security** tab, as this will override the changes.

The following are the capabilities that can be achieved by using the configuration files.

Increasing Response Buffer Capacity

In cases where you expect a large response, as is common when using MTOM, replay may issue the following error:

Error: The maximum message size quota for incoming messages (65536) has been exceeded. To increase the quota, use the MaxReceivedMessageSize property on the appropriate binding element.

To solve this, add the **maxReceivedMessageSize** attribute to the **httpTransport** element and configure it to use a larger size. For example:

```
<protocols scenario="customBinding xmlns="http://hp/ServiceTest/config">"
    <customization>
        <mtomMessageEncoding />
                    <httpTransport maxReceivedMessageSize="6000000" />
     </customization>
</protocols>
```

Customizing Windows Credentials

Some of the security scenarios use the Windows credentials. This is common with services utilizing WCF WsHttpBinding SPNEGO and Kerberos. By default, Service Test will use the currently logged in user as the client identity. You can instruct Service Test to use the identity of another user by modifying the appropriate configuration file. If any of the elements are already present in the configuration file, update them with the new information—do not duplicate the element.

```
<protocols …>
    <identities>
        <client>
            <windowsCredentials>
                <username>myUser</username>
                <password>myPassword</password>
                <domain>myDomain</domain>
            </windowsCredentials>
        </client>
    </identities>
    ...
</protocols>
```

Using a Logical Address (clientVia Behavior)

This section describes how to specify a logical address instead of a physical one emulating **clientVia** behavior. In this case, you send a message to an intermediate service that submits it to the actual server. This may also happen when you send the message to a debugging proxy.

In such cases it may be useful to separate the physical address to which the message is actually sent, from the logical address for which the message is intended. The logical address may be the physical address of the final server or any name. It appears in the SOAP message as follows:

```
<wsa:Action>http://myLogicalAddress<wsa:Action>
```

Configuring the Logical Address

The logical address is determined in the Service Test user interface. By default, it is the address specified in the WSDL. You can override this address from the Manage Services dialog box.

Configuring the Physical Address

To set the physical address, you need to modify the configuration file. Under the **protocols** element, modify the **behaviors** element. Make sure to change the logical address to the correct one.

```
<protocols scenario="customBinding" xmlns="http://hp/ServiceTest/config">
…
<behaviors>
    <endpointBehaviors>
        <behavior>
            <clientVia viaUri="http://MyLogicalAddress" />
        </behavior>
    </endpointBehaviors>
</behaviors>
</protocols>
```

Note: When the above behavior is not present in the configuration file (as in the default behavior), then the logical address will also be the physical one, and requests will be sent to it.

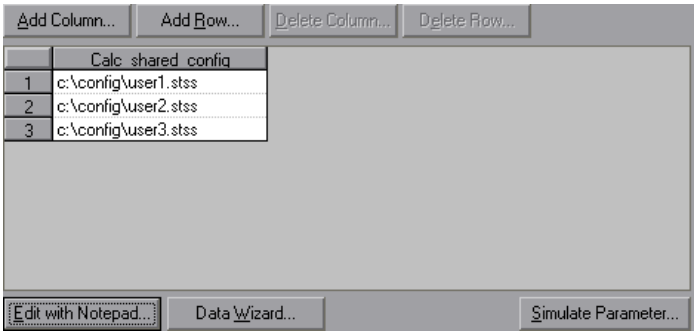
Parameterizing Security Elements

Scripts that utilize the new security model can be parameterized in the normal way. You can also parameterize the security elements independently. For example, in a username-based security scenario, you might want each Vuser or iteration to use a different user name.

To create parameters individually for each of the security elements:

1. Open the scenario editor. Select **SOA Tools** > **Security Scenario Editor**.
2. Set up and save a scenario for each Vuser. We recommend you use the names **user1**, **user2**, and so forth, and save them in a new folder, **%script root%/WSDL/referencedConfig**.
3. Open the Parameter List window. Select **Vuser** > **Parameters List**.
4. Create a new parameter, **<ServiceName>_shared_config**. Replace the <ServiceName> with the case-sensitive name of the service you are testing. To determine the exact name of the service, click **Manage Services** to see the list of services.

5 In the parameters dialog box, add the file names of the security scenarios with their .stss extensions as parameter values. You can use a relative path, which is relative to the script root folder. Click **Add Row** to add multiple values.



6 Click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.

7 Select **Shared Scenario**. Click the Browse button and enter the parameter name, **<ServiceName>_shared_config**, in the test box.

Simulating Users with Iterations

Many of the security scenarios establish a session with the server. For example, every scenario that uses **WS-SecureConversation** establishes a server session. This session is established when the first operation is executed and ends when the script is finished. By default, Service Test closes all sessions after each iteration and opens them again when the next iteration begins. This implies that every iteration simulates a new session and Vuser.

When working with multiple iterations, this may not be the desired effect—you may prefer to keep the original session active and not open a new session for each iteration. This applies when load testing through the LoadRunner Controller or when setting multiple iterations in the run-time settings.

You can override this behavior so that only the first iteration will establish a new session, while all subsequent ones will continue to use the open session. This simulates a user who repeatedly performs an action using the same session.

To determine which simulation mode to use, choose the one which is most appropriate to what you are simulating. For example, if you are simulating a load test where most of the actions are performed repeatedly by the same user in a single session, use the above configuration. If you are unsure, leave the default settings.

To configure your environment to use the same session for all iterations:

1. Open the script root directory. In Script view, click inside the script and choose **Open Script Directory** from the right-click menu.
2. Open **default.cfg** file in a text editor.
3. In the **[WebServices]** section, add in a row under the toolkit.

```
[WebServices]
Toolkit=.Net
SimulateNewUserInNewIteration=0
```

   If you are using the Axis toolkit or if you configured other settings, the file contents may differ.

4. Save and close the file.

Tips and Guidelines

This section provides a quick summary of using Service Test for WCF, general security, and advanced standards testing.

WCF

How do I test a WCF service?

Click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.

Expand the **WCF** node and choose the relevant scenario according to its binding. If you could not find an appropriate binding, choose the **customBinding** scenario since it can test any other binding.

How do I test a WCF service that uses WSHttpBinding?

WSHttpBinding is one of the most popular bindings in WCF. In order to use this binding, click **Manage Services** and select the **Protocol and Security** tab. Click **Edit Data**.

Expand the **WCF** > **By client authentication type** node and choose the client credential type that you use in your binding. This value corresponds to the **MessageClientCredentialType** property of the WCF's WSHttpBinding.

Windows authentication is the default value for a new WCF service. If you are using the WCF default settings for your service, use this option. Other options are username, certificate, or none. A username can be at the message level or at the transport level (equivalent to **TransportWithMessageCredential** in WCF).

For some scenarios you should indicate whether to use the WCF proprietary negotiation mechanism to get the service credentials.

Use the Advanced scenario properties to control the usage of a secure session.

How do I test a WCF service that uses CustomBinding?

Choose a scenario type of **WCF** > **Custom Binding** as described in "Scenario Types" on page 334. You can then customize many binding elements, such as your transport method, encoding, security, and reliable messaging.

How do I test a WCF service that uses netTcp or namedPipe transport?

Choose a scenario type of **WCF** > **Custom Binding** as described in "Scenario Types" on page 334. Configure the transport to **TCP** or **NamedPipe**.

How do I test a Federation scenario that uses an STS (Security Token Service)?

For this scenario, you must to define the communication properties for both the STS and the service. Additionally, you can test Federation scenarios that are compatible with Microsoft's WSE3 with the **web_service_set_security_saml** function. For more information, see the *Online Function Reference* (**Help** > **Function Reference**) or Chapter 14, "Web Services - Security."

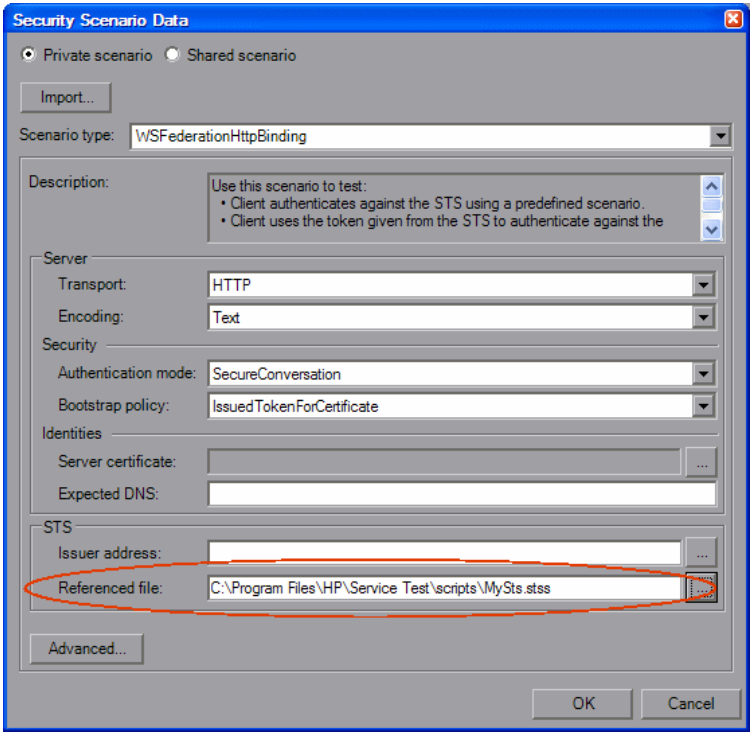
Choose the scenario **WCF** > **WSFederationHttpBinding**. For this scenario, you must to define the communication properties for both the STS and the application server.

To define the communication properties for the application server, use the **Protocol and Security** tab of the Manage Services dialog box.

To configure the communication with the STS:

1. Open the standalone security scenario editor. Select **SOA Tools** > **Security Scenario Editor**.
2. Click the **New** button. Configure the communication with the STS.
3. Click the **Save as** and specify a file name.

4 Open the Manage Services dialog box and select the **Protocol and Security** tab. Click **Edit Data**. In the **STS** section, reference the scenario file you created in the previous step.



General Security

How do I test a Web Service that uses SSL?

Testing a secure site does not require any special configuration. If your service URL begins with **https**, SSL is automatically used. If in addition to SSL you are using message-level security (for example a username) then you must configure the security for the message separately using the legacy or the scenario-based model. If you use the scenario-based model, you need to configure it to use SSL by choosing an HTTPS transport or a transport credentials mode in a WSHttpBinding scenario.

How do I test a Web Service that require Windows authentication at the HTTP level?

Use the **web_set_user** function. If additional standards are required, use the Legacy security based model in conjunction with or instead of the scenario-based model.

How to I test a Web Service that uses WS-Security?

Use the scenario-based as described in this section or the legacy security using **web_service_set_security**.

How do I configure the low-level details of my WS-Security tokens?

In most cases, you can configure the low-level details as described in "Advanced Settings" on page 343. In case a very low-level control over the WS-Security tokens is required use the legacy security model. For more information, see Chapter 14, "Web Services - Security."

How do I test a Federation scenario that uses an STS (Security Token Service)?

For this scenario, you must to define the communication properties for both the STS and the service. Additionally, you can test Federation scenarios that are compatible with Microsoft's WSE3 with the **web_service_set_security_saml** function. For more information, see the *Online Function Reference* (**Help > Function Reference**) or Chapter 14, "Web Services - Security."

Choose the scenario **WCF > WSFederationHttpBinding**. For this scenario, you must to define the communication properties for both the STS and the application server.

To define the communication properties for the application server, use the **Protocol and Security** tab of the Manage Services dialog box.

To configure the communication with the STS:

1. Open the standalone security scenario editor. Select **SOA Tools > Security Scenario Editor**.
2. Click the **New** button. Configure the communication with the STS.

3 Click the **Save as** and specify a file name.

4 Open the Manage Services dialog box and select the **Protocol and Security** tab. Click **Edit Data**. In the **STS** section, reference the scenario file you created in the previous step.

General Protocols

How do I indicate not to use any advanced configurations?

As a scenario type, choose **<no scenario>**.

If you selected a scenario and during replay you receive errors, it is possible that you do not need an advanced scenario. Choose **<no scenario>** to cancel the existing selection and rerun the script.

How do I test a Web Service that uses MTOM?

Choose the **MTOM** scenario. If additional security is required, use one of the other scenarios. In the Advanced dialog box, set the encoding to MTOM. For more information, see "Advanced Settings" on page 343.

How do I change the WS-Addressing version of a service?

By default, the .NET toolkit uses WS-Addressing 2004/03, while the Axis toolkit does not use any addressing. To override this behavior, choose the **Plain SOAP** scenario and select the WS-Addressing version. Other supported versions are 2004/08, 1.0, and None. If your service requires additional standards, such as security, use the appropriate scenario and configure the addressing version from the **Encoding** tab in the Advanced window. For more information, see "Advanced Settings" on page 343.

16

Web Services - Transport Layers and Customizations

You can customize Web Service calls by setting the transport layer properties and by writing user handlers to define the behavior of the Web Service calls.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Testing Web Service Transport Layers

Web services can be sent over various transport layers. The transport layer is the protocol used to transport messages to and from the server.

Service Test allows you to configure the transport layer for your services. It fully supports HTTP/HTTPS and JMS (Java Message Service) transport layers.

The Service Test solution allows you to emulate both synchronous and asynchronous messaging. If you are working with HTTP/HTTPS transport, you can also use WS-Addressing.

With user handlers, you can process SOAP requests and responses and assign them a custom behavior. For more information, see Chapter 17, "Web Services - User Handlers and Customization."

Configuring the Transport Layer

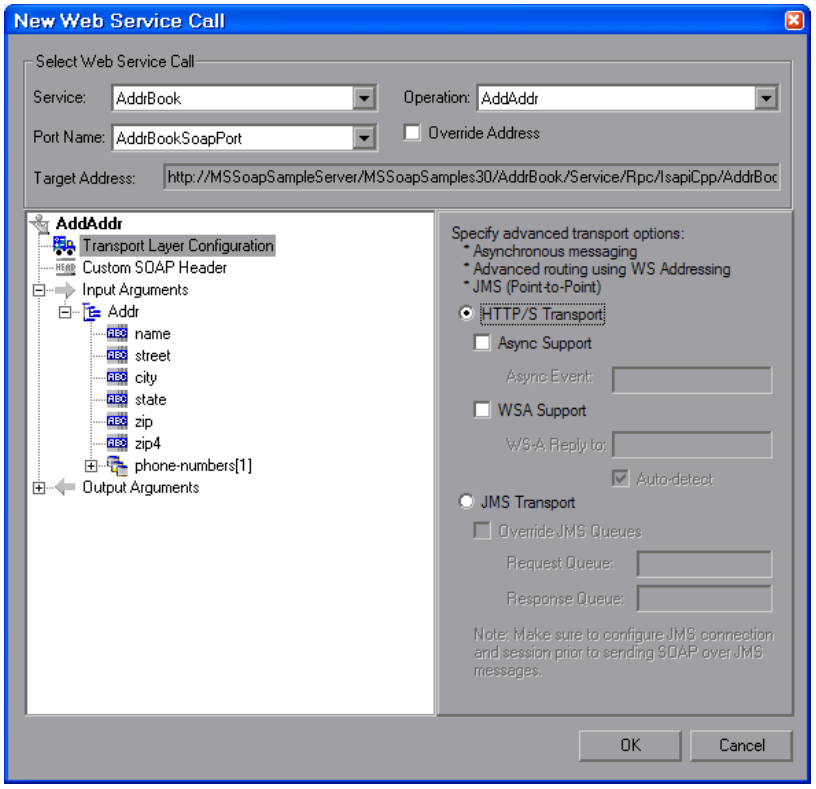
Service Test allows you to set the transport layer for your services. The transport layer indicates how to transport messages to and from the server—HTTP/HTTPS or JMS (Java Message Service).

For HTTP/HTTPS, see "Sending Messages over HTTP/HTTPS" on page 361. For more information about using the JMS transport, see "Understanding JMS" on page 362.

To choose a transport layer:

1 Create a new Web Service call. For an existing call, open Tree view, select the step, and select the **Step Properties** tab.

2 Select the **Transport Layer Configuration** node.



3 Choose the desired transport mode: **HTTP/S Transport** or **JMS Transport**.

For more options, see "Sending Messages over HTTP/HTTPS" below or "Using JMS as a Transport Layer" on page 363.

Sending Messages over HTTP/HTTPS

HTTP is used for sending requests from a Web client, usually a browser, to a Web server. HTTP is also used to return the Web content from the server back to the client.

HTTPS handles secure communication between a client and server. Typically, it handles credit card transactions and other sensitive data.

The typical request and response mechanism is synchronous. In synchronous messaging, the replay engine blocks script execution until the server sends its response. In asynchronous mode, the replay engine executes the script without waiting for server's response for previous messages.

If you are working with HTTP or HTTPS transport, you can use asynchronous calls in conjunction with WS-Addressing. For more information, see "Sending Asynchronous Calls with HTTP/HTTPS" on page 367.

Service Test also lets you emulate asynchronous messaging for your Web Service call. For more information, see "Sending Asynchronous Calls with JMS" on page 374.

Understanding JMS

JMS is a J2EE standard for sending messages, either text or Java objects, between Java clients.

There are two scenarios for communication:

Peer-to-Peer. Also known as **Point-to-Point**. JMS implements point-to-point messaging by defining a message queue as the target for a message. Multiple senders send messages to a message queue, and the receiver gets the message from the queue.

Publish-Subscribe. Each message is sent from one publisher to many subscribers through a designated topic. The subscribers only receive messages sent after they have subscribed.

Service Test supports point-to-point communication by allowing you to send and receive JMS messages to and from a queue.

Before you can send messages over JMS transport, you need to configure several items that describe the transport:

- **JNDI initial context factory.** The class name of the factory class that creates an initial context which will be used to locate the JMS resources such as JMS connection factory or JMS queue.
- **JNDI provider.** The URL of the service provider which will be used to locate the JMS resources such as JMS connection factory or JMS queue.
- **JMS connection factory.** The JNDI name of the JMS connection factory.

In addition, you can set a timeout for received messages and the number of JMS connection per process.

You can configure all of these settings through the JMS run-time settings, as described in "Web Services JMS Run-Time Settings" on page 286.

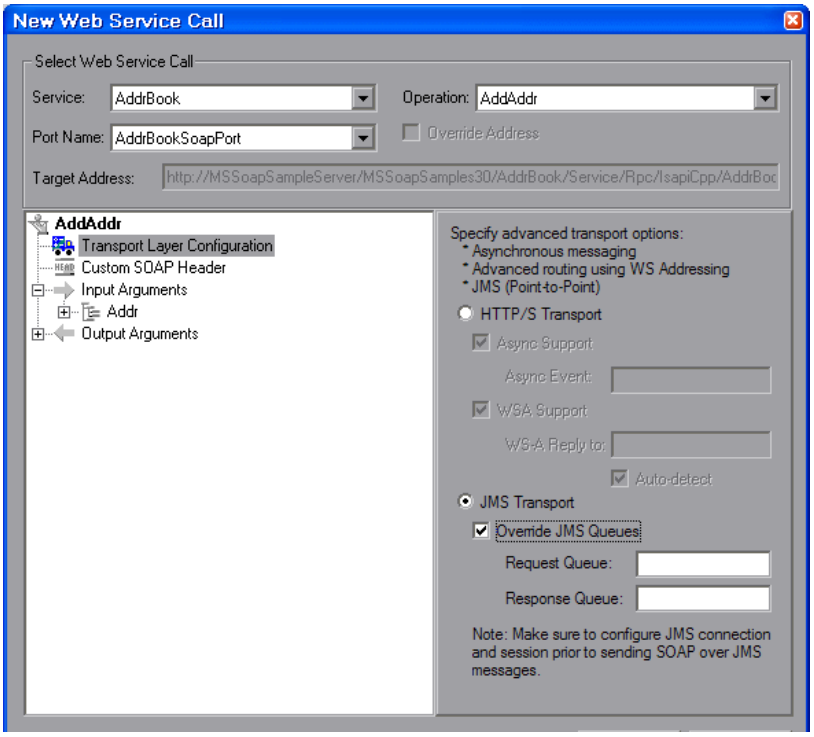
Using JMS as a Transport Layer

The JMS transport for Web services allows you to send SOAP messages using a JMS transport instead of the common HTTP transport.

To send messages using the JMS transport layer:

1 Create a new Web Service call. For an existing call, open Tree view, select the step, and select the **Step Properties** tab.

2 Select the **Transport Layer Configuration** node and select **JMS Transport**.



3 Configure the JMS run-time settings before running the script, as described in "Web Services JMS Run-Time Settings" on page 286.

The procedure above describes how to send synchronous JMS messages. For information on emulating asynchronous messages over JMS, see "Sending Asynchronous Calls with JMS" on page 374.

You can also send messages over JMS, even without creating Web Service calls. For example, using Service Test, you can:

- record SOAP messages using a standard Web protocol, and send them to a queue through JMS transport functions.
- manually send and receive general JMS messages from designated queues.

For more information, see "JMS Script Functions" below.

JMS Script Functions

Service Test uses its API functions to implement the JMS transport. Each function begins with a **jms** prefix:

Function Name	Description
jms_receive_message_queue	Receives a message from a queue
jms_send_message_queue	Sends a message to a queue.
jms_send_receive_message_queue	Sends a message to a specified queue and receives a message from a specified queue.
jms_set_general_property	Sets a general property in the user context.
jms_set_message_property	Sets a JMS header or property for the next message to be sent.

The JMS steps/functions are only available when manually creating scripts—you cannot record JMS messages sent between the client and server.

For additional information about the JMS functions, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

JMS Message Types

Each JMS message is composed of:

- **Header.** contains standard attributes (Correlation ID, Priority, Expiration date).
- **Properties.** custom attributes.
- **Body.** text or binary information.

JMS can be sent with several message body formats. Two common formats are **TextMessage** and **BytesMessage**.

Service Test attempts to resolve the desired format based on the message's content type. If the content type is **text/***, it sends the message in TextMessage format. Otherwise, it sends it in **BytesMessage** format.

To override the default behavior, use a **jms_set_general_property** function before sending the message. Set the JMS_MESSAGE_TYPE property to TextMessage, BytesMessage, or Default. For Example:

```
jms_set_general_property("step1","JMS_MESSAGE_TYPE","BytesMessage");
```

For more information, see the *Online Function Reference*.

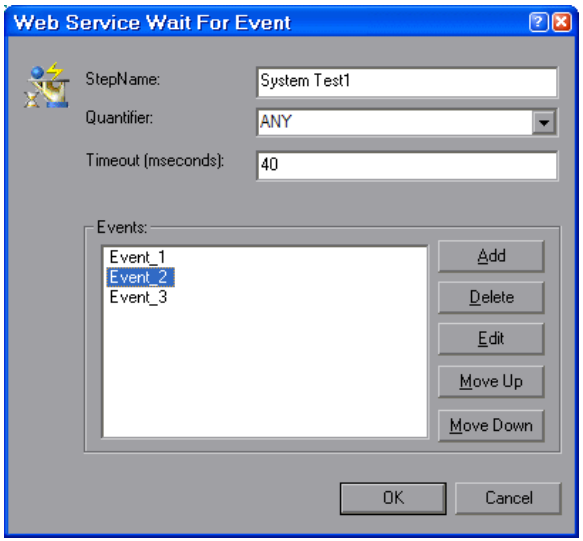
Sending Asynchronous Messages

You can use Service Test to emulate both synchronous and asynchronous messaging.

In synchronous messaging, the replay engine blocks script execution until the server sends its response. In asynchronous mode, the replay engine executes the script without waiting for server's response for previous messages.

Sending Asynchronous Calls with HTTP/HTTPS

This following section describes how to use asynchronous calls in HTTP/HTTPS. You use a **Wait for Event** step to instruct Vusers to wait for the response of previous asynchronous requests before continuing. The listener blocks the execution of the service until the server responds.



When adding a Web Service Wait for Event step:

- **Quantifier.** The quantifier indicates whether the Vuser should wait for **ALL** events to receive a response or **ANY,** just one of them. **ANY** returns the name of the first event to receive a response. **ALL** returns one of the event names.
- **Timeout.** the timeout in milliseconds. If no events receive responses in the specified timeout, then **web_service_wait_for_event** returns a NULL.
- **Events.** a list all of the asynchronous events for which you want to wait.

When running a script with asynchronous messaging, the Replay log provides information about the events and the input and output arguments.

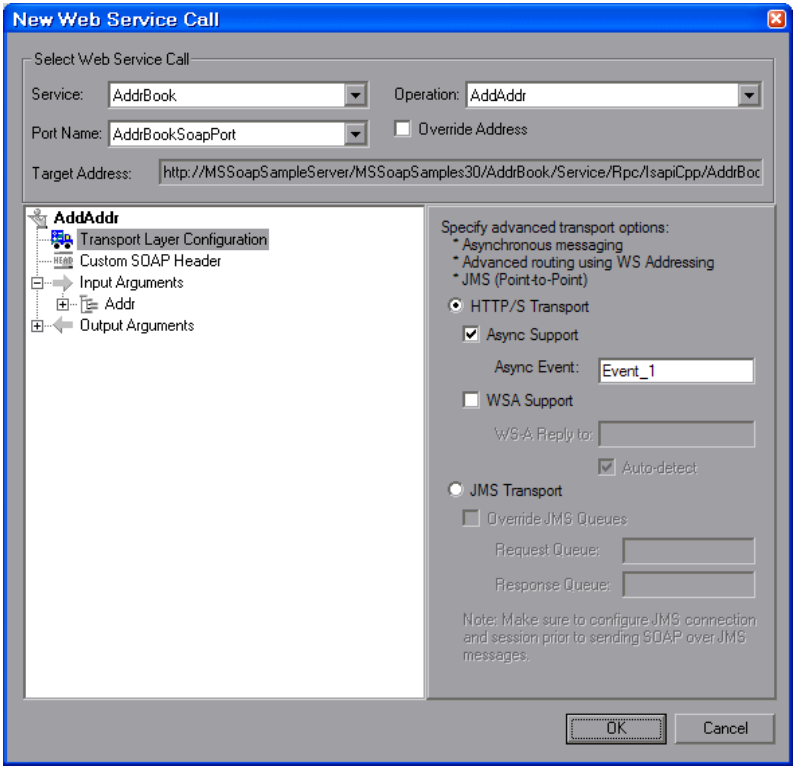
For additional information about the **Web Service Wait for Event** step or **web_service_wait_for_event** function, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

When setting up an asynchronous message, you can set the location to which the service responds when it detects an event using WS-Addressing. For more information, see "WS-Addressing" on page 370.

To send an asynchronous message using HTTP/S as a transport protocol:

1 In Tree view, select the **Step Properties** tab and select the **Transport Layer Configuration** node.

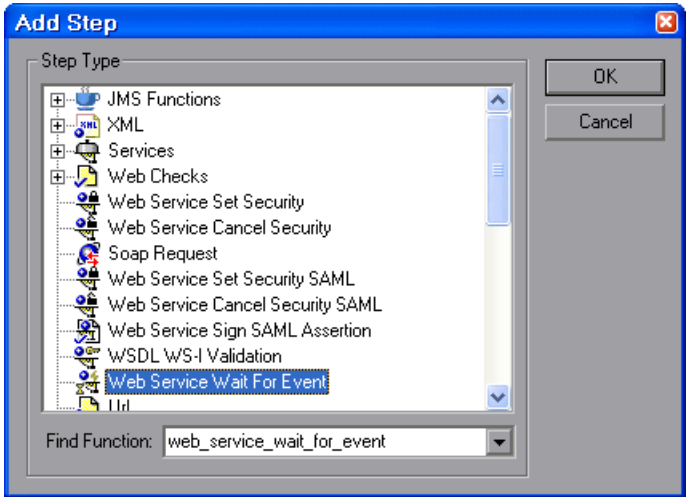
2 Choose **HTTP/S Transport** and select the **Async Support** option.



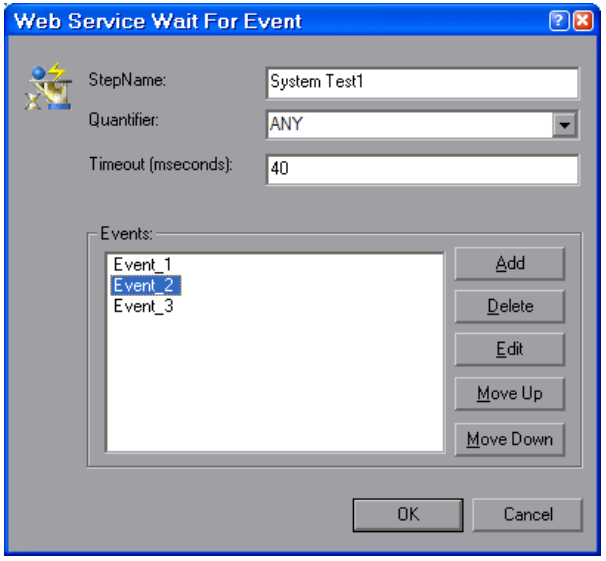
3 Type an event name in the **Async Event** box.

4 Click **OK** to generate the Web Service call.

5 Add a **Wait for Event** step. Select **Insert** > **New Step** and choose **Web Service Wait for Event**.



6 Specify a step name, a quantifier, and a timeout. Click **Add** and insert the name of the event that you defined in the previous step.



In Script view, Service Test indicates asynchronous messaging with the added parameter, **AsyncEvent**.

```
web_service_call("StepName=EchoString_101",
        "SOAPMethod=EchoRpcEncoded.EchoSoap.EchoString",
        "ResponseParam=response1",
        "Service=ExtendedECHO_rpc_encoded",
        "AsyncEvent=Event_1",
        "Snapshot=t1157371707.inf",
        BEGIN_ARGUMENTS,
        "sec=7",
        "strString=mytext",
        END_ARGUMENTS,
        BEGIN_RESULT,
        "EchoStringResult=first_call",
        END_RESULT,
        LAST);
```

The **AsyncEvent** flag instructs the Vuser to wait for the response of previous asynchronous service requests.

For synchronous messages, create a standard Web Service call, and do not enable the **Async Support** option.

WS-Addressing

WS-Addressing is a specification that allows Web Services to communicate addressing information. It does this by identifying Web service endpoints in order to secure end-to-end endpoint identification in messages. This allows you to transmit messages through networks that have additional processing nodes such as endpoint managers, firewalls, and gateways. WS-Addressing supports Web Services messages traveling over both synchronous or asynchronous transports.

The WS-Addressing specification requires a **WSAReplyTo** address—the location to which you want the service to reply.

An optional **WSAAction** argument allows you to define a SOAP action for instances where transport layers fails to send a message.

The following example illustrates a typical SOAP message using WS-Addressing, implemented in the background by Service Test.

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://www.w3.org/2004/12/addressing">
   <S:Header>
      <wsa:MessageID>
         http://example.com/SomeUniqueMessageIdString
      </wsa:MessageID>
      <wsa:ReplyTo>
         <wsa:Address>http://myClient.example/someClientUser</wsa:Address>
      </wsa:ReplyTo>
<wsa:Address>http://myserver.example/DemoErrorHandler</wsa:Address>
      </wsa:FaultTo>
      <wsa:To>http://myserver.example/DemoServiceURI</wsa:To>
      <wsa:Action>http://myserver.example/DoAction</wsa:Action>
   </S:Header>
   <S:Body>
      <!-- Body of SOAP request message -->
   </S:Body>
</S:Envelope>
```

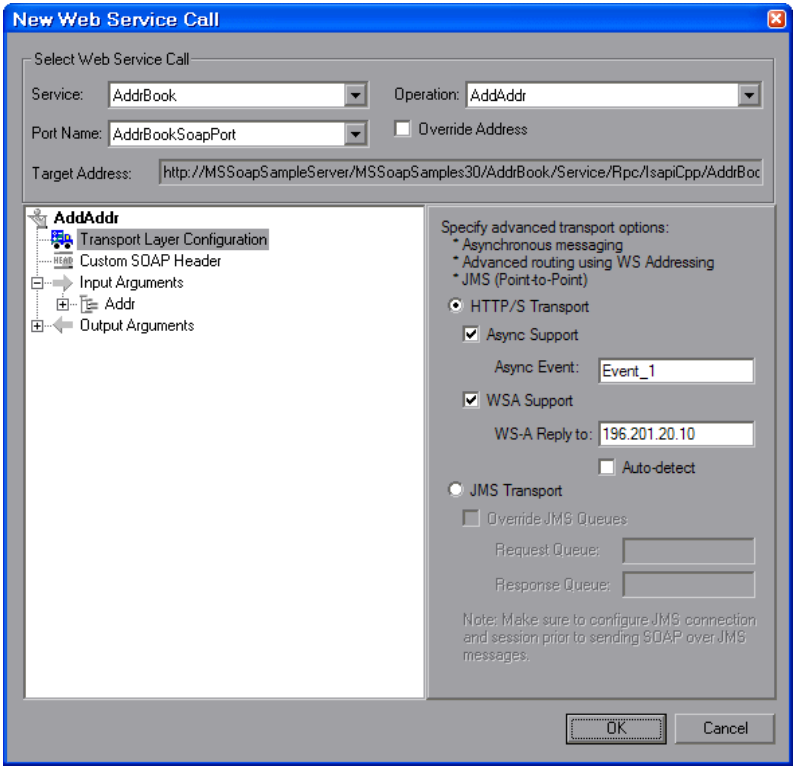
In the following example, the server responds to the interface 212.199.95.138 when it detects Event_1.

```
web_service_call("StepName=Add_101",
         "SOAPMethod=Calc.CalcSoap.Add",
         "ResponseParam=response",
         "AsyncEvent=Event_1",
            "WSAReplyTo=212.199.95.138",
         "WSDL=http://lab1/WebServices/CalcWS/Calc.asmx?wsdl",
         "UseWSDLCopy=1",
         "Snapshot=t1153825715.inf",
         BEGIN_ARGUMENTS,
            "first=1",
            "second=2",
         END_ARGUMENTS,
         BEGIN_RESULT,
            "AddResult=Param_AddResult1",
         END_RESULT,
         LAST);
```

You can issue WS-Addressing calls in both asynchronous and synchronous modes. To use WS-Addressing in synchronous mode, leave the **Async Event** box empty in the Transport Layer options. In Script view, remove the **AsyncEvent** argument. This instructs the replay engine to block script execution until the complete response is received from the server.

To add support for asynchronous messages and WS-Addressing:

1 For a new Web Service call, select the **Transport Layer Configuration** node. For an existing Web Service call, select the step in Tree view, and select the **Step Properties** tab. Select the **Transport Layer Configuration** node.



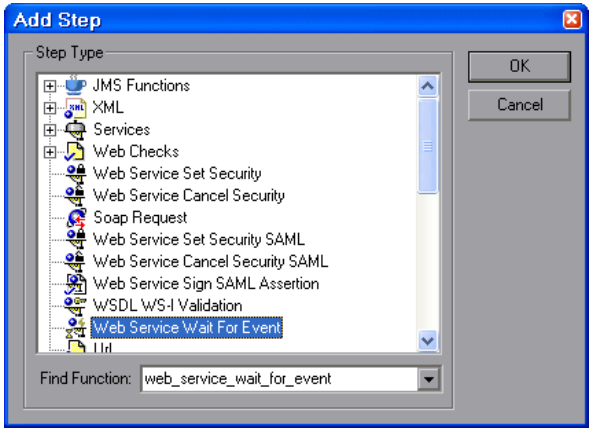
2 To mark the Web Service call as an asynchronous message:

- Select the **Async Support** option. This is only enabled for HTTP/S type transport.
- Provide an event name in the **Async Event** box. This can be an arbitrary name.

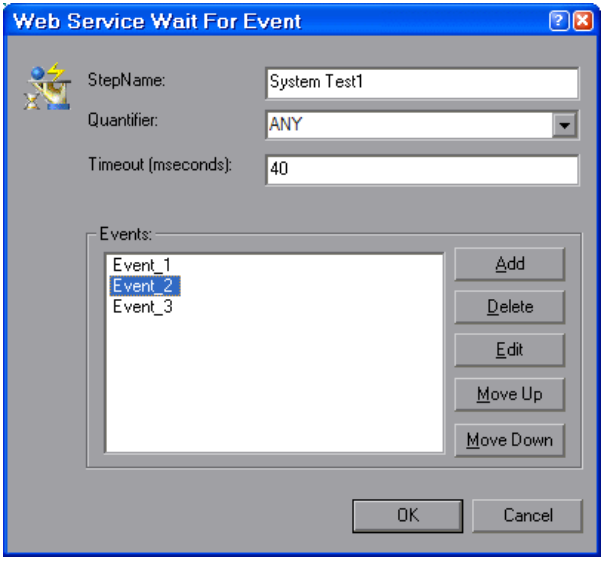
3 Select **WSA Support**. In the **WS-A Reply to** box, enter an IP address or **autodetect** to use the current host. Autodetect is useful when running the same script on several different machines. The server will reply to the specified location when the event occurs.

4 Click **OK** to save the settings.

5 Instruct the Vuser to wait for an event. Select **Insert** > **New Step** and add a **Web Service Wait For Event** step after the Web Service call step.



6 Specify a step name, quantifier, and timeout. To add an event name, click **Add**. The Web Service will wait for the specified event before responding.



7 Use the **Edit**, **Move Up**, and **Move Down** buttons to manipulate the events.

8 Click **OK**.

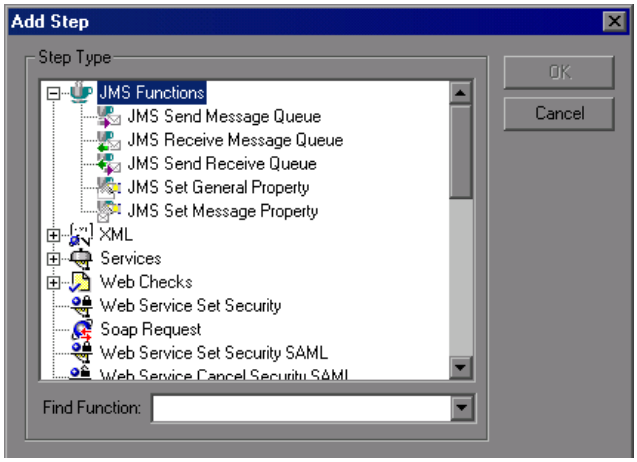
Sending Asynchronous Calls with JMS

This section describes how to send asynchronous messages using JMS.

To implement this, you send the request or retrieve the response using JMS steps—not Web Service calls. **JMS Send Message Queue** sends a message to a queue. **JMS Receive Message Queue** receives a message from the queue.

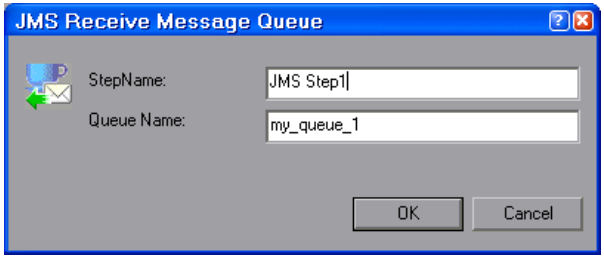
To send and receive asynchronous messages using JMS:

1 Click within the script at the desired location. Select **Insert** > **New Step** and expand the **JMS Functions** node.



2 Select a JMS function to set up the message queue information: **Send Message Queue** or **JMS Receive Message Queue**.

3 Click **OK** to open the properties the JMS functions.



4 Specify a queue name and click **OK** to generate the JMS functions.

For JMS synchronous messages, create a standard Web Service call, select JMS transport, and if desired, specify the queue information.

For additional information about these functions, see the *Online Function Reference* (**Help** > **Function Reference** or click **F1** on the function).

17

Web Services - User Handlers and Customization

Service Test allows you to customize your script with user handlers and configuration files.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Customizing Web Service Script Behavior

Service Test provides several advanced capabilities that allow you to customize the way your script behaves. These capabilities are user handlers and configuration files.

With user handlers, you can process SOAP requests and responses and assign them a custom behavior. For more information, see below.

Configuration files let you customize advanced settings such as security information and the WSE configuration. For more information, see "Using Custom Configuration Files" on page 387.

Setting up User Handlers

User Handlers are open API through which you can perform the following operations:

- ➤ Get and set the request/response SOAP envelopes
- ➤ Override the transport layer
- ➤ Get and set the request/response content type
- ➤ Get and Set values for LoadRunner parameters
- ➤ Retrieve a configuration argument from the script
- ➤ Issue messages to the execution log
- ➤ Fail an execution

You can set up a user handler directly in a script, or you can implement through a DLL. You can apply a handler locally or globally. For details, see the following sections:

- ❑ Defining a Handler Function in a Script
- ❑ Creating a Custom User Handler as a DLL
- ❑ Configuring the User Handler
- ❑ Implementing the User Handler

For sample user handlers, see "User Handler Examples" on page 383.

Defining a Handler Function in a Script

For basic implementation of a user handler, you define a user handler function within your Vuser script:

```
int MyScriptFunction(const char* pArgs, int isRequest)
{
…
}
```

The **pArgs** argument contains the string that is specified in **UserHandlerArgs** argument of **web_service_call** function. For more information, see the *Online Function Reference* (**Help** > **Function Reference**).

The **isRequest** argument indicates whether the function is being called during processing of a Request (1) or Response (0) SOAP envelope.

The content of SOAP envelope is passed to a parameter called **SoapEnvelopeParam** for both requests and responses. After the function processes the SOAP envelope, make sure to store it in the same parameter

To call the handler function, specify the function name as a value for the **UserHandlerFunction** argument in the relevant Web Service Call step.

```
web_service_call(
…
"UserHandlerFunction=MyScriptFunction",
"UserHandlerArgs=<handler arguments>",
  LAST);
```

Service Test recognizes the following return codes for the handler function.

Return Code		Description
LR_HANDLER_SUCCEEDED	0	The Handler succeeded, but the SOAP envelope did not change.
LR_HANDLER_FAILED	1	The Handler failed and further processing should be stopped.
LR_HANDLER_SUCCEEDED_AND_MODIFIED	2	The Handler succeeded and the updated SOAP envelope is stored in **SoapEnvelopeParam**.

In the following example, a script handler manipulates the outgoing envelope:

```
//This function processes the SOAP envelope before sending it to the server.
int MyScriptFunction(const char* pArgs, int isRequest)
{
    if (isRequest == 1) {

        //Get the request that is going to be sent
        char* str = lr_eval_string("{SoapEnvelopeParam}");

        //Manipulate the string...

        //Assign the new request content
        lr_save_string(str, "SoapEnvelopeParam");

        return LR_HANDLER_SUCCEEDED_AND_MODIFIED;
    }
    return LR_HANDLER_SUCCEEDED;
}
Action()
{
    //Instruct the web_service_call to use the handler
    web_service_call( "StepName=EchoAddr_102",
        "SOAPMethod=SpecialCases.SpecialCasesSoap.EchoAddr",
        "ResponseParam=response",
        "userHandlerFunction=MyScriptFunction",
        "Service=SpecialCases",
        "Snapshot=t1174304648.inf",
        BEGIN_ARGUMENTS,
        "xml:addr="
            "<addr>"
                "<name>abcde</name>"
                "<street>abcde</street>"
                "<city>abcde</city>"
                "<state>abcde</state>"
                "<zip>abcde</zip>"
            "</addr>",
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);

    return 0;
```

Creating a Custom User Handler as a DLL

You can also define a user handler by creating a DLL file through Visual Studio and the handler API. The API header file, **LrWsHandlerAPI.h**, located in the **ServiceTest/include** folder, contains many in-line comments and descriptions.

Service Test provides a sample Visual Studio project that can be used as a template for creating a handler. The sample retrieves the request and response envelope and saves it to a parameter. This sample is located in the **ServiceTest/samples/WebServices/SampleWsHandler** folder. To use this sample, open it in Visual Studio and modify it as required. If you do not need to save the request/response to a parameter, you can remove that section of the sample.

After editing the sample, save it and compile the DLL. When you compile the project, Visual Studio places the **<user_handler_name>.DLL** file in the **ServiceTest/bin** folder. If you compile the project from another location, or if you want to copy the DLL from one machine to another, make sure to place it in the bin folder.

Configuring the User Handler

You can declare the DLL user handler globally or locally.

To use the handler globally, for all requests in the script, add the following section to the **default.cfg** file located in the script's folder.

```
[UserHandler]
Function=<name>
Args=<arguments>
Order=<BeforeSecurity/AfterSecurity/AfterAttachments>
```

- **Name.** The name of the DLL.
- **Args.** A list of the configuration arguments for the handler. Use the **GetArguments** method to retrieve the arguments in your handler.
- **Order.** The order in which Vusers process the user handler in requests: **Before Security**, **After Security**, or **After Attachments**. You can also use this argument to override the transport layer, by entering the value **Replace Transport**.

Note: Setting the **UserHandlerFunction** property of a **web_service_call** function, overrides the definitions in the .cfg file.

By default, user handlers are processed before the security. For request messages, Vusers process the attachments handler after the security handler. For responses, Vusers process the handlers in a reversed order. In typical cases the order does not matter, so any value is acceptable.

To override the Transport layer, specify **Order=Replace Transport** and specify the new transport handler. If you implement the transport handler as a separate DLL, the **HandleRequest** function is called, while the **HandleResponse** function is ignored.

To use the handler locally, for a specific request, add the following arguments to the **web_service_call** function:

```
UserHandlerName=<name1>
UserHandlerArgs=<args1>
UserHandlerOrder=<BeforeSecurity/AfterSecurity/AfterAttachments/Replace
                  Transport>
```

Note: If you copy the script to another machine, it retains the handler information, since it is defined in script's folder. A user handler defined locally for a specific step in the script, overrides the global handler settings (defined in the script's **default.cfg** file).

Make sure that the user handler DLL is accessible to all Load Generator machines running scripts that call it. You may, for example, copy it to the **ServiceTest/bin** folder.

Implementing the User Handler

To implement a user handler, you use the entry functions **HandleRequest** or **HandleResponse**. Both functions have a single parameter, **context**, whose properties you can set in your handler. Use the Get functions to retrieve properties, and Set functions to pass information from the replay framework to the handlers or between the handlers.

- **GetEnvelope.** Gets the envelope content. For example, example:
  ```
  const char * pEnvelope = context->GetEnvelope();
  ```
- **GetEnvelopeLength.** Gets the envelope length
- **SetEnvelope.** Sets the envelope content and length. For example:
  ```
  string str("MySoapEnvelope...");
  context->SetEnvelope(str.c_str(), str.length());
  ```
- **SetContentType.** Sets a new value for HTTP header content type
- **LogMessage.** Issues a message to the replay log
- **GetArguments.** Gets the configuration arguments defined for the current handler in order to pass it to the DLL
- **GetProperty.** Gets a custom property value
- **SetProperty.** Sets a custom property value

For more information, see the comments in the **LrWsHandlerAPI.h** file located in the **ServiceTest/include** folder.

User Handler Examples

The following section describes how to create user handlers for several common issues:

- .NET Filters
- Overriding the Transport Layer
- Including MIME Attachments

.NET Filters

If you are familiar with Microsoft's Web Service Enhancements (WSE) 2.0, you can create a .NET filter and register it for incoming or outgoing SOAP messages. A .NET filter is a class that is derived from Microsoft.Web.Services2.SoapInputFilter or Microsoft.Web.Services2.SoapOutputFilter. By overriding the **ProcessMessage** function of this class, you can examine and modify the envelope's body and header.

You can apply a .NET filter to your messages using the user handler mechanism.

To define the filter globally for the entire script, add the following lines to the script's default.cfg file below.

```
[UserHandler]
Function=LrWsSoapFilterLoader
Args=<Filters InputFilterClass="class name" InputFilterLib="lib name"
OutputFilterClass="class name" OutputFilterLib="lib name" />
Order=BeforeSecurity/AfterSecurity/AfterAttachments
```

The **InputFilterClass** parameter indicates the name of your class, and **InputFilterLib** indicates the name of the assembly in which the class resides. For example:

```
    web_service_call(
        ...
        "UserHandlerName=LrWsSoapFilterLoader",
        "UserHandlerArgs=<Filters
InputFilterClass=\"MyFilterNamespace.MyFilterClassName\"
InputFilterLib=\"MyAssemblyName\" />",
        BEGIN_ARGUMENTS,
        ...
        END_ARGUMENTS,
        ...
        );
```

Use SoapOutputFilter to examine an outgoing **web_service_call** request, and SoapInputFilter to examine the response from the server. Use **InputFilterClass** and **InputFilterLib** if your filter is derived from SoapInputFilter, or **OutputFilterClass** and **OutputFilterLib** if your filter is derived from SoapOutputFilter.

To define the filter for a specific step, add the following arguments to the **web_service_call** function.

```
UserHandlerName= LrWsSoapFilterLoader
UserHandlerArgs=<Filters InputFilterClass=\"class name\" InputFilterLib=\"lib name\"
OutputFilterClass=\"class name\" OutputFilterLib=\"lib name\" />
UserHandlerOrder=BeforeSecurity/AfterSecurity/AfterAttachments
```

Overriding the Transport Layer

You can write a user handler function to override the transport layer. In this case, Service Test will not automatically send the SOAP request over HTTP transport—instead it follows the transport method indicated in the custom handler.

After you receive a response, you can set the response envelope with the command:

```
lr_save_string(someResponseEnvelopeStr, "SoapEnvelopeParam");
```

To apply an alternate transport layer, specify **ReplaceTransport** as an value for the **UserHandlerOrder** argument, and define the transport layer in the handler function.

```
web_service_call(
…
"UserHandlerFunction=<Transport HandlerFunction>",
"UserHandlerArgs=<handler arguments>",
"UserHandlerOrder=ReplaceTransport"
…
  LAST);
```

Including MIME Attachments

When working with Web Service scripts based on the .NET toolkit, the infrastructure does not support MIME attachments. Using the handlers mechanism, you can add MIME attachment functionality to .NET scripts.

The following sections describe how to send and receive MIME attachments for the .NET toolkit. You can receive and send a MIME attachment in the same operation.

Sending MIME Attachments

To send a MIME attachment, add the boldfaced code to the **web_service_call**:

```
web_service_call( "StepName=EchoComplex_101",
  "SOAPMethod=SimpleService|SimpleServiceSoap|EchoComplex",
  "ResponseParam=response",
  "Service=SimpleService",
  "UserHandlerName=LrWsAttachmentsHandler",
  "UserHandlerArgs=ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME;
  ContentType=text/plain; FileName=C:\\temp\\results.discomap",
  "ExpectedResponse=SoapResult",
  "Snapshot=t1208947811.inf",
  BEGIN_ARGUMENTS,
  "xml:cls="
   "<cls>"
    "<i>123456789</i>"
    "<s>abcde</s>"
   "</cls>",
  END_ARGUMENTS,
  BEGIN_RESULT,
  END_RESULT,
   LAST);
```

Modify the **FileName** and **ContentType** parameters to indicate the file you want to send and its content type.

Receiving MIME Attachments

To receive a MIME attachment, add the following code to the **web_service_call**:

```
"UserHandlerName=LrWsAttachmentsHandler",
 "UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;"
```

Sending and Receiving MIME Attachments

To send and receive a MIME attachment in the same **web_service_call**, add the following code:

```
"UserHandlerName=LrWsAttachmentsHandler",
 "UserHandlerArgs=ATTACHMENT_SAVE_ALL;ParamNamePrefix=attach;
ATTACHMENT_ADD; ATTACHMENTS_FORMAT_MIME; ContentType=text/plain;
FileName=C:\\temp\\results.discomap",
```

Using Custom Configuration Files

Configuration files let you customize advanced settings such as security information and the WSE configuration.

The standard .NET configuration file, **mmdrv.exe.config**, is located in the **Service Test/bin** folder.

If your application has its own configuration file, **app.config**, you can implement it in several ways:

- Save it as **mmdrv.exe.config**, overwriting the existing configuration file. This will apply your configuration information to all scripts on the machine.
- Save **app.config** to the script's folder. The settings in the **app.config** file override the ones in **mmdrv.exe.config**. In addition, if you save it to the script's file, it will always be associated with the script, not requiring you to copy it over separately to other machines.

Use the filter with the **Input** prefix if your filter is derived from SOAP input, or the **Output** prefix if your filter is derived from SOAP output.

In addition, the configuration file contains security information. You can configure whether or not to allow unsigned test certificates.

By default, Service Test allows unsigned certificates to facilitate testing. To disallow unsigned certificates, modify the **allowTestRoot** flag in the **mmdrv.exe.config** file to false.

```
<security>
  <x509 storeLocation=”currentuser” alllowTestRoot=”false”
```

18

Web Services - Negative Testing

Using Service Test, you can test your Web Service using different testing methodologies, such as positive or negative testing.

This chapter includes:

The following information only applies to Web Services/SOA Vuser scripts.

About Applying Testing Methodologies

When performing a functional test for your Web Service, you should approach the testing in a variety of ways. The most common type of testing is called **Positive Testing**—checking that the service does what it was designed to do.

In addition, you should perform **Negative Testing**, to confirm that the application did not perform a task that it was not designed to perform. In those cases, you need to verify that the application issued an appropriate error—a SOAP Fault.

To illustrate this, consider a form accepting input data—you apply positive testing to check that your Web Service has properly accepted the name and other input data. You apply negative testing to make sure that the application detects an invalid character, for example a letter character in a telephone number.

When your service send requests to the server, the server responds in one of the following ways:

- **SOAP Result.** A SOAP response to the request
- **SOAP Fault.** A response indicating that the SOAP request was invalid. Negative Testing applies only to SOAP faults.
- **HTTP Error.** An HTTP error, such as Page Not Found, unrelated to Web Services.

Service Test can check for a standard SOAP result or a SOAP fault responses—it always fails on HTTP errors. For example, if your Web Service attempts to access a Web page that cannot be found, resulting in a 404 HTTP error, the replay will fail, even if the SOAP is valid.

Understanding the Testing Settings

When creating a Web Service Call or SOAP Request in Service Test, you can indicate the type of testing you want to perform during replay:

Type of Testing	Description
Positive Testing	Accept SOAP result responses and fail on SOAP faults.
Negative Testing	Accept SOAP faults and fail on SOAP result responses.
Any Type	Accept both SOAP result and SOAP fault responses.

By default, Service Test, only performs positive testing and passes a test when it receives a SOAP result response. You can instruct Service Test to perform only negative testing, or to accept any SOAP response. If you enable negative testing only, and the server issues a regular SOAP result response, the step will have a **Failed** status.

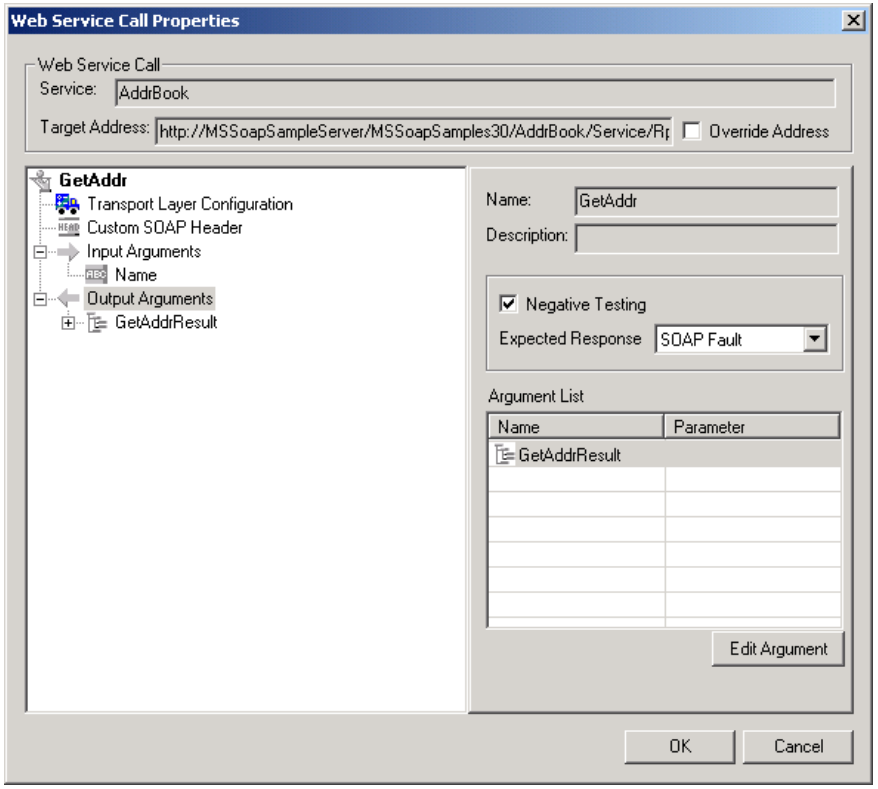
You can instruct Service Test to accept any SOAP response—a SOAP result or SOAP fault. This can be useful in testing environments where you only need to send the request, using a separate function to check the SOAP at a later time. In this testing mode, steps with either a SOAP result or SOAP fault will be issued a **Passed** status.

You can check the status of the replay by viewing the Replay log and Test Results report. A failed step will be marked in red as **Failed** in the Test Results.

If you are working with Quality Center or HP Service Test Manager, the application will list the test's status based on the Expected Response setting.

Defining a Testing Method

Before replaying the script, you indicate the testing method in the Web Service Call properties dialog box—positive or negative testing, or any SOAP.



To select a testing method:

1. Select the step whose response you want to test. Open the Properties from the right-click menu.
2. Select the **Output Argument** node.
3. Choose an Expected Response.
   - ➤ To perform positive testing only, clear the **Negative Testing** check box.
   - ➤ To perform negative testing only, select the **Negative Testing** check box and choose **SOAP Fault** as the **Expected Response**.
   - ➤ To accept any type of SOAP response, select the **Negative Testing** check box and choose **Any SOAP** as the **Expected Response**.

In Script view, Service Test represents the testing method with the **ExpectedResponse** argument. The possible values are **SoapResult**, **SoapFault**, or **AnySoap**. In the following example, the script performs negative testing, indicated by the **SoapFault** value:

```
web_service_call("StepName=AddAddr_101",
        "SOAPMethod=AddrBook|AddrBookSoapPort|AddAddr",
        "ResponseParam=response",
        "Service=AddrBook",
        "ExpectedResponse=SoapFault",
        "Snapshot=t1189409011.inf",
        BEGIN_ARGUMENTS,
        END_ARGUMENTS,
        BEGIN_RESULT,
        END_RESULT,
        LAST);
```

Evaluating the SOAP Fault Value

When you replay a script that results in a SOAP fault, Service Test saves the fault to a parameter called **response**. To check the value of the SOAP fault that was returned, you evaluate the **response** output parameter using **lr_xml_find.**

In the following example, **lr_xml_find** checks for a **VersionMismatch** SOAP fault and issues an output message.

```
lr_xml_find("XML={response}",
                "FastQuery=/Envelope/Body/Fault/faultString ",
                "Value=VersionMismatch",
                LAST);

if (soap_fault_cnt >0)
    lr_output_message{"A Version Mismatch SOAP Fault occurred")
```

For more information about **lr_xml_find**, see the *Online Function Reference*. (**Help** > **Function Reference**)

19

Web Services - Service Emulation

The Service Emulation tool lets you create an emulation of a Web service for the purpose of testing.

This chapter includes:

The following information only applies to Web Services and SOA Vuser scripts.

About Creating a Service Emulation

HP Service Test's **Service Emulator** allows you to create an emulation of a service in order to test other Web services in your environment.

The emulated service provides the following benefits:

- **Early stage development.** It lets you design and run tests at early stages of development when the actual service is inaccessible. For example if the development of the service is incomplete or if the service's host is unavailable, you can use an emulated service to test other services in your application.
- **Client testing.** Using the Service Emulator, you can test the functionality of your client application.
- **Isolating components.** If the service you want to test is part of a complex system, you can use the Service Emulator to test one service without its dependencies. The dependent services may be incomplete, temporarily unavailable, or simply distracting to your testing plans.

In the Service Emulation interface, you associate a WSDL document to the service. This indicates the service's operations and parameters. You specify rules and delays to define the service's behavior.

The steps in creating an emulated service are:

- Starting the Emulation Server
- Selecting a Host
- Adding a New Service
- Setting the Emulated Service's Behavior and Rules
- Manipulating Emulated Services

Starting the Emulation Server

The Service Emulation tool provides a Tomcat server through which you can run emulated services. The installation program installs an Axis servlet which allows you to run the service on the Tomcat server running on the local machine.

To create or run emulated services on your local machine, you must manually start the emulation service.

To start the emulation service:

➤ Choose **Start** > **All Programs** > **HP Service Test** > **Start Emulation Service**.

To stop the emulation service:

➤ Choose **Start** > **All Programs** > **HP Service Test** > **Stop Emulation Service.**

You can also manually start and stop the service from the Services dialog box (**Start Programs** > **Administrative Tools** > **Services**). The service is called **HP ServiceEmulation**.

To check whether or not the server is active, enter the following URL into your browser: http://localhost:8080/ServiceEmulation/index.jsp. If the server is active, the browser will display **HP Service Emulation.**

Troubleshooting the Server

If the Service Emulation console indicates that the server is not accessible, even after a manual start, you can try the following troubleshooting tips:

➤ Make sure the server is up. Enter the following URL into your browser: http://localhost:8080/ServiceEmulation/index.jsp. If the server is down, start it manually from the Start menu.

➤ Verify that port 8080 is available. If it is not, release the port and restart the server. Alternatively, edit the **Server.xml** file in the **apache-tomcat-5.5.17\conf** folder under the Service Test installation. Change the port from 8080 to another port. Create a new localhost server with the new port number.

- Open the Apache Tomcat log files under the <product install dir>\Service Test\apache-tomcat-5.5.17\logs directory and determine the reason that the server did not load. Fix the problem and reload the server.
- When emulating a service, if you encounter a warning "Service is not activated" in the endpoint location of your emulated service, perform one or more of the following actions:
  - Verify that the service is active. Select the service in the left pane and click the Activate Service button in the right pane, if it is visible.
  - If the service is already active, check its URL. Copy the URL from the WSDL Location in the Emulated Service's right pane, and paste it into a browser, removing the suffix **?wsdl** from the string. For example, instead of http://localhost:8080/axis/services/MyService?wsdl, use http://localhost:8080/axis/services/MyService. If your browser opens a valid page, then your service is active. To use this emulated service, import the original WSDL into Service Test. Then override the address and set it to the URL you previously used in the browser.

    For more information about integrating your emulated service into your test, see "Using Emulated Services in Scripts" on page 410.

Selecting a Host

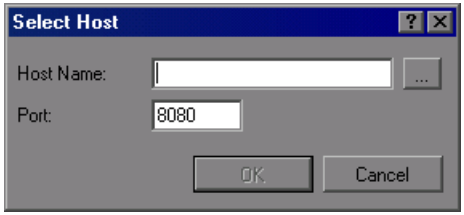
The first step in creating an emulated service is choosing a host for the emulated services. The setup installs a Tomcat server on your local machine. You can also specify external servers, upon which the Tomcat server is installed.

By default, the Service Emulation tool uses localhost as the emulation server.

To add an external host:

1 Open the Service Emulation tool. Choose **Start** > **All Programs** > **HP Service Test** > **Service Emulation Console**.

2 Choose **Main** > **New Host** or click the **Add Host** button to open the Select Host dialog box.



3 Browse for a host in your network, or enter the host name directly. If applicable, change the port setting from the default value, 8080.

4 Click **OK**. The host is added to the list in the window 's left pane.

Tip: If you have a conflict with the 8080 port, edit the **Server.xml** file in the **apache-tomcat-5.5.17\conf** folder under the Service Test installation. Change the port from 8080 to another port. Create a new localhost server with the new port number.

Adding a New Service

For each host that you added, you can create emulated services.

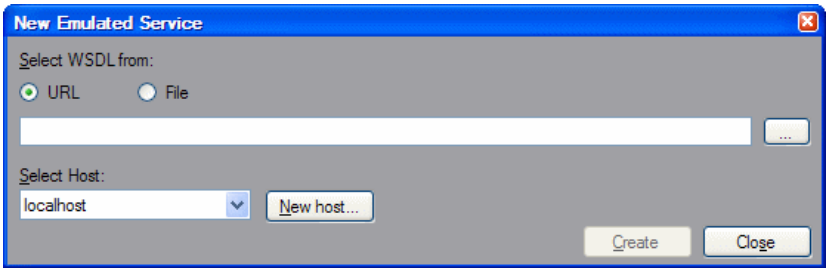
You create a service by specifying a WSDL file that defines the operations and parameters of the service. When you specify a WSDL file, the Service Emulation tool uses the current structure of the WSDL to define the structure of the service's input and output data.

If the original WSDL changes, it will not be reflected in your emulated service. To use an updated WSDL, recreate the emulated service.

Note: The importing of WSDLs is not supported for WSDLs that import schemas. To import this type of WSDL, deploy it on a local server and then import it using the generated URL.

To define a new service:

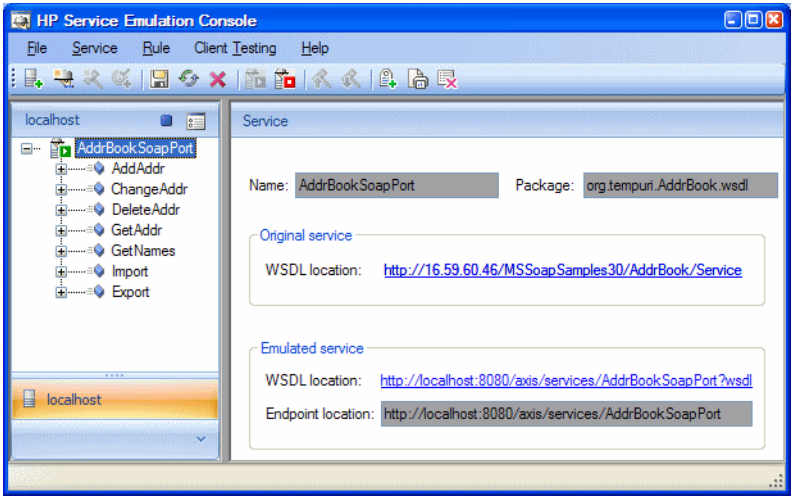
1 Choose **Main** > **New Emulated Service** or click the **New Emulated Service** button to open the Create a New Emulate Service dialog box.



2 Enter the full path or URL of the WSDL in the **select WSDL From** box, or click the Browse button to the right of the box, to locate the WSDL.

3 Choose a host from the list, or click **New Host** to add a new one.

4 Click **Create** to add the new service. The Service Emulator creates the service and lists all of its operations in the window's left pane. It also deploys and launches the service on the host.



5 Repeat the above steps to add additional services.

6 To delete a service, select it and choose **Delete** from the right-click menu.

7 To add a description about a specific operation, select it and enter text in the **Comments** box.

Setting the Emulated Service's Behavior and Rules

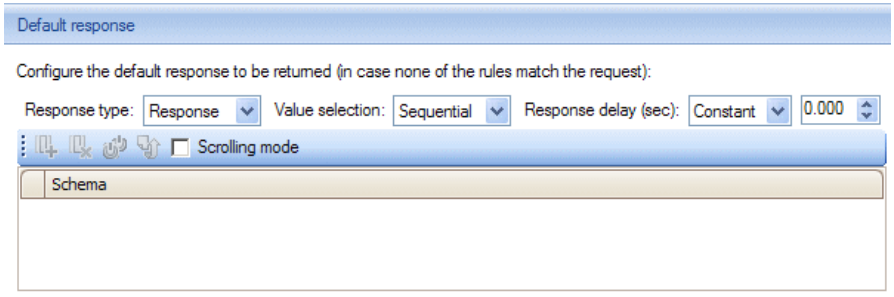
The Service Emulation lets you specify a behavior for each of the operations defined in the WSDL. You specify the behavior of the service's operations by:

- Delaying an Operation's Response
- Providing a Default Response
- Setting Rules

Delaying an Operation's Response

To more accurately emulate a service, you can insert a delay time for the response of each operation. This time emulates a delay in the response of the server after the application submits a request.

You can set a default delay for the operation's response, and a delay per rule. The rule delay overrides the default. The delay can be a constant one, or a random delay within a specified range.



To set the delay for the operation's default response:

1 Select an operation in the left pane. Expand it and click the **Default Response** node.

2 Choose the type of delay—**Constant** or **Random**.

3 For a constant delay, specify the delay in seconds. To indicate milliseconds, use the values to the right of the decimal. For example, to indicate 20 milliseconds, enter 0.02.

   For a random delay, specify a range of time.

Providing a Default Response

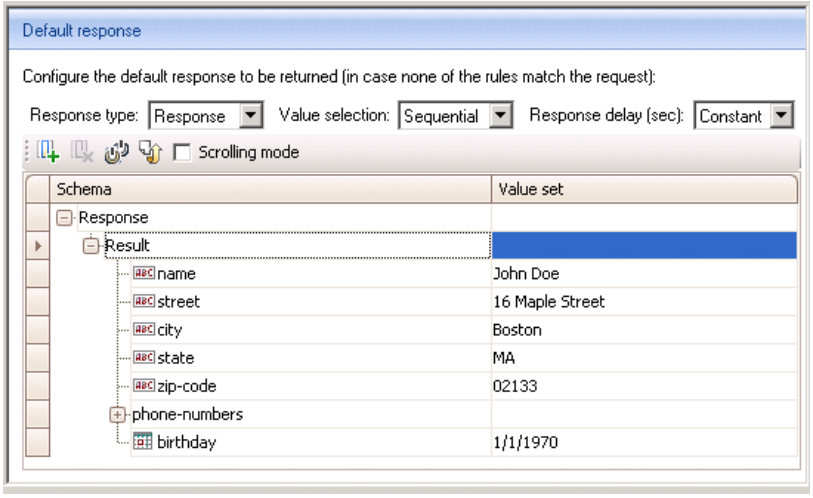
The default response is the set of values that the operation will use, if no rules are present.

You can manually specify the values for the default response, or you can import them from an XML file containing sample results.

To specify a default response:

1 Expand the service and desired operation in the left pane.

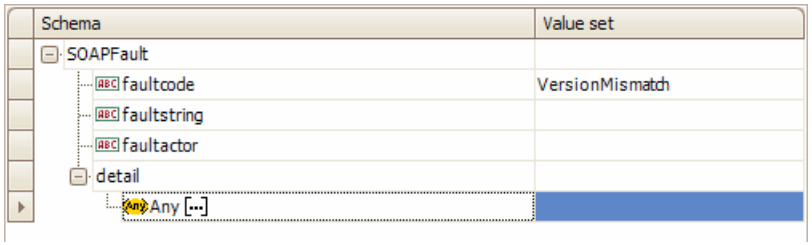
2 Expand an operation and select its **Default Response** node. The right pane shows a Default Response table.



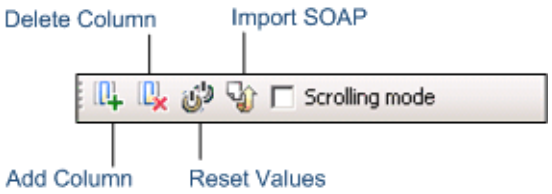
3 Select a type of SOAP response—**Response** or **Fault**.

4 For a SOAP Response, specify default response values for the result in the **Values** column.

For a SOAP Fault, specify the SOAP Fault argument values.



Use the toolbar to import values and create multiple value sets.



5 To import values from the SOAP response, click **Import SOAP**. Select the XML file with a valid SOAP response and click **OK**.

6 To add more value sets:

a Click the **Add Column** button.

b Provide their values or select a value set click **Import SOAP into the selected column.**

c Choose a Value Selection method: **Sequential** or **Random**.

d Select the **Scrolling mode** option to freeze the schema column, allowing you to scroll freely through the value sets.

7 To instruct the service to use the WSDL's default response values, click **Reset Values**.

8 Set the **Response delay**, using the decimal point to specify milliseconds.

Setting Rules

In addition to setting the default response, you can also set behavior rules. Through rules, you define a distinct behavior for the service—the expected response based on the request, input data.

You can set the element values manually, or by importing an existing XML SOAP file.

To set rules manually, you insert values for one or more of the arguments and a corresponding response. For example, for an Add operation, you could specify 4 as the first argument, 5 as the second argument, and a result of 9.

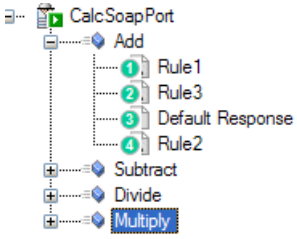
You do not need to specify an exact value. For strings, you can specify one of the following operators: **Equals**, **Not Equal**, **Contains**, **EndsWith**, or **StartsWith**. For numbers, choose one of the standard comparison operators such as **Equals**, **GreaterOrEqual**, **LessOrEqual**, **GreaterThan**, **or LessThan**.

To determine the data type of the value, place your mouse over the **ABC** or **123** icon adjacent to the element.

You can exclude arguments from the rule definition. This is useful if you want your service to return a particular response, ignoring the value of one of the arguments. For example, for a multiplication operation, you can set a rule indicating that if the first argument is 0, the result will be 0, regardless of the value of the second argument.

For each rule, you can provide comments which describe the rule and its behavior.

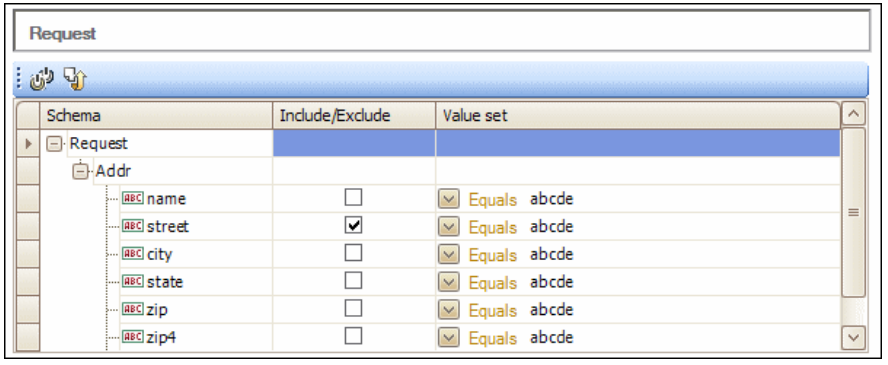
You can set multiple rules for your operation. You arrange the rules in order of priority. If there is a conflict between rules, the emulated service follows the position of the rules. A number icon indicates the priority of the rule. In the following example, **Rule 1** has the highest priority.



Request Rules

When setting up a rule for the request, you can indicate which elements to ignore, and which will impact your emulation, through the **Include/Exclude** box. Clear the boxes adjacent to the arguments that you want the emulation to ignore.

You can also use comparison operators such as **Equals** and **Not Equal**.



To set up a rule for the request:

1 Expand the service and desired operation in the left pane.

2 Click on an operation in the left pane. Select **Add New Rule** from the right-click menu or click the **New Rule** button.

3 Describe the rule in the **Comments** box, above the rule's schema.

4 Click in the upper section—**Request**. In the **Include/Exclude** column, select the check boxes adjacent to each of the arguments that you want to impact the rule. Select a comparison operator for the values such as **Equals** and **Not Equal**.

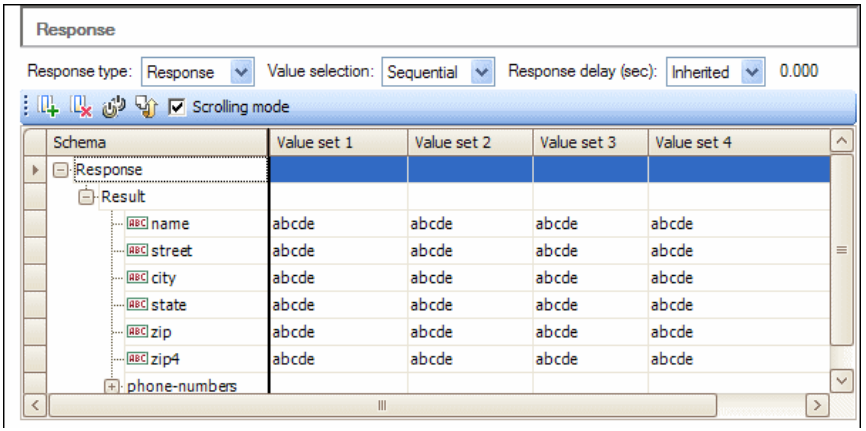
5 Specify values for the request arguments that you included. To import values from SOAP file, click **Import SOAP** and specify an XML file containing the SOAP message.

6 To reset the original values, click the **Reset Values** button.

Response Rules

When setting up a response rule, you can provide multiple result sets. You can then indicate how to use the value sets—in a sequential or random manner.

The response window provides a scrolling mode to enhance the data view. When you enable this mode, the response window freezes the schema column and provides scrolling for the value sets.



Using the toolbar, you can configure the following items:

- Response type—Response or SOAPFault
- Value selection (for multiple value sets)—sequential or random
- Delay type and duration—constant, random, or default

To set up a rule for the response:

1 Click in the lower part of the window, **Response**. Choose the type of response: **Response** or **Fault**.

2 Specify values for the response elements. To import values from SOAP file, select the value set column and click **Import SOAP.**

3 To add more value sets:

a click the **Add Column** button.

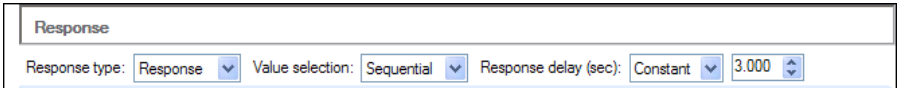
b Provide their values or select a value set click **Import SOAP into the selected column.**

c Choose a Value Selection method: **Sequential** or **Random**.

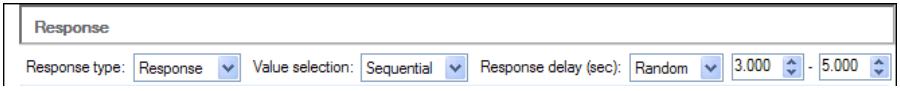
d Select the **Scrolling mode** option to freeze the schema column, allowing you to scroll freely through the value sets.

4 Choose the type of delay:

- **Default.** The delay values specified for the operation's default response.
- **Constant.** A constant delay. Specify the delay in seconds. To indicate milliseconds, use the values to the right of the decimal. For example, to indicate 20 milliseconds, enter 0.02.



- **Random.** A random delay. Specify a range for the delay.



5 To reset the rule to its default values, click the **Reset Values** button.

Handling Rules

6 After you set up a rule, click **Save** to store all of the modifications.

7 To add more rules for an operation, repeat the above steps.

8 To move a rule up to raise its priority, use the toolbar button or right-click and choose **Move up**.

To move a rule down, use the toolbar button or right-click and choose **Move down**.

9 To rename a rule, select **Rule** > **Rename** or press F2.

Manipulating Emulated Services

After you create your emulated services you can manipulate them by:

- Reloading an Emulated Service
- Deactivating and Activating a Service
- Changing Port Numbers

Reloading an Emulated Service

Reloading emulated services reverts to the last saved version of the service on the host. Any rules which were not saved will be lost. If another user saved changes to the service from a different machine, by reloading the service, you will bring those changes on to your machine.

This feature is useful if you made changes and you want to discard them and revert back to the saved version. Another use for this feature is if another user modified the service behavior, you can reload the service to get their changes.

To reload a service:

1 Select the service you want to reload.

2 Click the **Reload** button or choose **Reload** from the right-click menu.

Deactivating and Activating a Service

After you create a service, you can deactivate it temporarily instead of deleting it. If you deactivate a service, it will be ignored in your tests, but it will still be available if you need to implement it in the future.

In the list of emulated services, a green box indicates that a service is active and a red box indicates that it not active.

To deactivate a service, select it in the left pane and click the **Deactivate** button or click **Deactivate Service** in the right pane.

To activate a service that was previously deactivated, select it in the left pane and click the **Activate** button or click **Activate Service** in the right pane.

Changing Port Numbers

You can modify the port of your emulated service by changing the port number in the emulation's configuration file. The configuration file, **httpd.conf**, is stored in Service Test's **apache/conf** directory.

To modify the port number:

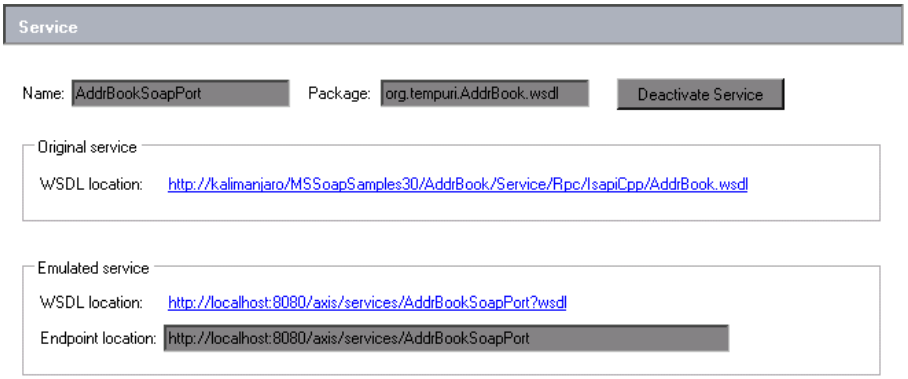
1. Open the **httpd.conf** file with any text editor.
2. Modify the entry "Listen 80" to the desired port number, for example "Listen 8080. "
3. Modify the entry which contains "ServerName localhost:80" to indicate the desired port, for example "ServerName localhost:8080."
4. Restart the Apache server. Select **Programs** > **Start** > **HP Service Test** > **Start Emulation Server**.

Using Emulated Services in Scripts

After you create an emulated service, you can import it into your Vuser script for testing purposes.

To use an emulated service:

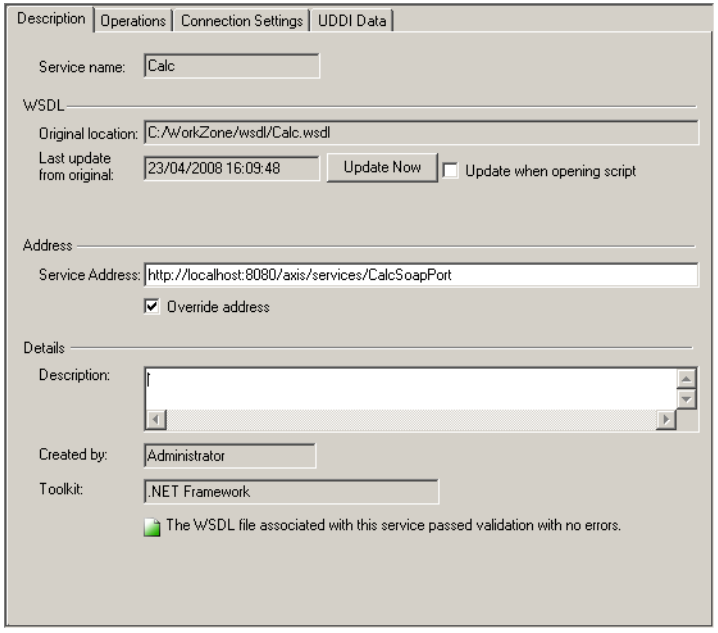
1 In the Service Emulation console, select a service in the left pane and copy the **WSDL location** from the right pane to the clipboard.



2 In Service Test, Open the Service Management window. Select **SOA Tools** > **Manage Services** or click the Service Management toolbar button.

3 If you did not yet import a WSDL for the service you want to emulate, click **Import**. If you have already imported the service that you want to emulate, select it in the left pane.

4 Enable the **Override address** option. Paste the WSDL location into the **Service Address** box. During the test run, the service will respond to that location.



The above description shows you how to use the original service's WSDL, but the emulated service address as the temporary **Service Address**.

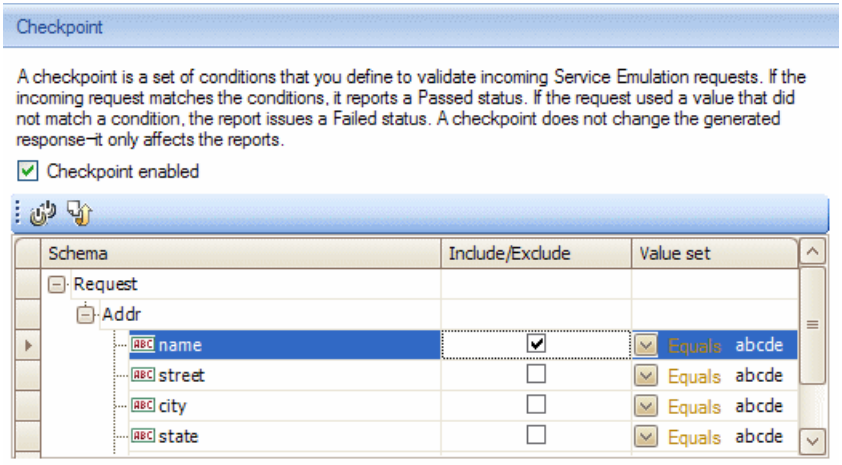
Client Testing

Service Emulation provides you with additional client testing tools:

- Service Emulation Checkpoints
- Service Emulation Labels
- Service Emulation Reports
- Clearing the Service Call Log

Service Emulation Checkpoints

A Service Emulation checkpoint is a set of conditions that you define to validate incoming requests. The conditions are the argument values together with comparison operators, with which you expect the request to comply. For example, you can set a checkpoint to verify that the value of argument **A** is greater than or equal to 2.



If the emulated service's incoming request matches the conditions, it reports a **Passed** status. If, however, the request used a value that did not match a condition, the report issues a **Failed** status. For more information on reports, see "Service Emulation Reports" on page 415.

A checkpoint does not change the generated response—it only affects the reports.

You can set a checkpoint for an entire operation or for a specific rule.

To set a checkpoint for an entire operation:

1 Select an operation in the left pane.

2 Click the New Checkpoint button or select **New Checkpoint** from the right-click menu.

3 For each value that you want to validate:

   a In the **Include/Exclude** column, enable the validation for that value.

b In the **Value Set** column, choose a comparison operator (Equals, Does not equal, and so forth).

c After the comparison operator, specify a value.

4 Repeat the above step for all values that you want to validate.

5 Select **Main** > **Save** or click the Save button on the toolbar.

Rule Checkpoints

You can set a checkpoint for a specific rule.

If you define a checkpoint for both an operation and a rule below it, the incoming request will need to comply with both checkpoints to be considered **Passed**.

A rule checkpoint is only activated if its parent rule is activated. This means that a rule's checkpoint will only be validated when the request complies with the rule.

To set a checkpoint for a rule:

1 Select a rule.

2 Click the New Checkpoint button or select **New Checkpoint** from the right-click menu.

3 For each value that you want to validate:

a In the **Include/Exclude** column, enable the checkpoint.

b In the **Value Set** column, choose a comparison operator (Equals, Does not equal, and so forth).

c After the comparison operator, specify a value.

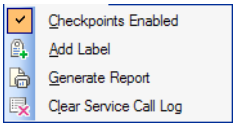
4 Repeat the above step for all values that you want to validate.

5 Select **Main** > **Save** or click the Save button on the toolbar.

Handling Service Emulation Checkpoints

You can define checkpoints and temporarily disable them. The checkpoints will be available for use when necessary. You can also disable individual checkpoints or remove them from your emulation.

The following table describes how to enable, disable, or delete checkpoints:

Action		Description
Enable/Disable all checkpoints		Use the Checkpoint Enable/Disable option in the **Client Testing** menu.
Enable/Disable specific checkpoints		Select or clear the **Checkpoint enabled** box in the Checkpoint area.
Remove checkpoints		Select the checkpoint and click the Delete button or select **Delete** from the right-click menu.

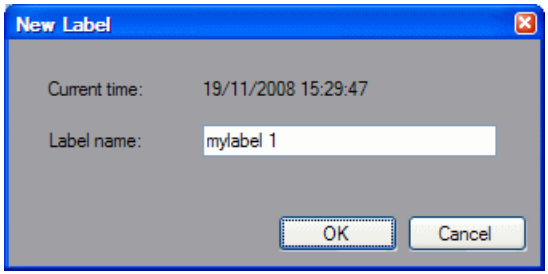
Service Emulation Labels

Labels allow you to put a date and time stamp on incoming service calls.

These labels will allow you to filter you data when generating reports. For more information, see the "Service Emulation Reports" on page 415.

To create a label:

1 Choose **Client Testing** > **Add Label**. or click the Add Label button. The New Label dialog box opens.



2 Specify a label name to be used for filtering the data. Click **OK**.

For more information on reports, see "Service Emulation Reports" on page 415.

Service Emulation Reports

You can create reports to view, filter and sort the information about your emulated service. You can select which information to include and the display order.

To create a report:

1 Choose **Client Testing** > **Generate Report** or click the Generate Report button. The Report Wizard Welcome screen opens.

2 Click **Next**. The Column Selection screen opens.

3 Use the right and left arrows to move the fields that you want in your report, to the right column. Use the up and down arrows to set the order of the columns in the report.

4 Click **Next**. The Filtering screen opens.

5 Select the items you want to filter, and provide the filter information. You can filter by date or label, service, operation, checkpoint status, client IP, and Session ID. The report supports multiple selections for all items except for date and label.

6 Click **Next**. The Sorting screen opens.

7 Specify the sorting order for the values in your report. Use the right facing arrows to include an item in the sorting list. Use the up and down arrows to specify the priority of each item. For each field, select a sorting order—**Ascending** or **Descending**. This sorting order is not global and may be set differently for each item. The default for each item is ascending order.

8 Click **Finish**. The wizard creates the report.

Clearing the Service Call Log

Service Test allows you to clear the database log, containing all the service calls sent through the emulated services.

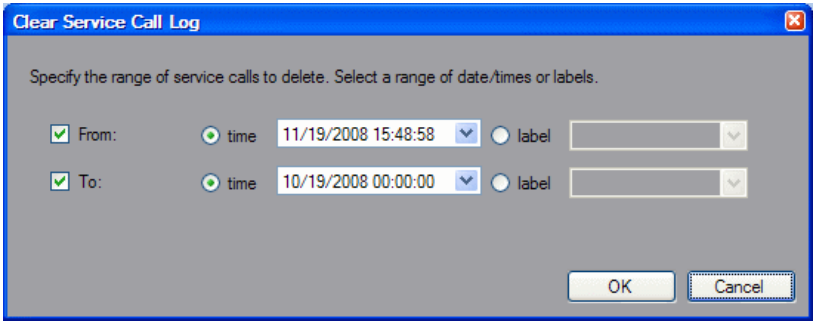
The custom defined settings such as rules, checkpoints, and default responses, remain unaffected.

To clear the log:

1 Choose **Client Testing** > **Clear Service Call Log** or click the **Clear Service Log** button.

2 Select a range of dates or labels whose service call's you want to delete.

To clear all service calls from the log, clear the **From** and **To** checkboxes. If you do not select a **From** range, all records from the beginning of the log until the specified date/ label will be deleted. If you do not select a **To** range, all records from the **From** date/label until the end of the log will be deleted.



3 Click **OK**. The program clears the service calls within the specified range.

Part IV

Parameters

20

Working with VuGen Parameters

When you record a business process, VuGen generates a script that contains the actual values used during recording. Suppose you want to perform the script's actions (query, submit, and so forth) using different values from those recorded. To do this, you replace the recorded values with parameters. This is known as *parameterizing* the script.

This chapter includes:

About VuGen Parameters

When you record a business process, VuGen generates a Vuser script composed of functions. The values of the arguments in the functions are the actual values used during the recording session.

For example, assume that you recorded a Vuser script while operating a Web application. VuGen generated the following statement that searches a library's database for the title "UNIX":

```
web_submit_form("db2net.exe",
        ITEMDATA,
        "name=library.TITLE",
        "value=UNIX",
        ENDITEM,
        "name=library.AUTHOR",
        "value=",
        ENDITEM,
        "name=library.SUBJECT",
        "value=",
        ENDITEM,
        LAST);
;
```

When you replay the script using multiple Vusers and iterations, you do not want to repeatedly use the same value, UNIX. Instead, you replace the constant value with a parameter:

```
web_submit_form("db2net.exe",
        ITEMDATA,
        "name=library.TITLE",
        "value={Book_Title}",
        ENDITEM,
        "name=library.AUTHOR",
        "value=",
        ENDITEM,
        "name=library.SUBJECT",
        "value=",
        ENDITEM,
        LAST);
```

The resulting Vusers then substitute the parameter with values from a data source that you specify. The data source can be either a file, or internally generated variables. For more information about data sources, see "Understanding Parameter Types" on page 425.

Parameterizing a Vuser script has the following advantages:

- It reduces the size of the script.
- It provides the ability to test your script with different values. For example, if you want to search a library's database for several titles, you only need to write the submit function once. Instead of instructing your Vuser to search for a specific item, use a parameter. During replay, VuGen substitutes different values for the parameter.

Parameterization involves the following tasks:

- Replacing the constant values in the Vuser script with parameters
- Setting the properties and data source for the parameters

Understanding Parameter Limitations

You can use parameterization only for the arguments within a function. You cannot parameterize text strings that are not function arguments. In addition, not all function arguments can be parameterized. For details on which arguments you can parameterize, see the *Online Function Reference* (**Help** > **Function Reference**) for each function.

For example, consider the lrd_stmt function. The function has the following syntax:

```
lrd_stmt (LRD_CURSOR FAR *mptCursor, char FAR *mpcText, long mliTextLen,
LRDOS_INT4 mjOpt1, LRDOS_INT4 mjOpt2, int miDBErrorSeverity);
```

The *Online Function Reference* indicates that you can parameterize only the *mpcText* argument.

A recorded **lrd_stmt** function could look like this:

```
lrd_stmt(Csr4, "select name from sysobjects where name =\"Kim\" ", -1, 148, -99999, 0);
```

You could parameterize the recorded function to look like this:

```
lrd_stmt(Csr4, "select name from sysobjects where name =\"<name>\" ", -1, 148, -99999, 0);
```

Note: You can use the **lr_eval_string** function to "parameterize" a function argument that you cannot parameterize by using standard parameterization. In addition, you can use the **lr_eval_string** function to "parameterize" any string in a Vuser script.

For VB, COM, and Microsoft .NET protocols, you must use the **lr.eval string** function to define a parameter. For example, lr.eval_string("**{Custom_param}**").

For more information on the **lr_eval_string** function, see the *Online Function Reference*.

Creating Parameters

You create a parameter by giving it a name, and specifying its type and properties. There is no limit to the number of parameters you can create in a Vuser script.

Step 1: Select the argument that you want to parameterize.

If you are in Script view:

Select the argument that you want to parameterize, and select **Replace with a Parameter** from the right-click menu.

Notes:

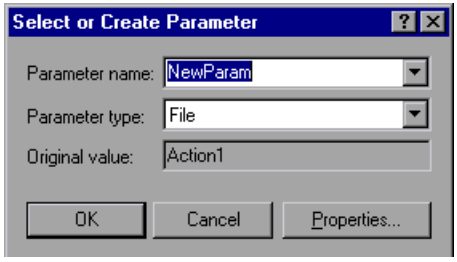
- When creating XML parameters in script view, you must select only the inner xml, without the bounding tags. For example, to parameterize the complex data structure <A><B>Belement</B><C>Celement</C></A>, select the whole string, <B>Belement</B><C>Celement</C>, and replace it with a parameter.
- When parameterizing Java Record Replay or Java Vuser scripts, you must parameterize complete strings, not parts of a string.

If you are in Tree view:

1 Right-click the step you want to parameterize, and select **Properties** from the menu. The appropriate Step Properties dialog box opens.

2 Click the **ABC** icon next to the argument that you want to parameterize.

The Select or Create Parameter dialog box opens.



Step 2: Name the parameter.

Type a name for the parameter in the **Parameter name** box. The parameter name is displayed in the script in place of the original argument.

The parameter name should be suitable to the type of information that will replace the parameter during a script run.

For example, if you typically enter a username, then name the parameter Username.

Note: Do not name a parameter *unique*, since this name is used by VuGen.

Step 3: Select a parameter type.

When you create a parameter, you specify the source of the parameter data. This determines the *parameter type*.

Data can be generated internally - such as the date and time, or can be returned as a result of a user-defined function.

Another, very common method for using parameters, is instructing Vusers to take values from an data table or an external file which contains values that the user has defined. These parameters are called File and Table type parameters.

From the **Parameter type** list, select **File**.

For more detailed information about the different parameter types, see "Understanding Parameter Types" on page 425.

Step 4: Define properties for the parameter type.

1 Click **Properties**. The Parameter Properties dialog box opens.

2 Click **Create Table**. A message box opens. Click **OK**.

VuGen creates a table with one cell containing the argument's original value.

3 To add another value to the table, click **Add Row**, and enter the value.

Repeat this step to add more values to the table.

4 Click **Close** to close the Parameter Properties dialog box.

For more information, see "Defining Parameter Properties" on page 427.

Step 5: Replace the argument with the parameter.

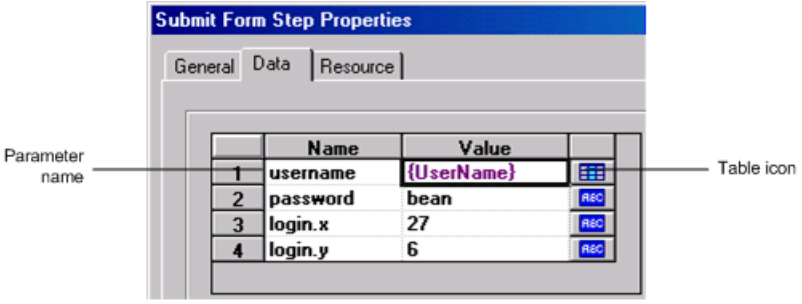
Click **OK** to close the Select or Create Parameter dialog box.

VuGen replaces the selected string in your script with the name of the parameter, surrounded by curly or round brackets.

Note: The default parameter braces are either curly or angle brackets, depending on the protocol type. You can check the proper parameter braces in the Parameterization tab (select **Tools** > **General Options**). For more information, see "Setting Parameterization Options" on page 434.

In Tree view, VuGen replaces the **ABC** icon with the table icon.

In the following example, the original **username** value was **jojo**. It has been replaced with the parameter **{UserName}**.



Understanding Parameter Types

When you create a parameter, you specify the source for the parameter data. You can specify any one of the following data source types:

- File or Table Parameter Types
- XML Parameter Types
- BPT Type Parameters

File or Table Parameter Types

Data that is contained in a file—either an existing file or one that you create with VuGen or MS Query. A very common method for using parameters, is instructing Vusers to take values from an external file or a data table.

Data Files

Data files hold data that a Vuser accesses during script execution. Data files can be local or global. You can specify an existing ASCII file, use VuGen to create a new one, or import a database file. Data files are useful if you have many known values for your parameter.

The data in a data file is stored in the form of a table. One file can contain values for many parameters. Each column holds the data for one parameter. Column breaks are marked by a delimiter, for example, a comma.

In the following example, the data file contains ID numbers and first names:

```
id,first_name
120,John
121,Bill
122,Tom
```

Note: When working with languages other than English, save the parameter file as a UTF-8 file. In the Parameter Properties window, click **Edit with Notepad**. In Notepad, save the file as a text file with UTF-8 type encoding.

Data Tables

The Table parameter type is meant for applications that you want to test by filling in table cell values. Whereas the file type uses one cell value for each parameter occurrence, the table type uses several rows and columns as parameter values, similar to an array of values. Using the table type, you can fill in an entire table with a single command. This is common in SAPGUI Vusers where the **sapgui_fill_data** function fills the table cells.

For information about defining data file or data table parameter properties, see Chapter 21, "File, Table, BPT, and XML Parameter Types."

XML Parameter Types

Used as a placeholder for multiple valued data contained in an XML structure. You can use an XML type parameter to replace the entire structure with a single parameter. For example, an XML parameter called **Address** can replace a contact name, an address, city, and postal code. Using XML parameters for this type of data allows for cleaner input of the data, and enables cleaner parameterization of Vuser scripts. We recommend that you use XML parameters with Web Service scripts or for SOA services.

BPT Type Parameters

You typically use BPT (Business Process Test) type parameters to share parameters between business components in Quality Center. In the Parameter Properties dialog box, you can configure properties such as Input/Output, value, data type and description. For information on setting parameter properties in Service Test, see "Defining Parameter Properties" on page 427.

For more information, see the section on Business Process Tests or see the *Business Process Testing User Guide*.

Defining Parameter Properties

You can define a parameter's properties in the Parameter Properties dialog box or in the Parameter List dialog box.

To define parameter properties in the Parameter Properties dialog box:

1 Open the Parameter Properties dialog box.

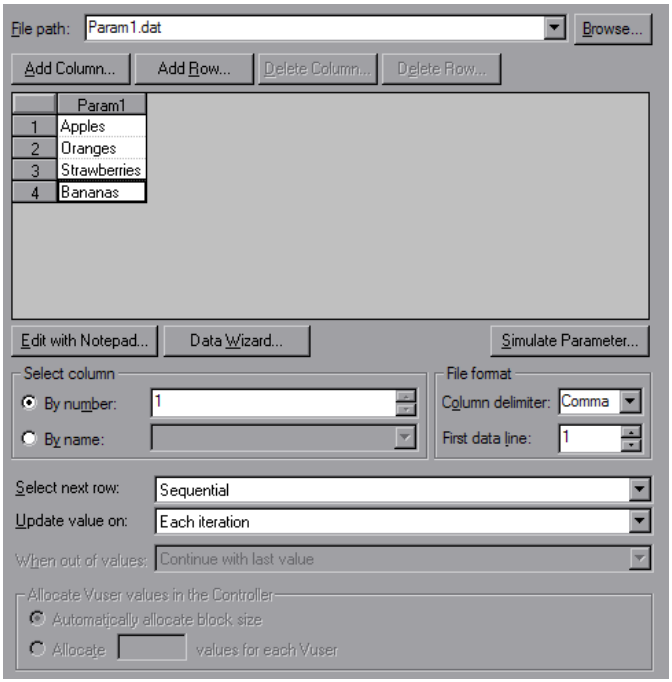
You open the Parameter Properties dialog box in one of the following ways:

- When you create a new Parameter as described in "Creating Parameters" on page 422, you click **Properties** in the Select or Create Parameter dialog box to open the Parameter Properties dialog box.
- In Script view, select the parameter, and select **Parameter Properties** from the right-click menu.

➤ In Tree view, right-click the step containing the parameter whose properties you want to define, and select **Properties**. The Step Properties dialog box for the selected step opens.

Click the table icon beside the parameter whose properties you want to define, and select **Parameter Properties** from the pop-up menu.

In the following example, the properties of a **file** type parameter are displayed:



2 Define the parameter properties.

➤ To define properties for File and Table type parameters, see Chapter 21, "File, Table, BPT, and XML Parameter Types."

➤ To define properties for internal data parameter types, see "Setting Properties for Internal Data Parameter Types" on page 466.

➤ To define BPT properties, see "Setting Properties for BPT Type Parameters" on page 446.

3 Close the Parameter Properties dialog box.

Click **Close** to close the Parameter Properties dialog box.

To define parameter properties in the Parameter List dialog box:

Click the **Parameter List** button, or select **Vuser** > **Parameter List**. Select a parameter to show its properties.

For more information, see "Using the Parameter List" on page 431.

Using Existing Parameters

When you create a parameter, VuGen stores it in a parameter list. You can use an existing parameter to replace an argument, or to replace multiple occurrences of an argument.

Using the **Parameter List** is convenient when you want to replace an argument with a previously defined parameter and, at the same time, view or modify that parameter's properties. For details on using the Parameter List, see "Using the Parameter List" on page 431.

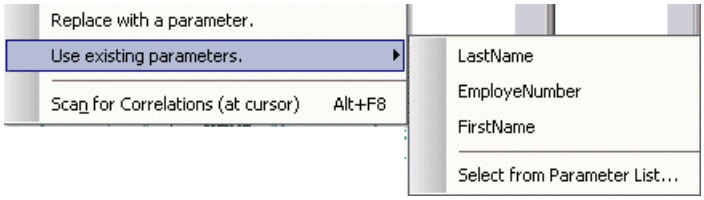
Replacing Strings Using Pre-defined Parameters

You can assign a pre-defined parameter to an argument.

To replace a string with a pre-defined parameter:

1 Enter Script view.

2 Right-click on the argument that you want to parameterize, and select **Use existing parameters**. A submenu opens.



3 Use one of the following options to select a parameter:

- Select a parameter from the submenu list.
- Choose **Select from Parameter List** to open the Parameter List dialog box, and select a parameter from the left pane.

Replacing Multiple Occurrences

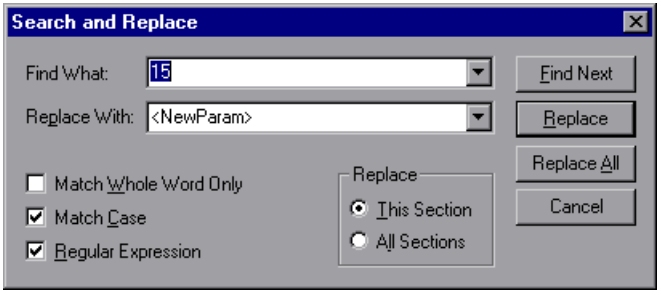
When you create a parameter, the system remembers the original value of the argument. You can use the **Search and Replace** function to replace selected or all occurrences of the same argument with the same parameter or another existing parameter.

To replace multiple occurrences of an argument with a specific parameter:

1 Right-click a parameter and select **Replace more occurrences** from the menu.

The Search and Replace dialog box opens. The **Find What** box displays the value or argument you want to replace. The **Replace With** box displays the parameter name in brackets.

2 Select the appropriate check boxes for matching whole words or case. To search with regular expressions (., !, ?, +, and so forth.) select the **Regular Expressions** check box.



3 Click **Replace** or **Replace All**.

In the above example, all arguments of value **15** are replaced with the parameter, **{NewParam}**.

Note: Use caution when using **Replace All**, especially when replacing number strings. VuGen changes all occurrences of the string.

Restoring Original Strings

VuGen lets you undo the parameterization and restore the originally recorded argument.

To restore a parameter to its original value:

➤ In Script view, right-click on the parameter and select **Restore original value**.

➤ In Tree view:

➤ Right-click on the step that contains the parameter and click **Properties**.

➤ Click the table icon next to the parameter that you want to restore to its original value, and select **Undo Parameter**.

The original argument is restored.

Using the Parameter List

You use the Parameter List to view, create, delete, select, and modify parameters.

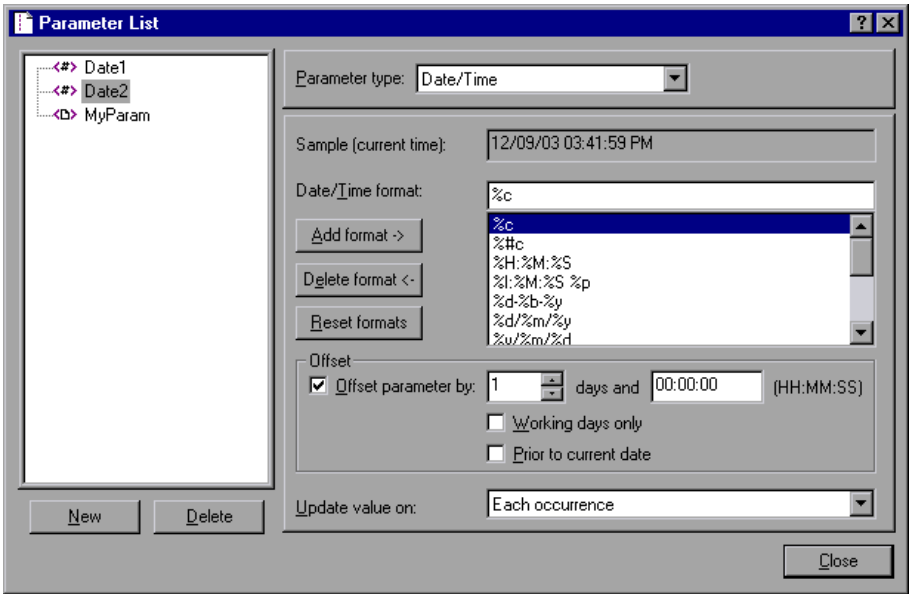
To view the Parameter List and view a parameter's properties:

Click the **Parameter List** button, or select **Vuser** > **Parameter List**. Select a parameter to show its properties.

The Parameter list shows all of the parameters that you created, including both input and output parameters. Input parameters are parameters whose value you define in the design stage before running the script. Output parameters you define during design stage, but they acquire values during test execution. Output parameters are often used with Web Service calls.

Use care when selecting a parameter for your script during design stage, make sure that it is not an empty Output parameter.

In the following example, the properties of a **Date/Time** type parameter are displayed:



To modify a parameter's properties:

Select the parameter from the parameter tree on the left, and edit the parameter's type and properties in the right pane.

For more information on setting parameter properties, see Chapter 22, "Setting Parameter Properties," and Chapter 21, "File, Table, BPT, and XML Parameter Types."

To create a new parameter:

1. In the Parameter List dialog box, click **New**. The new parameter appears in the parameter tree with a temporary name.
2. Type a name for the new parameter, and press Enter.

   Note: Do not name a parameter *unique*, since this name is used by VuGen.

3 Set the parameter's type and properties.

4 Click **Close** to close the Parameter List dialog box.

Note: VuGen creates a new parameter, but does not automatically replace any selected string in the script.

To delete an existing parameter:

1 Select the parameter from the parameter tree, and click **Delete**. The Delete Parameter dialog box opens.

2 If you want to delete the parameter file from the disk, select **Delete parameter data file from disk**.

3 Click **Yes**.

4 If you selected **Delete parameter data file from disk**, VuGen sends a warning message. Click **Yes** to confirm your action.

Setting Parameterization Options

You set the parameter options in the Parameterization tab of VuGen's General Options window. These options refer to:

- Parameter Braces
- Global Directory

Parameter Braces

When you insert a parameter into a Vuser script, VuGen places parameter braces on either side of the parameter name. The **Parameterization** tab (**Tools** > **General Options**) shows the default braces for your protocol. In the following example, the Web protocol uses curly brackets:

```
web_submit_form("db2net.exe",
        ITEMDATA,
        "name=library.TITLE",
        "value={Book_Title}",
        ENDITEM,
        "name=library.AUTHOR",
        "value=",
        ENDITEM,
        "name=library.SUBJECT",
        "value=",
        ENDITEM,
        LAST);
```

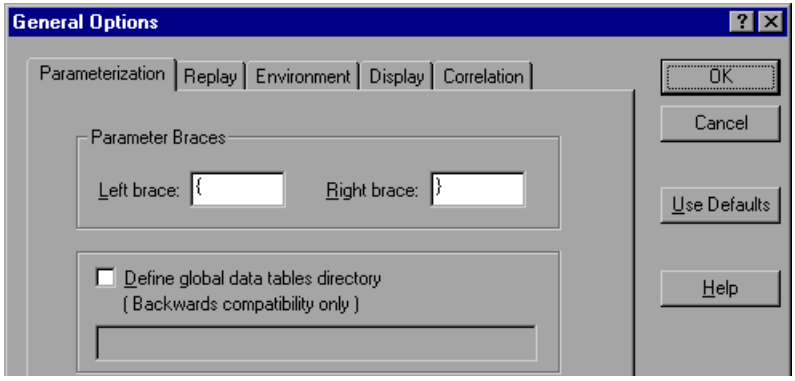
You can change the style of parameter braces by specifying a string of one or more characters. All characters are valid with the exception of spaces.

Note: The default parameter braces are either angle or curly brackets, depending on the protocol type. To check the default brace type, create a new script and check the **Parameterization** tab (select **Tools** > **General Options**).

To change the parameter brace style:

1 Select **Tools** > **General Options** in VuGen. The General Options dialog box opens.

2 Select the **Parameterization** tab and enter the desired brace.



3 Click **OK** to accept the settings and close the dialog box.

Global Directory

This option is provided only for backward compatibility with earlier versions of VuGen. In earlier versions, (4.51 and below), when you created a new data table, you specified local or global. A local table is saved in the current Vuser script directory and is only available to Vusers running that script. A global table is available to all Vuser scripts. The global directory can be on a local or network drive. Make sure that the global directory is available to all machines running the script. Using the General Options dialog box, you can change the location of the global tables at any time.

In newer versions of VuGen, you specify the location of the data table either in the Parameter Properties dialog box or in the Parameter List dialog box. VuGen is able to retrieve the data from any location that you specify, be it the default script directory or another directory on the network. For more information, see "Data Files" on page 426.

By default, the **Define global data tables directory** option is disabled.

To set the global directory:

1. Select **Tools** > **General Options**. The General Options dialog box opens.
2. Select the **Parameterization** tab.
3. Select the **Define global data tables directory** check box, and specify the directory containing your global data tables.
4. Click **OK** to accept the settings and close the dialog box.

21

File, Table, BPT, and XML Parameter Types

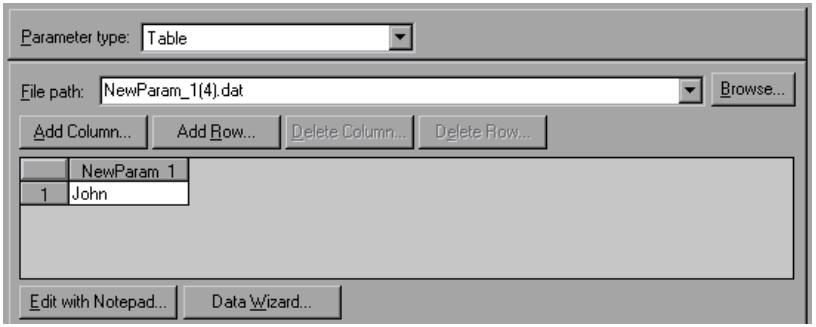
A very common method for using parameters is instructing Vusers to take values from a data table or an external file. The data is contained either in an existing file or in a file that you create with VuGen or MS Query.

This chapter includes:

Selecting or Creating Data Files or Data Tables

When you create a File or Table parameter you have to create a .dat file to store the data, or open an existing one. Then you define the other properties for the parameter, such as how the Vuser should assign values to the parameter.

You can create a new data table or select an existing data source from the File Path list.



To select a source file or table for your data:

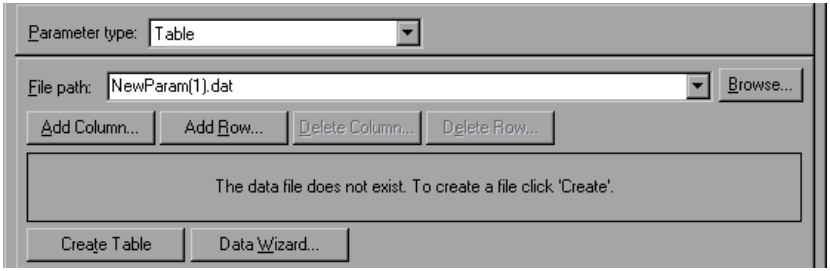
1 Open the Parameter Properties dialog box or the Parameter List.

For instructions, see "Defining Parameter Properties" on page 427.

2 Select a table or create a new one.

- If there are no tables (**.dat** files) listed in the file path list, or you want to create a new table, click **Create Table**. VuGen creates a new table with one cell, displaying the original value of the argument in the first column of the table.
- To open an existing data file, type the name of the **.dat** file in the **File path** box or select a name from the drop-down list.

Alternatively, click **Browse** to specify the file location of an existing data file. By default, all new data files are named <*parameter_name*>**.dat** and are stored in the script's directory.



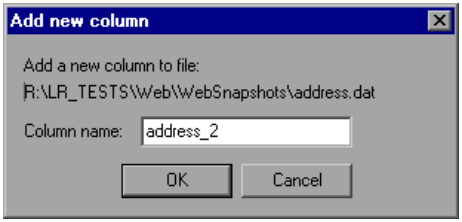
VuGen opens the data file and displays the first 100 rows. To view all of the data, click **Edit with Notepad** and view the data in a text editor.

Note: You can also specify a global directory. Global directories are provided only for backward compatibility with earlier versions of VuGen. For more information, see "Global Directory" on page 435.

➤ To import data from an existing database, click **Data Wizard** and follow the wizard's instructions. For more information, see "Importing Data from an Existing Databases" on page 440.

3 Add columns and rows to the table.

➤ To add additional columns to the table, select **Add Column**. The Add new column dialog box opens. Enter a column name and click **OK.**

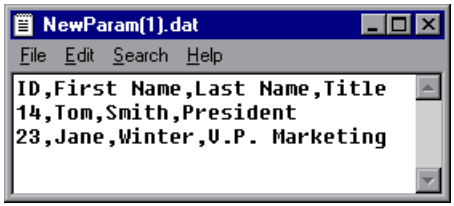


➤ To add additional rows to the table, select **Add Row**.

4 Edit the data file.

➤ Click within any cell to enter a value.

➤ To edit the data file from within Notepad, click **Edit with Notepad**. Notepad opens with the parameter's name in the first row and its original value in the second row. Enter additional column names and values into the file using a delimiter such as a comma or a tab to indicate a column break. Begin a new line for each table row (for each new row of data).



Importing Data from an Existing Databases

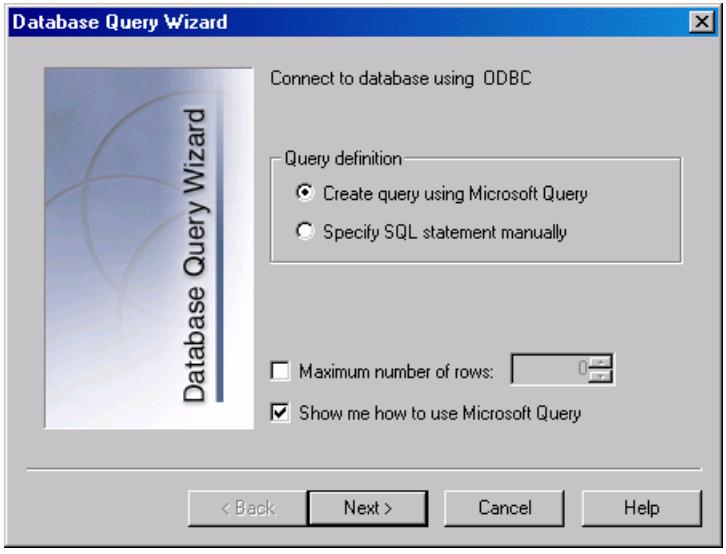
VuGen allows you to import data from a database for use with parameterization. You can import the data in one of two ways:

➤ Creating a New Query

➤ Specifying an SQL Statement

VuGen provides a wizard that guides you through the procedure of importing data from a database. In the wizard, you specify how to import the data—create a new query via an MS Query or by specifying an SQL statement. After you import the data, it is saved as a file with a *.dat* extension and stored as a regular parameter file.

To begin the procedure of importing a database, click **Data Wizard** in the Parameter List dialog box (**Vuser** > **Parameter List**). The Database Query Wizard opens.



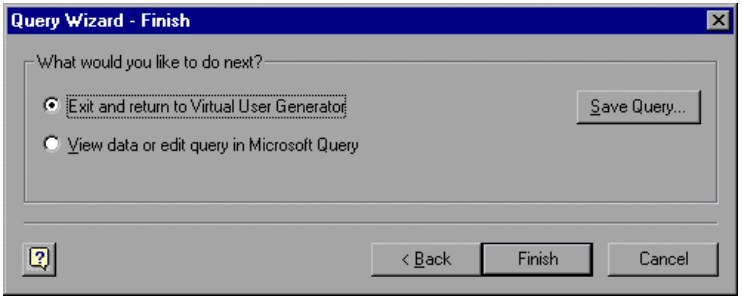
Creating a New Query

You use Microsoft's Database Query Wizard to create a new query. This requires the installation of MS Query on your system.

To create a new query:

1. Select **Create query using Microsoft Query**. If you need instructions on Microsoft Query, select **Show me how to use Microsoft Query**.
2. Click **Finish**. If Microsoft Query is not installed on your machine, VuGen issues a message indicating that it is not available. Install MS Query from Microsoft Office before proceeding.
3. Follow the instructions in the wizard, importing the desired tables and columns.

4 When you finish importing the data, select **Exit and return to the Virtual User Generator** and click **Finish**. The database records appear in the Parameter Properties box as a data file.



To edit and view the data in MS Query, select **View data or edit in Microsoft Query**.

5 Set the data assignment properties. See "Setting Properties for File Type Parameters" on page 444.

Specifying an SQL Statement

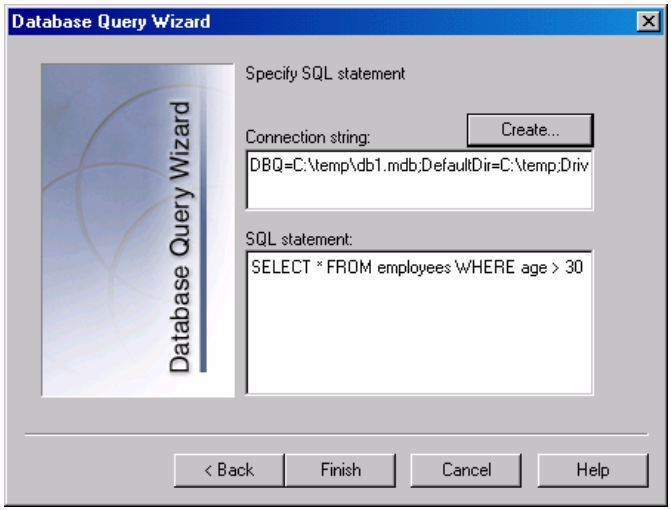
To specify a database connection and SQL statement:

1 Select **Specify SQL Statement**. Click **Next**.

2 Click **Create** to specify a new connection string. The Select Data Source window opens.

3 Select a data source, or click **New** to create a new one. The wizard guides you through the procedure for creating an ODBC data source. When you are finished, the connection string appears in the **Connection String** box.

4 In the **SQL statement** box, enter an SQL statement.



5 Click **Finish** to process the SQL statement and import the data. The database records appears in the Parameter Properties box as a data file.

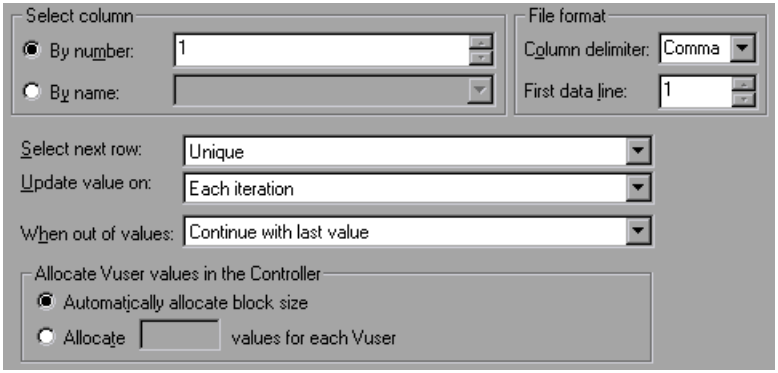
6 Set the data assignment properties. See "Setting Properties for File Type Parameters" on page 444.

After creating table or file data, you set the assignment properties. The properties specify the columns and rows to use, and whether to use the data randomly or sequentially. You set the properties separately for the File and Table type parameters.

Note: You can also set the properties for a parameter from the Parameter List dialog box. In the left pane, select the parameter and then specify its properties in the right pane. See "Using the Parameter List" on page 431.

Setting Properties for File Type Parameters

After you select a source of data, you set the assignment properties for your file. These properties instruct VuGen how to use the data. For example, they indicate which columns to use, how often to use new values, and what do to when there are no more unique values.



To set the File parameter properties:

1 Specify the column in the table that contains the values for your parameter. In the **Select column** section, specify a column number or name.

To specify a column number, select **By number** and the column number. The column number is the index of the column containing your data. For example, if the data for the parameter is in the table's first column, set it to 1.

To specify a column name, select **By name** and select the column name from the list. The column header is the first row of each column (row 0). If column numbers might change, or if there is no header, use the column name to select a column.

2 In the **Column delimiter** box of the **File format** section, enter the column delimiter—the character used to separate the columns in the table. You can specify a comma, tab, or space.

3 In the **First data line** box of the **File format** section, select the first line of data to be used during Vuser script execution. The header is line 0. To begin with the first line after the header, specify 1. If there is no header, specify 0.

4 Select a Data Assignment method from the **Select next row** list to instruct the Vuser how to select the file data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information, see "Choosing Data Assignment Methods for File/Table Parameters" on page 450.

5 Select an update option from the **Update value on** list. The choices are **Each Iteration**, **Each Occurrence**, and **Once**. For more information, see "Data Assignment and Update Methods for File/Table/ XML Parameters" on page 452.

6 If you chose **Unique** as the Data Assignment method (in step 4):

- **When out of values.** Specify what to do when there is no more unique data: **Abort the Vuser**, **Continue in a cyclic manne***r*, or **Continue with last value**.
- **Allocate Vuser values in the Controller** (for LoadRunner users only)**.** Indicate whether you want to manually allocate data blocks for the Vusers. You can allow the Controller to automatically allocate a block size or you can specify the desired number of values. Select **Automatically allocate block size** or **Allocate** x **values for each Vuser.** For the second option, specify the number of values to allocate.

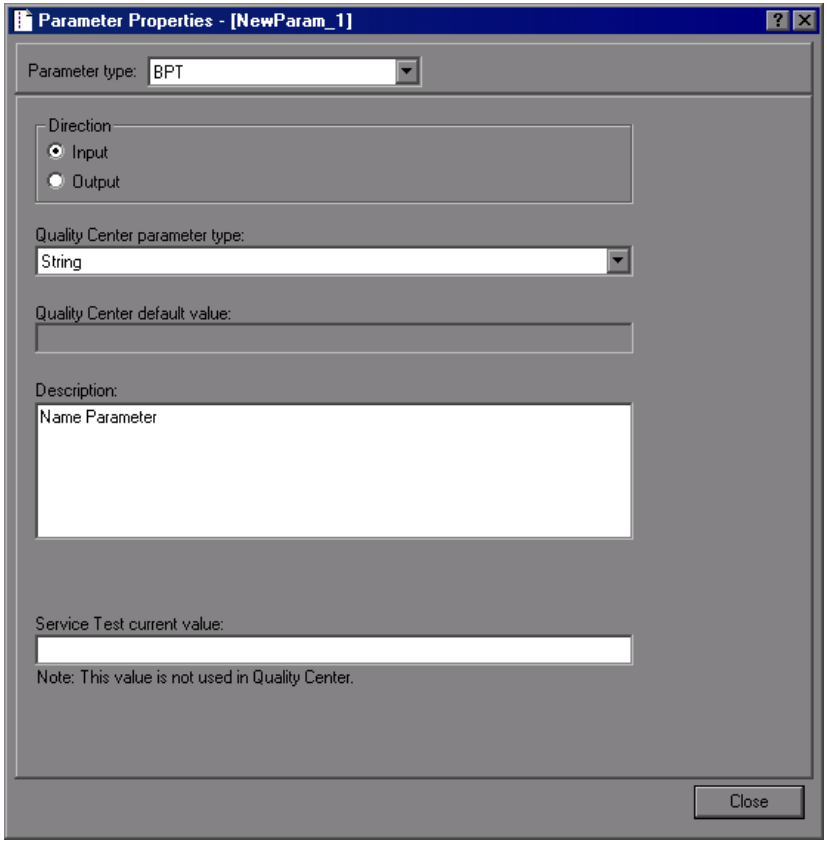
  To track this occurrence, enable the **Extended Log** > **Parameter Substitution** option in the Log Run-Time settings. When there is not enough data, VuGen writes a warning message to the Vuser log "No more unique values for this parameter in table <*table_name*>".

Setting Properties for BPT Type Parameters

You set BPT type parameters to allow you to share values between business components in Quality Center. You can set the following values:

- **Direction.** Input or Output. Select **Input** to use the parameter as an input parameter for the current component. Select **Output** to assign the current component's parameter as an output value.
- **Quality Center parameter type.** Select a data type for which the parameter will be used in Quality Center: String, Boolean, Date, Number, or Password.
- **Description.** A textual description of the parameter.
- **Quality Center default value.** (Input only) The default value of the parameter in Quality Center, if it had already been created in Quality Center.

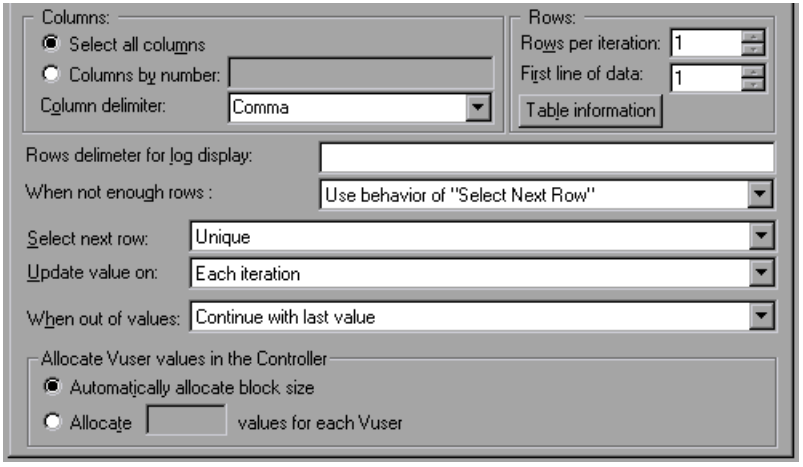
➤ **Service Test current value.** The current value of the parameter in Service Test. This value is not used by Quality Center.



Specify the desired values and click **Close**. When you save your script, the parameter is saved in Quality Center, provided that you have an open connection to the server.

Setting Properties for Table Type Parameters

After you select a table of data, you set its assignment properties. These properties instruct VuGen how to use the table data. For example, they indicate which columns and rows to use, how often to use them, and what to do when there are no more unique values.



To set the Table parameter properties:

1 Specify the columns in the table that contains the values for your parameter. In the **Columns** section, specify which columns you want to use. Alternatively, you can select **Select all columns.**

To specify one or more columns by their number, select **Columns by number** and enter the column numbers separated by a comma or a dash. The column number is the index of the column containing your data. For example, if the data for the parameter is in the table's first column, select 1.

2 In the **Column delimiter** box, select a column delimiter—the character used to separate the columns in the table. The available delimiters are: comma, tab, space.

3 In the **Rows** section, specify how many rows to use per iteration in the **Rows per iteration** box.

Note: This only relevant when the **Update value on** field is set to **Each iteration**. If **Update value on** is set to **Once**, then the same rows will be used for all iterations.

4 In the **First line of data** box, select the first line of data to be used during script execution. To begin with the first line after the header, enter 1. To display information about the table, including how many rows of data are available, click **Table information**.

5 Specify a row delimiter for your data presentation in the **Rows delimiter for log display** box. This delimiter is used to differentiate between rows in the output logs. If you enable parameter substitution logging, VuGen sends the substituted values to the Replay log. The row delimiter character in the Replay log indicates a new row.

6 In the **When not enough rows** box, specify a handling method when there are not enough rows in the table for the iteration. For example, assume that the table you want to fill has 3 rows, but your data only has two rows. Select **Parameter will get less rows than required** to fill in only two rows. Select **Use behavior of "Select Next Row"** to loop around and get the next row according the method specified in the **Select next row** box—**Random** or **Sequential**.

7 Select a Data Assignment method from the **Select next row** list to instruct the Vuser how to select the table data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information, see "Choosing Data Assignment Methods for File/Table Parameters" on page 450.

8 Select an Update method from the **Update value on** list. The options are **Each Iteration** or **Once**. For more information, see "Data Assignment and Update Methods for File/Table/ XML Parameters" on page 452.

9 If you chose to assign data using the **Unique** method:

- **When out of values.** Specify how to proceed when there is no more unique data: **Abort the Vuser**, **Continue in a cyclic manner**, or **Continue with last value**.
- **Allocate Vuser values in the Controller** (for LoadRunner users only). Indicate whether you want to manually allocate data blocks for the Vusers. You can allow the Controller to automatically allocate a block size or you can specify the desired number of values. Select **Automatically allocate block size** or **Allocate x values for each Vuser.** For the second option, specify the number of values to allocate.

  To track this occurrence, enable the **Extended Log** > **Parameter Substitution** option in the Log Run-Time settings. When there is not enough data, VuGen writes a warning message to the Vuser log "No more unique values for this parameter in table <*table_name*>".

Choosing Data Assignment Methods for File/Table Parameters

When using values from a file, VuGen lets you specify the way in which you assign data from the source to the parameters. The following methods are available:

- Sequential
- Random
- Unique

Sequential

The **Sequential** method assigns data to a Vuser sequentially. As a running Vuser accesses the data table, it takes the next available row of data.

If there are not enough values in the data table, VuGen returns to the first value in the table, continuing in a loop until the end of the test.

Random

The **Random** method assigns a random value from the data table to each Vuser at the start of the test run.

When running a scenario or Business Process Monitor profile, you can specify a seed number for random sequencing. Each seed value represents one sequence of random values used for test execution. Whenever you use this seed value, the same sequence of values is assigned to the Vusers in the scenario. You enable this option if you discover a problem in the test execution and want to repeat the test using the same sequence of random values.

For more information see the *HP LoadRunner Controller*, *HP Performance Center*, or *HP Business Availibility Center User Guides*.

Unique

The **Unique** method assigns a unique sequential value to the parameter for each Vuser.

In this case you must make sure there is enough data in the table for all the Vusers and their iterations. If you have 20 Vusers and you want to perform 5 iterations, your table must contain at least 100 unique values.

If there are not enough values in the data table, you can instruct VuGen how to proceed. For more details, see "Setting Properties for File Type Parameters" on page 444, or "Setting Properties for Table Type Parameters" on page 448.

Note: For LoadRunner users: If a script uses Unique file parameterization, running more than one Vuser group with that script in the same scenario may cause unexpected scenario results. For more information about Vuser groups in scenarios, see the *HP LoadRunner Controller User's Guide*.

Data Assignment and Update Methods for File/Table/ XML Parameters

For File, Table, and XML type parameters, the Data Assignment method that you select, together with your choice of Update method, affect the values that the Vusers use to substitute parameters during the scenario run.

The following table summarizes the values that Vusers use depending on which Data Assignment and Update properties you selected:

Update Method	Data Assignment Method		
	Sequential	**Random**	**Unique**
Each iteration	The Vuser takes the *next* value from the data table for each iteration.	The Vuser takes a *new random* value from the data table for each iteration.	The Vuser takes a value from the next unique position in the data table for each iteration.
Each occurrence (Data Files only)	The Vuser takes the *next* value from the data table for each occurrence of the parameter, even if it is within the same iteration.	The Vuser takes a *new random* value from the data table for each occurrence of the parameter, even if it is within the same iteration.	The Vuser takes a *new unique* value from the data table for each occurrence of the parameter, even if it is within the same iteration.
Once	The value assigned in the first iteration is used for all subsequent iterations for each Vuser.	The random value assigned in the first iteration is used for all iterations of that Vuser.	The unique value assigned in the first iteration is used for all subsequent iterations of the Vuser.

Examples

Assume that your table/file has the following values:

Kim; David; Michael; Jane; Ron; Alice; Ken; Julie; Fred

Sequential Method

- If you specify update on **Each iteration**, all the Vusers use Kim in the first iteration, David in the second iteration, Michael in the third iteration, and so on.
- If you specify update on **Each occurrence**, all the Vusers use Kim in the first occurrence, David in the second occurrence, Michael in the third occurrence, and so on.
- If you specify update **Once**, all Vusers take Kim for all iterations.

Note: If you select the **Sequential** method and there are not enough values in the data table, VuGen returns to the first value in the table, continuing in a loop until the end of the test.

Random Method

- If you specify update on **Each iteration**, the Vusers use random values from the table for each iteration.
- If you specify update on **Each occurrence**, the Vusers use random values for each occurrence of the parameter.
- If you specify update **Once**, all Vusers take the first randomly assigned value for all the iterations.

Unique Method

- If you specify update on **Each iteration**, for a test run of 3 iterations, the first Vuser takes Kim in the first iteration, David in the second, and Michael in the third. The second Vuser takes Jane, Ron, and Alice. The third Vuser, Ken, Julie, and Fred.
- If you specify update on **Each occurrence**, then the Vuser uses a unique value from the list for each occurrence of the parameter.

➤ If you specify update **Once**, the first Vuser takes Kim for all iterations, the second Vuser takes David for all iterations, and so on.

Vuser Behavior in the Controller (LoadRunner Only)

When you set up a scenario to run a parameterized script, you can instruct the Vusers how to act when there are not enough values. The following table summarizes the results of a scenario using the following parameter settings:

➤ Select next row = **Unique**

➤ Update Value on = **Each iteration**

➤ When out of values = **Continue with last value**

Situation	Duration	Resulting Action
More iterations than values	**Run until completion**	When the unique values are finished, each Vuser continues with the last value, but a warning message is sent to the log indicating that the values are no longer unique.
More Vusers than values	**Run indefinitely or Run for …**	Vusers take all of the unique values until they are finished. Then the test issues an error message **Error: Insufficient records for param <param_name> in table to provide the Vuser with unique data**. To avoid this, change the **When out of values** option in the Parameter properties or the **Select next row** method in the Parameter properties.
One of two parameters are out of values	**Run indefinitely or Run for …**	The parameter that ran out of values, continues in a cyclic manner until the values of the second parameter are no longer unique.

Setting Properties for XML Parameters

When you create a Web Service call to emulate a specific operation, the arguments in the operation may include complex structures with many values. You can use an XML type parameter to replace the entire structure with a single parameter.

You can create several value sets for the XML elements and assign a different value set for each iteration.

The XML parameter type supports complex schema types such as arrays. Choice, and <any> elements.

This section describes:

- Creating New XML Parameters
- Defining Value Sets
- Setting an Assignment Method
- Modifying XML Parameter Properties

Creating New XML Parameters

When working with Web Service **Input Arguments**, you may encounter arrays and their sub-elements. You can define a single XML parameter that will contain values for all of the array elements.

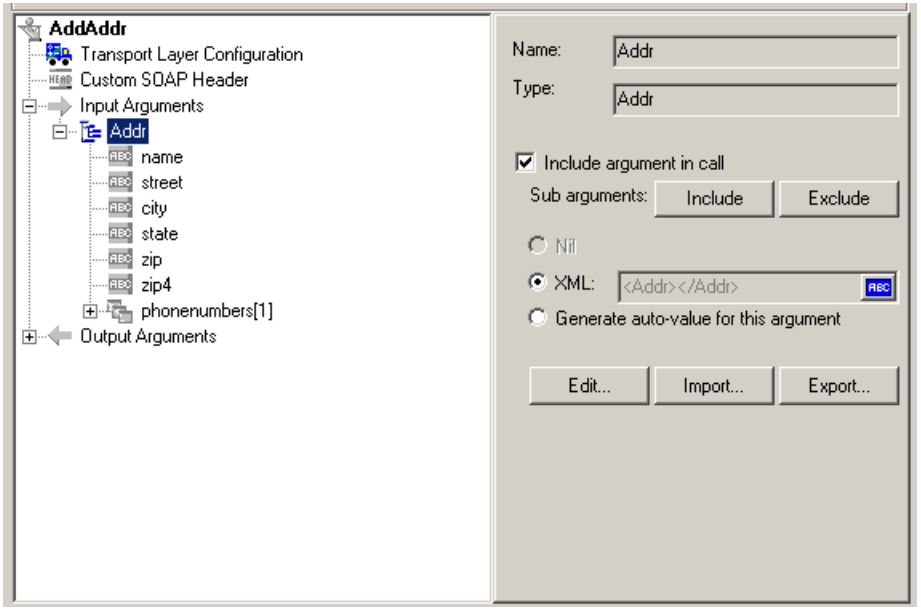
You can create new XML type parameters directly from the **Insert** menu, similar to all other parameter types. For Web Services type scripts, you create an XML parameter directly from the Web Services Call properties.

Creating XML Parameters From a Web Service Call

This section describes how to create an XML parameter from the Web Service properties.

To create an XML parameter from the Web Service call properties:

1 Select the root element of the complex data structure. The right pane displays the argument's details.



2 Select **XML** in the right pane, and click the **ABC** icon. The Select or Create Parameter dialog box opens.

3 In the **Parameter name** box, enter a name for the parameter.

4 In the **Parameter type** box, select **XML** if it is not already selected.

5 Click **Properties** to assign a value set now, or **OK** to close the dialog box and assign values later.

Creating XML Parameters - Standard Method

This section describes how to create an XML type parameter without viewing the properties of a Web Service call. This is the most common way of parameterizing values for most protocols and parameter types.

For Web Service Scripts, we recommend that you create parameters from within a Web Service Call, as described above.

To create a new XML parameter:

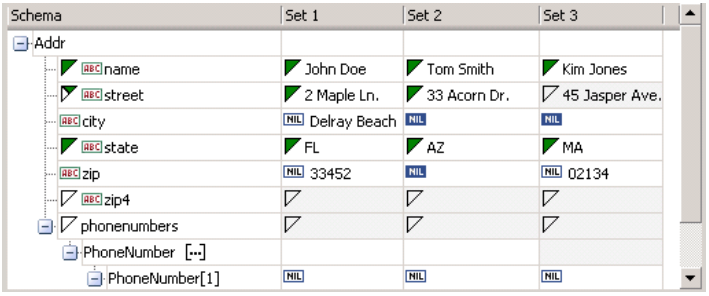
1. Select **Insert** > **New Parameter** or select a constant value in the Script view and select **Replace with a Parameter** from the right-click menu. The Select or Create Parameter dialog box opens.
2. In the **Parameter name** box, enter a name for the parameter.
3. In the **Parameter type** box, select **XML** if it is not already selected.
4. Click **Properties** to assign a value set now, or **OK** to close the dialog box and assign values later.

For information on how to set the properties, see "Setting Properties for XML Parameters" on page 455.

Defining Value Sets

This section describes how to create value sets for XML parameters.

Value sets are arrays that contain a set of values. Using the **Add Column** and **Duplicate Column** buttons, you can create multiple value sets for your parameter and use them for different iterations.

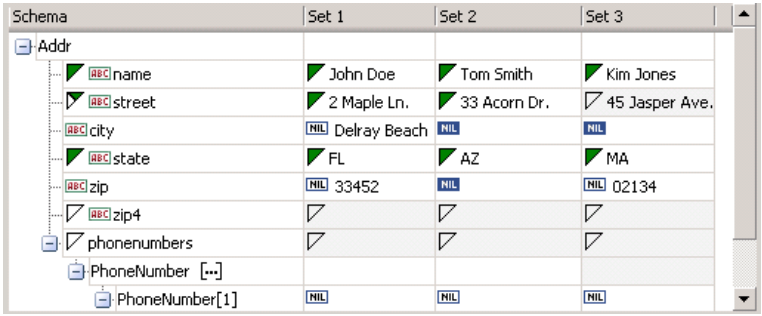


When using value sets, the number of array elements per parameter does not have to be constant.

You can use optional elements that will appear in one value set, but not in another. This allows you to vary the values you send for each of the iterations—some iterations can include specific array elements, while other iterations exclude them.

To exclude an optional element, click the small triangle in the upper left corner of the cell and insure that it is not filled in.

In the following example, **Set 1** and **Set 2** use the optional elements: **name**, **street**, and **state**. **Set 3** does not use a street name.

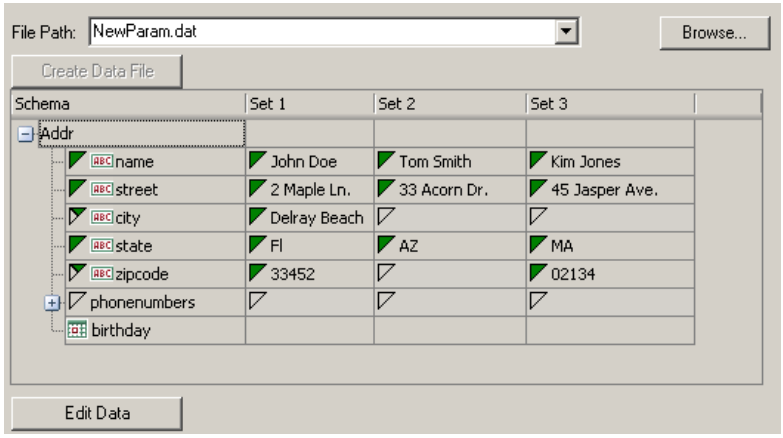


For more information about editing the values, see "XML Editing" on page 163.

To set parameter element values:

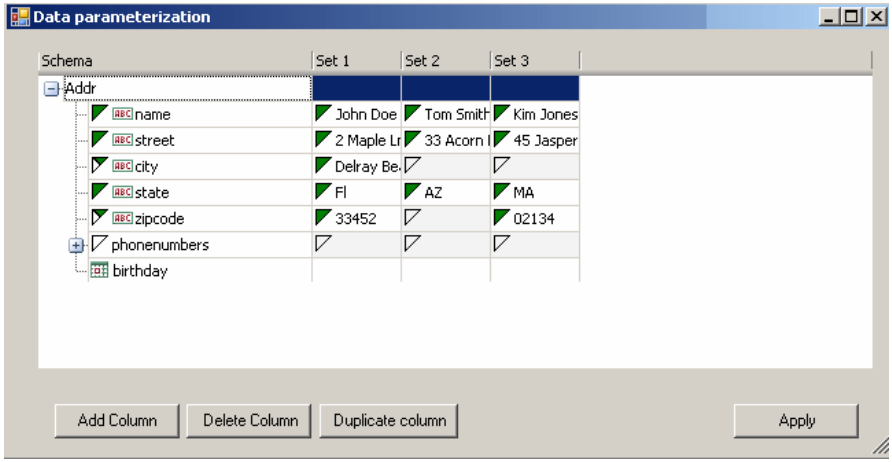
1 View the Parameter Properties.

If the Parameter Properties dialog box is not open, select **Vuser** > **Parameter List** and select the desired parameter. The dialog box shows a read-only view of the parameter values.



2 Open the Data Parameterization box.

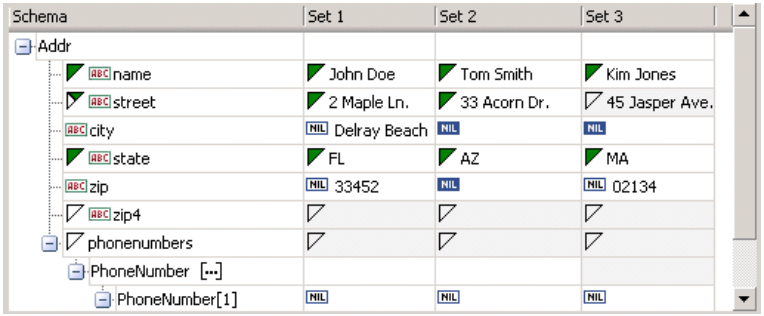
Click the **Edit Data** button to open the Data Parameterization dialog box.



3 Define value sets for the XML parameter.

In the **Set** columns, insert values corresponding to the schema.

If a row says **NIL**, it implies that the element is nillable. To include a value for the nillable element, enter the value as usual. To mark a value as **nil**, click the NIL icon to fill it in. This erases any value that you may have assigned to the element. In the following example, the **city** element is nillable, but it is only marked as nil in **Set 2** and **Set 3**—not in **Set 1**.

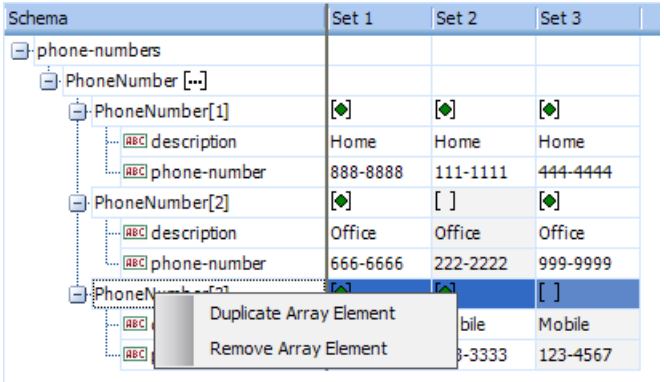


4 Create additional value sets.

To insert more value sets, click **Add Column** and insert another set of values in the new column. To copy an existing value set, select a row in the value set you want to copy and click **Duplicate Column**.

5 Copy arrays.

To duplicate an array element and its children, select the parent node and choose **Duplicate Array Element** from the right-click menu.

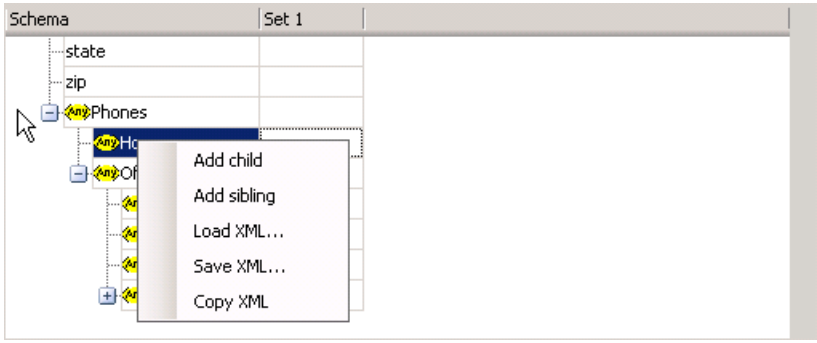


6 Handle the <any> elements.

For **any** type elements, right-click **<any>** in the **Schema** column and select one of the available options. These options may vary depending on the location of the cursor.

- **Add Array Element.** Adds a sub-element under the root element.
- **Insert child.** Adds a sub-element to the selected element.
- **Insert sibling.** Adds a sub-element on the same level as the selected element.
- **Load XML.** Loads the element values from an XML file.
- **Save XML.** Saves the array as an XML file.
- **Copy XML**. Copies the full XML of the selected element to the clipboard.

Click the **Rename** text to provide a meaningful name for each array element.



7 Remove unwanted columns.

To remove a value set, select it and click **Delete Column**.

8 Save the changes.

Click **Apply** to save the changes and update the view in the Parameter Properties dialog box.

Setting an Assignment Method

The assignment method indicates which of the value sets to use and how to use them. For example, you can instruct Vusers to use a new value set for each iteration and use the value sets sequentially or at random. For more information, see "Data Assignment and Update Methods for File/Table/ XML Parameters" on page 452.

To define an assignment method:

1 Open the Parameter Properties and select a parameter.

2 Define a data assignment method.

In the **Select next value** list, select a data assignment method to instruct the Vuser how to select the file data during Vuser script execution. The options are: **Sequential**, **Random**, or **Unique**. For more information, see "Choosing Data Assignment Methods for File/Table Parameters" on page 450.

3 Select an update option for the parameter.

In the **Update value on** list, select an update option. The choices are **Each Iteration**, **Each Occurrence**, and **Once**. For more information, see "Data Assignment and Update Methods for File/Table/ XML Parameters" on page 452.

4 If you chose **Unique** as the data assignment method the **When out of values** and **Allocate Vuser values in the Controller** options become enabled.

➤ **When out of values.** Specify what to do when there is no more unique data: **Abort Vuser**, **Continue in a cyclic manne***r*, or **Continue with last value**.

➤ **Allocate Vuser values in the Controller (**for LoadRunner users only)**.** Indicate whether you want to manually allocate data blocks for the Vusers. You can allow the Controller to automatically allocate a block size or you can specify the desired number of values. Select **Automatically allocate block size** or **Allocate x values for each Vuser.** For the second option, specify the number of values to allocate.

To track this occurrence, enable the **Extended Log** > **Parameter Substitution** option in the Log Run-Time settings. When there is not enough data, Service Test writes a warning message to the Vuser log: **No more unique values for this parameter in table <table_name>**.

5 In the Parameter Properties dialog box, click **Close**.

The list of input arguments is replaced by the parameter name, and ABC button is replace by a table icon which you can click to edit the parameter properties or un-parameterize the parameter.

Modifying XML Parameter Properties

If you need modify a value set of a parameter, you can do so from the Web Service's Step Properties tab.

To modify XML parameter properties:

1 In the Web Service script's tree view, click the **Step Properties** tab.

2 Under **Input Arguments**, select the XML parameter. The right pane displays the parameter details.

3 To modify the XML parameter properties, click the table icon button adjacent to the **XML** box and select **Parameter Properties**.

4 Modify the parameter properties as described in "Defining Value Sets" on page 457.

22

Setting Parameter Properties

A parameter is defined according to the type of information it replaces.

This chapter includes:

About Setting Parameter Properties

When you define a parameter's properties, you specify the source for the parameter data.

You define properties for any one of the following data source types:

Data type	Data Source
Internal Data Parameter Types	Data that is generated internally by the Vuser: Date/Time, Group Name, Iteration Number, Load Generator Name, Random Number, Unique Number, and Vuser ID.

User-Defined Functions	Data that is generated using a function from an external DLL.
Files and Tables	Data that is contained in a file—either an existing file or one that you create with VuGen or MS Query.

This chapter describes how to assign properties to Internal Data and User-Defined Function parameters.

For information on defining properties for Table or File type parameters, see Chapter 21, "File, Table, BPT, and XML Parameter Types."

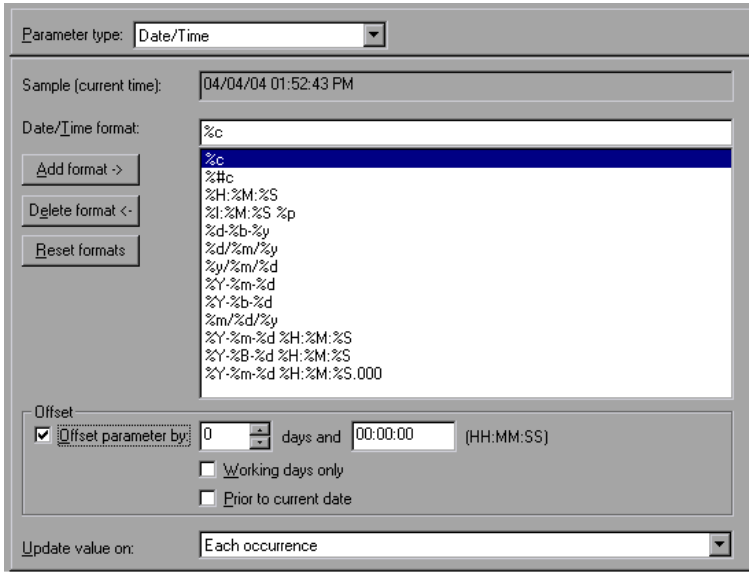
Setting Properties for Internal Data Parameter Types

This section discusses setting the properties for data that is generated internally by the Vuser. Internal data includes data such as:

- Date/Time
- Group Name
- Iteration Number
- Load Generator Name
- Random Number
- Unique Number
- Vuser ID

Date/Time

Date/Time replaces the parameter with the current date and/or time. To specify a date/time format, you can select a format from the format list or specify your own format. The format should correspond to the date/time format recorded in your script.



VuGen lets you set an offset for the date/time parameter. For example, if you want to test a date next month, you set the date offset to 30 days. If you want to test your application for a future time, you specify a time offset. You can specify a forward, future offset (default) or a backward offset, a date or time that already passed. In addition, you can instruct VuGen to use date values for work days only, excluding Saturdays and Sundays.

The following table describes the date/time symbols:

Symbol	Description
c	complete date and time in digits
#c	complete date as a string and time

Symbol	Description
H	hours (24 hour clock)
I	hours (12 hour clock)
M	minutes
S	seconds
p	AM or PM
d	day
m	month in digits (01-12)
b	month as a string - short format (e.g. Dec)
B	month as a string - long format (e.g. December)
y	year in short format (e.g. 03)
Y	year in long format (e.g. 2003)

To set the properties for Date/Time parameters:

1 Select one of the existing date/time formats or create a new format. You can view a sample of how VuGen will display the value, in the **Sample (Current time)** box. For information on customizing parameter formats, see "Customizing Parameter Formats" on page 477.

2 To set the date and time offsets, select **Offset Parameter by** and specify the desired offset for the date and time values.

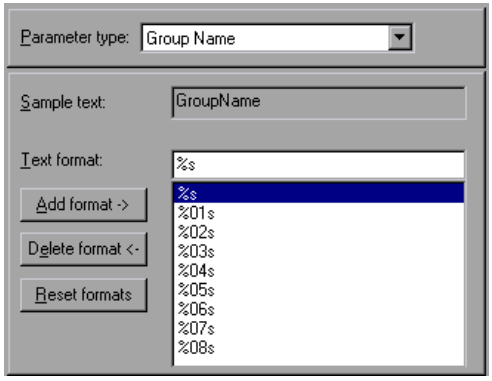
To instruct VuGen to use working day dates only, excluding weekends, select **Working days only**. To indicate a negative offset to test a date prior to the current, select **Prior to current date**.

3 Select an update method, instructing the Vuser when to update parameter values—**Each occurrence**, **Each iteration**, or **Once**. For more information, see "Selecting an Update Method" on page 478.

4 Click **Close** to accept the settings and close the Parameter Properties dialog box.

Group Name

Group Name replaces the parameter with the name of the Vuser Group. You specify the name of the Vuser Group when you create a scenario. When you run a script from VuGen, the Group name is always *None*.

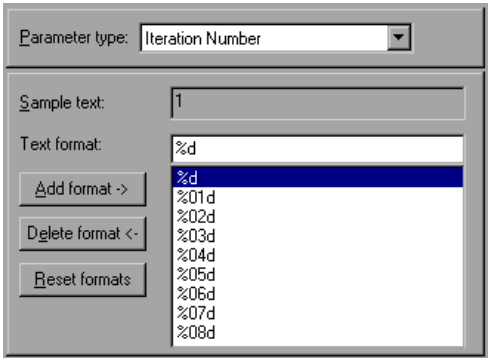


To set properties for the Group Name parameter type:

1. Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see "Customizing Parameter Formats" on page 477.
2. Click **Close** to accept the settings and close the Parameter Properties dialog box.

Iteration Number

Iteration Number replaces the parameter with the current iteration number.

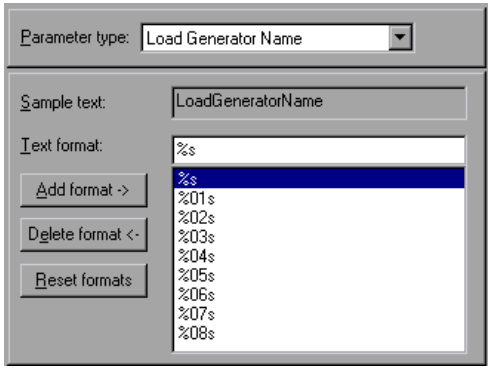


To set the properties for the Iteration Number parameter type:

1. Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see "Customizing Parameter Formats" on page 477.
2. Click **Close** to save the settings and close the Parameter Properties dialog box.

Load Generator Name

Load Generator Name replaces the parameter with the name of the Vuser script's load generator. The load generator is the computer on which the Vuser is running.



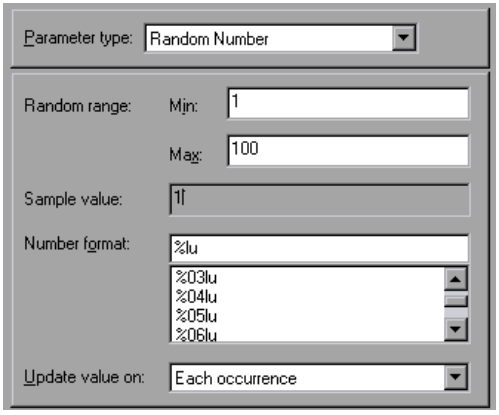
To set the properties for the Load Generator Name parameter type:

1. Select one of the available formats or create a new one. You select a format to specify the length of the parameter string. For details, see "Customizing Parameter Formats" on page 477.
2. Click **Close** to save the settings and close the Parameter Properties dialog box.

Random Number

Random Number replaces the parameter with a random number. You set a range of numbers by specifying minimum and maximum values.

You can use the Random Number parameter type to sample your system's behavior within a possible range of values. For example, to run a query for 50 employees, where employee ID numbers range from 1 through 1000, create 50 Vusers and set the minimum to 1 and maximum to 1000. Each Vuser receives a random number, from within the range of 1 to 1000.



To set the properties for the Random Number parameter type:

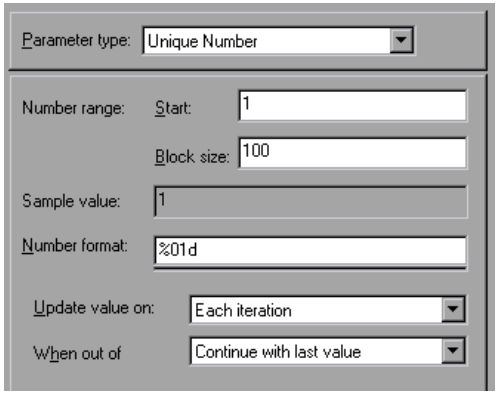
1. Enter a range defining the set of possible parameter values. You specify minimum and maximum values for the range of random numbers.
2. Select a **Number format**, indicating the length of the random number. Specify **%01lu** (or **%lu**) for one digit, **%02lu** for two digits, and so on. You can view a sample of how VuGen will display the value, in the **Sample value** box.
3. Select an update method, instructing the Vuser when to update parameter values—**Each occurrence**, **Each iteration**, or **Once**. For more information, see "Selecting an Update Method" on page 478.
4. Click **Close** to accept the settings and close the Parameter Properties dialog box.

Unique Number

Unique Number replaces the parameter with a unique number.

When you create a Unique Number type parameter, you specify a start number and a block size. The block size indicates the size of the block of numbers assigned to each Vuser. Each Vuser begins at the bottom of its range and increments the parameter value for each iteration. For example, if you set the Start number at 1 with a block of 500, the first Vuser uses the value 1 and the next Vuser uses the value 501, in their first iterations.

The number of digits in the unique number string together with the block size determine the number of iterations and Vusers. For example, if you are limited to five digits using a block size of 500, only 100,000 numbers (0-99,999) are available. It is therefore possible to run only 200 Vusers, with each Vuser running 500 iterations.



You can also indicate what action to take when there are no more unique numbers in the block: **Abort Vuser**, **Continue in a cyclic manner**, or **Continue with last value** (default).

You can use the Unique Number parameter type to check your system's behavior for all possible values of the parameter. For example, to perform a query for all employees, whose ID numbers range from 100 through 199, create 100 Vusers and set the start number to 100 and block size to 100. Each Vuser receives a unique number, beginning with 100 and ending with 199.

Note: VuGen creates only one instance of Unique Number type parameters. If you define multiple parameters and assign them the Unique Number Parameter type, the values will not overlap. For example, if you define two parameters with blocks of 100 for 5 iterations, the Vusers in the first group will use 1, 101, 201, 301, and 401. The Vusers in the next group, using the second parameter will use 501, 601, 701, 801, and 901.

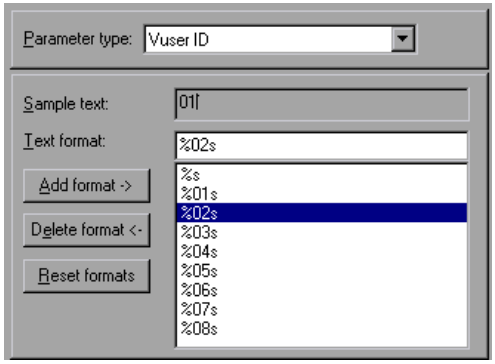
After you define a Unique parameter, you can use it in other scripts by pointing to the same parameter file. The parameter retains all of the properties that you assigned to the original parameter.

To set the properties for the Unique Number parameter type:

1. Enter a start number and the desired block size. For example, if you want 500 numbers beginning with 1, specify 1 in the **Start** box, and 500 in the **Block size per Vuser** box.
2. Select a Number format, indicating the length of the unique number. Specify **%01d** (or **%d**) for one digit, **%02d** for two digits, and so on. You can view a sample of how VuGen will display the value, in the **Sample value** box.
3. Select an update method, instructing the Vuser when to update parameter values—**Each occurrence**, **Each iteration**, or **Once**. For more information, see "Selecting an Update Method" on page 478.
4. Indicate what to do when there are no more unique values, in the **When out of values** box: **Abort Vuser**, **Continue in cyclic manner**, or **Continue with last value**.
5. Click **Close** to accept the settings and close the Parameter Properties dialog box.

Vuser ID

Vuser ID replaces the parameter with the ID number assigned to the Vuser by the Controller during a scenario run. When you run a script from VuGen, the Vuser ID is always -1.



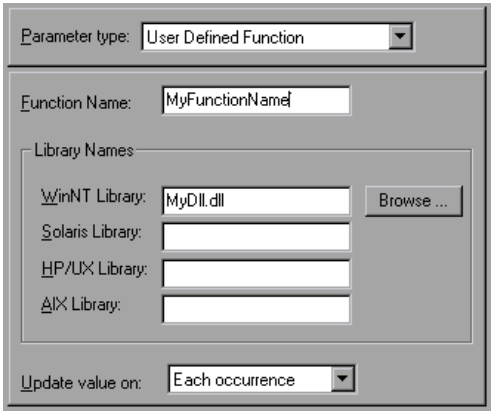
Note: This is not the ID number that appears in the Vuser window—it is a unique ID number generated at runtime.

To set the properties for the Vuser ID parameter type:

1. Select one of the available formats or create a new one. You select a format to specify the length and structure of the parameter string. For details, see "Customizing Parameter Formats" on page 477.
2. Click **Close** to accept the settings and close the Parameter Properties dialog box.

Setting Properties for User-Defined Functions

In the Parameter Properties dialog box, select **User Defined Function** from the **Parameter type** list.



To set the properties for user-defined functions:

1. Specify the function name in the **Function Name** box. Use the name of the function as it appears in the DLL file.
2. In the **Library Names** section, specify a library in the relevant **Library** box. If necessary, locate the file using the **Browse** command.
3. Select an update method for the values. For more information on update methods for user-defined functions, see "Selecting an Update Method" on page 478.

Customizing Parameter Formats

For most data types, you can customize a format for a parameter by selecting an existing format or specifying a new one.

Note: The parameter format should match the recorded values. If the format of the parameter differs from the format of the original recorded value, the script may not run correctly.

The format specifies the length and structure of the resulting parameter string. The resulting parameter string is the actual parameter value together with any text that accompanies the parameter. For example, if you specify a format of "%05s," a Vuser ID of 5 is displayed as "00005," padding the single digit with four zeros. To pad the number with blank spaces, specify the number of spaces without a "0." For example, %4s adds blank spaces before the Vuser ID so that the resulting parameter string is 4 characters long.

You can specify a text string before and after the actual parameter value.

For example, if you specify a format of "Vuser No: %03s," then a Vuser ID of 1 is displayed as "**Vuser No: 001**."

You can add and delete formats for the following parameter types: Date/Time; Group Name; Iteration Number; Load Generator Name; Vuser ID.

To add a format to a parameter type:

1. In the Parameter Properties dialog box, select the parameter type that you want to format.
2. Enter the format symbols in the editable box and click **Add Format**.

   Note: When you add a format to the list, VuGen saves it with the Vuser, making it available for future use.

To delete a format:

In the Parameter Properties dialog box, select an existing format from the list, and click **Delete format**.

To restore the original formats:

Click **Reset formats**.

Selecting an Update Method

When using several of the parameter types, VuGen lets you specify how to update the values for the parameters. To set an Update method, select a method from the **Update value on** list. The available update methods are:

- Each Occurrence
- Each Iteration
- Once

Each Occurrence

The **Each occurrence** method instructs the Vuser to use a new value for each occurrence of the parameter. This is useful when the statements using a parameter are unrelated. For example, for random data, it may be useful to use a new value for each occurrence of the parameter.

Each Iteration

The **Each iteration** method instructs the Vuser to use a new value for each script iteration. If a parameter appears in a script several times, the Vuser uses the same value for all occurrences of the parameter, for the entire iteration. This is useful when the statements using a parameter are related.

Note: If you create an action block with parameters using its own iteration count—if you instruct VuGen to update their values each iteration, it refers to the global iteration and not the block iteration. For more information about action blocks, see "Creating Action Blocks" on page 51.

Once

The **Once** method instructs the Vuser to update the parameter value only once during the scenario run. The Vuser uses the same parameter value for all occurrences and all iterations of the parameter. This type may be useful when working with dates and times.

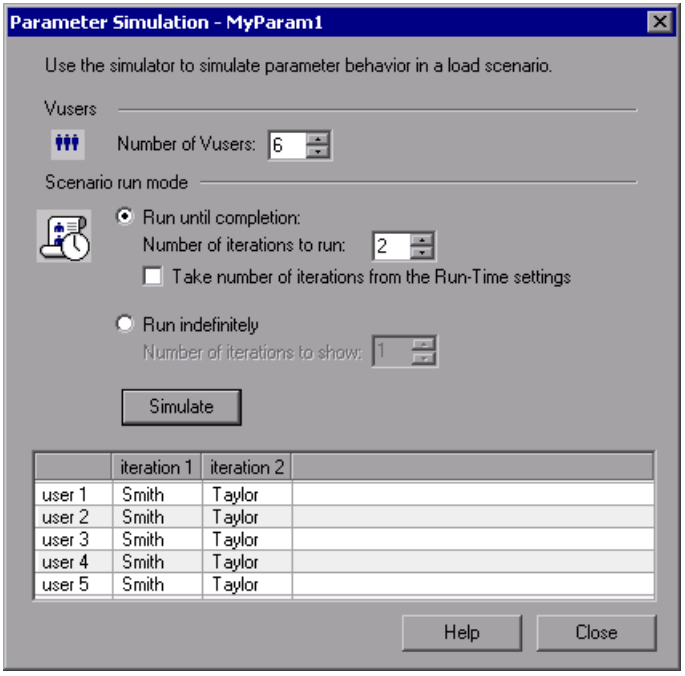
Simulating File Type Parameters

After you have created a File type parameter, you can use the File Parameter Simulator to simulate the parameter substitution in an actual scenario. This allows you to correct any wrong parameters before you run the script in the Controller. The File Parameter Simulator is only relevant for LoadRunner users.

Note: Not all types of Parameter Substitution can be simulated. If you select **Select next row: Same line as...** or **Update value on: Each occurrence**, then the File Parameter Simulator will not open.

To run a File Parameter Simulation:

1 From the Parameter List dialog box, click **Simulate Parameter**. The Parameter Simulation dialog box opens.



2 Select the number of Vusers to run in the simulation from the **Number of Vusers** box.

3 Select Scenario run mode:

- If you select **Run until completion**, select the number of iterations to run from the **Number of iterations to run** box, or take the number from the Run-Time settings.
- If you select **Run indefinitely**, you must define how many iterations to show. The number of iteration in the Run-Time settings is ignored. The number you select does not change the value distribution for each Vuser; it is only for viewing purposes.

Note:

- **Run Indefinitely** is compliant with the **Real-life schedule** in the Scheduler of the Controller.
- If you select **Select next row: Unique** in the Parameter List dialog, then each Vuser is assigned a unique range of rows from which the Simulator will substitute values (for that Vuser).

  With this setting, the default selection in the Allocate Vuser values in the Controller section is **Automatically allocate block size**. In this case, when you run the simulation, the range allocation takes place in accordance with your Scenario run mode selection.

  If you change the default selection to **Allocate x values for each Vuser,** then the Vusers will be allocated the amount of values you specify, ignoring of your Scenario run mode selection.

4 Click **Simulate**.

Note: The File Parameter Simulator can simulate up to 256 iterations and 256 Vusers.

Using the File Parameter Simulator

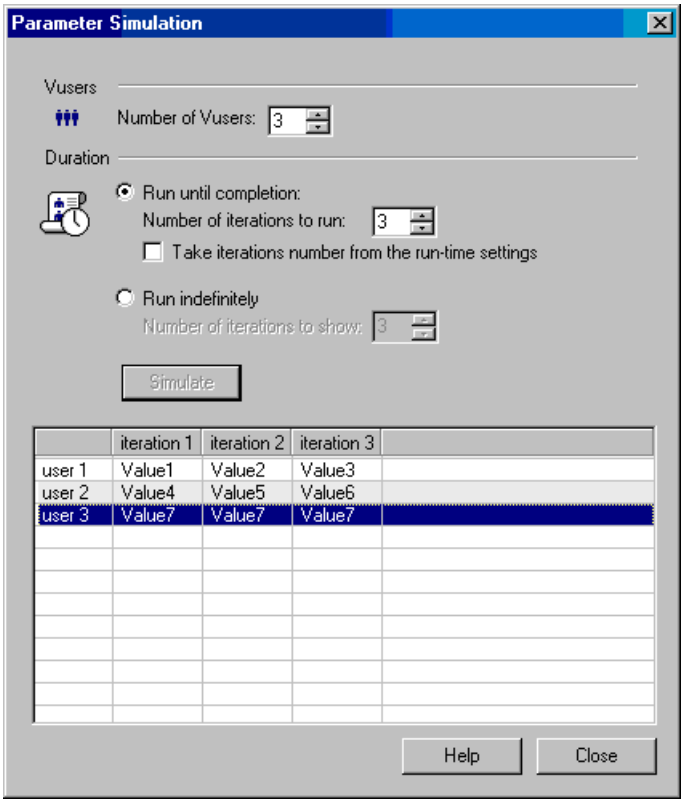
The following section illustrates how the File Parameter Simulator operates, demonstrating the difference between the Scenario run mode settings.

In the following examples, the settings in the Parameter List dialog box are:

- **Values for the new parameter**. Value1 to Value7
- **Select next row**. Unique
- **When out of rows**. Continue with last value
- **Allocate Vuser values in the Controller**. Automatically allocate block size

Scenario run mode: Run until completion

In the following example, the user has selected three Vusers, set the Scenario run mode to **Run until completion**, and selected three iterations.



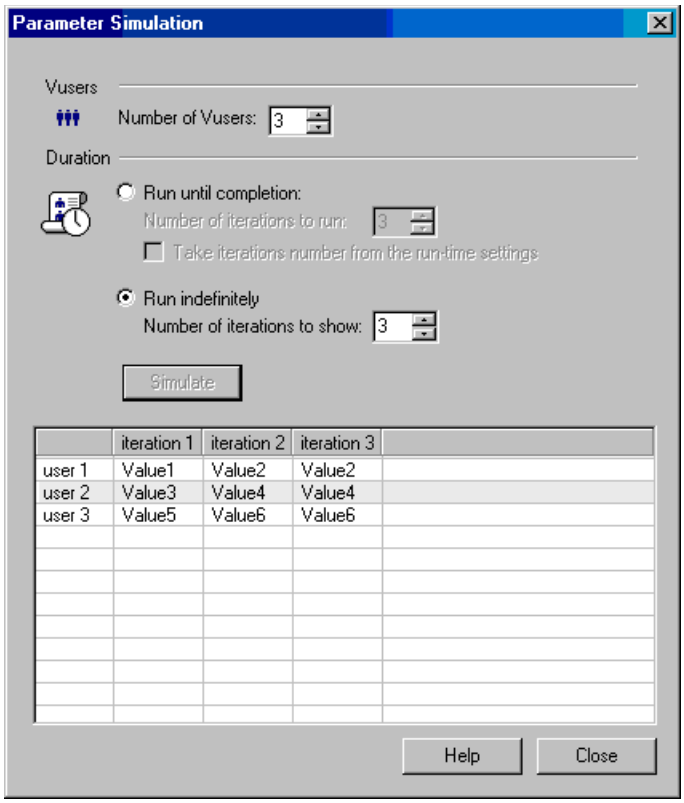
When the scenario run mode is set to **Run until completion**, the number of rows that each Vuser receives is the same as the number of iterations. The range allocation stops when there are no longer enough rows in the table.

As the simulation is run, the first Vuser takes the first three values (because this was the number of iterations). The second Vuser takes the next three values. The third Vuser takes the remaining value in the first iteration. For the remaining iterations, since the **When out of values** option in the Parameter List dialog box was set to **Continue with last value,** the third Vuser continues with the same value.

A fourth Vuser would have failed.

Scenario run mode: Run indefinitely

In the following example, the user has selected 3 Vusers and set the Scenario run mode to Run indefinitely and selected to show 3 iterations.



When the Scenario run mode is set to Run indefinitely, the allocated range for each Vuser is calculated by dividing the number of cells in the .dat file by the number of Vusers. In this scenario, that is 7/3 = 2 (The simulator takes the closest smaller integer.).

As the simulation is run, the first Vuser takes Value1 and Value2. The second Vuser takes Value3 and Value4 and the third Vuser takes Value5 and Value6. Since there are were only 3 Vusers, Value7 was not distributed.

Note: If you hold the mouse over the cells in the first column of the table, a tool tip appears with information about which values were assigned to that Vuser.

If you hold the mouse over cells which were not assigned values, a tool tip appears with the reason no values were assigned.

A tool tip does not appear if a proper value was assigned.

Part V

Appendix

A

Using Keyboard Shortcuts

The following list describes the keyboard shortcuts available in the Virtual User Generator.

ALT+F8	Compares the Current Snapshots (Web Vusers only)
ALT+INS	Create New Step
CTRL+A	Select All
CTRL+C	Copy
CTRL+F	Find
CTRL+G	Go To Line
CTRL+H	Replace
CTRL+N	New
CTRL+O	Open
CTRL+P	Print
CTRL+S	Save
CTRL+V	Paste
CTRL+X	Cut
CTRL+Y	Redo
CTRL+Z	Undo
CTRL+F7	Recording Options
CTRL+F8	Scan for Correlations

CTRL+SHIFT+SPACE	Show Function Syntax (Intellisense)
CTRL+SPACE	Complete Wizard (completes the function name)
F1	Help
F3	FIND Next Downward
SHIFT+F3	Find Next Upward
F4	Run-Time Settings
F5	Run Vuser
F6	Move Between Panes
F7	Show EBCDIC Translation Dialog (for WinSocket data)
F9	Toggle Breakpoint
F10	Run Vuser Step by Step

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

T

U

V

W

X