Oracle® Functional Testing

OpenScript User's Guide

Release 12.2.0.1

E15488-10

November 2012

ORACLE®

Oracle Functional Testing OpenScript User's Guide Release 12.2.0.1

E15488-10

Copyright © 2009, 2012, Oracle and/or its affiliates. All rights reserved.

Primary Author: Rick Santos

Contributing Author: Theresa Bandy, Orlando Cabrero, Leo Cloutier, Matt Demeusy, Joe Fernandes, Dan Hynes, Rich Kuzsma.

Contributor:

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle America, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Oracle Enterprise Manager Application Testing Suite contains Classic IDE 3.2.2 with the OpenScript product and certain Equinox jar files from the Eclipse SDK (the "EPL Programs"). The authors and/or contributors to the EPL Programs disclaim (i) all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose and (ii) all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits. Any provision of any license provided by Oracle is offered by Oracle alone and not by any other party. The source code for the EPL Programs and a copy of the Eclipse Public License is available from Oracle at the following URL:
http://oss.oracle.com/projects/eclipse-member-downloads/.

Contents

Preface xix

Audience xix
Documentation Accessibility xxi
Related Documents xxi
Conventions xxi

1 Getting Started With OpenScript

1.1 OpenScript Features 1-1
1.2 Installing OpenScript 1-3
1.3 Starting the OpenScript Workbench 1-3
1.4 Overview of the OpenScript Main Window (Workbench) 1-4
1.4.1 Tester Perspective 1-4
1.4.2 Developer Perspective 1-4
1.4.3 OpenScript Menu Options 1-5
1.4.3.1 File 1-5
1.4.3.2 Edit 1-6
1.4.3.3 Search 1-7
1.4.3.4 Script 1-7
1.4.3.5 View 1-9
1.4.3.6 Run 1-9
1.4.3.7 Tools 1-10
1.4.3.8 Help 1-11
1.4.3.9 Navigate 1-11
1.4.3.10 Project 1-12
1.4.3.11 Window 1-12
1.4.4 OpenScript Tool Bar 1-13
1.4.5 Script View 1-15
1.4.5.1 Tree View 1-15
1.4.5.2 Java Code 1-15
1.4.5.3 Assets 1-15
1.4.5.4 Assets 1-16
1.4.6 Details View 1-16
1.4.7 Problems View 1-17
1.4.8 Properties View 1-17
1.4.9 Console View 1-17

1.4.10 Results View 1-17
1.4.11 Error Log View 1-18
1.4.12 Data Table View 1-18
1.4.13 Object Details View 1-19
1.4.14 Script Variables View 1-20
1.4.15 Treeview Breakpoint View 1-20
1.4.16 Navigator and Package Explorer Views 1-21
1.4.17 Debug View 1-21
1.4.18 Declaration View 1-21
1.4.19 Variables and Breakpoints Views 1-21
1.5 About Multi-User Execution 1-21
1.6 About Script Assets 1-22

2 Setting Preferences

2.1 Setting OpenScript Preferences 2-1
2.2 Correlation and Validation Category 2-1
2.2.1 Module Correlation Preferences 2-1
2.2.2 Add Library 2-2
2.2.3 Add/Edit Rule 2-2
2.3 General Category 2-2
2.3.1 General Preferences 2-3
2.3.2 Browser Preferences 2-4
2.3.3 Encryption Preferences 2-4
2.3.4 Keys Preferences 2-5
2.3.4.1 Default OpenScript Keybindings 2-6
2.3.5 Large Data Preferences 2-7
2.3.6 Repository Preferences 2-7
2.4 Playback Category 2-8
2.4.1 General Playback Preferences 2-8
2.4.1.1 General 2-8
2.4.1.2 Error Handling 2-9
2.4.1.3 Debug 2-9
2.4.1.4 System 2-10
2.4.2 Error Recovery Preferences 2-10
2.4.2.1 General 2-11
2.4.2.2 Flex Load Test (AMF) 2-11
2.4.2.3 Functional Test 2-11
2.4.2.4 HTTP 2-12
2.4.2.5 Oracle Forms Functional Test 2-12
2.4.2.6 Oracle Forms Load Test 2-12
2.4.2.7 Utilities 2-13
2.4.2.8 Web Functional Test 2-13
2.4.3 HTTP Preferences 2-13
2.4.3.1 Proxy 2-13
2.4.3.2 Compression 2-13
2.4.3.3 Headers 2-14
2.4.3.4 Connections 2-14

2.4.3.5 SSL ... 2-15
2.4.3.6 Download Manager ... 2-15
2.4.3.7 Caching ... 2-16
2.4.3.8 Miscellaneous ... 2-16
2.4.4 Oracle EBS/Forms Functional Test Preferences ... 2-17
2.4.4.1 Event Timeout ... 2-17
2.4.4.2 Miscellaneous ... 2-18
2.4.5 Oracle EBS/Forms Load Test Preferences ... 2-18
2.4.5.1 Connection ... 2-18
2.4.5.2 Miscellaneous ... 2-18
2.4.6 Shared Data Service Preferences ... 2-18
2.4.7 Web Functional Test Preferences ... 2-19
2.4.7.1 Object Timeout ... 2-19
2.4.7.2 Capture ... 2-19
2.4.7.3 Browser ... 2-19
2.4.7.4 Cache and Cookies ... 2-20
2.4.7.5 Object Identification ... 2-20
2.4.7.6 Miscellaneous ... 2-22
2.5 Record Category ... 2-22
2.5.1 General Preferences ... 2-22
2.5.2 HTTP Preferences ... 2-22
2.5.2.1 General ... 2-22
2.5.2.2 Proxy Settings ... 2-24
2.5.2.3 URL Filters ... 2-25
2.5.2.4 Certificates ... 2-25
2.5.2.5 Object Identification ... 2-25
2.5.3 Oracle ADF Functional Test Preferences ... 2-26
2.5.3.1 Object Identification ... 2-26
2.5.4 Oracle EBS/Forms Functional Test Preferences ... 2-27
2.5.4.1 General ... 2-27
2.5.4.2 Object Identification ... 2-27
2.5.4.3 Applet Object Identification ... 2-28
2.5.5 Oracle EBS/Forms Load Test Preferences ... 2-28
2.5.6 Oracle JDE EnterpriseOne Functional Test Preferences ... 2-29
2.5.6.1 Object Identification ... 2-29
2.5.7 Siebel Functional Test Preferences ... 2-30
2.5.7.1 General ... 2-30
2.5.8 Web Functional Test Preferences ... 2-31
2.5.8.1 General ... 2-31
2.5.8.2 Object Identification ... 2-32
2.5.9 Web Services Preferences ... 2-33
2.5.9.1 General ... 2-33
2.5.9.2 Parser Tools ... 2-33
2.5.9.3 Proxy Configuration ... 2-33
2.5.9.4 Certificates ... 2-33
2.6 Step Group Category ... 2-34
2.6.1 ADF Load Test Preferences ... 2-34

2.6.2 Basic Module Preferences ... 2-34
2.6.3 Flex (AMF) Load Test Preferences ... 2-35
2.6.4 HTTP Preferences ... 2-36
2.6.5 Oracle EBS/Forms Functional Test Preferences ... 2-36
2.6.6 Oracle EBS/Forms Load Test Preferences ... 2-37
2.6.7 Siebel Functional Test Preferences ... 2-38
2.6.8 Siebel Load Test Preferences ... 2-39
2.6.9 Web Functional Test Preferences ... 2-39
2.7 Setting Project Preferences ... 2-40

3 Creating and Modifying Scripts

3.1 Creating Repositories and Workspaces ... 3-1
3.1.1 Creating a Repository ... 3-2
3.1.2 Managing Repositories ... 3-3
3.1.3 Managing Folders (Workspaces) ... 3-3
3.1.4 Managing Scripts ... 3-3
3.2 Creating a Script Project ... 3-3
3.2.1 Recording Scripts ... 3-7
3.2.1.1 Recording Scripts from Another Machine ... 3-8
3.2.2 Opening Existing Scripts ... 3-9
3.2.2.1 Opening Older Scripts in OpenScript ... 3-9
3.2.2.2 Migrating Older Scripts in OpenScript ... 3-10
3.2.2.3 Running Mixed Versions of Scripts ... 3-10
3.2.2.4 Multiple Users Opening Scripts ... 3-10
3.2.3 Exporting and Importing Scripts ... 3-10
3.2.3.1 Exporting Scripts ... 3-10
3.2.3.2 Importing Scripts ... 3-12
3.2.4 Migrating Scripts ... 3-12
3.2.5 Creating New Scripts from Templates ... 3-13
3.2.6 Setting Script Properties ... 3-13
3.2.6.1 About ... 3-14
3.2.6.2 Correlation and Validation ... 3-14
3.2.6.3 Modules ... 3-14
3.2.6.4 Step Groups ... 3-14
3.2.7 Importing Database Capture Files ... 3-14
3.2.8 Importing Oracle Real User Experience Insight (RUEI) Session Logs ... 3-17
3.2.9 Exporting Script Playback Settings ... 3-17
3.3 Modifying Scripts ... 3-18
3.3.1 Adding Step Groups to a Script ... 3-18
3.3.2 Adding a Delay to a Script ... 3-19
3.3.3 Adding a Log Message to a Script ... 3-19
3.3.4 Adding a For Statement to a Script ... 3-19
3.3.5 Adding a Function to a Script ... 3-20
3.3.5.1 Adding Functions that Use Lists ... 3-23
3.3.5.2 Adding Functions that Use Maps ... 3-24
3.3.5.3 Adding Functions that use Enumerated Lists ... 3-25
3.3.5.4 Inputting Values from a File ... 3-25

3.3.6 Using a Script as a Dedicated Function Library ... 3-27
3.3.6.1 About Function Libraries ... 3-27
3.3.6.2 Creating a Dedicated Function Library Script ... 3-30
3.3.6.3 Calling Functions from a Function Library Script ... 3-31
3.3.7 Converting a Script to a Dedicated Function Library ... 3-32
3.3.8 Adding Script Assets ... 3-33
3.3.9 Adding a Script to Run from a Script ... 3-34
3.3.10 Adding a Synchronization Point to a Script ... 3-35
3.3.11 Adding a Set Variable to a Script ... 3-35
3.3.11.1 Variables with Scope ... 3-36
3.3.12 Removing Unchanging Variables ... 3-37
3.3.13 Parameterizing URLs ... 3-38
3.3.14 Adding Comments to Script Results ... 3-38
3.3.15 Adding Error Recovery to a Script ... 3-39
3.3.15.1 Script Types ... 3-39
3.3.15.2 Constants ... 3-40
3.3.15.3 Actions ... 3-40
3.3.16 Verifying Script Actions ... 3-41
3.3.16.1 Adding an Error Recovery Action ... 3-41
3.3.16.2 Adding a Has Error Control Statement ... 3-41
3.3.16.3 Adding a Result Object Message ... 3-42
3.3.16.4 Actions That Can Be Verified ... 3-42
3.3.17 Chaining Multiple Scripts ... 3-43
3.3.17.1 Setting the Browser Preferences ... 3-43
3.3.17.2 Recording Scripts ... 3-43
3.3.17.3 Creating a Shell Script ... 3-43
3.3.18 Moving Nodes in a Script ... 3-44
3.3.19 Aborting and Resuming a Script Programmatically ... 3-45
3.4 Changing Text File Encoding ... 3-46
3.5 Debugging Scripts ... 3-46
3.5.1 Adding Views to the Tester Perspective ... 3-46
3.5.2 Adding Breakpoints to a Script ... 3-47
3.5.3 Adding a Java Exception Breakpoint ... 3-49
3.5.4 Pausing and Resuming Script Playback in Debug Mode ... 3-49
3.5.5 Inspecting and Changing Script Variable Values ... 3-50
3.6 Enabling Debug Logging ... 3-51

4 Using Data Parameterization

4.1 Understanding Data Driven Testing (Parameterization) ... 4-1
4.2 Using Script Databanks ... 4-2
4.2.1 Configuring Databanks ... 4-3
4.2.2 Creating or Editing Databank Files ... 4-6
4.2.3 Getting Databank Records ... 4-7
4.2.3.1 Getting Databank Records Using the API ... 4-8
4.2.3.1.1 Databank API Usage Notes ... 4-8
4.2.3.1.2 Loading a Databank ... 4-8
4.2.3.1.3 Getting a Record Count ... 4-9

4.2.3.1.4 Getting a Specific Record 4-9
4.2.3.1.5 Getting the First Record 4-9
4.2.3.1.6 Getting the Last Record 4-9
4.2.4 Playing Back Scripts With Iterations 4-9
4.2.4.1 Notes and Limitations 4-12
4.2.4.2 Using Very Large Databanks 4-13
4.3 Using Data Tables 4-13
4.3.1 Enabling the Data Table Service 4-14
4.3.2 Setting the First Row Policy 4-14
4.3.3 Entering Data Manually 4-15
4.3.4 Importing Data from a Spreadsheet File 4-16
4.3.5 Exporting Data to a Spreadsheet File 4-17
4.3.6 Changing Data During Script Playback 4-17
4.3.6.1 Getting and Setting Cell Values 4-17
4.3.6.1.1 Getting Data by Row and Column Value 4-17
4.3.6.1.2 Getting Data by Sheet, Row, and Column Value 4-17
4.3.6.1.3 Setting Data by Row and Column Value 4-18
4.3.6.1.4 Setting Data by Sheet, Row, and Column Value 4-18
4.3.6.2 Adding and Deleting Rows and Columns 4-18
4.3.6.2.1 Adding Columns 4-18
4.3.6.2.2 Deleting Columns 4-19
4.3.6.2.3 Adding Rows 4-19
4.3.6.2.4 Deleting Rows 4-19
4.3.6.3 Adding and Deleting Worksheets 4-19
4.3.6.3.1 Adding Worksheets 4-19
4.3.6.3.2 Deleting Worksheets 4-19
4.3.6.4 Getting Worksheet, Row, and Column Counts 4-19
4.3.6.4.1 Getting Worksheet Counts 4-20
4.3.6.4.2 Getting Row Counts 4-20
4.3.6.4.3 Getting Column Counts 4-20
4.3.6.5 Getting the Current Sheet and Row 4-20
4.3.6.5.1 Getting the Current Sheet 4-20
4.3.6.5.2 Getting the Current Row 4-20
4.3.6.6 Setting Next and Previous Rows 4-20
4.3.6.6.1 Setting the Next Row 4-21
4.3.6.6.2 Setting the Previous Row 4-21
4.3.6.7 Importing and Exporting Documents and Sheets 4-21
4.3.6.7.1 Importing an Excel Spreadsheet Document 4-21
4.3.6.7.2 Importing Worksheets 4-21
4.3.6.7.3 Exporting an Excel Spreadsheet Document 4-22
4.3.6.7.4 Exporting Worksheets 4-22
4.3.6.8 Using Data Tables with Parent and Child Scripts 4-22
4.3.6.8.1 Accessing the Parent Data Table from a Child Script 4-22
4.3.6.8.2 Accessing the Top-Most Data Table in Chain of Parent Scripts 4-23

5 Using the Web Functional Test Module

5.1 About the Web Functional Test Module 5-1

5.1.1 Key Features of the Web Functional Test Module ... 5-2
5.2 Recording Web Functional Tests ... 5-2
5.2.1 Setting Web Functional Test Record Preferences ... 5-2
5.2.2 Adding/Editing Object Identifiers ... 5-3
5.2.2.1 Available Attributes for Web DOM Elements ... 5-5
5.2.3 Recording Web Functional Test Scripts ... 5-6
5.3 Playing Back Scripts ... 5-7
5.3.1 Setting Web Functional Test Playback Preferences ... 5-7
5.3.2 Playing Back Web Functional Scripts ... 5-7
5.3.3 Playing Back Web Functional Scripts with Iterations ... 5-7
5.4 Modifying Scripts ... 5-8
5.4.1 Path Editor Toolbar ... 5-8
5.4.2 Adding Browser Navigation to a Script ... 5-9
5.4.3 Adding Web Actions on Browser Objects ... 5-9
5.4.4 Adding Object Libraries to a Script ... 5-10
5.4.5 Adding a Server Response Test ... 5-11
5.4.6 Adding Text Matching Tests to a Script ... 5-12
5.4.7 Adding Object Tests ... 5-13
5.4.8 Adding Table Tests ... 5-15
5.4.8.1 Testing Images in Tables ... 5-16
5.4.9 Adding a Page Title Test ... 5-17
5.4.10 Adding an HTML Test ... 5-18
5.4.11 Adding an XML Test ... 5-19
5.4.12 Adding a Wait for Page ... 5-21
5.4.13 Inspecting Object Paths ... 5-22
5.4.14 Using the Object Details View ... 5-23
5.4.14.1 Viewing the Object Path ... 5-24
5.4.14.2 Adding an Object Test ... 5-24
5.4.14.3 Adding a Table Test ... 5-24
5.4.14.4 Saving an Object Path to an Object Library ... 5-24
5.4.15 Setting Script Properties ... 5-25
5.4.16 Substituting Databank Variables ... 5-25
5.4.17 Using the Web Functional Test Module API ... 5-26
5.5 Editing Object Libraries ... 5-27

6 Using the HTTP Module

6.1 About the HTTP Module ... 6-1
6.1.1 Key Features of the HTTP Module ... 6-1
6.2 Navigation Editing (Correlation) ... 6-2
6.2.1 Setting Correlation Preferences ... 6-3
6.2.2 Adding Correlation Libraries ... 6-3
6.2.3 Adding and Editing Correlation Rules ... 6-3
6.2.3.1 Client Set Cookie ... 6-4
6.2.3.2 Correlate Cookie Header ... 6-4
6.2.3.3 Correlate Header ... 6-4
6.2.3.4 Correlate Referer Header ... 6-5
6.2.3.5 DOM Correlation Rules ... 6-5

6.2.3.6 Function/Text Substitution Rules ... 6-6
6.2.3.7 Java Session id ... 6-8
6.2.3.8 Substitute Recorded Date ... 6-9
6.2.3.9 Title Verification ... 6-9
6.2.3.10 Variable Substitution Rules ... 6-10
6.3 Recording Scripts ... 6-11
6.3.1 Setting HTTP Record Preferences ... 6-11
6.3.2 Recording a New HTTP Script ... 6-12
6.3.3 Using Client-Side Digital Certificates ... 6-12
6.3.3.1 Exporting Client Certificates from Internet Explorer ... 6-12
6.3.3.2 Configuring OpenScript to use the Client Certificate ... 6-13
6.4 Playing Back Scripts ... 6-13
6.4.1 Setting HTTP Playback Preferences ... 6-13
6.4.2 Playing Back HTTP Scripts ... 6-14
6.4.3 Playing Back HTTP Scripts With Iterations ... 6-14
6.4.4 Viewing Script Playback Results ... 6-14
6.4.5 Resetting Encoding ... 6-15
6.4.6 Comparing Recorded/Playback Results ... 6-15
6.4.7 Playing Back HTTP Scripts In Oracle Load Testing ... 6-15
6.4.8 Posting Binary or XML File Data ... 6-16
6.5 Modifying Scripts ... 6-17
6.5.1 Understanding the HTTP Module Script View ... 6-17
6.5.2 Using Script Variables ... 6-18
6.5.3 Adding a Variable to a Script ... 6-20
6.5.4 Adding a Solve XPath to a Script ... 6-20
6.5.5 Finding a Variable in a Script ... 6-21
6.5.6 Deleting Variables from a Script ... 6-21
6.5.7 Adding Authentication to a Script ... 6-21
6.5.8 Adding Text Matching Tests to a Script ... 6-22
6.5.9 Adding Server Response Tests to a Script ... 6-23
6.5.10 Substituting Databank Variables ... 6-23
6.5.11 Substituting Post Data Variables ... 6-25
6.5.12 Adding a Cookie to a Script ... 6-25
6.5.13 Removing a Cookie From Script ... 6-26
6.5.14 Adding a User Agent to a Script ... 6-26
6.6 Adding Navigation ... 6-27
6.6.1 Understanding Navigation Editing (Correlation) ... 6-27
6.6.2 Adding HTTP Get Navigation ... 6-29
6.6.3 Adding HTTP Post Navigation ... 6-29
6.6.4 Adding an HTTP Multipart Post Navigation ... 6-30
6.6.5 Adding an HTTP XML Post Navigation ... 6-31
6.6.6 Using the HTTP Module API ... 6-32

7 Using the Oracle EBS/Forms Functional Test Module

7.1 About the Oracle EBS/Forms Functional Test Module ... 7-1
7.1.1 Key Features of the Oracle EBS/Forms Functional Test Module ... 7-1
7.1.2 Prerequisites ... 7-2

7.2 Recording Oracle EBS/Forms Functional Tests ... 7-2
7.2.1 Setting Oracle EBS/Forms Functional Test Record Preferences ... 7-3
7.2.2 Adding/Editing Object Identifiers ... 7-3
7.2.3 Recording Oracle EBS/Forms Functional Test Scripts ... 7-4
7.2.4 Event-Driven Recording ... 7-5
7.3 Playing Back Scripts ... 7-7
7.3.1 Setting Oracle EBS/Forms Functional Test Playback Preferences ... 7-7
7.3.2 Playing Back Oracle EBS/Forms Functional Scripts ... 7-7
7.3.3 Playing Back Oracle EBS/Forms Functional Scripts with Iterations ... 7-7
7.4 Modifying Scripts ... 7-8
7.4.1 Adding Forms Actions ... 7-8
7.4.2 Using the Oracle EBS/Forms Functional Test Module API ... 7-8

8 Using the Oracle EBS/Forms Load Test Module

8.1 About the Oracle EBS/Forms Load Test Module ... 8-1
8.1.1 Key Features of the Oracle EBS/Forms Load Test Module ... 8-1
8.1.2 Prerequisites ... 8-2
8.2 Recording Oracle EBS/Forms Load Tests ... 8-3
8.2.1 Setting Oracle EBS/Forms Load Test Record Preferences ... 8-3
8.2.2 Recording Oracle EBS/Forms Load Test Scripts ... 8-3
8.3 Playing Back Scripts ... 8-4
8.3.1 Setting Oracle EBS/Forms Load Test Playback Preferences ... 8-4
8.3.2 Playing Back Oracle EBS/Forms Load Scripts ... 8-4
8.3.3 Playing Back Oracle EBS/Forms Load Scripts with Iterations ... 8-5
8.3.4 Playing Back Oracle EBS/Forms Load Scripts on Multiple "Same" Environments ... 8-5
8.4 Modifying Scripts ... 8-6
8.4.1 Adding Forms Actions ... 8-6
8.4.2 Converting Forms Actions to XML Messages ... 8-7
8.4.3 Using the Oracle EBS/Forms Load Test Module API ... 8-8
8.5 Setting Oracle EBS/Forms Load Test Correlation Preferences ... 8-8
8.6 Oracle EBS/Forms Load Test Correlation Library ... 8-9
8.7 Troubleshooting Oracle EBS/Forms Load Test Scripts ... 8-11
8.7.1 Debugging Using the Message Log ... 8-12
8.7.1.1 During Recording ... 8-12
8.7.1.2 Format of the Recorded Log ... 8-12
8.7.1.3 During Playback ... 8-12
8.7.1.4 After Playback ... 8-13
8.7.2 Analyzing Message Logs ... 8-13
8.7.3 Resolving "Component does not exist" Errors ... 8-13
8.7.4 Troubleshooting Forms ifError Messages ... 8-14

9 Using the Oracle Fusion/ADF Functional Test Module

9.1 About the Oracle Fusion/ADF Functional Test Module ... 9-1
9.1.1 Prerequisites ... 9-1
9.1.2 Key Features of the Oracle Fusion/ADF Functional Test Module ... 9-2
9.2 Configuring the ADF Server ... 9-2

9.2.1 Configuring the WEB-INF/web.xml File ... 9-2
9.2.2 Configuring the trinidad-config.xml File ... 9-3
9.2.3 Verifying the Compression Settings ... 9-3
9.3 Recording Oracle Fusion/ADF Functional Tests ... 9-3
9.3.1 Setting Oracle ADF Functional Test Record Preferences ... 9-4
9.3.2 Adding/Editing Object Identifiers ... 9-4
9.3.3 Recording Oracle Fusion/ADF Functional Test Scripts ... 9-5
9.4 Playing Back Scripts ... 9-6
9.4.1 Playing Back Oracle Fusion/ADF Functional Scripts ... 9-6
9.4.2 Playing Back Oracle Fusion/ADF Functional Scripts with Iterations ... 9-6
9.5 Modifying Scripts ... 9-7
9.5.1 Adding Fusion/ADF Actions ... 9-7
9.5.2 Oracle Fusion/ADF Functional Test Module API ... 9-8

10 Using the Oracle Fusion/ADF Load Test Module

10.1 About the Oracle Fusion/ADF Load Test Module ... 10-1
10.1.1 Key Features of the Oracle Fusion/ADF Load Test Module ... 10-1
10.1.2 Prerequisites ... 10-2
10.2 Recording Oracle Fusion/ADF Load Tests ... 10-2
10.2.1 Recording Oracle Fusion/ADF Load Test Scripts ... 10-2
10.2.1.1 Editing ADF Variables Groups ... 10-4
10.2.1.2 Editing ADF Variables XPaths ... 10-4
10.2.1.3 ADF Component Tree View ... 10-5
10.3 Playing Back Scripts ... 10-5
10.3.1 Playing Back Oracle Fusion/ADF Load Scripts ... 10-6
10.3.1.1 ADF Comparison View ... 10-6
10.3.2 Playing Back Oracle Fusion/ADF Load Scripts with Iterations ... 10-6
10.4 Setting Oracle Fusion/ADF Load Test Correlation Preferences ... 10-6
10.5 Oracle Fusion/ADF Load Test Correlation Library ... 10-7
10.6 Oracle Fusion/ADF Load Test Module API ... 10-7

11 Using the Adobe Flex Functional Test Module

11.1 About the Adobe Flex Functional Test Module ... 11-1
11.1.1 Key Features of the Adobe Flex Functional Test Module ... 11-1
11.1.2 Prerequisites ... 11-2
11.2 Recording Adobe Flex Functional Tests ... 11-3
11.2.1 Recording Adobe Flex Functional Test Scripts ... 11-3
11.3 Playing Back Scripts ... 11-4
11.3.1 Adobe Flex Object Identification ... 11-4
11.3.2 Playing Back Adobe Flex Functional Scripts ... 11-5
11.3.3 Playing Back Adobe Flex Functional Scripts with Iterations ... 11-5
11.4 Modifying Scripts ... 11-6
11.4.1 Adding Flex Actions ... 11-6
11.4.2 Adobe Flex Action Dialog Box ... 11-7
11.4.3 Using the Adobe Flex Functional Test Module API ... 11-10

12 Using the Adobe Flex (AMF) Load Test Module

12.1 About the Adobe Flex (AMF) Load Test Module 12-1
12.1.1 Key Features of the Adobe Flex (AMF) Load Test Module 12-1
12.2 Recording Adobe Flex (AMF) Load Tests 12-2
12.2.1 Recording Adobe Flex (AMF) Load Test Scripts 12-2
12.3 Playing Back Scripts 12-3
12.3.1 Playing Back Adobe Flex (AMF) Load Scripts 12-3
12.3.2 Playing Back Adobe Flex (AMF) Load Scripts with Iterations 12-3
12.4 Modifying Scripts 12-4
12.4.1 Adding Adobe Flex (AMF) Load Actions 12-4
12.4.2 Using the Adobe Flex (AMF) Load Test Module API 12-5
12.5 Setting Adobe Flex (AMF) Load Test Correlation Preferences 12-5
12.6 Adobe Flex (AMF) Load Test Correlation Library 12-5

13 Using the Hyperion Load Test Module

13.1 About the Hyperion Load Test Module 13-1
13.1.1 Key Features of the Hyperion Load Test Module 13-1
13.2 Recording Hyperion Load Tests 13-1
13.2.1 Recording Hyperion Load Test Scripts 13-1
13.3 Playing Back Scripts 13-2
13.3.1 Playing Back Hyperion Load Scripts 13-2
13.3.2 Playing Back Hyperion Load Scripts with Iterations 13-2
13.4 Setting Hyperion Load Test Correlation Preferences 13-3
13.5 Hyperion Load Test Correlation Library 13-3

14 Using the JD Edwards Functional Test Module

14.1 About the JD Ewards Functional Test Module 14-1
14.1.1 Key Features of the JD Edwards EnterpriseOne Functional Test Module 14-1
14.2 Recording JD Edwards EnterpriseOne Functional Tests 14-1
14.2.1 Setting JD Edwards EnterpriseOne Functional Test Preferences 14-2
14.2.2 Adding/Editing Object Identifiers 14-2
14.2.3 Recording JD Edwards EnterpriseOne Functional Test Scripts 14-4
14.3 Playing Back Scripts 14-4
14.3.1 Playing Back JD Edwards EnterpriseOne Functional Scripts 14-4
14.3.2 Playing Back JD Edwards EnterpriseOne Functional Scripts with Iterations 14-5
14.4 Modifying Scripts 14-5
14.4.1 Capturing JD E EnterpriseOne Grid Control Attributes 14-5
14.4.2 Adding JD Edwards EnterpriseOne Grid Control Actions 14-5
14.4.3 Oracle JD Ewards EnterpriseOne Functional Test Module API 14-6

15 Using the JD Edwards Load Test Module

15.1 About the JD Edwards Load Test Module 15-1
15.1.1 Key Features of the JD Edwards Load Test Module 15-1
15.2 Recording JD Edwards Load Tests 15-1
15.2.1 Recording JD Edwards Load Test Scripts 15-1

15.3 Playing Back Scripts 15-2
15.3.1 Playing Back JD Edwards Load Scripts 15-2
15.3.2 Playing Back JD Edwards Load Scripts with Iterations 15-2
15.4 Setting JD Edwards Load Test Correlation Preferences 15-3
15.5 JD Edwards Load Test Correlation Library 15-3

16 Using the PeopleSoft Load Test Module

16.1 About the PeopleSoft Load Test Module 16-1
16.1.1 Key Features of the PeopleSoft Load Test Module 16-1
16.2 Recording PeopleSoft Load Tests 16-1
16.2.1 Recording PeopleSoft Load Test Scripts 16-1
16.3 Playing Back Scripts 16-2
16.3.1 Playing Back PeopleSoft Load Scripts 16-2
16.3.2 Playing Back PeopleSoft Load Scripts with Iterations 16-2
16.4 Setting PeopleSoft Load Test Correlation Preferences 16-3
16.5 PeopleSoft Load Test Correlation Library 16-3

17 Using the Web Services Module

17.1 About the Web Services Module 17-1
17.1.1 Key Features of the Web Services Module 17-1
17.2 Creating Web Services Scripts Using WSDL Manager 17-2
17.2.1 Creating the Web Services Script Tree 17-2
17.2.2 Adding WSDL Files to the WSDL Manager View 17-2
17.2.3 Adding Methods to the Script Tree 17-2
17.2.4 Editing Method Parameters in the Details View 17-3
17.3 Modifying Scripts 17-3
17.3.1 Adding a Web Services Post Navigation 17-4
17.3.2 Adding a Text Matching Test 17-4
17.3.3 Adding Security Extensions 17-5
17.3.4 Adding Attachments 17-6
17.3.5 Web Services Module API 17-8
17.4 Recording Web Services Scripts 17-8
17.4.1 Setting Web Services Record Preferences 17-8
17.4.2 Recording Web Services Scripts 17-9

18 Using the Siebel Functional Test Module

18.1 About the Siebel Functional Test Module 18-1
18.1.1 Key Features of the Siebel Functional Test Module 18-1
18.2 Functional Testing Siebel Applications 18-2
18.2.1 Prerequisites 18-2
18.2.2 Setting up the Siebel Test Environment 18-2
18.2.3 Enabling Siebel Test Automation 18-3
18.2.3.1 Siebel 7.x 18-3
18.2.3.2 Siebel 8.x 18-3
18.2.4 Script Creation Techniques 18-3
18.2.5 Setting Browser Options 18-4

18.2.6 Starting the Siebel Application ... 18-4
18.2.7 Determining a Siebel Component Type ... 18-5
18.3 Recording Siebel Functional Test Scripts ... 18-5
18.3.1 Setting Siebel Functional Test Record Preferences ... 18-6
18.3.2 Adding/Editing SI Element and Site Map Link Paths ... 18-6
18.3.3 Recording Siebel Functional Test Scripts ... 18-7
18.4 Modifying Scripts ... 18-7
18.4.1 Adding Siebel Actions ... 18-7
18.4.2 Handling Non-Standard Siebel Dialog Boxes ... 18-8
18.4.3 Siebel Functional Test Module API ... 18-10

19 Using the Siebel Load Test Module

19.1 About the Siebel Load Test Module ... 19-1
19.1.1 Key Features of the Siebel Load Test Module ... 19-1
19.1.2 Prerequisites ... 19-2
19.2 Load Testing Siebel Applications ... 19-2
19.2.1 Setting Up Siebel Load Test Environments ... 19-2
19.2.1.1 Basic Configuration ... 19-2
19.2.1.2 Floating Load Balancing Test Server ... 19-3
19.2.1.3 Clustered Web Server Configuration ... 19-3
19.2.1.4 Clustered Siebel Servers Configuration ... 19-3
19.2.1.5 Clustered Database Server Configuration ... 19-3
19.2.2 Siebel Correlation Library ... 19-3
19.2.3 Script Creation Techniques ... 19-3
19.2.4 Recording Scripts for Load Tests ... 19-4
19.2.5 Starting the Siebel Application ... 19-4
19.2.6 Playing Back Scripts ... 19-4
19.2.7 Resolving Script Issues ... 19-5
19.2.7.1 Siebel Entities to Parameterize ... 19-5
19.2.8 Using Databanks with Siebel ... 19-6
19.2.9 Preparing the Siebel Server Manager Commands ... 19-7
19.2.9.1 Creating the Batch File ... 19-7
19.2.9.2 Creating the Command Input File ... 19-8
19.2.9.3 Siebel Statistics ... 19-8
19.2.9.4 Batch File Location ... 19-10
19.2.10 Defining ServerStats Metrics ... 19-10
19.2.11 Defining a ServerStats Configuration ... 19-11
19.2.12 Importing Pre-Configured Metrics and Profiles to Oracle Load Testing ... 19-12
19.2.13 Running Load Tests in the Oracle Load Testing Console ... 19-13
19.2.13.1 Viewing VU Grid ... 19-13
19.2.13.2 Viewing ServerStats ... 19-13
19.2.14 Generating Graphs and Reports Using Oracle Load Testing ... 19-13
19.2.14.1 Creating Custom Runtime Graphs ... 19-13
19.2.14.2 Creating Custom Reports ... 19-14
19.3 Setting Siebel Correlation Preferences ... 19-14
19.3.1 Enabling the Java Correlation Mode ... 19-15
19.4 Siebel Correlation Library ... 19-15

19.5 Siebel Script Functions 19-16

20 Using the Utilities Module

20.1 About the Utilities Module 20-1
20.1.1 Key Features of the Utilities Module 20-1
20.2 Using Text File Processing 20-1
20.2.1 Working with Text Files 20-1
20.2.2 Working with CSV Files 20-2
20.2.3 Working with XML Files 20-3
20.3 Getting Values from a Database 20-3
20.3.1 Adding a SQL Query Test 20-5
20.3.2 Calling a Database Procedure Statement 20-7
20.4 Using the XPath Generator 20-8

21 Using the Shared Data Module

21.1 About the Shared Data Module 21-1
21.1.1 Key Features of the Shared Data Module 21-1
21.2 Setting Shared Data Preferences 21-1
21.3 Using the Shared Data Service 21-2
21.3.1 Basic Scenarios 21-2
21.3.2 Enabling the Shared Data Service 21-2
21.3.3 Setting the Password Encryption 21-3
21.3.4 Setting the Connection Parameters 21-3
21.3.5 Creating a Shared Data Queue 21-4
21.3.6 Inserting Data into a Shared Data Queue 21-4
21.3.7 Getting Data from a Shared Data Queue 21-5
21.3.8 Clearing a Shared Data Queue 21-6
21.3.9 Destroying a Shared Queue 21-6
21.3.10 Creating a Shared Data Hash Map 21-6
21.3.11 Inserting Data into a Shared Data Hash Map 21-6
21.3.12 Getting Data from a Shared Data Hash Map 21-7
21.3.13 Clearing a Shared Data Hash Map 21-8
21.3.14 Destroying a Shared Data Hash Map 21-8
21.4 Using The Shared Data API 21-8

22 Using the Block Scenarios Module

22.1 About the Block Scenario Module 22-1
22.2 Creating Block Scenario Projects 22-2
22.2.1 Adding Script Assets to Block Scenario Projects 22-2
22.3 Modifying Block Scenarios 22-3
22.3.1 Adding Blocks 22-3
22.3.2 Adding Scripts 22-4
22.3.3 Adding Child Blocks 22-5
22.3.4 Adding Child Scripts 22-6
22.3.5 Editing Block and Script Settings 22-6
22.3.6 Moving Blocks and Scripts within a Scenario 22-6

22.3.7 Deleting Blocks and Scripts from a Scenario 22-7
22.4 Playing Back Block Scenario Scripts 22-7

A Script Command Line Reference

A.1 Specifying Command Line Settings A-1
A.2 Supported Agent Command Line Settings A-2
A.2.1 General Settings A-2
A.2.2 Browser Settings A-10
A.2.3 HTTP Settings A-11
A.2.3.1 Proxy A-11
A.2.3.2 Compression A-12
A.2.3.3 Headers A-12
A.2.3.4 Connections A-13
A.2.3.5 Other A-13
A.2.3.6 Download Manager A-15
A.2.4 Functional Test Settings A-15
A.2.5 Oracle EBS/Forms Functional Test Settings A-16
A.2.6 Oracle EBS/Forms Load Test Settings A-16
A.2.7 Siebel Load Test Settings A-17
A.2.8 Shared Data Settings A-17
A.2.9 Web Functional Test Settings A-18
A.2.10 Error Recovery Settings A-20
A.2.10.1 General A-20
A.2.10.2 Flex Load Testing (AMF) A-21
A.2.10.3 Functional Testing A-21
A.2.10.4 HTTP A-21
A.2.10.5 Oracle EBS/Forms Functional Testing A-21
A.2.10.6 Oracle EBS/Forms Load Testing A-21
A.2.10.7 Web Functional Testing A-22
A.2.10.8 Utilities A-22

B Proxy Command Line Reference

B.1 Specifying Command Line Settings B-1
B.2 Supported Proxy Command Line Settings B-1
B.2.1 General Settings B-2
B.2.2 Chain Proxy Settings B-3
B.2.3 Logging Settings B-3
B.2.4 Security Settings B-4

C Command Line Tools Reference

C.1 Using the Command-Line Tools Interface C-1
C.2 Supported Command Line Tools Interface Options C-2
C.2.1 Command-Line Compiler Options C-2
C.2.2 Command-Line Asset Updater Options C-3

D Error Message Reference

D.1 Basic Module Error Messages D-1
D.1.1 General Script Exceptions D-1
D.1.2 Binary Decoding Exceptions D-1
D.1.3 Script Creation Exceptions D-2
D.1.4 Segment Parser Exceptions D-3
D.1.5 Script Service Exceptions D-4
D.1.6 URL Encoding Exceptions D-4
D.1.7 Variable Exceptions D-4
D.2 Platform Error Messages D-4
D.2.1 Browser Exceptions D-5
D.2.2 SSL Exceptions D-5
D.2.3 TCP Exceptions D-5
D.2.4 HTTP Exceptions D-7
D.3 HTTP Error Messages D-7
D.3.1 HTTP Service Exceptions D-7
D.4 Oracle EBS/Forms Functional Test Error Messages D-8
D.4.1 Oracle EBS/Forms Functional Test D-8
D.5 Oracle Forms Load Test Error Messages D-10
D.5.1 Connect Errors D-10
D.5.2 I/O Errors D-12
D.5.3 Match Errors D-13
D.5.4 Component Not Found Errors D-13
D.5.5 Playback Errors D-14
D.6 Shared Data Error Messages D-14
D.6.1 Shared Data Exceptions D-15
D.7 Siebel Error Messages D-16
D.7.1 Siebel Exceptions D-16
D.8 Web Error Messages D-16
D.8.1 Web Service Exceptions D-16

E Troubleshooting

E.1 Installation E-1
E.2 OpenScript Script Execution in Oracle Test Manager E-1
E.3 Manual Installation of Firefox Extension E-2
E.4 Installation of Security Certificate in Internet Explorer E-2

F Third-Party Licenses

Index

Preface

Welcome to the Oracle OpenScript User's Guide. Oracle OpenScript is an extensible, standards-based test automation platform designed to test the next generation of Web applications. This guide explains how to use the features and options of Oracle OpenScript for testing Web applications.

Audience

This document is intended for test engineers who will be developing Oracle OpenScript scripts for regression and performance (load and scalability) testing of a Web site or application. The guide does require an understanding of software or Web application testing concepts. Test engineers using Oracle OpenScript should be familiar with the concepts of regression testing, load testing, and scalability testing.

The record/playback paradigm of Oracle OpenScript does not require any programming experience to develop scripts for testing. However, the advanced programming features available in Oracle OpenScript do require experience with the Java programming language. The programming sections and code examples of this manual assume that you understand programming concepts in Java.

Using This Guide

This guide is organized as follows:

Chapter 1, "Getting Started With OpenScript" introduces OpenScript and provides an overview of the features and user interface.

Chapter 2, "Setting Preferences" explains the available options in the OpenScript Preferences categories.

Chapter 3, "Creating and Modifying Scripts" explains the procedures for creating and modifying basic scripts in OpenScript.

Chapter 4, "Using Data Parameterization" explains the concepts and procedures of Data Driven Testing using Databanks.

Chapter 5, "Using the Web Functional Test Module" provides instructions on configuring and using the OpenScript Web Functional Test Module for functional testing of applications through the Document Object Model (DOM) of the Web browser.

Chapter 6, "Using the HTTP Module" provides instructions on configuring and using the OpenScript HTTP Module for load testing of Web applications through the underlying HTTP protocol traffic.

Chapter 7, "Using the Oracle EBS/Forms Functional Test Module" provides instructions on configuring and using the OpenScript Oracle Forms Functional Test Module for functional testing of Oracle Forms web applications.

Chapter 8, "Using the Oracle EBS/Forms Load Test Module" provides instructions on configuring and using the OpenScript Oracle Forms Load Test Module for load testing of Oracle Forms web applications.

Chapter 9, "Using the Oracle Fusion/ADF Functional Test Module" provides instructions on configuring and using the OpenScript Oracle Fusion/ADF Functional Test Module, which provides support for functional testing of Oracle Application Development Framework (ADF)-based applications.

Chapter 10, "Using the Oracle Fusion/ADF Load Test Module" provides instructions on configuring and using the OpenScript Oracle Fusion/ADF Load Test Module, which provides support for load testing of Oracle Application Development Framework (ADF)-based applications.

Chapter 11, "Using the Adobe Flex Functional Test Module" provides instructions on configuring and using the OpenScript Adobe Flex Functional Test Module, which provides support for functional testing of Adobe Flex-based web applications that use the Adobe Flex Automation Framework.

Chapter 12, "Using the Adobe Flex (AMF) Load Test Module" provides instructions on configuring and using the OpenScript Adobe Flex (AMF) Load Test Module, which provides support for load testing of Adobe Flex-based web applications that use the Action Message Format (AMF).

Chapter 13, "Using the Hyperion Load Test Module" provides instructions on configuring and using the OpenScript Hyperion Load Test Module, which provides support for load testing of Hyperion applications.

Chapter 14, "Using the JD Edwards Functional Test Module" provides instructions on configuring and using the OpenScript JD Edwards EnterpriseOne Functional Test Module, which provides support for testing of JD Edwards EnterpriseOne applications with Grid controls.

Chapter 15, "Using the JD Edwards Load Test Module" provides instructions on configuring and using the OpenScript JD Edwards Load Test Module, which provides support for load testing of Oracle JD Ewards EnterpriseOne-based web applications.

Chapter 16, "Using the PeopleSoft Load Test Module" provides instructions on configuring and using the OpenScript PeopleSoft Load Test Module, which provides support for load testing of PeopleSoft applications.

Chapter 17, "Using the Web Services Module" provides instructions on using the OpenScript Web Services Module for testing Web Services.

Chapter 18, "Using the Siebel Functional Test Module" provides instructions on configuring and using the OpenScript Siebel Functional Test Module for testing Siebel applications through the Document Object Model (DOM) of the Web browser and the Siebel test automation framework.

Chapter 19, "Using the Siebel Load Test Module" provides instructions on configuring and using the OpenScript Siebel Load Test Module for load testing Siebel web applications through the underlying HTTP protocol traffic.

Chapter 20, "Using the Utilities Module" provides instructions on using the OpenScript Utilities Module, which provides commonly used testing functions.

Chapter 21, "Using the Shared Data Module" provides instructions for using the Shared Data Module for transferring data using message queues and hash maps.

Chapter 22, "Using the Block Scenarios Module" provides instructions on using the OpenScript Block Scenarios Module, which provides support for generating complex Virtual User scenarios.

Appendix A, "Script Command Line Reference" provides reference information for agent command line settings.

Appendix B, "Proxy Command Line Reference" provides reference information for proxy command line settings.

Appendix C, "Command Line Tools Reference" provides reference information for command line tools options.

Appendix D, "Error Message Reference" provides reference information for error messages.

Appendix E, "Troubleshooting" provides basic troubleshooting information.

Appendix F, "Third-Party Licenses" contains copyright information about certain third-party products used with Oracle Application Testing Suite.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Application Testing Suite documentation set:

- *Oracle Application Testing Suite Release Notes*
- *Oracle Application Testing Suite Getting Started Guide*
- *Oracle Functional Testing OpenScript User's Guide*
- *Oracle Functional Testing OpenScript Programmer's Reference*
- *Oracle Load Testing Load Testing User's Guide*
- *Oracle Load Testing Load Testing ServerStats Guide*
- *Oracle Test Manager Test Manager User's Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Getting Started With OpenScript

OpenScript is an updated scripting platform for creating automated extensible test scripts in Java. Combining an intuitive graphical interface with the robust Java language, OpenScript serves needs ranging from novice testers to advanced QA automation experts.

OpenScript is built on a standards-based platform and provides the foundation for OpenScript Modules and Application Programming Interfaces (APIs). OpenScript APIs are used to build scripts for testing Web applications. The OpenScript API consists of a set of procedures that can be used to customize the scripts within the development environment. The API can also be used by advanced technical users to enhance scripts for unique testing needs.

1.1 OpenScript Features

OpenScript is the next generation environment for developing Oracle Application Testing Suite scripts for Web application testing. OpenScript provides the following features:

- **Scripting Workbench** - OpenScript provides an Eclipse -based scripting Workbench where you can create and run your automated test scripts. Users can use the Tree View graphical scripting interface for creating and editing scripts through the UI. Users can also switch to the Java Code View programming interface and leverage the integrated Eclipse IDE for creating and editing their scripts programmatically.

  Functional test scripts created in OpenScript can be played back to test and validate application functionality. Load test scripts created in OpenScript will run in Oracle Load Testing for application load testing, allowing users to simulate hundreds our thousands of users executing scripts at the same time.

- **Test Modules** - The OpenScript Test Modules provide application-specific test automation capabilities. Each Test Module is custom built to test a specific application or protocol. OpenScript includes several functional and load testing modules for testing Web-based applications. Additional modules can be developed for the OpenScript platform.

  OpenScript's Test Module interface is completely open and extendable by end-users. Users can leverage the Test Module API to build their own modules for testing specific applications or can extend an existing module to add custom functionality.

- **Graphical/Tree View Scripting Interface** - The OpenScript Tree View scripting interface provides a graphical representation of the test script. Multiple script

windows can actually be open at the same time. Within each script window, the Tree View is broken down into 3 main script sections:

- Initialize: For script commands that only execute once on the first iteration
- Run: Main body of the script for commands that will run on every iteration
- Finish: For script commands that only execute once on the last iteration

Within each section, script Steps and Navigation nodes can be created automatically during script recording or manually through the Tree View user interface. Additional script commands will also be represented as nodes in Tree View including test cases, data inputs, log messages, etc. Each Tree View node has a corresponding representation in the Java Code View.

- **Programming/Code View Scripting Interface** - The OpenScript Java Code View scripting interface provides a Java representation of the test script. This view provides full access to Eclipse IDE for creating, editing & debugging script code. Script commands in Java are mapped to a corresponding representation in the Tree View. Users can edit their script in either the code or tree view and changes will be automatically reflected in both views.
- **Properties View & Results View** - The OpenScript Properties View allows users to view detailed properties for selected script nodes in the Tree View. The Results View shows detailed step-by-step results of script playback which are linked to the OpenScript display window.
- **Data Parameterization** - OpenScript allows users to parameterize script data inputs to perform data driven testing. OpenScript uses the following types of data sources:
  - Databank - one or more external comma-separated value (CSV) or text (TXT) files that provides inputs to script parameters. Multiple Databank files can be attached to a single script and users can specify how OpenScript assigns data during script playback. Script playback iterations can cycle through the Databank sequentially, randomly, and by shuffling the data. Databanks can be used with functional and load test scripts. See Section 4.2, "Using Script Databanks" for additional information.
  - Database - a SQL query that extracts data from an Oracle Database in the same format as a Databank CSV or TXT file. See Section 4.2, "Using Script Databanks" for additional information.
  - Data Table - a spreadsheet table that specifies the data by row and column. The data in the table can be entered manually of imported from an Excel spreadsheet file. The Data Table API provides methods for accessing the data in the table programmatically. See Section 4.3, "Using Data Tables" for additional information.
  - Shared Data Service - a Shared Data Module API that provides methods for sharing data between Virtual User agents using a shared data queue or hash map. See Chapter 21, "Using the Shared Data Module" for additional information.

  Users can select any data inputs for their script and then substitute a variable to drive the input from the data source during playback.
- **Correlation** - The OpenScript Correlation interface allows users to create correlation libraries to automatically parameterize dynamic requests during playback. Correlation libraries contain rules for automatically handling dynamic request parameters such as urls, query strings and post data for the load testing modules.

- **OpenScript Preferences** - The OpenScript Preferences interface is where users specify settings to control script recording, script playback, correlation and general preferences for the OpenScript Workbench.
- **Multi-User Execution** - launch more than one OpenScript instance under separate named Windows user accounts. Playback for multiple scripts is supported using any of the following:
  - OpenScript Playback button
  - Command-Line Interface
  - Oracle Load Testing
  - Oracle Test Manager

1.2 Installing OpenScript

To install OpenScript:

> **Note:** OpenScript requires Administrator Privileges to be installed and uninstalled. It does not require Administrator Privileges to run.

1. Go to: http://www.oracle.com/technetwork/oem/app-test/index.html.
2. Download the product Zip file from the Web site and save it to a temporary directory on your hard disk. See the *Oracle Application Testing Suite Release Notes* for additional information about the product zip files.
3. Unzip the download Zip file and run setup.bat.
4. Follow the setup instructions to install the Oracle Application Testing Suite.

> **Note:** A product establishes its Default Repository in $installDir/OFT, where $installDir is the directory where Oracle Application Testing Suite is installed or, if Oracle Application Testing Suite is not installed, where OpenScript is installed.

During the Oracle Application Testing Suite installation, you will be required to enter a master password to be used with Oracle Application Testing Suite products. *Remember this password*. It will be required to log in to the Administrator, Oracle Load Testing, and Oracle Test Manager. The password is not required for OpenScript.

5. Select **OpenScript** from the **Oracle Application Testing Suite** start menu to start the OpenScript Workbench.

1.3 Starting the OpenScript Workbench

To start the OpenScript workbench:

From the Start menu:

- Select **Programs** from the **Start** menu and then select **OpenScript** from the **Oracle Application Testing Suite** menu.

1.4 Overview of the OpenScript Main Window (Workbench)

The OpenScript main window (Workbench) is where you perform the majority of your test development activities. The main window consists of the perspectives used for developing scripts. OpenScript includes a Tester Perspective and a Developer Perspective. The menu bar, toolbar, and the views and editors vary depending upon which perspective is being used. The following sections describe the functionality and various elements of the OpenScript Workbench.

Some dialogs and views require the user to hit the popup menu keyboard button in order to access some features in the UI that are normally accessible using the right-click menu.

1.4.1 Tester Perspective

The OpenScript Tester Perspective provides a convenient way to record and edit scripts and view the playback results. The Tester Perspective opens the following views by default.

- **Script View**: Shows the recorded script in two tabs: Tree View and Java Code. The Tree View tab shows the steps and pages and the Initialize, Run, and Finish nodes of each step using a graphical tree view. The Java Code tab shows the underlying Java code used for the script. The Assets view tab shows the script assets (databanks, jar files, Object libraries and child scripts) that have been added to the script.
- **Details View**: Shows the content details for URL navigations or pages added to the script.
- **Problems View**: Shows any problems in the script code that may produce errors or prevent compiling the script.
- **Properties View**: Shows the properties for the selected node in the script.
- **Console View**: Shows the playback command output and status information for the script. Script log message also appear in the Console.
- **Results View**: Shows the results of script playback.

The following views are also available from the **View** menu but do not open by default:

- **Error Log View**: Shows the error log information for the project and script.
- **Data Table View**: Shows a spreadsheet-like data table for Functional testing scripts.
- **Object Details View**: Shows the attributes and values for the object selected in the browser connected to the view.
- **Script Variables View**: Shows the name and value of script variables. The script variables are only shown when a running script is paused during playback.
- **Treeview Breakpoint View**: Shows the location of breakpoints set in the script tree view.

The views are described in more detail in the following sections.

1.4.2 Developer Perspective

The OpenScript Developer Perspective provides advanced options for developers when creating and editing scripts using the advanced features of OpenScript and the

Eclipse development platform. The Developer Perspective opens the following views by default:

- **Navigator and Package Explorer Views**: Shows hierarchical views of the script project resources. You can open the resource files in an editor to view and edit the contents.
- **Script View**: Shows the recorded script in two tabs: Tree View and Java Code. The Tree View tab shows the steps and pages and the Initialize, Run, and Finish nodes of each step using a graphical tree view. The Java Code tab shows the underlying Java code used for the script. The Assets view tab shows the script assets (databanks, jar files, Object libraries and child scripts) that have been added to the script.
- **Debug View**: Shows the debugging options and provides options for stepping through code.
- **Variables and Breakpoint Views**: Shows the script variables and breakpoints set in the code.
- **Problems View**: Shows any problems in the script code that may produce errors or prevent compiling the script.
- **Declaration View**: shows the source of the element selected in the Java code view.

The following views are also available but do not open by default:

- **Details View**: Shows recorded page details in three tabs: HTML, Browser, and Header. The HTML tab shows the page HTML source. The Browser tab shows the page. The Header tab shows the page response header.
- **Properties View**: Shows the properties for the selected node in the script.
- **Console View**: Shows the playback command output and status information for the script. Script log message also appear in the Console.
- **Results View**: Shows the results of script playback.
- **Error Log View**: Shows the error log information for the project and script.
- **Data Table View**: Shows a spreadsheet-like data table for Functional testing scripts.
- **Object Details View**: Shows the attributes and values for the object selected in the browser connected to the view.
- **Script Variables View**: Shows the name and value of script variables. The script variables are only shown when a running script is paused during playback.
- **Treeview Breakpoint View**: Shows the location of breakpoints set in the script tree view.

The views are described in more detail in the following sections.

1.4.3 OpenScript Menu Options

The menu options that appear change depending upon which perspective is set in the Workbench (Tester or Developer) and which view is the active view. Specific test modules may also add or remove menu options.

1.4.3.1 File

- **New** - opens the New Project wizard. You can select the type of project including OpenScript scripts, jobs and modules

- **Open Script** - opens a window for selecting the OpenScript Repository, workspace, and script to open.
- **Open Object Library** - opens a dialog box for selecting the object library file to open. This option only appears for functional test scripts.
- **Open File** - opens a window for selecting the file to open.
- **Close** - closes the script editor.
- **Close All** - closes all script editors.
- **Save** - saves the data in the currently active editor.
- **Save As** - saves the data in the currently active editor using a new name.
- **Save All** - saves the data in all open editors.
- **Restart** - restarts the OpenScript application and the Eclipse IDE.
- **Import Script** - opens a window for importing an archived OpenScript script project from a .zip file.
- **Export Script** - opens a window for exporting the OpenScript script project to an archive .zip file.
- **Exit** - exits OpenScript.
- [Recent files list] - shows the most recent script files/repository location opened on OpenScript Click a file name to open it in OpenScript. You can set the number of files in the list using the Workbench Preferences (select **Preferences** from the **Window** menu in the Developer Perspective. Expand the **General** preferences and then select **Editors**).

The following options are also available in the Developer Perspective:

- **Revert** - reverts changes to the last saved file contents.
- **Move** - opens a dialog box for selecting where to move the resource.
- **Rename** - opens a dialog box for specifying a new name for the resource.
- **Refresh** - refreshes the resources in the Navigator or Package view.
- **Convert Line Delimiters To** - opens a sub menu for selecting the type of line delimiters for the conversion: Windows, UNIX, or Mac OS.
- **Print** - prints the contents of the selected editor view.
- **Switch Workspace** - opens a dialog box for selecting the workspace to use.
- **Restart** - restarts OpenScript.
- **Import** - opens a window for selecting the type of project to import.
- **Export** - opens a window for exporting the type of project to export.
- **Properties** - opens the properties information for the selected resource.

1.4.3.2 Edit

- **Undo** - undoes the last action.
- **Redo** - redoes the last action.
- **Cut** - cuts the selected text/data to the clipboard.
- **Copy** - copies the selected text/data to the clipboard.
- **Paste** - pasted text/data on the clipboard to cursor location.

- **Delete** - deletes the selected text/data.
- **Select All** - selects all text/data in the currently active editor.
- **Find/Replace** - opens a dialog box for setting the text search and replace options. The menu option is available when an editor is open.
- **Search** - opens a dialog box for specifying the search criteria.

The following options are also available when the Java Code editor is open in the Script view.

- **Expand Selection To** - opens a sub menu for selecting which element to use to expand the selection.
- **Find/Replace** - opens a dialog box for specifying the text to find and replace. You can specify the search direction, scope, and search options.
- **Find Next** - finds the next instance of the Find text specified in the Find/Replace settings.
- **Find Previous** - finds the previous instance of the Find text specified in the Find/Replace settings.
- **Incremental Find Next** - finds the next instance of the Find text specified in the Find/Replace settings.
- **Incremental Find Previous** - finds the previous instance of the Find text specified in the Find/Replace settings.
- **Add Bookmark** - opens a dialog box for specifying the bookmark name.
- **Add Task** - opens a dialog box for defining a task to perform on a resource.
- **Smart Insert Mode** - when selected, code typing aids such as automatic indentation and closing of brackets are enabled in the code view.
- **Show Tooltip Description** - opens a description for the current selection in the Code View.
- **Content Assist** - opens a context assist menu to bring up Java code assist proposals and templates. See the Templates preference page for available templates (Window > Preferences > Java > Editor > Templates) and go to the Editor preference page (Window > Preferences > Java > Editor > Code Assist) for configuring the behavior of code assist.
- **Word Completion** - completes typing of a partial word.
- **Quick Fix** - If the cursor is located at a location with problem indication this opens a context assist dialog at the current cursor to present possible corrections.
- **Set Encoding** - opens a dialog box for setting the type of text file encoding.

1.4.3.3 Search

- **Search** - opens a dialog box for specifying the search criteria.
- **File** - opens a dialog box for specifying the file search criteria.
- **Text** - opens a sub menu for selecting the text search location.

1.4.3.4 Script

- **Add** - opens a sub menu for adding options to the script tree.
- **Record** - starts the selected OpenScript script recorder.

- **Playback** - plays back the currently open OpenScript
- **Iterate** - plays back the script repeatedly, with or without a Databank.
- **Pause/Resume** - pauses and resumes script playback. These options are only active during script playback.
- **Stop** - stops the OpenScript script recorder.
- **Step** - runs the currently selected node and moves the execution point to the next sibling node. If the selected node has a child node, the execution pointer is moved to the first child node. This option is only active during script playback and script execution is suspended while stepping through the script code.
- **Step Into** - steps into the function or sub procedure. This option is only active during script playback and script execution is suspended while stepping through the script code. The execution pointer is moved into the beginning of the function.
- **Configure Recorders** - opens a window for pausing and restarting the current script recorder. This option is only available while in Record mode.
- **Set Record Section** - opens a submenu for selecting the section of the script where script recording will begin. The selected section will be highlight in bold in the script tree. The Run section is the default. The submenu has the following options:
  - **Initialize** - starts script recording in the Initialize section of the script.
  - **Run** - starts script recording in the Run section of the script.
  - **Finish** - starts script recording in the Finish section of the script.
- **Add Step Group** - opens a dialog box for manually adding a Step Group to a script. This option is only available when a script is open.
- **Revert all Navigations to Recorded** - reverts changes back to the recorded version of the script.
- **Update Step Groups** - opens a dialog box for specifying the Update Step Group options.
- **Create/Update Step Groups** - creates step groups in the script based on page navigations.
- **Correlate Script** - opens a dialog box for selecting a defined correlation library to manually apply to the script. Correlation libraries are used to convert dynamic data in page navigations to variable values for script playback. Use the Correlation options in the OpenScript Preferences to define the correlation libraries and rules.
- **Script Properties** - opens a window for setting script-related properties such as correlation, modules, and step groups. Script properties will vary depending upon the script type.

The Script menu includes additional options for functional test scripts:

- **Select Browser** - opens a submenu for selecting the browser to use for recording functional test scripts. The list of browsers shows all running browsers that belong to the current user that are not in record, capture, or playback status. When you record, playback, or capture, OpenScript will use the selected browser. The default browser is the browser instance started by OpenScript.
- **Inspect Path** - starts the object capture mode and opens a browser for selecting the object path to capture. The object path is used by functional test scripts for object identification.

- **Apply Object Libraries** - opens a dialog box for selecting an object library to apply to the current functional test script.
- **Add Object Test** - opens a dialog box for defining an object test for a functional test script.
- **Add Table Test** - opens a dialog box for defining a table test for a functional test script.
- **Add Text Matching Test** - opens a dialog box for defining a Text Matching test for a functional test script.
- **Add Capture Page** - opens a dialog box for specifying a page to capture for a functional test script. The Capture Page option captures a screenshot and the page HTML.

1.4.3.5 View

- **Tester Perspective** - changes the Workbench to the Tester Perspective.
- **Developer Perspective** - changes the Workbench to the Developer Perspective.
- **Reset Perspective** - resets the current perspective to the default settings.
- **Error Log** - toggles the Error Log View. When selected, the Error Logs view is displayed. When cleared, the Error Log View is hidden.
- **Properties** - toggles the Properties View. When selected, the Properties view is displayed. When cleared, the Properties View is hidden.
- **Problems** - toggles the Problems View. When selected, the Problems view is displayed. When cleared, the Problems View is hidden.
- **Details** - toggles the Details View. When selected, the Details view is displayed. When cleared, the Details View is hidden.
- **Console** - toggles the Console View. When selected, the Console view is displayed. When cleared, the Console View is hidden.
- **Results** - toggles the Results View. When selected, the Results view is displayed. When cleared, the Results View is hidden.
- **Data Table** - toggles the Data Table View. When selected, the Data Table view is displayed. When cleared, the Data Table View is hidden.
- **Object Details** - toggles the Object Details View. When selected, the Object Details view is displayed. When cleared, the Object Details View is hidden.
- **Script Variables** - toggles the Script Variables View. When selected, the Script Variables view is displayed. When cleared, the Script Variables view is hidden.
- **Treeview Breakpoint** - toggles the breakpoint indicators in the script tree view and the Treeview Breakpoint view. When selected, any breakpoints set in the script are displayed and the Treeview Breakpoint view is shown. When cleared, the breakpoints in the script tree and the Treeview Breakpoint view are hidden.
- **OpenScript Preferences** - opens the OpenScript Preferences dialog box for specifying default settings and options.

1.4.3.6 Run

- **Resume** - resumes suspended code execution or script playback.
- **Suspend** - suspends the current code execution or script playback.
- **Terminate** - ends the current code execution or script playback.

- **Step Into** - single steps code execution into the highlighted statement or method. The Step options are active in debug mode.
- **Step Over** - single steps code execution over the current statement or method to the next statement or method.
- **Step Return** - steps code execution out of the current method and stops after exiting the current method.
- **Run to Line** - resumes execution until the specified line is executed. Used When a thread is suspended.
- **Use Step Filters** - toggles step filters on and off. When set to on, all step functions apply step filters.
- **External Tools** - opens a sub menu for selecting the external tools option.

The following options are also available in the Developer Perspective:

- **Run** - runs the last launched code or script playback.
- **Debug** - opens the debug configuration options for the last launched code or script playback. You can customize the debug configuration before launching the code or script playback for debugging.
- **Run History** - opens a sub menu listing run configurations. Selecting a run configuration shows the run history in the debug view.
- **Run As** - opens a sub menu listing available external run tools. External tools need to be configured to appear on the sub menu by selecting **External Tools** from the **Run** menu.
- **Run Configurations** - opens the run configuration options for the last launched code or script playback. You can customize the run configuration before launching the code or script playback.
- **Debug History** - opens a sub menu listing debug configurations. Selecting a debug configuration shows the run history in the debug view.
- **Debug As** - opens a sub menu listing available external run tools. External tools need to be configured to appear on the sub menu by selecting **External Tools** from the **Run** menu.
- **Debug Configurations** - opens the debug configuration options for the last launched code or script playback. You can customize the debug configuration before launching the code or script playback.

1.4.3.7 Tools

- **Manage Scripts** - opens a window for managing OpenScript scripts.
- **Manage Folders** - opens a window for managing folders used for OpenScript Workspaces.
- **Manage Repositories** - opens a window for managing OpenScript Repositories.
- **Remove Unchanging Variables** - opens a window for selecting and removing script variables that it is known will never change. Removing unchanging variables can improve script playback performance because unchanging variables will not need to be evaluated during script playback.
- **Import Database File** - opens the Import Database File Wizard for importing a Database Replay capture file, SQL statements from a plain SQL and PL/SQL statements .SQL script file, or SQL statements captured and stored in an SQL Tuning Set (STS) in Oracle Database to generate an OpenScript load testing script.

- **Merge Object Libraries** - opens a dialog box for specifying two Object Library files to merge and the resulting output file.
- **Parameterize URLs** - opens a window for specifying a variable name to use to replace a URL.
- **Generate XPaths** - opens a dialog box for generating an XPath from an XML file.
- **Export Playback Settings** - opens a file save dialog box for specifying the folder and file name of the playback properties file to save.
- **Import Oracle Real User Experience Insight (RUEI) Session Log** - opens a dialog box for selecting a RUEI captured user session .tab log file and specifying the script creation options. The imported RUEI session log files record to HTTP-based OpenScript scripts.
- **Convert to Function Library** - converts the current script to a function library script. A dialog box opens for specifying a Package and Class Name for the function library. The option is only enabled if the current script is not a function library script. See Section 3.3.6, "Using a Script as a Dedicated Function Library" for additional information about function library scripts.
- **Migrate Scripts** - opens the Script Migration Manager for migrating pre-version 9.10 scripts to the current version. The Script Migration Manager provides options for migrating top-level scripts and locating all dependent child scripts.

1.4.3.8 Help

- **Welcome** opens the welcome page with links to the Workbench product documentation.
- **OpenScript Diagnosis Tool** - opens the Diagnosis Wizard for verifying the connection status of the OpenScript Internet Explorer Browser Helper Object (BHO), Firefox extensions, and Forms Internet Explorer helper object.
- **Help Contents** - opens the help table of contents.
- **Search** - opens the help search view.
- **Dynamic Help** - opens the available help topics for the currently active view and perspective.
- **Key Assist** - opens the list of keyboard shortcuts.
- **Tips and Tricks** - opens the help tip and tricks window.
- **Cheat Sheets** - opens the Cheat Sheets view.
- **About OpenScript** - provides version and copyright information and configuration details.

1.4.3.9 Navigate

The Navigate menu appears when the Developer perspective is open.

- **Go Into** - refocuses the active view so that the current selection is at the root. This allows web browser style navigation within hierarchies of artifacts.
- **Go To** - opens a sub menu with options for selecting the location to which to navigate. The sub menu options change depending upon the current view.
- **Open Type** - opens a dialog box for selecting the type library to open in an editor view.

- **Open Type in Hierarchy** - opens a dialog box for selecting the type library to open in a hierarchy view.
- **Open Resource** - opens a dialog box for selecting the resource file to open in an editor view.
- **Show In** - opens a sub menu for selecting where to show the statement or method selected in the code view: Package Explorer, Navigator, or Outline view.
- **Next Annotation** - moves the selection to the next annotation in the Code View.
- **Previous Annotation** - moves the selection to the next annotation in the Code View.
- **Last Edit Location** - moves the selection to the location of the last edit made in the Code View.
- **Go to Line** - opens a dialog box for specifying the Java code line number to go to when in the Java Code view.
- **Back** - moves the selection back through the list of locations previously selected in the Code view.
- **Forward** - moves the selection forward through the list of locations previously selected in the Code view.

Additional menu options may appear depending upon the current Java Code editor selection.

1.4.3.10 Project

The Project menu appears when the Developer perspective is open.

- **Open Project** - opens the project selected in the Navigator View.
- **Close Project** - closes the project selected in the Navigator View.
- **Build All** - builds all projects. This option is only available if the Build Automatically option is not selected.
- **Build Project** - builds the current project. This option is only available if the Build Automatically option is not selected.
- **Build Working Set** - opens a sub menu for selecting or creating a working set of projects. Working set projects are only available if the Build Automatically option is not selected.
- **Clean** - opens a dialog box for selecting the project to clean of build problems.
- **Build Automatically** - toggles the automatic build option on and off.
- **Generate Javadoc** - opens the Generate Javadoc window.
- **Properties** - opens the properties window for the current project.

1.4.3.11 Window

- **New Window** opens a new OpenScript window.
- **New Editor** opens a new editor view of the current file.
- **Open Perspective** opens a sub menu for selecting the perspective to open.
- **Show View** opens a sub menu for selecting the view to show.
- **Customize Perspective** opens a window for selecting the shortcuts and commands to customize.

- **Save Perspective As** opens a dialog box for specifying a name for the saved perspective.
- **Reset Perspective** resets the current perspective to the default settings.
- **Close Perspective** closes the currently open perspective.
- **Close All Perspectives** closes all perspectives.
- **Navigation** opens a sub menu for selecting navigation options.
- **Preferences** opens a window for specifying the project preferences.

1.4.4 OpenScript Tool Bar

The following toolbar buttons are available in the Tester and Developer Perspectives:

- **New** - opens a Wizard for creating new OpenScript scripts or Java platform objects and resources.
- **Open** - opens a dialog box for selecting an existing OpenScript script.
- **Save** - saves the changes in the currently active editor. The button is only active if an editor is open with changes to be saved.
- **Print** - prints the information in the currently selected editor. The button is only active if an editor with printable content is open.
- **Record** - starts OpenScript script recording using the selected script recorder. Clicking the menu button opens the a menu listing the available recorder types.
- **Playback** - starts playback of the currently open Visual Script.
- **Iterate** - opens a dialog box for specifying the playback iterations options.
- **Pause/Resume** - pauses and resumes script playback. These buttons are only active during script playback.
- **Stop** - stops OpenScript script recording.
- **Step** - runs the currently selected node and moves the execution point to the next sibling node. If the selected node has a child node, the execution pointer is moved to the first child node. This button is only active during script playback and script execution is suspended while stepping through the script code.
- **Step Into** - steps into the function or sub procedure. This button is only active during script playback and script execution is suspended while stepping through the script code. The execution pointer is moved into the beginning of the function.
- **Select Browser** - opens a submenu for selecting the browser to use for recording functional test scripts. The list of browsers shows all running browsers that belong to the current user that are not in record, capture, or playback status. When you record, playback, or capture, OpenScript will use the selected browser. The default browser is the browser instance started by OpenScript.
- **Configure Recorders** - opens a window for pausing and restarting the current script recorder. This option is only available while in Record mode.
- **Set Record Section** - opens a submenu for selecting the section of the script where script recording will begin. The selected section will be highlight in bold in the script tree. The Run section is the default.
  - **Initialize** - starts script recording in the Initialize section of the script.
  - **Run** - starts script recording in the Run section of the script.

  - **Finish** - starts script recording in the Finish section of the script.
- **Add Step Group** - opens a dialog box for manually adding a Step Group to a script. This option is only available when a script is open.
- **Inspect Path** - starts the object capture mode and opens a browser for selecting the object path to capture. The object path is used by functional test scripts for object identification.
- **Add Object Test** - opens a dialog box for defining an object test for a functional test script.
- **Add Table Test** - opens a dialog box for defining a table test for a functional test script.
- **Add Text Matching Test** - opens a dialog box for defining a Text Matching test for a functional test script.
- **Add Capture Page** - opens a dialog box for specifying a page to capture for a functional test script.
- **Toolbar** - opens the floating toolbar.
- **Debug** - opens a dialog box for specifying a debug configuration. This button is only available when the Developer Perspective is open.
- **Run** - Runs the selected configuration type or application. This button is only available when the Developer Perspective is open.
- **Run External Tools** - Runs an external application. This button is only available when the Developer Perspective is open.
- **New Java Project** - opens a window for creating a new Java project.This button is only available when the Developer Perspective is open.
- **New Java Package** - opens a window for creating a new Java package. This button is only available when the Developer Perspective is open.
- **New Java Class** - opens a window for creating a new Java class. This button is only available when the Developer Perspective is open.
- **Open Type** - opens a window for specifying the type library to open. This button is only available when the Developer Perspective is open.
- **Search** - opens a dialog box for specifying search options.
- **Next Annotation** - goes to the next annotation in the Java code. This button is only available when the Developer Perspective is open. and a Java Code view is open.
- **Previous Annotation** - goes to the previous annotation in the Java code. This button is only available when the Developer Perspective is open. and a Java Code view is open.
- **Last Edit Location** - opens the edit view that was open and goes to the last edit location. This button is only active when a Java Code view is open.
- **Back to** (location) - browses back to the last OpenScript script view. This button is only available when the Developer Perspective is open.
- **Forward** - browses forward to the previous OpenScript script view. This button is only available when the Developer Perspective is open.

1.4.5 Script View

Shows the recorded script in three tabs: Tree View, Java Code, and Assets. The Tree View tab shows the steps and pages and the Initialize, Run, and Finish nodes of each step using a graphical tree view. The Java Code tab shows the underlying Java code used for the script. The Assets view tab shows the script assets (databanks, jar files, Object libraries and child scripts) that have been added to the script.

The script view is where you perform the majority of script editing actions. The Script view has the following tab views:

1.4.5.1 Tree View

The Tree View shows the script navigations and data as nodes in a collapsible tree view. The Tree View corresponds to the Java Code view. Any changes in the Tree View will be automatically updated in the Java Code view. The Tree View has the following standard nodes:

- Initialize - specifies script actions to perform once at the beginning of script playback.
- Run - specifies script actions to perform one or more times during script playback depending upon databanks or other custom programming.
- Finish - specifies script actions to perform once at the end of script playback.

Use the Record options and right-click shortcut menu to add options to script nodes or modify the properties of script nodes in the Tree View.

1.4.5.2 Java Code

The Java Code view shows the script navigations and data as Java programming code. The Java Code view corresponds to the Tree View. Any changes in the Code View will be automatically updated in the Tree View. The Java Code view has the following standard procedures:

- `initialize()` - corresponds to the Initialize node of the Tree View and executes any custom code added once at the beginning of script playback.
- `run()` - corresponds to the Run node of the Tree View and executes recorded and custom code one or more times during script playback depending upon databanks or other custom programming.
- `finish()` - corresponds to the Finish node of the Tree View and executes any custom code added once at the end of script playback.

Use Ctrl-space to open an Intellisense window listing available procedures. See the API Reference in the OpenScript Platform Reference help for additional programming information.

1.4.5.3 Assets

The Assets view tab shows the script assets (databanks, jar files, Object libraries and child scripts) that have been added to the script. If assets have been added to a script, the tree shows a plus/minus next to the asset name for expanding or collapsing the asset tree. The Assets tab has the following options:

- **Databanks**: Lists the Databank assets added to a script.
- **Object Libraries**: Lists the Object Library assets added to a script.
- **Jar files**: Lists generic Jar file assets added to a script.
- **Script**: Lists the child script assets added to a script.

- **Add**: Opens a file selection dialog box for selecting the file to add as an asset. Expand the My Repositories tree to navigate to a workspace folder containing the file. For Databanks, a submenu opens for selecting CSV file or database-type databanks. For CSV file databanks, you select the file. For database-type databanks, you specify the database driver and connection information.

  Note: Any scripts you plan to run, along with any associated assets, in the Oracle Load Testing application must be stored in a repository/workspace that can be accessed by the Oracle Load Testing Controller. If you create new repositories in OpenScript, you should also add the new repositories in Oracle Load Testing.

- **Edit**: Opens a file selection dialog box for changing which file is added as an asset.
- **Open**: Opens the selected asset file in the appropriate editor.
- **Remove**: Removes the selected asset file from the Assets tree. The file still exists in repository/workspace.

1.4.5.4 Assets

The Assets view tab shows the script assets (databanks, jar files, Object libraries and child scripts) that have been added to the script.The Java Code view has the following standard procedures:

- Databanks - lists the databanks that have been added to the script.
- `run()` - corresponds to the Run node of the Tree View and executes recorded and custom code one or more times during script playback depending upon databanks or other custom programming.
- `finish()` - corresponds to the Finish node of the Tree View and executes any custom code added once at the end of script playback.

Use Ctrl-space to open an Intellisense window listing available procedures. See the API Reference in the OpenScript Platform Reference help for additional programming information.

1.4.6 Details View

The Details view shows the content details for URL navigations added to the script. The Details view may have the following tab views depending upon the selected script node and type of script:

- ScreenShot - shows a screen capture of the web page.
- Browser - shows the Browser rendered page for the script navigation selected in the tree view.
- HTML - shows the HTML source for the script navigation selected in the tree view.
- Headers - shows the Request Header and Response Header source for the script navigation selected in the tree view.
- Comparison - shows the recorded and playback text for the Content, Request Header, or Response Header selected in the Compare list. The Comparison tab appears only after a script is played back and a navigation is selected in the Results View.

- Results Report - shows the results report for the script playback. The Results Report tab appears only after a script is played back and a navigation is selected in the Results View.

1.4.7 Problems View

The Problems view shows any problems in the script code that may produce errors or prevent compiling the script. The Problems view shows the following information:

- # error, # warnings, # infos - shows the number of errors, warning messages, and information messages in the problems view.
- Description - shows a description of the errors, warning messages, and information messages.
- Resource - shows the name of the resource file where the error, warning, or information message was generated.
- Path - shows the script name, workspace, and repository path where the resource file is located.
- Location - shows the location/line number where the error, warning, or information message was generated.

The following toolbar button is available in the Problems View:

- Configure the filters to be applied to this view - opens a dialog box for configuring the filters to apply to the Problems View.

1.4.8 Properties View

The Properties view shows the properties for the selected node in the script. The Properties view shows the following information:

- Property - shows the names of the properties for the script node. The properties vary depending upon which type of script node is selected.
- Value - shows the value of the script node properties. Property values can be edited in the properties view.

The following toolbar buttons are available in the Properties View:

- Show Categories - toggles the property categories.
- Show Advanced Properties - toggles the advanced properties.
- Restore Default Value - restores any changed property values to the default values.

1.4.9 Console View

The Console view shows the playback command output and status information for the script. Script log message also appear in the Console. See the Process Console View topics in the reference section of the Java development user guide online help for additional information about console toolbar options.

1.4.10 Results View

The Results view shows the playback results for the script. The Results View shows the following information:

- Name - shows the test date or navigation name.

- Duration (sec) - shows the absolute time delta in secondsbetween when the action began and when it ended.
- Response Time (sec) - shows the DNS Lookup time plus Establish Connect time plus Write Request time plus Read Response time.
- Result - shows the playback result: Passed or Failed.
- Summary - shows the data values from the Databank that are passed to parameters or it shows failure descriptions.

The following toolbar buttons are available in the Results View:

- Delete Result - deletes the selected result row.
- Delete All Results - deletes all rows from the Results View.
- Scroll Lock - toggles scroll lock on and off for the Result View.
- Properties - opens the Properties for the selected result.

1.4.11 Error Log View

The Error Log view shows the error log information for the project and script. The following toolbar buttons are available in the Error Log View:

- Export Log - exports the error log to a text file.
- Import Log - imports an error log text file to the Error Log View.
- Clear Log Viewer - clears all entries from the Error Log View.
- Delete Log - deletes all logged events.
- Open Log - opens the log file in a text editor.
- Restore Log - restores the error log entries from the log file.

You can right-click on a log entry to open the shortcut menu. The shortcut menu includes the same options as the toolbar. The following additional options are available on the shortcut menu:

- Copy - copies the text for the selected log entry to the clipboard.
- Event Details - opens the event details dialog with the details for the selected log entry.

1.4.12 Data Table View

The Data Table view is a spreadsheet-like data table for Functional testing scripts. The Data Table content can be changed by manually inputting data into cells or by importing an Excel file before playback. The Data Table content can be changed at runtime using the `datatable` API or using the user interface when playback is paused by a breakpoint, paused by exception, or paused by clicking the Pause toolbar button. Changes to the Data Table are saved as part of script playback results only. The Data Table and Result Data Table can be exported to an Excel file. See Section 4.3, "Using Data Tables" for additional information.

The following toolbar buttons are available in the Data Table view:

- Import - opens a dialog for selecting an Excel spreadsheet file to import into the Data Table.
- Export - opens a dialog for specifying where to save the Data Table as an Excel spreadsheet.

You can right-click on a data table cell to open the shortcut Edit menu. The following additional options are available on the shortcut menu:

- Edit - changes the selected cell to text edit mode. Type data into the cell and press Enter.
- Cut - cuts the data from the selected cell.
- Copy - copies the text for the selected cell to the clipboard.
- Paste - pastes text from the clipboard to the selected cell.
- Delete - deletes the text from the selected cell.
- Insert Row Before - inserts a new row into the table before the selected row.
- Insert Row After - inserts a new row into the table after the selected row.
- Delete Row - deletes the selected row from the table.
- Use First Row as Column Header - sets the policy for how the first row of data in each Data Table sheet is used. When selected, the first row of data in the sheet is used as the column header value. When not selected, the first row of data is not used as the column header. The first row policy should be set before any data is manipulated within a Data Table sheet. See Section 4.3.2, "Setting the First Row Policy" for additional information.
- Insert Column Before - inserts a new column into the table before the selected column.
- Insert Column After - inserts a new column into the table after the selected column.
- Rename Column - opens a dialog box for specifying a new heading name for the selected column.
- Delete Column - deletes the selected column from the table.

You can right-click on a worksheet tab in the Data Table view to open the worksheet shortcut menu. The following additional options are available on the shortcut menu:

- Insert Sheet Before - inserts a worksheet tab into the Data Table before the selected worksheet tab.
- Insert Sheet After - inserts a worksheet tab into the Data Table after the selected worksheet tab.
- Rename Sheet - opens a dialog box for specifying a new name for the selected worksheet tab.
- Delete Sheet - deletes the selected worksheet from the Data Table. A confirmation dialog box appears to confirm the deletion.

1.4.13 Object Details View

The Object Details view provides options for viewing the attributes and values for the object selected in the browser connected to the view. The Object Details view is available for Functional tests. See Section 5.4.14, "Using the Object Details View" for additional information about using the Object Details view.

The following toolbar buttons are available in the Object Details view:

- Refresh Tree - refreshes the Object Details tree pane.

- Find a node to inspect by selecting in browser - starts the capture mode for selecting the Web page object in the browser. Highlight the object in the browser and press F10 to select it and show the attributes in the Object Details View.
- Connect to browser/Disconnect from browser - connects or disconnects the Object Details View to the browser. The Object Details View must be connected to the browser to capture objects.

The following options are available in the Object Details view:

- Module - selects the type of OpenScript module. The opjects in the tree view change to the specific module type. For example, the Web module shows the HTML DOM tree. The ADF module shows the ADF object tree.
- Find - provides search capabilities to locate specific text in the Object Details. Type the text to find and click **Next** or **Previous** to locate the attributes and values within the tree.
- Partial Match - when selected the **Next** or **Previous** search will match partial text strings specified in **Find**. When cleared, the **Next** or **Previous** search will match the entire **Find** string.
- Next - searches down the tree for the next object that matches the **Find** string.
- Previous - searches up the tree for the previous object that matches the **Find** string.
- Tree pane - shows a tree view of the Document Object Model (DOM). The right-click shortcut menu includes the following options for working with the object selected in the tree:
  - View Object Path - opens a dialog box showing the full path of the object.
  - Add Object Test - opens the Object Test dialog for defining an object test for the object selected in the tree.
  - Add Table Test - opens the Table test dialog box for defining a table test for the table object selected in the tree. This option is only available for table objects.
  - Save to Object Library - opens the Save to Object Library dialog box for saving the object path to an object library.
- Attribute - shows the attribute name of the object selected in the DOM tree.
- Value - shows the value of the object attribute selected in the DOM tree.

1.4.14 Script Variables View

The Script Variables view shows the name value of script variables. The script variables are only shown when a running script is paused during playback. To view the script variable values select **Script Variables** from the **View** menu, playback the script and click the pause button on the toolbar.

1.4.15 Treeview Breakpoint View

The Treeview Breakpoint view shows the location of breakpoints set in the script tree view. To add a breakpoint to the script tree view, right-click on a script tree node and select **Add Breakpoint** from the shortcut menu. The Treeview Breakpoint view shows the node, file name, and line number of the breakpoint.

1.4.16 Navigator and Package Explorer Views

The Navigator and Package Explorer view shows the Java resources for the script and Java package. Double-click on a resource to open it in an editor view. See the Package Explorer View topics in the reference section of the Java development user guide online help for additional information about Package Explorer toolbar options.

1.4.17 Debug View

The Debug view provides options for debugging script playback. See the Debug View topics in the reference section of the Java development user guide online help for additional information about debugging toolbar options.

1.4.18 Declaration View

The Declaration view shows the source of the element selected in the Java code view. You can open the Java code view of a script and select a script method to view the declaration.

1.4.19 Variables and Breakpoints Views

The Variables and Breakpoints view shows variable values and breakpoints for debugging script playback. See the Breakpoints and Variables View topics in the reference section of the Java development user guide online help for additional information about breakpoints and variables toolbar options.

1.5 About Multi-User Execution

The Multi-User Execution feature lets multiple users run OpenScript from a single installation using multiple concurrent interactive desktop sessions (for example, Terminal Server or Remote Desktop sessions). All users must have Administrator privileges.

Users must login as a different, unique user accounts for each instance of OpenScript they want to run. One installation serves all user accounts on the machine. Users manage their settings and private data in their own workspaces. Any user who opens OpenScript creates a workspace under their "User Profile Folder", for example, C:\Documents and Settings*username*\osworkspace. All of the user's OpenScript Settings and private data are stored under this workspace, including the user's Http Trust Stores (ostruststore).

If more than one user attempts to open the same the script in the same workspace, the second or subsequent users will receive a message indicating the script is in use and locked. The second or subsequent users can make a copy of the script and files in use.

Note: Teams should enable Write access on their root Repository folders for all users and encourage teams to store their databanks and object library files in the Repository. This way, by default all newly created scripts, databanks, and object library files inherit the permissions of the Repository folder and be "Writeable" by all users.

Only load-test scripts can be played in the same desktop session. There no script type limitations for playback in different desktop sessions.

You configure the port range to use for each user in the OpenScript General Preferences. The ports within the configured port range are checked and if none are available, an error message will appear.

1.6 About Script Assets

Script assets are resources that can be used by scripts such as, databanks, generic Jar files, object libraries, or other scripts containing recorded steps or custom functions. Assets have the following characteristics:

- They are external resources that a script can use or run.
- They are resources that are not part of a script like the java code itself.
- They are resources that can be shared among a team of users.
- They are resources that can be added or removed from one or more scripts.

> **Note:** Any scripts you plan to run, along with any associated assets, in the Oracle Load Testing application must be stored in a repository/workspace that can be accessed by the Oracle Load Testing Controller. If you create new repositories in OpenScript, you should also add the new repositories in Oracle Load Testing.

Script assets can be used for testing projects that include multiple users executing a suite of tests or any QA team that uses a complex structure of scripts and assets developed and used by different people.

Script assets provide for the following:

- Creating, editing and obtaining (discovering and assigning) assets independently from scripts.
- Assigning or removing assets from the script using the script properties GUI.
- Viewing all assets associated with a script, and all dependent assets using the script properties GUI.
- Viewing and editing asset properties (alias, location, etc.) from the script properties GUI.
- Creating self-contained script .zip files that include all assets referenced by a script and its assets. Self-contained .zip files can be used as a script export file for Customer support or script execution on computers other than the computer where script was created.
- Importing self-contained script .zip files and running them without having to manually resolve file locations of assets.
- Running self-contained .zip files from the Command line or Oracle Load Testing.
- Notification of missing assets before a script run starts.
- Storing scripts inside subfolders X levels deep in a workspace.
- Loading object libraries and/or databanks into a script without concern about modifying the behavior of a parent script because of conflicting object libraries and/or databanks.
- Understanding the order in which assets are loaded to know which object in which library has precedence.

2
Setting Preferences

The OpenScript Preferences let you specify default values and settings to use for OpenScript options. This chapter explains the available options in the OpenScript Preferences categories. The OpenScript preferences are under the OpenScript node. The available preferences may vary depending upon installed modules.

2.1 Setting OpenScript Preferences

To set OpenScript preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and select the preference category.
4. Specify the preferences as necessary for the selected category.

The following sections explain the available options for each category.

2.2 Correlation and Validation Category

The OpenScript Correlation and Validation interface allows users to create correlation libraries to automatically parameterize dynamic requests during playback. Correlation libraries contain rules for automatically handling dynamic request parameters such as urls, query strings and post data for the specific modules.

This category lets you specify libraries and rules for transforming dynamic data in recorded script URLs and related parameters (headers, post data, etc.) to variable names that will be recognized by the script playback engine (OpenScript or Oracle Load Testing). Correlation rules must be defined within OpenScript modules and are not available with the basic platform.

2.2.1 Module Correlation Preferences

Selected Module: Shows the names of the defined correlation libraries. Use the Add Library button to define libraries. After you define a library you can use the Add Rule button to specify the rules to include in the library.

- **Add Library**: Opens a dialog box for adding a correlation library name.
- **Add Rule**: Opens a dialog box for adding a correlation rule to the selected library.
- **Edit**: Opens a dialog box for editing the selected correlation library name.

- **Delete**: Deletes the selected correlation library from the Preferences list. The defined rules for the library are also removed from the preferences. The correlation library .XML file is not deleted from disk.
- **Up**: Moves the correlation rule up in the priority list.
- **Down**: Moves the correlation rule down in the priority list.
- **Import**: Opens a dialog box for selecting the correlation library file to import.
- **Export**: Opens a dialog box for selecting the location where you want to export the selected correlation library .XML file.
- **Revert**: Reverts the library to the default values.

Tab view: Shows the library or rule details for the selected correlation library or rule. The tab view information changes depending upon whether a library or rule node is selected and which type of correlation rule is selected. See the module chapters for details about specific correlation libraries.

2.2.2 Add Library

This dialog box lets you specify a new correlation library for transforming dynamic data in recorded script URLs and related parameters (headers, post data, etc.) to variable names that will be recognized by the script playback engine (OpenScript or Oracle Load Testing).

- **Name**: Specifies the name of the correlation library. After you define a library you can use the Add Rule button to specify the rules to include in the library. The name is required. You can also select Copy rules to copy correlation rules from an existing library.
- **Copy rules from existing library**: Lets you copy correlation rules from an existing library to a new library.Specifies the name of the correlation library. After you define a library you can use the Add Rule button to specify the rules to include in the library. The name is required. You can also select Copy rules to copy correlation rules from an existing library.
  - **Copy Rules**: When selected, a list of existing correlation rule libraries will be enabled for copying.
  - **Library**: Lists the correlation rule libraries available for copying.

2.2.3 Add/Edit Rule

This dialog box lets you specify or edit a correlation rule for transforming dynamic data in recorded script URLs and related parameters (headers, post data, etc.) to variable names that will be recognized by the script playback engine (OpenScript or Oracle Load Testing).

- **Type**: Specifies the type of correlation rule. The available Source and Target options change depending upon the rule type.
- **Name**: Specifies the name of the correlation rule. The name is required.
- **Source**: Specifies which object(s) to substitute as dynamic data.
- **Target**: Specifies which object(s) to use as the target location of the transform.

2.3 General Category

This category lets you specify general preferences.

2.3.1 General Preferences

These preferences set the default general preferences. The resulting dialog presents the following fields:

Maximum Number of Results to Save: Specifies the maximum number of script playback session results to save. The script playback session results appear in the Results view and are saved to the `results` subdirectory under the script directory. When the maximum number has been reached, subsequent playback sessions delete the oldest playback results.

- **Results**: Specifies the number of results to save.

Record and Playback Port Range: Specifies the range of ports to use to avoid port conflicts when multiple users run OpenScript from a single installation using multiple concurrent interactive desktop sessions (for example, Terminal Server or Remote Desktop sessions).

- **Minimum**: Specifies the port number to use as the minimum.
- **Maximum**: Specifies the port number to use as the maximum.

Opening Script that is Already Opened by Another User: Specifies the response to a user attempting to open a script that is currently opened by another user in a multiple user installation.

- **Ask to open a copy of the script**: When selected, the second or subsequent users attempting to open a script that is in use by another user will receive a prompt asking if they want to open a copy of the script.
- **Do not allow opening a copy of the script**: When selected, the second or subsequent users attempting to open a script that is in use by another user will receive a prompt indicating the script is in use and cannot be opened.

Date Format: Specifies the date format to use for all modules and tests.

- **Use the default short date format for this locale**: When selected, the Date Pattern follows standard Java SimpleDateFormat string conventions. The default value is `MMM d, yyyy h:mm:ss a` (month, day, year, hour, minutes, seconds, am/pm).
- **Use a specific date format**: When selected, the Date Pattern follows the selected format.

Databanks: Specifies the databank preferences.

- **Databank Setup Timeout**: Specifies how much time to spend preparing a databank for use before timing out. The value is in seconds. This setting includes the total time to do all of the following activities:

  If using a Database-backed databank:

  – Connect to the database

  – Query

  – Read records, write into the file

  – Create the index simultaneously

  – Disconnect

  If using a CSV-backed databank:

  – Time required to parse the CSV file and create the index

  If using Random Unique:

  - Time to shuffle the index

- **Use system editor for opening Databanks**: When selected, the databank file opens in the system defined editor for the file type. If there is no system defined editor registered for the file type (for example, Excel for CSV files), OpenScript produces an OLE Exception error when you try to open the file with this option selected. When cleared, the databank file opens in a text editor view. You can open the databank from the **Assets** tab of the Script view. See Section 4.2.1, "Configuring Databanks" for more information about adding databanks to scripts.

Confirm exit when closing last window: When selected, a confirmation message appears when closing the OpenScript workbench.

Confirm exit deleting from script tree: When selected, a confirmation message appears when deleting items from the script tree.

Automatically upgrade scripts: When selected, scripts that were created in OpenScript prior to version 9.10 will be automatically upgraded to the current version. When cleared, the user will be prompted when opening scripts that were created in OpenScript prior to version 9.10 if they want to upgrade the script.

2.3.2 Browser Preferences

The Browser preferences specify the browser and any additional arguments. The resulting dialog presents the following fields:

Browser Type: Select which browser to use.

- **Internet Explorer**: When selected, OpenScript uses the Internet Explorer browser.
  - **Path Override**: Specifies the path and file name to use to override the default Internet Explorer browser location.
  - **Additional Arguments**: Specifies any additional command arguments to include when starting the browser.
- **Firefox**: When selected, OpenScript uses the Firefox browser.
  - **Path Override**: Specifies the path and file name to use to override the default Firefox browser location.
  - **Additional Arguments**: Specifies any additional command arguments to include when starting the browser.
  - **Base Profile**: Specifies the name and location of the Firefox browser user profile to use if not using the default profile. See the Firefox Profile Manager for additional information about user profiles.

Start Up: Specifies the browser start up settings.

- **Startup Timeout**: Specifies the amount of time, in seconds, to use for the browser startup timeout.

2.3.3 Encryption Preferences

These preferences set the default encryption preferences. The resulting dialog presents the following fields:

Do not encrypt script data: When selected, passwords are stored and displayed as plain text in the script.

Obfuscate script data: When selected, passwords are obfuscated before storing and displaying in the script. Obfuscated passwords are hidden but *not* securely encrypted.

Encrypt script data: When selected, passwords are encrypted in the script. Specify an encryption password to use.

- **Password**: Specifies the password to use when encrypting and decrypting data strings using the `encrypt()` and `decrypt()` functions. Only users who use the same encryption password in their copy of OpenScript will be able to decrypt script passwords. This same encryption password must be used for authentication on all agent machines that will run or open the script in Oracle Load Testing Authentication Manager or Oracle Test Manager.

  To get the correct encryption password onto the agent machine(s):

  1. Set the Encryption Password in OpenScript's Encryption Preferences. Make sure the same password is used when creating all scripts.
  2. Copy the following file from your OpenScript machine to some place local on each Agent machine that will run the scripts:

     ```
     <Documents and Settings>\<your username>\osworkspace\..metadata
      \.plugins\oracle.oats.scripting.utilities\encryption.properties
     ```

     Because this file contains the encryption password, you should not put this file in a public folder. However it should be in a location where each agent can access it.
  3. In the Oracle Load Testing or Oracle Test Manager application, add the following argument into the **Additional Arguments** setting:

     ```
     -ENCRYPTION_PROPERTIES_FILE <path>/encryption.properties
     ```

     where `<path>/encryption.properties` is the same encryption key file used by OpenScript, but in a location accessible to the agent machine itself.

Caution: HTTP scripts do not automatically obfuscate/encrypt sensitive script passwords.

2.3.4 Keys Preferences

These preferences set the default keyboard shortcut preferences (keybindings) for OpenScript. You can specify or change the keyboard shortcut key or key combination bindings used to execute commands in OpenScript. The resulting dialog presents the following options:

Scheme: Specifies which key scheme. A 'scheme' is a set of bindings in the Eclipse development environment. The OpenScript keyboard shortcuts are an extension to the keyboard shortcuts in the "default" scheme available in the Eclipse development environment. See the Keys preferences in the Reference section of the Workbench User Guide online help for additional information about schemes.

[Filter]: Filters the commands listed in the Keys preferences dialog box by category. The OpenScript keyboard shortcuts are filtered by the "OpenScript Keys" category. Backspace the text or click the Clear icon to remove the filter text and show other keyboard shortcuts.

Command: Shows the command that will be executed when the shortcut key or key combination is pressed.

Binding: Shows the key or key combination to use to execute the command.

When: Shows the context in which the key or key combination will execute the command. See the **When** field later in this section for additional information.

Category: Shows the category to which the key or key combination belongs. The Category text can be used when filtering keybindings.

User: Shows the user to which the key or key combination belongs.

Copy Command: Copies the currently selected command to a new entry in the command list. The new entry can be customized to use another key or key combination.

Unbind Command: Removes the current key or key combination binding from the selected command. Use this option to change a command's current key or key combination to a different key or key combination.

Restore Command: Restores the default key or key combination binding to the selected command.

Name: Shows the name of the currently selected command for editing a key or key combination binding.

Description: Shows the description of the currently selected command for editing key or key combination binding.

Binding: Specifies the key or key combination to bind to the currently selected command. Press the key or key combination to use as the keyboard shortcut for the selected command.

When: Specifies the context in which the key or key combination will execute the selected command. This list will appear after a key or key combination is entered into the **Binding** field. For OpenScript, the primary contexts for the **When** setting are "Treeview Debugging" and "In Windows".

See the Keys preferences in the Reference section of the Workbench User Guide online help for additional information about contexts.

Conflicts: Shows any conflicts a key or key combination for the currently selected command has with other commands. You can use this to change any keybindings that conflict with each other.

- **Command**: Shows the name of the command(s) that conflict with the currently selected command.
- **When**: Shows the name of the context in which the conflict occurs with the currently selected command.

Filters: Opens a dialog box for specifying the When context filters to apply to the Keyboard shortcuts command list.

Export: Opens a file save dialog box for exporting the keyboard shortcuts to a comma separated value (csv) file. The exported file will contain all keybindings specified in the Eclipse development environment.

2.3.4.1 Default OpenScript Keybindings

The following table lists the default keybindings for OpenScript commands:

Table 2–1 Default OpenScript Keybindings

Command	Binding	When Context
Execute	F4	Treeview Debugging
Execute	F7	Treeview Debugging
Iterate	Alt+I	In Windows

Table 2–1 (Cont.) Default OpenScript Keybindings

Command	Binding	When Context
Pause	Alt+Shift+P	In Windows
Playback	Alt+P	In Windows
Record	Alt+R	In Windows
Step	F6	Treeview Debugging
Step Into	F5	Treeview Debugging
Stop	Alt+S	In Windows

2.3.5 Large Data Preferences

These preferences let you specify the thresholds at which large amounts of data will be stored as part of a load test script or stored in a resource file outside of the script. The large data settings are used only for load test scripts.

These settings are used in cases where an application-under-test either requires or returns large amounts of data in the request-response transaction. Large data that is stored as part of the script code can exceed compiler limitations and cause a script to become too large to compile. Also, in some cases, large data in scripts can consume too much heap, which can cause out of memory issues. Examples of cases where applications can require or return large data include, but are not limited to, the following:

- HTTP post data containing a long list of Name-Value Pairs
- Web Table Tests on tables containing thousands of table cells
- Database SQL Query Result Validation containing hundreds of cells
- Flex HTTP scripts that use very long action messages.

The Large Data preferences include the following settings:

Large Data Settings: Specify the thresholds at which large amounts of data will be stored as part of a script or stored in a resource file and the maximum number items to show in a tree view.

- **Threshold of large data [] KB**: Specifies the number of kilobytes to use as the threshold that determines if large data is stored as part of the script or in a resource file outside of the script. Resource files are saved under the "postData" folder of current script project with a "rsc" file extension.
- **Max count of data items shown**: Specifies the number of data items that will be shown in the treeview for large data items such as HTTP post data, multipart post data, or XML post data. The treeview shows a node such as (x items are omitted) if the number of data items exceeds the **Max count of data items shown** threshold. Clicking the (x items are omitted) node shows the additional items.

2.3.6 Repository Preferences

These preferences let you specify the name and location of the repository to use to store script files. The resulting dialog presents the following fields:

Name: Shows the names of the defined repositories. Use the Add button to define repositories and locations.

Location: Shows the location of defined repositories.

Add: Opens a dialog box for specifying a repository and location.

Edit: Opens a dialog box for editing the selected repository and location. The Default repository location can be changed but it cannot be renamed.

Delete: Deletes the selected repository and location from the Preferences list. The script files and directory are not deleted. The Default repository cannot be deleted.

2.4 Playback Category

This category lets you specify script playback preferences.

2.4.1 General Playback Preferences

This category lets you specify general playback preferences. The resulting dialog box displays the following sections and fields:

2.4.1.1 General

This section lets you specify general playback preferences and displays the following fields:

VU Pacing (Think Time): Specifies the script playback delay between pages for each virtual user. This is the amount of time the user looks at a page before making the next request and is commonly referred to as "think time." There are four options:

- **Recorded**: Uses the delay times that were recorded in the Script. You can set minimum and maximum delay times (in seconds) that override the script delay times in the Minimum and Maximum edit boxes.
- **Recorded/Random**: Uses random delay times based upon the recorded user delay. The low end of the random range as the actual recorded user delay minus the Lower percentage setting. The high end of the random range as the actual recorded user delay plus the Upper percentage setting. For example, if the actual recorded delay time was 100 seconds and the Lower and Upper settings are 10% and 25% respectively, Oracle Load Testing uses random delay times between 90 and 125 seconds.
- **Random**: Uses random times for Virtual User pacing. You can set minimum and maximum delay times for random delay in the Minimum and Maximum edit boxes.
- **No Delay**: Plays back the Visual Scripts at the fastest possible speed with no time between page requests.

Preserve variables between iterations: Used to preserve or automatically clear variables defined in the **Run** section between successive iterations of the **Run** section.

Variables defined in the **Initialize** section will be preserved forever, unless explicitly removed in script code.

Variables set in the **Run** section will always be preserved between the final iteration of the **Run** section and the **Finish** section.

Variables include all items that are added into the script variables collection (see: `getVariables()` script method). This includes variables for elements such as HTTP form fields defined using `http.solve(...)` and `http.solveXPath(...)`.

Execute User Defined Tests: When selected, user defined tests (such as text matching, server response, title, and XPath tests) are executed during playback. When cleared, user defined tests are not executed.

Multiple Playback Warning: When selected, OpenScript generates a warning if a script is currently playing back when attempting to play back another script. Clicking **No** in the warning message stops playback of the script currently playing back. Clicking **Yes** allows the script currently playing back to continue.

Additional Arguments: Used to specify custom OpenScript script.java code arguments. You can create your own settings in OpenScript scripts. For example, you can create custom settings in OpenScript script.java code, as follows:

```
if (getSettings().get("MyCustomSetting").equals("abc")) {
  info("We're running in ABC mode.");
}
```

You can then set the additional arguments in the **Additional Arguments** field as follows:

```
-MyCustomSetting abc
```

Replace URLs: Specifies the URL replacement string in the form:

originalURL1=*replacementURL1*, *originalURL2*=*replacementURL2*,[...]

During playback, anytime the agent makes a request to a URL starting with a segment, *originalURL*, the agent replaces the original URL segment with *replacementURL*. This feature is only supported for Load Test scripts.

- *originalURL* - Specify the starting segment of the *URL*:*port* that appears in the script that should be replaced. This value is case-sensitive.
- *replacementURL* - Specify the new starting segment *URL*:*port* that the agent requests instead of *originalURL*.

For both parameters, if the protocol is omitted, HTTP protocol is assumed. If no port is specified after the host, port 80 is assumed for HTTP protocol, and port 443 is assumed for HTTPS protocol. URLs are replaced after all correlations are applied. One or more URL replacement pairs may be specified, separating each replacement pair with a comma. The following examples show the format of **Replace URLs** strings:

```
test_server:7789=production_server:7789

test:7789=prod:7789,https://stage.oracle.com/main=https://prod.oracle.com/home
```

2.4.1.2 Error Handling

This section specifies the default playback error handling settings.

On iteration failure, do not run more iterations: When selected, virtual user playback is stopped if an error occurs between playback iterations.

Use recorded value if variable is not found: When selected, the recorded data value will be used if a variable is not found. Databank variables will always use the recorded value if the databank attached to the script cannot be found. If the databank cannot be found and **Use recorded value if variable is not found** is not selected, the recorded data value will still be used.

2.4.1.3 Debug

This section specifies the default playback debug settings.

Pause on exceptions: When selected, script playback is paused if an AbstractScriptException occurs during playback. You can use the Execute debug option to run actions after an error occurs. You may also be able to modify the script

and save it to reposition execution pointer at start of the function where the exception occurred.

Activate the workbench when a breakpoint is hit: When selected, the OpenScript workbench becomes the active window when a breakpoint is reached during script execution. When cleared, The OpenScript workbench does not become the active window. When playing back a script with a breakpoint in it, if the browser is the top-level window, it might not be clear that OpenScript is stopped on a breakpoint. By selecting this option, it ensures that focus is brought to the OpenScript window when a breakpoint is reached.

Open and activate the Debug View when a breakpoint is hit: When selected, the Debug view becomes the active view when a breakpoint is reached during script execution. When cleared, the Debug view remains in its current state. This is an advanced option. The Debug View is helpful when debugging long nested functions, such as in order to view the full stack trace of program execution.

2.4.1.4 System

This section specifies the default playback system settings.

Maximum JVM Heap Size: Specifies the maximum size of the JVM heap. The default is 256MB. This value cannot be more than 90% of the total memory size.

JVM Arguments: Used to specify additional Java Virtual Machine (JVM) or program arguments to pass to a script upon playback. It can accept all standard JVM arguments. For example, if you specify a custom argument `-Dmyvariable=myvalue` in the Additional Arguments, the argument will be passed to the script upon playback. Within the script code you can use `System.getProperty("myvariable")` to get "myvalue".

If the specified argument conflicts with existing OpenScript playback setting, (for example the "Maximum Heap Size" playback setting) the playback setting replaces the setting being specified in the JVM Arguments.

Debug logging: When selected, debug logging is enabled and DEBUG messages appear in the Console view during script playback. When cleared, debug logging is disabled and DEBUG messages do not appear in the console view during script playback.

2.4.2 Error Recovery Preferences

This category lets you specify error recovery actions for exceptions that occur during playback. You can set the error recovery action for individual playback exceptions. Expand specific sections and set the error recovery action. You can set the action as Fail, Warn, or Ignore, as follows:

- **Fail**: Report the error as failure and stop script execution.
- **Warn**: Report the error as a warning and continue script execution.
- **Ignore**: Ignore the error and continue script execution.
- **ReportErrorAndContinue**: Report the error to the results log and continue script execution.
- **Pause**: Pause playback and wait for user's decision to continue or abort script execution.

You can use the options on the **Set All** menu to set all the error recovery options to the same setting with a single selection.

Error Recovery playback preferences specified in the OpenScript Preferences are stored on the local machine and only apply when the script is played back from inside OpenScript on that machine. If you upload your script to Oracle Load Testing on another server and your script depends on an error recovery setting being a certain way in order for it to work, then you can set the error recovery setting in the OpenScript script Java code.

In OpenScript scripts, error settings can be turned on and off at any time, overriding the default Oracle Load testing and OpenScript Preferences using script Java code. For example:

```
getSettings().setErrorRecovery("http.zeroLengthDownloads", "IGNORE");
// user code executed in script, such as http.get(), http.post(), ...
getSettings().setErrorRecovery("http.zeroLengthDownloads", "FAIL");
```

2.4.2.1 General

This section lets you specify the default General error recovery actions, as follows:

Variable Not Found - specifies the error recovery action if a variable cannot be found when parsing transformed strings.

Create Variable Failed - specifies the error recovery action if a script fails to create a variable.

File Not Found - specifies the error recovery action if a file is not found.

Segment Parser Failed - specifies the error recovery action if the XPath Segment Parser cannot verify the correctness of an XPath.

Binary Decode Failed - specifies the error recovery action if a binary post data parameter error occurs.

Encryption Service Not Initialized - specifies the error recovery action when the password encryption service was not initialized.

Unexpected Script Error - specifies the error recovery action if any unexpected script error occurs.

Child Script Failed - specifies the error recovery action if a child (a script called from another script) script fails during playback.

Function Failed - specifies the error recovery action if a called function fails during playback.

2.4.2.2 Flex Load Test (AMF)

This section lets you specify the default Flex Load Test error recovery actions, as follows:

Playback Error - specifies the error recovery action if a playback error occurs.

Operation Invocation Error - specifies the error recovery action if an error occurs while invoking an operation on an object.

2.4.2.3 Functional Test

This section lets you specify the default Functional Test error recovery actions, as follows:

Text Matching Failed - specifies the error recovery action if a text matching test fails.

Object Test Failed - specifies the error recovery action if an object test fails.

Table Test Failed - specifies the error recovery action if a table test fails.

XML Test Failed - specifies the error recovery action if an XML test fails.

2.4.2.4 HTTP

This section lets you specify the default HTTP error recovery actions, as follows:

Zero Length Downloads - specifies the error recovery action if a server response indicates zero bytes length.

Text Matching Failed - specifies the error recovery action if a text matching test fails.

Response Time Error - specifies the error recovery action if a Server Response Time test fails.

Solve Variable Failed - specifies the error recovery action if the value of any variable cannot be solved.

HTML Parsing Error - specifies the error recovery action if an HTML parsing error occurs.

Invalid URL - specifies the error recovery action if the server returns an Invalid URL response code.

Invalid HTTP Response Code - specifies the error recovery action if the sever returns an invalid HTTP response code.

Client Certificate Keystore Error - specifies the error recovery action if the Client Certificate Keystore indicates an error.

Element node not found with xpath - specifies the error recovery action if a node for an object element is not found with the specified XPath notation.

Failure to create DOM object - specifies the error recovery action if there is a failure to create a Document Object Model object.

2.4.2.5 Oracle Forms Functional Test

This section lets you specify the default Oracle Forms Functional test error recovery actions, as follows:

Oracle Forms Error - specifies the error recovery action if any Oracle Forms Functional test error occurs.

Status Bar Test Error - specifies the error recovery action if an Oracle Forms Status Bar test error occurs.

2.4.2.6 Oracle Forms Load Test

This section lets you specify the default Oracle Forms Load test error recovery actions, as follows:

Form Connect Error - specifies the error recovery action if a server connection error occurs.

Forms Input/Output Communication Error - specifies the error recovery action if a read or write error occurs with an Oracle Forms message.

Forms Content Match Failed - specifies the error recovery action if the content of a form does not match on playback.

Forms Playback Error - specifies the error recovery action if there is an error playing back a form.

Forms Component Not Found - specifies the error recovery action if a component of a form is not found.

2.4.2.7 Utilities

This section lets you specify the default Utilities error recovery actions, as follows:

SQL Execute Error - specifies the error recovery action if an SQL execute error occurs.

XML Parsing Error - specifies the error recovery action if any XML parsing error occurs.

CSV Loading Error - specifies the error recovery action if an error occurs while loading a CSV file.

SQL Validation Row Count Error - specifies the error recovery action if an error occurs while validating the row count using a SQL statement.

2.4.2.8 Web Functional Test

This section lets you specify the default Web Functional Test error recovery actions, as follows:

Response Time Error - specifies the error recovery action if a Server Response Time test fails.

Solve Variable Failed - specifies the error recovery action if the value of any variable cannot be solved.

Wait for Page Timeout - specifies the error recovery action if a page timeout error occurs.

Object Not Found - specifies the error recovery action if a web page object is not found.

Playback Failed - specifies the error recovery action if script playback fails.

Title Test Failed - specifies the error recovery action if a page Title test fails.

HTML Test Failed - specifies the error recovery action if an HTML test fails.

2.4.3 HTTP Preferences

This category lets you specify HTTP playback preferences. The resulting dialog box displays the following sections and fields:

2.4.3.1 Proxy

This section lets you specify HTTP playback preferences and displays the following fields:

Proxy Host: Specifies the host name of the proxy server.

Proxy Port: Specifies the port to use on the proxy server.

Username: Specify the user name to use for authentication.

Password: Specify the password to use for authentication.

Non-Proxy Hosts: Specifies the host name of the non-proxy servers.

2.4.3.2 Compression

This section lets you specify specifies the default HTTP compression playback settings.

Enable GZIP: When selected, support for gzip compression is enabled. The browser Request includes the `Accept-Encoding: gzip` header indicating a gzip compressed page response will be accepted. If the server uses gzip compression, the response includes the `Content-Encoding: gzip` header indicating the returned

page is in gzip compressed format. The browser unzips the compressed file before rendering the HTML page. Gzip compression is typically used to provide faster transfer of large HTML pages between the browser and the server.

Enable Deflate: When selected, when selected, support for deflate compression is enabled. The browser Request includes the `Accept-Encoding: deflate` header indicating a deflate compressed page response will be accepted. If the server uses deflate compression, the response includes the `Content-Encoding: deflate` header indicating the returned page is in deflate compressed format. The browser inflates the compressed file before rendering the HTML page. Deflate compression is typically used to provide faster transfer of large HTML pages between the browser and the server.

2.4.3.3 Headers

This section specifies the default HTTP header playback settings.

Browser Emulation: Specifies which browser to emulate for script playback. The Default is the recorded browser.

Language: Specifies which language to use for script playback. The default is the locale assigned by the JVM.

HTTP Version: Specifies the HTTP protocol version to specify in the GET or POST request/response between client and server. The HTTP/1.0 protocol is an early implementation of the Hypertext Transfer Protocol. HTTP/1.1 is a standards-based enhancement to the HTTP/1.0 protocol. See the Key Differences between HTTP/1.0 and HTTP/1.1 at
http://www8.org/w8-papers/5c-protocols/key/key.html.

Accept String: Specifies the Accept: HTTP header value looks like. The default in the JavaAgent.properties file is: text/html, image/gif, image/jpeg, */*. If you modify a navigation in a script by adding a custom Accept: header, the custom header value from the script is used instead.

Global Headers: Specifies any custom "Global Headers: string to use in the Request header for script playback. The format is in the form:
name1`:`*value1*`;`*name2*`:`*value2*`;`*name3*`:`*value3*. For example:
`x-oracle-slm-message-id: bcn=<`*beacon_name*`>; svc=<`*service_name*`>;test=<test_name>;step={{@getTopLevelStepName()}}.`

2.4.3.4 Connections

This section specifies the default HTTP playback connections settings.

Enable Keep Alive: When selected, the `Connection: Keep-Alive` header is set to indicate requests should use a persistent connection. The "Keep-Alive" keyword indicates that the request should keep the connection open for multiple requests. For HTTP/1.0, the socket connection is kept open until either the client or the server drops the connection. For HTTP/1.1 all connections are kept alive unless a Connection: close header is specified.

Max Number of Keep Alive Requests: Specifies the maximum number of requests to make on a keep alive connection before closing it or select Unlimited for an unlimited number.

Max HTTP Connections Per User: Specifies the maximum number of server connections per process per server. Each VU makes multiple connections to request additional resources for images and additional frames for example. Setting this option specifies a limit on the total number of connections that the VUs can make to the server. The default setting is 'Default', which means use the default connection limits

as configured on the agent machine. (See Microsoft KBase article Q183110 for more information.)

Max Connection Idle Duration: Specifies the socket 'idle timeout' and uses a new connection when reusing an idle-timeout socket. This is used to specify the timeout for a socket that gets closed by the server side after a long idle period.

2.4.3.5 SSL

This section specifies the default HTTP playback Secure Sockets Layer (SSL) settings.

SSL Version: Specifies the Secure Socket Layer version to use for the proxy server. When recording a secure site in the browser, the user only sees the Proxy Recorder's certificate not the secure web site's certificate. The Browser, Proxy Recorder, and Secure Server each have their own private and public keys which are used to encrypt/decrypt data.

- **SSL**: Use Secure Socket Layer protocol with the proxy server. OpenScript uses Sun Java Secure Socket Extension (JSSE). Sun JSSE by default supports SSLv2, ASSLv3, ASSL, ATLSv1, ATLS, and SSL_TLS.
- **SSL without TLS**: Use Secure Socket Layer without Transport Layer Security. In some cases, a JSSE issue may cause a TLS Protocol connect failure. Use this option if a protocol connect failure occurs when using the **SSL** option.

2.4.3.6 Download Manager

This section lets you specify the default settings for which resources (embedded objects, images, css, js, jars, etc.) to download for a page during playback of a script.

The Download Manager does the following:

- Parses resource URLs embedded in an HTML page during playback.
- Simplifies scripts by filtering out resource URLs from scripts.
- Provides user control over which resource URLs are downloaded or executed during playback.

> **Note:** Certain resource URLs that do not appear directly in the HTML page contents are not parsed by the Download Manager. For example, an HTML page that imports a '.css' file. When the browser loads the HTML page, it automatically loads the '.css' file and downloads any '.gif' resources.
>
> Certain resource URLs that are not inside an HTML tag are not parsed by the Download Manager. For example, a resource URL that is dynamically composed by Javascript cannot be parsed by Download Manager.

The resource downloads will be included in the playback results in the Results view and the HTML Results report. Each downloaded resource counts as a hit in the Hits per Second and Total Hits reports. The size in kilobytes for each resource is included in the Kilobytes per Second and Total Kilobytes reports.

The Download Manager section has the following options:

Use Download Manager: When selected, the Download Manager is enabled during playback. When cleared, the Download Manager is not enabled during playback.

CSS Resource: When selected, css resources in <Link> tags are downloaded during playback. When cleared, css resources are not downloaded during playback.

Image Resource: When selected, image resources in <Img> tags, in the "background" attribute of a tag, or in <style> tags with "background:url" patterns are downloaded during playback. When cleared, image resources are not downloaded during playback.

Embedded Object Resource: When selected, object resources in <Embed> tags or in <Object> tags are downloaded during playback. When cleared, object resources are not downloaded during playback.

Script Resource: When selected, script resources in <Script> tags are downloaded during playback. When cleared, script resources are not downloaded during playback.

Applet Resources: When selected, applet resources in <Applet> tags are downloaded during playback. When cleared, applet resources are not downloaded during playback.

2.4.3.7 Caching

This section specifies the default HTTP caching playback settings.

Cache download pages: When selected, downloaded pages are stored in a local cache and caching options are enabled. Caching places less of a load on the server as only newer pages are requested and brought down from the Web server. When cleared, caching is not used. No caching places more of a load on the Web server because pages and images are brought down from the Web server for every request.

- **Clear cache each iteration**: When selected, the browser's cache is cleared after the script completes each iteration of its `run()` section.
- **Check for newer versions of cached pages**: Specifies when to check for newer versions of cached pages.
  - **Automatically (when page is out of date)**: When selected, the web server is checked for newer versions if the page is out of date. The Web server is not checked for newer versions of unexpired cached pages. This setting behaves like the "Automatically" cache setting in Internet Explorer.
  - **Every visit to the page**: When selected, the Web server is always checked for newer versions of all cached pages. This setting behaves like the "Every time I visit the web page" cache setting in Internet Explorer.

Maximum In-Memory Cache Size - Specifies the maximum amount of in-memory storage to allocate for cached document contents. This setting applies to all virtual users in the process, even though each virtual user keeps its own cached documents. After the in-memory cache is exhausted, document contents will be cached to a temporary folder on disk in <*installDir*>\agent\cache. There is no upper bound on how much disk storage may be used to store cached documents. The disk cache is cleared every time the agent process starts. The default value is 16MB.

2.4.3.8 Miscellaneous

This section specifies the default miscellaneous HTTP playback settings.

Do Not Request URLs Ending In: Specifies the URLs that will not be requested when the URL ends with one of the specified patterns or file types. Specify the ending pattern or file type separated by commas.

Ignore URLs that Match Regex: Specifies the Regular Expression(s) string to use to ignore specific resources. For example, the expression `Login_Banner(.+?)` would

not download resources such as Login_Banner1.gif and Login_Banner2.gif. Multiple Regular Expressions can be separated using a comma (,).

Enable Cookies: When selected, the virtual user profiles will use cookies. Use this setting if your Web application uses cookies to manage session and other context information.

Download Local Files: When selected, the Java Agent retrieves the requested local file contents.

Preserve Cookies between iterations: Used to preserve or automatically clear cookies added to the browser in the **Run** section between successive iterations of the **Run** section.

- Cookies added to the browser in the **Initialize** section will be preserved forever, unless explicitly removed in script code.
- Cookies added to the browser in the **Run** section will always be preserved between the final iteration of the **Run** section and the **Finish** section.

Preserve Connections Between Iterations: Used to preserve connections between OpenScript and the browser between successive iterations of the script. When selected, the browser should attempt to reuse any open browser connections if possible between iterations. Each virtual user maintains its own set of connections that it never shares with other virtual users.

Max Content Size (KB): Specifies the maximum number of KB to download from a server for a given request. The default value of this option is "Unlimited". However, the maximum Virtual User Display Size is set to 1024KB, which may cause content in the Details view to be truncated if the content size exceeds the Virtual User Display Size. You can configure the Maximum Virtual User Display Size by adding the following setting in the **Additional Arguments** field of the General Playback Preferences:

```
-MAX_VUDATA_BYTES xxxxxxxx
```

Where *xxxxxxxx* is the size in KB to set as the Maximum Virtual User Display Size. For example:

```
-MAX_VUDATA_BYTES 10000000
```

See Section 2.4.1, "General Playback Preferences" for additional information about specifying Additional Arguments.

Socket Timeout: Specifies the maximum number of seconds to wait for a socket connection before timing out.

2.4.4 Oracle EBS/Forms Functional Test Preferences

This category lets you specify playback preferences for Oracle EBS/Forms Functional Tests. The resulting dialog box displays the following sections and fields:

2.4.4.1 Event Timeout

This section lets you specify the default forms event timeout setting.

Forms Startup Timeout: Specifies the maximum number of seconds OpenScript should wait for a form to appear before considering the form not found. This is the default timeout when waiting for a form to appear before invoking an action against it. This is also the default timeout when waiting for a form to appear before continuing the script.

Forms Action Timeout: Specifies the maximum number of seconds OpenScript should wait for forms action playback until success.

Forms Response Timeout: Specifies the maximum number of seconds OpenScript should wait for forms response before timing out.

2.4.4.2 Miscellaneous

This section lets you specify screenshot capture preferences.

Capture screenshots: When selected, screenshots of the pages are captured during playback. Screenshots can be viewed by selecting a WaitForPage result in the Results view and then selecting the Screenshot tab in the Details view. Captured screenshots will increase the size of scripts when exported to zip files.

2.4.5 Oracle EBS/Forms Load Test Preferences

This category lets you specify playback preferences for Oracle EBS/Forms Load Tests. The resulting dialog box displays the following sections and fields:

2.4.5.1 Connection

This section lets you specify playback connection preferences.

Heart Beat Interval in seconds: Specifies how often to notify the forms server that the forms client is still alive when there is no user activitiy in the forms client. This value is used to override the timeout configured for the EBS Application that indicates how long the client has no activities. The default "0" value means no heart beat is sent to the server.

2.4.5.2 Miscellaneous

This section lets you specify playback log preferences.

Capture Message Details: Specifies if forms message details are captured during playback. When selected, OpenScript captures and stores Forms message requests, responses, and information about all loaded Forms components during playback. This information is useful to have when debugging the script.

OpenScript displays captured details in the "Messages" and "Object Details" tabs of the Details view. Oracle Load Testing displays this information in the Virtual User Display based on the "Virtual User Display" settings.

Capturing message details is a memory-intensive operation. During heavy load testing, it is recommended to clear this setting to reduce the amount of heap space required by the agent.

Show Message Log In the Console: Specifies if forms message log details are shown it the Console tab. When selected, the message log details are shown in the console. When cleared, the message log details are not shown in the console.

2.4.6 Shared Data Service Preferences

This category lets you specify playback preferences for the agent Shared Data Service. The resulting dialog box displays the following fields:

OATS Credentials: Specifies the authentication credentials to use to establish the communication between the shared queue and the Virtual User.

- **Enable global shared data access credentials**: When selected, when selected, the shard data access credentials are enabled. Specify the **Address**, **User Name**, and **Password**.
- **Address**: Specifies the address of the Oracle Load Testing for Web Application server to use for the shared data service.
- **User name**: Specifies the user name to use for authentication. The default name is `oats` unless changed in the Oracle Application Testing Suite configuration.
- **Password**: Specifies the password to use for authentication. This should be the same password specified in the Encryption setting of the General preferences if the **Encrypt script data** setting is selected.

Actions on Shared Data: Specifies actions on shared data.

- **Timeout**: Specifies the maximum number of seconds to wait for actions on shared data to occur before timing out.

2.4.7 Web Functional Test Preferences

This category lets you specify the default preferences for Web Functional Test script playback. The resulting dialog box displays the following sections and fields:

2.4.7.1 Object Timeout

This section lets you specify the default object playback timeout setting.

Timeout: Specifies the maximum number of seconds OpenScript should wait for an object to appear before considering the object not found. This is the default timeout when waiting for an object to appear before invoking an action against it. This is also the default timeout when waiting for an object to appear before continuing the script.

You can override individual object wait timeouts in `waitForPage()` or `object.waitFor()` by editing their "timeout" properties. Action timeouts cannot be overridden.

2.4.7.2 Capture

This section lets you specify the default screen capture settings.

Capture HTML: When selected, the page HTML will be captured.

Capture screenshots: When selected, screenshots are captured during playback.

- **Only capture browser content**: When selected, only the contents of the browser window is captured as a screenshot.
- **Capture entire screen**: When selected, the entire screen is captured as a screenshot.

Screenshots can be viewed by selecting a WaitForPage result in the Results view and then selecting the Screenshot tab in the Details view. Captured screenshots will increase the size of scripts when exported to zip files.

Capture URLs: When selected, the page URL will be captured.

Capture frames: When selected, the HTML frames on the page will be captured.

Delay time for capture screenshot: Specifies the amount of time to wait before capturing a screenshot of the page.

2.4.7.3 Browser

This section lets you specify the browser settings for functional tests.

Always launch a new browser when playing back a different script: When selected, script playback always launches a new browser when playing back a different script. When cleared, a script never reuses a browser launched by a previously run script.

Hide browser during playback: Specifies if the browser appears or is hidden during script playback. However, the browser will be changed to be visible automatically when focus is set to an element inside of that browser by a mouse click or key press event.

Close browser after playback: Specifies if the browser automatically closes after playback.

2.4.7.4 Cache and Cookies

This section lets you specify the default playback settings for Web functional test-type scripts.

Clear cache before playing back: When selected, pages are cleared from cache before playback.

Clear cache between iterations: When selected, pages are cleared from cache between playback iterations.

Clear session cookies before playing back: When selected, session cookies are cleared from cache before playback.

Clear session cookies between iterations: When selected, session cookies are cleared from cache between playback iterations.

Clear persistent cookies before playing back: When selected, persistent cookies are cleared from cache before playback.

Clear persistent cookies between iterations: When selected, persistent cookies are cleared from cache between playback iterations.

2.4.7.5 Object Identification

This section lets you specify the default Match Format and Object Identification setting.

Object Identification: specifies which Object Identification method to use. Select one of the following options:

- **Use XPath**: When selected, the OpenScript object identification uses the standard XPath query language to find an object within a page by searching for it in the HTML based on its tag + attributes.
- **Use XPath with Smart Match**: When selected, the OpenScript uses an XPath with OpenScript Smart Match, which provides additional functionality to rank the choices in cases where XPath alone returns multiple matches. The following example explains how the Smart Match ranking feature enhances object identification in an XPath. With the following XPath,

  ```
  /web:a[@text='Search' OR @href='search.jsp' OR @index='0']
  ```

  it is possible for multiple links on a page to match the XPath criteria. For example:

  ```
  link A: text='Logout', href='logout.jsp', index=0
  link B: text='Search', href='search.jsp', index=3
  link C: text='Search', href='doNotSearch.jsp', index=15
  link D: text='Find', href='search.jsp', index=22
  ```

When Smart Match is *not* enabled, OpenScript returns the first result found on the page (Link A in the above example).

When Smart Match is enabled, OpenScript ranks all the results based on how well they match the specified attributes in the XPath. OpenScript evaluates the XPath from left-to-right and produce a list of attribute name=value pairs. For example:

```
Attribute 1: text=Search
Attribute 2: href=search.jsp
Attribute 3: index=0
```

OpenScript then builds a table and assigns a score to each attribute for each result. OpenScript assigns a 0 or a 1 based on whether or not each result matches a particular attribute name=value pair. The result with the highest numerical ranking will be used during playback. For example:

Link	Attr 1 text=Search	Attr 2 href=search.jsp	Attr 3 index=0	Smart Mode Score
A	0	0	1	001
B	1	1	0	110 (Best Match)
C	1	0	0	100
D	0	1	0	010

Logical operators (AND, OR) in the XPath are ignored when Smart Match is enabled during playback. In Smart Match mode, all attributes are matched as one group in left-highest priority.

You can specify required attributes by using the Logical AND operator. All attributes joined together using the Logical OR operator are optional. The AND operator has a higher priority than the OR operator when both operators are used in a single XPath. Parenthetical groups of attributes are also permitted. For example:

```
/web:a[@text='Search' AND (@alt='Find' OR @title='Find')]
```

In this XPath, the text attribute is required, and the alt and title attributes are ranked using the Smart Match ranking system.

You can turn on/off Smart Mode for an individual action(s) by using the `getSettings().set()` API.

Match Format: Specifies which format to use to match attributes in an object path. The match format can be a wildcard-formatted or a regular-expression-formatted expression. *`format`* is one of the following settings:

- **Wildcard**: (default) Attributes in the given path may contain wildcards for unknown characters. For example, title="Welcome, user *". An asterisk "*" matches any number of characters. A question mark "?" matches any single character.
- **Wildcard then Regular Expression**: Attributes in the given path may contain a wildcard-formatted expression, or a regular-expression-formatted expression. During playback, an attempt is first made to find the object assuming a wildcard format, then an attempt is made to find the object assuming a regular-expression format.
- **Regular Expression**: Attributes in the given path may contain a regular expression.

2.4.7.6 Miscellaneous

This section lets you specify the default playback settings for Web functional test-type scripts.

Log JavaScript event for actions: When selected, script playback creates a log of the Javascript events (such as onmouseover, onmousedown, click, etc.) fired on HTML elements. This is useful for troubleshooting scripts that do not playback properly pages that include Dynamic HTML(DHTML) using javascript.

With Internet Explorer browsers, the log file is save to <*installdir*>\OpenScript\Oracle IE ToolBar\WebDOMToolBar*.log. With Fire Fox browsers, the logs can be accessed from the **Tools** menu. Select **Error Console** then **Message**.

The Web Functional Test module API provides these API methods that can be added to the script Java Code, if necessary, to handle events that do not playback properly: `web.element("path").fireEvent("eventName")` and `web.element.setSelectedJSElement()`.

When cleared, no logs are created.

Automatically dismiss javascript alert dialogs: When selected, JavaScript alert dialog boxes are automatically dismissed if they appear during playback.

Capture entire screen on fail: When selected, a screen capture of the entire screen is saved to the results if a failure occurs during playback.

2.5 Record Category

This category lets you specify recording preferences.

2.5.1 General Preferences

Selecting the Record preferences node let you specify the following general recording preference:

- **Show external toolbar while recording**: When selected, the floating recording toolbar will be shown while recording. When cleared the floating toolbar is not shown.
- **Do not record any think time**: When selected, think time is not added to the script during recording. When cleared think time is added to the script during recording.

2.5.2 HTTP Preferences

This dialog box lets you specify recording preferences for the HTTP module. The resulting dialog box displays the following sections and fields:

2.5.2.1 General

This tab lets you specify the general browser recorder settings.

Setup: Specifies the network settings for proxy recording.

- **Network Interface**: Enter or select the network IP address of the proxy server.
- **Additional Arguments**: Specify any additional command line arguments to use when starting the proxy server. Two of the more commonly used arguments are as follows:

  `-timeout [milliseconds]` - Specifies how long the proxy should wait on a connection before timing out waiting for a response. The default is 120000ms.

Example usage:

```
-timeout 200000
```

`-replace "serverUrl=localFilePath"` - When *serverUrl* is downloaded, replace its contents with the contents from the local File located at the *localFilePath*. This is used to inject a different response content into the browser than what the server would normally return.

Example usage:

```
-replace "test.server.com\mocApp.htm =
C:\Temp\Replacement\mocApp.htm,http://production.com =
C:\Temp\Replacement\testFile.txt"
```

See Appendix B, "Proxy Command Line Reference" for additional proxy command line arguments.

- **Maximum Download Size (MB)**: Specify the maximum file size for file downloads.
- **Only record requests originating from the local machine**: When selected, only requests originating from the local machine are recorded by the HTTP proxy recorder. When cleared, OpenScript can record requests originating from another machine or device (phone, tablet pc, etc.) which has the proxy in Internet Explorer configured to use the IP address of the OpenScript machine.

  For example, with two machines as machine A and machine B:

  On machine A,

  – launch OpenScript,

  – create an HTTP script,

  – clear **Only record requests originating from the local machine**,

  – Start recording.

  On machine B,

  – launch Internet Explorer browser,

  – set the proxy in Internet Explorer to the IP address of machine A,

  – open a web site in the browser and navigate pages.

  OpenScript records HTTP requests that originate from machine B.

Miscellaneous: Specifies various settings for proxy recording.

- **Record Mode**: Specifies the record mode to use for HTTP scripts.
  - **Web**: When selected, the script recorder generates the Web mode HTTP script Java code with for the requests. This Java code is less verbose than the HTTP mode to simplify Java coding of the scripts. The advantage of the Web mode compared to the HTTP mode is that it simplifies script creation and makes the script easier to read when testing Web browser applications. The Web mode can be used for any Web browser application that communicates via HTTP.
  - **HTTP**: When selected, the script recorder generates the verbose HTTP script Java code with detailed GET and POST requests. This can be used for any HTTP application including Web browser applications and other applications that communicate via HTTP. This is the record mode used for HTTP scripts prior to version 9.20 of OpenScript.

- **SSL Version**: Specifies the Secure Socket Layer version to use for the proxy server. When recording a secure site in the browser, the user only sees the Proxy Recorder's certificate not the secure web site's certificate. The Browser, Proxy Recorder, and Secure Server each have their own private and public keys which are used to encrypt/decrypt data.
  - **SSL**: Use Secure Socket Layer protocol with the proxy server. OpenScript uses Sun Java Secure Socket Extension (JSSE). Sun JSSE by default supports SSLv2, ASSLv3, ASSL, ATLSv1, ATLS, and SSL_TLS.
  - **SSL without TLS**: Use Secure Socket Layer without Transport Layer Security. In some cases, a JSSE issue may cause a TLS Protocol connect failure. Use this option if a protocol connect failure occurs when using the **SSL** option.
- **IE Cache** - specifies the clear cache option for the Internet Explorer browser.
  - **Prompt to clear the cache**: When selected, a prompt dialog box appears when you start recording a script asking if you want to clear the Internet Explorer browser cache.
  - **Always clear the cache**: When selected the Internet Explorer browser cache is always cleared when recording is started.
  - **Never clear the cache**: When selected the Internet Explorer browser cache is never cleared when recording is started.
- **Clear persistent cookies before browser starts**: When selected, all persistent cookies are cleared before the browser starts when recording scripts.
- **Always Launch a new browser when starting recorder**: When selected, the browser launches automatically when recording is started.
- **Close browser when stopping recorder**: When selected, the browser closes automatically when recording is stopped.
- **Record navigations that return error code 404**: When selected, the HTTP recorder records navigations that return a Server Status Code 404: Not Found
- **Capture screenshots**: When selected, screenshots of the pages are captured during recording. Screenshots can be viewed by selecting a Wait For Page node in the Tree view and then selecting the Screenshot tab in the Details view. Captured screenshots will increase the size of scripts when exported to zip files.

2.5.2.2 Proxy Settings

This tab lets you specify the default Proxy recorder settings.

Chain Proxy: Specifies if the OpenScript proxy is chained to another proxy.

- **Chain Proxy**: When selected, the OpenScript proxy is chained to another proxy.
- **Use browser's proxy**: When selected, the HTTP recorder uses the proxy configuration specified by the browser.
- **Use specified proxy**: When selected, OpenScript uses the specified proxy.
  - **Use proxy configuration script**: When selected, the specified configuration scripts will be used.
  - **Address**: Specify the URL to the JavaScript file containing the FindProxyForURL JavaScript function supplied by the system administrator for the intranet environment.
  - **Use proxy server**: When selected, the specified proxy server will be used.

  - **Address:** specify the network IP address of the proxy server to which to chain the OpenScript proxy.
  - **Port**: Specify the port to use on the chained proxy server.

Proxy Authentication: Specifies the log in credentials for authentication.

- **Username**: Specify the user name to use for authentication.
- **Password**: Specify the password to use for authentication.

2.5.2.3 URL Filters

This tab lets you specify the URL type(s) to filter during recording.

Name: Shows the name(s) of the defined filters. Select the checkbox to enable the filter. Clear the checkbox to disable the filter.

Pattern: Shows the pattern(s) specified for the defined filters.

Match by: Shows the match setting(s) (Content Type or URL) specified for the defined filters.

Add: Opens a dialog box for specifying a URL filter.

Edit: Opens a dialog box for editing the selected URL filter.

Delete: Deletes the selected URL filter.

Automatically filter download manager resources: When selected, the proxy recorder automatically filters the URL resources based upon the settings specified in the Download Manager section of the HTTP Playback preferences. When cleared, the Download Manager settings are not used during recording.

2.5.2.4 Certificates

This tab lets you specify the Client-Side Digital Certificate Store to use when recording.

Store Client-Side Digital Certificate File (.PFX format): Specifies the .PFX-formatted digital certificate information.

- **Last stored certificate**: Specifies the name of the certificate PFX file. Enter the name or click **Store Certificate** to select the file from a drive and directory location. Click **Store Certificate** and enter the file name and private-key password defined for the client certificate PFX file when the certificate was exported from Internet Explorer. Click **Delete Certificate** to remove the certificate set for **Last stored certificate**.

Set Customized Certificate File: Specifies a custom digital certificate file and password. Click **Browse** to select the file. The Proxy Recorder will attempt to load the customized certificate first. If there is no customized certificate specified, the proxy recorder will load the default proxy certificate. If an incorrect certificate file is specified, the proxy recorder will fail to initialize and throw an exception.

- **Password**: Specifies the password to use for the custom certificate file.

2.5.2.5 Object Identification

This tab lets you specify the HTTP/Web object identification attributes. Object identification attributes define how OpenScript recognizes and records specific controls used in HTTP/Web-based applications.

Active Profile: Specifies which object identification profile to use as the active profile when recording scripts. Profiles define a specific set of object identifiers to use when recording HTTP/Web tests. Use the **Add Profile** option to create a new custom profile.

Once you have created a profile, select the profile name in the **Name** column and use **Add Object** to define custom objects and attributes in the custom profile.

Name: Shows the name(s) of the defined HTTP/Web object identifiers and profiles.

Attributes: Shows the pattern(s) specified for the defined HTTP/Web object identifiers.

Add Profile: Opens a dialog box for specifying a new HTTP/Web object identifier profile.

Add Object: Opens a dialog box for specifying a new HTTP/Web object identifier.

Edit: Opens a dialog box for editing the selected HTTP/Web object identifier.

Delete: Deletes the selected HTTP/Web object identifier or custom profile. The default profile cannot be deleted.

Export: Opens a dialog box for exporting the currently selected HTTP/Web object identifier profile to an XML file.

Import: Opens a dialog box for importing a saved object identifier profile XML file.

Revert: Reverts the default HTTP/Web object identification profile to the default profile. Any changes to the default profile are removed. Select the default profile name in the **Name** column to activate the revert option.

For each object element, you specify a name (typically an HTTP/Web object attribute), an operator, a value and a value type. As you add object elements, OpenScript builds the object identifier using logical OR between each object identifier element. Click **Edit** to change between logical OR and AND.

2.5.3 Oracle ADF Functional Test Preferences

This dialog box lets you specify recording preferences for the Oracle Application Development Framework (ADF) Functional Test module. The resulting dialog box displays the following sections and fields:

2.5.3.1 Object Identification

This tab lets you specify the Oracle ADF object identification attributes. Object identification attributes define how OpenScript recognizes and records specific controls used in ADF-based applications.

Active Profile: Specifies which object identification profile to use as the active profile when recording scripts. Profiles define a specific set of object identifiers to use when recording ADF functional tests. Use the **Add Profile** option to create a new custom profile. Once you have created a profile, select the profile name in the **Name** column and use **Add Object** to define custom objects and attributes in the custom profile.

Name: Shows the name(s) of the defined Oracle ADF object identifiers and profiles.

Attributes: Shows the pattern(s) specified for the defined Oracle ADF object identifiers.

Add Profile: Opens a dialog box for specifying a new Oracle ADF object identifier profile.

Add Object: Opens a dialog box for specifying a new Oracle ADF object identifier.

Edit: Opens a dialog box for editing the selected Oracle ADF object identifier.

Delete: Deletes the selected Oracle ADF object identifier or custom profile. The default profile cannot be deleted.

Export: Opens a dialog box for exporting the currently selected ADF object identifier profile to an XML file.

Import: Opens a dialog box for importing a saved object identifier profile XML file.

Revert: Reverts the default ADF object identification profile to the default profile. Any changes to the default profile are removed. Select the default profile name in the **Name** column to activate the revert option.

For each object element, you specify a name (typically an Oracle ADF object attribute), an operator, a value and a value type. As you add object elements, OpenScript builds the object identifier using logical OR between each object identifier element. Click **Edit** to change between logical OR and AND.

2.5.4 Oracle EBS/Forms Functional Test Preferences

This dialog box lets you specify recording preferences for the Oracle EBS/Forms Functional Test module. The resulting dialog box displays the following sections and fields:

2.5.4.1 General

This tab lets you specify the general Oracle Forms recorder settings.

Miscellaneous: Specifies if screenshots are captured.

- **Capture screenshots**: When selected, screenshots are captured during recording.
- **Suppress JRE Plug-in Security Dialog**: When selected, "Java Security Windows" and "Warning - Security" pop up windows are not shown during either script recording or play back when the client machine has Java Runtime Environement (JRE) that is 1.6.0_24 or newer. If the client machine has JRE that is 1.6.0_23 or older, no matter if this setting is enabled or disabled, "Java Security Windows" and "Warning - Security" pop up windows will not show during either script recording or play back.
- **Activate Event-Driven Recording** - when selected, browser events recording is activated.

2.5.4.2 Object Identification

This tab lets you specify the Oracle Forms object identification attributes. Object identification attributes define how OpenScript recognizes and records specific controls used in EBS/Forms-based applications.

Active Profile: Specifies which object identification profile to use as the active profile when recording scripts. Profiles define a specific set of object identifiers to use when recording EBS/Forms functional tests. Use the **Add Profile** option to create a new custom profile. Once you have created a profile, select the profile name in the **Name** column and use **Add Object** to define custom objects and attributes in the custom profile.

Name: Shows the name(s) of the defined Oracle EBS/Forms object identifiers.

Attributes: Shows the pattern(s) specified for the defined Oracle EBS/Forms object identifiers.

Add Profile: Opens a dialog box for specifying a new Oracle EBS/Forms object identifier profile.

Add Object: Opens a dialog box for specifying a new Oracle EBS/Forms object identifier.

Edit: Opens a dialog box for editing the selected Oracle EBS/Forms object identifier or profile.

Delete: Deletes the selected Oracle Forms object identifier.

Export: Opens a dialog box for exporting the currently selected Forms object identifier profile to an XML file. Select the profile name in the **Name** column to activate the export option.

Import: Opens a dialog box for importing a saved object identifier profile XML file.

Revert: Reverts the default EBS/Forms object identification profile to the default profile. Any changes to the default profile are removed. Select the default profile name in the **Name** column to activate the revert option.

2.5.4.3 Applet Object Identification

This tab lets you specify the Oracle Forms applet object identification attributes. Applet object identification attributes define how OpenScript recognizes and records specific applet controls used in EBS/Forms-based applications.

Active Profile: Specifies which object identification profile to use as the active profile when recording scripts. Profiles define a specific set of applet object identifiers to use when recording EBS/Forms functional tests. Use the **Add Profile** option to create a new custom profile. Once you have created a profile, select the profile name in the **Name** column and use **Add Object** to define custom applet objects and attributes in the custom profile.

Name: Shows the name(s) of the defined Oracle EBS/Forms applet object identifiers.

Attributes: Shows the pattern(s) specified for the defined Oracle EBS/Forms applet object identifiers.

Add Profile: Opens a dialog box for specifying a new Oracle EBS/Forms applet object identifier profile.

Add Object: Opens a dialog box for specifying a new Oracle EBS/Forms applet object identifier.

Edit: Opens a dialog box for editing the selected Oracle EBS/Forms applet object identifier or profile.

Delete: Deletes the selected Oracle Forms applet object identifier.

Export: Opens a dialog box for exporting the currently selected Forms applet object identifier profile to an XML file. Select the profile name in the **Name** column to activate the export option.

Import: Opens a dialog box for importing a saved applet object identifier profile XML file.

Revert: Reverts the default EBS/Forms object identification profile to the default profile. Any changes to the default profile are removed. Select the default profile name in the **Name** column to activate the revert option.

2.5.5 Oracle EBS/Forms Load Test Preferences

This dialog box lets you specify recording preferences for the Oracle Forms Load Test module. The resulting dialog box displays the following section and fields:

Mode: Specifies the Forms message recording mode.

- **Record all forms messages**: When selected, all forms messages between the client and the server are recorded. This option creates a verbose script which generates the same load as an actual user environment.
- **Record critical forms messages only**: When selected, only forms messages that are considered critical between the client and the server are recorded. "Critical" messages are the minimum amount of messages needed by the EBS/Forms Load Test module to be able simulate a business flow as an actual user scenario. The EBS/Forms Load Test recorder marks all critical messages according to known Forms protocol.

  This option creates a less verbose script with fewer recorded statements than the **Record all forms messages** option and generates less (approximately 60-80 percent) of the load as an actual user environment. In some cases, the critical only mode may experience record/playback issues if unknown Forms protocol messages are not recorded as critical messages.

  In some circumstances, this option may also be useful if a script requires a large amount of custom logic and programming as the script is less verbose and does not require knowing how or where to add terminal messages.

Applet Parameters: Specifies the Forms Applet Class property to use to record Forms applications.

- **EBS Forms**: When selected, the OpenScript Applet Class property oracle.forms.engine.main is used to record EBS/Forms applications.
- **Custom Forms**: When selected, the specified Applet Class property is used to record custom non-Web deployed Forms applications.

Miscellaneous: Specifies the miscellaneous record preferences.

- **Force HTTP Recording**: When selected, recording communicates over HTTP disregarding the connection parameter values of the Applet page. This setting may be necessary if the site has an Applet loading page that is set to communicate over a socket instead of using HTTP. If the contents of the Applet loading page are set to Socket, OpenScript cannot record the socket traffic.

2.5.6 Oracle JDE EnterpriseOne Functional Test Preferences

This dialog box lets you specify recording preferences for the Oracle JDEdwards EnterpriseOne Functional Test module. The resulting dialog box displays the following sections and fields:

2.5.6.1 Object Identification

This tab lets you specify the JDE EnterpriseOne object identification attributes. Object identification attributes define how OpenScript recognizes and records specific controls used in JDE EnterpriseOne-based applications.

Active Profile: Specifies which object identification profile to use as the active profile when recording scripts. Profiles define a specific set of object identifiers to use when recording JDE EnterpriseOne functional tests. Use the **Add Profile** option to create a new custom profile. Once you have created a profile, select the profile name in the **Name** column and use **Add Object** to define custom objects and attributes in the custom profile.

Name: Shows the name(s) of the defined JDE EnterpriseOne object identifiers and profiles.

Attributes: Shows the pattern(s) specified for the defined JDE EnterpriseOne object identifiers.

Add Profile: Opens a dialog box for specifying a new JDE EnterpriseOne object identifier profile.

Add Object: Opens a dialog box for specifying a new JDE EnterpriseOne object identifier.

Edit: Opens a dialog box for editing the selected JDE EnterpriseOne object identifier.

Delete: Deletes the selected JDE EnterpriseOne object identifier or custom profile. The default profile cannot be deleted.

Export: Opens a dialog box for exporting the currently selected JDE EnterpriseOne object identifier profile to an XML file.

Import: Opens a dialog box for importing a saved object identifier profile XML file.

Revert: Reverts the default JDE EnterpriseOne object identification profile to the default profile. Any changes to the default profile are removed. Select the default profile name in the **Name** column to activate the revert option.

For each object element, you specify a name (typically an JDE EnterpriseOne object attribute), an operator, a value and a value type. As you add object elements, OpenScript builds the object identifier using logical OR between each object identifier element. Click **Edit** to change between logical OR and AND.

2.5.7 Siebel Functional Test Preferences

This dialog box lets you specify recording preferences for the Siebel Functional Test module. The resulting dialog box displays the following sections:

2.5.7.1 General

This tab lets you specify the Siebel general preferences.

SI Elements Paths: Specifies if the Siebel script recorder uses Siebel specific object identifier paths for the webdom elements that are marked in the Siebel application as Standard Interactivity (SI) controls instead of normally recorded attributes such as text, href, and index.

- **Use special paths for SI elements**: When selected, the Siebel script recorder records only the Siebel tag attributes when normal html elements (A, TD, INPUT, DIV, etc.) are used as SI controls in a Siebel application.
- **Path**: Specifies the object identifier path to use for Siebel SI controls. The Siebel attributes are RN (repository name), OT (object type) and UN (unique name).
- **Edit**: Opens a dialog box for editing the object identifier path.

Sitemap Links: Specifies if the Siebel script recorder uses Siebel site map page specific object identifier paths when recording actions on links within the Siebel site map page instead of normally recorded object identifier paths that use the standard path. The standard path includes a particular document index or frame name, which may change dynamically on playback of the script.

- **Use global paths for Sitemap link**: When selected, the Siebel script recorder records only a site map page specific path for the object identifier path.
- **Path**: Specifies the object identifier path to use for Siebel site map page links. The Siebel attributes are RN (repository name), OT (object type) and UN (unique name).

- **Edit**: Opens a dialog box for editing the object identifier path.

Miscellaneous: Specifies the miscellaneous Siebel record preferences.

- **Record "waitForPage" actions**: When selected, the script recorder generates "wait for page" actions for test steps that generate a page transition in the browser. When cleared, the script recorder generates a "capture page" action for test steps in Siebel applications. Some Siebel actions may not generate the page transition needed to reliably play back "wait for page" actions. Clearing this setting for Siebel Functional test scripts allows scripts to record the "capture page" action instead of the "wait for page" action for more accurate script playback.

2.5.8 Web Functional Test Preferences

This dialog box lets you specify recording preferences for the Web Functional Test module. The resulting dialog box displays the following sections and fields:

2.5.8.1 General

This tab lets you specify the general Web Functional test recorder settings.

Browser: Specify the browser options that will be used during Web Functional test recording.

- **Always launch a new browser when recording a different script**: When selected, a new instance of the Internet Explorer browser is launched for each new script recording. When cleared, a new browser is launched only for the first script recording of the OpenScript session. The general case is to launch a new browser instance for each specific script. However, when chaining scripts using a shell script where each script needs to use the same instance of the browser, clearing this setting will cause subsequent scripts recordings to use the same browser instance as the first recording.

Miscellaneous: Specifies the miscellaneous record settings.

- **Capture screenshots**: When selected, screen images are captured during recording. Screenshots can be viewed in the Results view.
  - **Only capture browser content**: When selected, only the content of the browser window is captured during recording.
  - **Capture entire screen**: When selected, the entire screen is captured during recording.
- **Capture HTML**: When selected, page source HTML is captured during recording.
- **Capture URLs**: When selected, the page URL will be captured.
- **Capture frames**: When selected, the HTML frames on the page will be captured.
- **Ignore auto page**: When selected, server-side auto pages are ignored during recording.
- **Action cache interval(s)**: Specifies how often to cache page actions during recording. The following cases are determined by this setting:

  If while recording, the text on the same Web page element is changed within the Action Cache Interval time setting, the previously recorded value will be replaced by the changed value. In the Java code, the `setText` action will be replaced with the changed value.

  If while recording, a browser window closes within the Action Cache Interval after a user performs an action on a web page (for example, a button click) the window

close event will not be recorded, as the window close event is considered to be caused by the previously performed action.

- **Record "waitForPage" actions**: When selected, the script recorder generates "wait for page" actions for test steps that generate a page transition in the browser. When cleared, the script recorder generates a "capture page" action for test steps that generate a page transition in the browser. For Web functional test scripts, the "wait for page" action is the normal record option. See Section 2.5.7, "Siebel Functional Test Preferences" for additional information.
- **Record "mouseClick" actions**: When selected, the script recorder generates mouse click actions to support record and playback of actions against embedded browser objects such as Flash or ActiveX.
- **Create Title Test for every Page**: When selected, the Web Functional Test recorder automatically inserts a page title test for every page recorded. The page title test compares the recorded page title to the page title received during playback. The default test does not stop playback if the page title comparison fails. When cleared, page title tests are not inserted during recording.
- **Create HTML Test for every Page**: When selected, the Web Functional Test recorder automatically inserts an HTML test for every page recorded. The HTML test compares the recorded HTML to the HTML received during playback. The default test stops playback if the HTML comparison fails. Select the result in the **Results** view and view the differences in the **Comparison** tab of the **Details** view. When cleared, page HTML tests are not inserted during recording.
- **Record actions on "AccElements"**: When selected, actions on Accessibility elements are recorded. When cleared, actions on Accessibility elements are not recorded.

2.5.8.2 Object Identification

This tab lets you specify recording preferences for the Web Functional Test module.

Active Profile: Specifies which object identification profile to use as the active profile when recording scripts. Profiles define a specific set of object identifiers to use when recording Web functional tests. Use the **Add Profile** option to create a new custom profile. Once you have created a profile, select the profile name in the **Name** column and use **Add Object** to define custom objects and attributes in the custom profile.

Name: Shows the name(s) of the defined Web object identifiers.

Attributes: Shows the pattern(s) specified for the defined Web object identifiers.

Add Profile: Opens a dialog box for specifying a new Web object identifier profile.

Add Object: Opens a dialog box for specifying a new Web object identifier.

Edit: Opens a dialog box for editing the selected Web object identifier or profile.

Delete: Deletes the selected Web object identifier or profile. The default profile cannot be deleted.

Export: Opens a dialog box for exporting the currently selected Web object identifier profile to an XML file. Select the profile name in the **Name** column to activate the export option.

Import: Opens a dialog box for importing the currently selected Web object identifier profile to an XML file. Select a profile name in the **Name** column to activate the import option.

Revert: reverts the default Web object identification profile to the default profile. Any changes to the default profile are removed. Select the default profile name in the **Name** column to activate the revert option.

2.5.9 Web Services Preferences

This tab lets you specify recording preferences for the Web Services module. The resulting dialog box displays the following sections and fields:

2.5.9.1 General

This tab lets you specify the general browser recorder settings.

Request Timeout: Specifies the amount of time in seconds to wait for a response to a request before timing out.

Generate default values for requests: When selected, the OpenScript XML parser generates the specified values for the primitive data types by default. The values may be empty. (For other Axis or Oracle parsers the parameters of the methods must be specified.)

- **xsd:string**: Specifies the default value for String data type parameters.
- **xsd:int**: Specifies the default value for Integer data type parameters.
- **xsd:long**: Specifies the default value for Long data type parameters.
- **xsd:float**: Specifies the default value for Float data type parameters.
- **xsd:double**: Specifies the default value for Double data type parameters.
- **xsd:boolean**: Specifies the default value for Boolean data type parameters.

2.5.9.2 Parser Tools

This tab lets you specify additional Apache AXIS parsers to use with the Web Services module.

Apache AXIS 1.X: Specifies the root folder of the Apache AXIS 1.X implementation of the SOAP ("Simple Object Access Protocol") parser. Download "AXIS 1.4 Final" binary ZIP file (axis-bin-1_4.zip) from `http://ws.apache.org/axis/`, unpack the zip file, and then specify the AXIS 1.X root folder using the **Browse** button.

Apache AXIS 2: Specifies the root folder of the Apache AXIS 2 implementation of the SOAP ("Simple Object Access Protocol") parser. Download the AXIS 2 Standard Binary Distribution ZIP file (axis2-1.3-bin.zip) from `http://ws.apache.org/axis2/`, unpack the zip file, and specify the AXIS 2 root folder using the **Browse** button.

2.5.9.3 Proxy Configuration

The Web Services module uses the integrated HTTP Proxy recorder to record SOAP/HTTP protocol requests. Specify the proxy settings for the parsers to be able to parse the internet WSDL file from an internal network using the HTTP Record Preferences Proxy Settings tab.

2.5.9.4 Certificates

The Web Services module uses the integrated HTTP module to specify certificates. Specify the certificate settings using the HTTP Record Preferences Certificates tab.

2.6 Step Group Category

This category lets you specify script step group creation, naming, and numbering preferences. Step groups allow you to optionally organize your OpenScript script commands into logical groupings based on the type of script you are creating. If step groups are enabled during recording, your script commands will be listed within a step group node (or sections) in the tree view (or code view) of the script. Step groups can also be added or modified manually or completely disabled if you prefer not to use them.

2.6.1 ADF Load Test Preferences

This dialog box lets you specify how step groups are created for ADF Load Tests. See the Basic module Step Group preferences on on page 2-34 for additional information. The resulting dialog box displays the following options:

Step Creation: Specifies if step groups are created or not by default during script recording.

- **ADF Load**: When selected, step groups will be created based on the ADF component. Groups are created whenever an ADF component is changed.
- **Based on time threshold**: When selected, step groups are created based upon the specified recording time threshold. Specify the **Threshold** time value in seconds.
- **By page navigation**: When selected, step groups are created based upon page navigation in the browser.
- **Do not create steps**: When selected, step groups will not be created automatically during script recording.

Step Naming: Specifies if step groups are named or not by default during script recording.

- **By page title**: When selected, step groups are named based upon the title of the Web page as defined in the HTML <Title> tag for the main page and the page URL will also be shown in parentheses in the step group name. If a title is not specified then the step group will be named "No Title" but URL will still be displayed.
- **ADF Component**: When selected, step groups are named based upon the component title specified by the ADF JavaScript. If no title is found in the ADF Javascript, step groups are named based upon the title of the Web page as defined in the HTML <Title> tag. If no is title found, step groups are named based upon the subsequent child elements.
- **Do not name steps**: When selected, step groups will not be named automatically during script recording.

Step Numbering: Specifies if step groups are numbered or not by default during script recording.

- **Auto number**: When selected, step groups are numbered sequentially starting with step 1.
- **Do not number steps**: When selected, step groups will not be numbered automatically during script recording.

2.6.2 Basic Module Preferences

This dialog box lets you specify how step groups are created. The resulting dialog box displays the following options:

Step Creation: Specifies if step groups are created or not by default during script recording.

- **Based on time threshold**: When selected, step groups are created based upon the specified recording time interval threshold. Specify the **Threshold** time value in seconds. Script commands that occur within the specified time interval relative to each other, will be organized into the same step group. For example, if a user performs multiple actions on a page within the specified time interval would result in those action commands being grouped into the same step group. This may be useful for grouping commands into step groups for AJAX applications where full Web page transitions may not occur which would allow you to group commands by page.
- **Do not create steps**: When selected, step groups will not be created automatically during script recording.

Step Naming: Specifies if step groups are named or not by default during script recording. Step names will be displayed in the step nodes of the tree view and also shown in the code view. Step names can also be edited manually in either view.

- **Do not name steps**: When selected, step groups will not be named automatically during script recording.

Step Numbering: Specifies if step groups are numbered or not by default during script recording.

- **Auto number**: When selected, step groups are numbered sequentially starting with step 1.
- **Do not number steps**: When selected, step groups will not be numbered automatically during script recording.

2.6.3 Flex (AMF) Load Test Preferences

This dialog box lets you specify how step groups are created for Flex (AMF) Load Tests. See the Basic module Step Group preferences on on page 2-34 for additional information. The resulting dialog box displays the following options:

Step Creation: Specifies if step groups are created or not by default during script recording.

- **Based on time threshold**: When selected, step groups are created based upon the specified recording time threshold. Specify the **Threshold** time value in seconds.
- **By page navigation**: When selected, step groups are created based upon page navigation in the browser.
- **Flex LT (AMF)**: When selected, step groups will be created based on the windows in which the actions occur. Groups are created whenever a Window Activate action is recorded.
- **Do not create steps**: When selected, step groups will not be created automatically during script recording.

Step Naming: Specifies if step groups are named or not by default during script recording.

- **By page title**: When selected, step groups are named based upon the title of the Web page as defined in the HTML <Title> tag for the main page and the page URL will also be shown in parentheses in the step group name. If a title is not specified then the step group will be named "No Title" but URL will still be displayed.

- **Flex LT (AMF)**: When selected, step groups will be named the same as the window titles. Groups will be named the same as the window that is active.
- **Do not name steps**: When selected, step groups will not be named automatically during script recording.

Step Numbering: Specifies if step groups are numbered or not by default during script recording.

- **Auto number**: When selected, step groups are numbered sequentially starting with step 1.
- **Do not number steps**: When selected, step groups will not be numbered automatically during script recording.

2.6.4 HTTP Preferences

This dialog box lets you specify how step groups are created, named, and numbered for HTTP scripts. See the Basic module Step Group preferences on on page 2-34 for additional information. The resulting dialog box displays the following options:

Step Creation: Specifies if step groups are created or not by default during script recording.

- **Based on time threshold**: When selected, step groups are created based upon the specified recording time threshold. Specify the Threshold time value in seconds.
- **By page navigation**: When selected, step groups are created based upon page navigation in the browser.
- **Do not create steps**: When selected, step groups will not be created automatically during script recording.

Step Naming: Specifies if step groups are named or not by default during script recording.

- **By page title**: When selected, step groups include the title of the web page.
- **Do not name steps**: When selected, step groups will not be named automatically during script recording.

Step Numbering: Specifies if step groups are numbered or not by default during script recording.

- **Auto number**: When selected, step groups are numbered sequentially starting with step 1.
- **Do not number step**: When selected, step groups will not be numbered automatically during script recording.

2.6.5 Oracle EBS/Forms Functional Test Preferences

This dialog box lets you specify how step groups are created for EBS Forms Functional Tests. See the Basic module Step Group preferences on on page 2-34 for additional information. The resulting dialog box displays the following options:

Step Creation: Specifies if step groups are created or not by default during script recording.

- **Based on time threshold**: When selected, step groups are created based upon the specified recording time threshold. Specify the **Threshold** time value in seconds.

- **Forms Functional Test**: When selected, step groups will be created based on the windows in which the actions occur. Groups are created whenever a Window Activate action is recorded.
- **Web Functional**: When selected, step groups are created based upon the loading of a new Web page being loaded in the browser. When a new page is finished loading, the page and subsequent user actions performed on that page prior to the next page load will be grouped into the same step group.
- **Do not create steps**: When selected, step groups will not be created automatically during script recording.

Step Naming: Specifies if step groups are named or not by default during script recording.

- **Forms Functional Test**: When selected, step groups will be named the same as the window titles. Groups will be named the same as the window that is active.
- **Web Functional**: When selected, step groups are named based upon the title of the Web page as defined in the HTML <Title> tag for the main page and the page URL will also be shown in parentheses in the step group name, If a title is not specified then the step group will be named "No Title" but the URL will still be displayed.
- **Do not name steps**: When selected, step groups will not be named automatically during script recording.

Step Numbering: Specifies if step groups are numbered or not by default during script recording.

- **Auto number**: When selected, step groups are numbered sequentially starting with step 1.
- **Do not number steps**: When selected, step groups will not be numbered automatically during script recording.

2.6.6 Oracle EBS/Forms Load Test Preferences

This dialog box lets you specify how step groups are created for EBS Forms Load Tests. See the Basic module Step Group preferences on on page 2-34 for additional information. The resulting dialog box displays the following options:

Step Creation: Specifies if step groups are created or not by default during script recording.

- **Based on time threshold**: When selected, step groups are created based upon the specified recording time threshold. Specify the **Threshold** time value in seconds.
- **By page navigation**: When selected, step groups are created based upon page navigation in the browser.
- **Oracle EBS/Forms Load**: When selected, step groups will be created based on the windows in which the actions occur. Groups are created whenever a Window Activate action is recorded.
- **Do not create steps**: When selected, step groups will not be created automatically during script recording.

Step Naming: Specifies if step groups are named or not by default during script recording.

- **By page title**: When selected, step groups are named based upon the title of the Web page as defined in the HTML <Title> tag for the main page and the page URL

will also be shown in parentheses in the step group name. If a title is not specified then the step group will be named "No Title" but URL will still be displayed.

- **Oracle EBS/Forms Load**: When selected, step groups will be named the same as the window titles. Groups will be named the same as the window that is active.
- **Do not name steps**: When selected, step groups will not be named automatically during script recording.

Step Numbering: Specifies if step groups are numbered or not by default during script recording.

- **Auto number**: When selected, step groups are numbered sequentially starting with step 1.
- **Do not number steps**: When selected, step groups will not be numbered automatically during script recording.

2.6.7 Siebel Functional Test Preferences

This dialog box lets you specify how step groups are created for Siebel Functional Tests. See the Basic module Step Group preferences on on page 2-34 for additional information. The resulting dialog box displays the following options:

Step Creation: Specifies if step groups are created or not by default during script recording.

- **Based on time threshold**: When selected, step groups are created based upon the specified recording time threshold. Specify the **Threshold** time value in seconds.
- **Web Functional**: When selected, step groups are created based upon the loading of a new Web page being loaded in the browser. When a new page is finished loading, the page and subsequent user actions performed on that page prior to the next page load will be grouped into the same step group.
- **Do not create steps**: When selected, step groups will not be created automatically during script recording.

Step Naming: Specifies if step groups are named or not by default during script recording.

- **Siebel Functional**: When selected, step groups will be named based upon the Siebel URL pattern. OpenScript uses a heuristic to evaluate the recorded URL of pages to determine a meaningful title.
- **Web Functional**: When selected, step groups are named based upon the title of the Web page as defined in the HTML <Title> tag for the main page and the page URL will also be shown in parentheses in the step group name, If a title is not specified then the step group will be named "No Title" but the URL will still be displayed.
- **Do not name steps**: When selected, step groups will not be named automatically during script recording.

Step Numbering: Specifies if step groups are numbered or not by default during script recording.

- **Auto number**: When selected, step groups are numbered sequentially starting with step 1.
- **Do not number steps**: When selected, step groups will not be numbered automatically during script recording.

2.6.8 Siebel Load Test Preferences

This dialog box lets you specify how step groups are created, named, and numbered for Siebel scripts. See the Basic module Step Group preferences on on page 2-34 for additional information. The resulting dialog box displays the following options:

Step Creation: if step groups are created or not by default during script recording.

- **Based on time threshold**: When selected, step groups are created based upon the specified recording time threshold. Specify the Threshold time value in seconds.
- **By page navigation**: When selected, step groups are created based upon page navigation in the browser.
- **Do not create steps**: When selected, step groups will not be created automatically during script recording.

Step Naming: Specifies if step groups are named or not by default during script recording.

- **By Siebel URL Pattern**: When selected, step groups will be named based upon the Siebel URL pattern. OpenScript uses a heuristic to evaluate the recorded URL of pages to determine a meaningful title.
- **By page title**: When selected, step groups include the title of the web page.
- **Do not name steps**: When selected, step groups will not be named automatically during script recording.

Step Numbering: Specifies if step groups are numbered or not by default during script recording.

- **Auto number**: When selected, step groups are numbered sequentially starting with step 1.
- **Do not number step**: When selected, step groups will not be numbered automatically during script recording.

2.6.9 Web Functional Test Preferences

This dialog box lets you specify how step groups are created for Web Functional Tests. See the Basic module Step Group preferences on on page 2-34 for additional information. The resulting dialog box displays the following options:

Step Creation: Specifies if step groups are created or not by default during script recording.

- **Based on time threshold**: When selected, step groups are created based upon the specified recording time threshold. Specify the **Threshold** time value in seconds.
- **Web Functional**: When selected, step groups are created based upon the loading of a new Web page being loaded in the browser. When a new page is finished loading, the page and subsequent user actions performed on that page prior to the next page load will be grouped into the same step group.
- **Do not create steps**: When selected, step groups will not be created automatically during script recording.

Step Naming: Specifies if step groups are named or not by default during script recording.

- **Web Functional**: When selected, step groups are named based upon the title of the Web page as defined in the HTML <Title> tag for the main page and the page URL

will also be shown in parentheses in the step group name, If a title is not specified then the step group will be named "No Title" but the URL will still be displayed.

- **Do not name steps**: When selected, step groups will not be named automatically during script recording.

Step Numbering: Specifies if step groups are numbered or not by default during script recording.

- **Auto number**: When selected, step groups are numbered sequentially starting with step 1.
- **Do not number steps**: When selected, step groups will not be numbered automatically during script recording.

2.7 Setting Project Preferences

To set Project preferences:

1. Start OpenScript.
2. Switch to the Developer Perspective.
3. Select **Preferences** from the **Window** menu.
4. Expand the desired node and select the Preferences category.
5. Specify the preferences as necessary for the selected category.

3
Creating and Modifying Scripts

This chapter explains the procedures for creating and modifying basic scripts in OpenScript. The module chapters provide additional information about creating scripts using the features and capabilities provided within specific modules.

3.1 Creating Repositories and Workspaces

Repositories and workspaces store project related script files and results log files. You can use repositories and workspaces to organize your various testing projects. OpenScript lets you create multiple workspaces.You can create repositories to organize the storage of your script projects.

> **Note:** Any scripts you plan to run, along with any associated assets, in the Oracle Load Testing application must be stored in a repository/workspace.

A repository is the directory location where you store workspaces. Workspaces are user-specified subdirectories of the repository. As of version 9.10, OpenScript no longer uses an exclamation point at the end of the directory name to identify the directory as a Workspace directory. Any folder (directory) below the specified repository can be a workspace folder.

When you record and save scripts, or play back a script and save the log file, OpenScript stores the script or log file in the specified Workspace.

OpenScript does not create any new repository if you have at least 1 repository kept from a previous installation. If there was no Open Script installed on this machine by the current User, then OpenScript will create a repository named "Default" in the location <*installDir*>/OFT. You can create your own repositories and workspaces using OpenScript.

Repositories specify the location to use to store scripts and related asset files. Repositories also provide a way to share files between OpenScript and Oracle Load Testing. Oracle Load Testing requires that all assets live inside of a named Repository. Oracle Load Testing will not be able to find an asset located in the local file system outside of a repository. Any shared directory can be used as a repository. However, all repositories shared between Oracle Load Testing, Oracle Test Manager, OpenScript, and team members must share the same repository name. For example, if one member of a team calls a shared repository `SharedRepo1`, but another member of a team calls the same shared repository `Shared_Repository_1`, it is possible that some script assets may not be found when the team members share scripts.

To reduce the chance of local repository name conflicts, it is recommended that you create a new local repository named something unique to the user, such as *<machineName>.<windowsUserName>.MyRepository*. Store in this folder all scripts that are not intended to be shared among team members.

Best Practices:

- Always store scripts and assets (i.e. databanks, .JAR files, etc.) inside named repositories.
- Avoid selecting the **Save path relative to current script** option in OpenScript when saving scripts.
- Establish a consistent repository naming scheme across all Oracle Load Testing, Oracle Test Manger, and OpenScript installations.
- Avoid using the repository named "Default" for storing local scripts. Use "*machineName*.Default" instead.
- **Do not** use the file system (i.e. Windows Explorer) to copy/move script folders or files. Scripts are Eclipse bundles (with unique IDs) and have file dependencies within script folders. If you need to move scripts, use **Save As** on the OpenScript **File** menu. Use the **Manage** options on the **Tools** menu to rename scripts and manage folders and repositories.

3.1.1 Creating a Repository

To create a repository:

1. Select **OpenScript Preferences** from the **View** menu.
2. Expand the OpenScript node.
3. Expand the General node.
4. Select the Repository node.
5. Click **Add**.

   This dialog box lets you specify the name and location of the repository to use to store script files.
6. Enter a repository name. The name is required.

   Name: Enter any name to identify the repository.

   > **Note:** If you plan to use OpenScript scripts with Oracle Load Testing, the repository names you specify should match the repository name specified in Oracle Load Testing (including case).

7. Enter the drive and directory location or click **Browse** to select the location to use for the repository.

   Location: Enter the drive and directory path to the repository or use the Browse button to select a location. The location must be a valid drive and directory path.
8. Click **OK** to add the new repository to the list of repositories.
9. Click **OK** to close the preferences.

When you create new a script project, you can select the repository to use to store the project.

3.1.2 Managing Repositories

To add, edit, or delete repositories:

1. Select **Manage Repositories** from the **Tools** menu.
2. Select the repository where you want to create the workspace.
3. Click the **Add**, **Edit** or **Delete** buttons to manage repositories.
4. Click **Close** when finished.

3.1.3 Managing Folders (Workspaces)

When starting a new testing project, you should create a project-specific workspace folder to store related files.

To create, rename, or delete workspace folders:

1. Select **Manage Folders** from the **Tools** menu. OpenScript opens a dialog box for managing workspace folders in repositories.
2. Expand the tree and select the workspace folder to manage.
3. Click **New**, **Rename**, or **Delete** buttons to manage workspace folders.
4. Click **Close** when finished.

3.1.4 Managing Scripts

To rename or delete scripts:

1. Select **Manage Scripts** from the **Tools** menu.
2. Select the script.
3. Click the **Rename** or **Delete** buttons to manage script files.
4. Click **Close** when finished.

3.2 Creating a Script Project

You must create a script project to generate the basic structure that you can then customize.

To create a script project:

1. Select **New** from the **File** menu.
2. Expand a group node and select the type of asset or script to create:

   Functional Testing (Browser/GUI Automation): The Functional Testing group contains the following script types:

   - **Adobe Flex**: This option lets you create a new script for automated functional testing of web applications that use the Adobe Flex Automation Framework at the browser/gui level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded Web navigations based upon the defined Web Functional test Step Group preferences and the Web navigations and Flex actions performed during recording. You can edit the script tree or Java code to customize the script.
   - **Oracle EBS/Forms**: This option lets you create a new script for automated functional testing of Oracle E-Business Suite and other applications that utilize Web and Oracle Forms components at the browser/gui level. The resulting

script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded Web navigations based upon the defined Step Group preferences and the Web navigations and Forms actions performed during recording. You can edit the script tree or Java code to customize the script.

- **Oracle Fusion/ADF**: This option lets you create a new script for automated functional testing of Oracle Application Development Framework (ADF)-based applications and other applications that utilize Web and ADF components at the browser/gui level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded Web navigations based upon the defined Step Group preferences and the Web navigations and ADF actions performed during recording. You can edit the script tree or Java code to customize the script.
- **Oracle JD Edwards EnterpriseOne**: This option lets you create a new script for automated functional testing of Oracle JD Edwards EnterpriseOne applications that utilize Web and JD Edwards EnterpriseOne Grid Control components at the browser/gui level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded Web navigations based upon the defined Step Group preferences and the Web navigations and Grid Control actions performed during recording. You can edit the script tree or Java code to customize the script.
- **Siebel**: This option lets you create a new script for automated functional testing of Siebel applications that utilize Siebel High Interactivity and Standard Interactivity/Web controls at the browser/gui level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded Web navigations based upon the defined Step Group preferences and the Web navigations performed during recording. You can edit the script tree or Java code to customize the script.
- **Web**: This option lets you create a new script for automated functional testing of Web applications at the browser/gui level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded Web navigations based upon the defined Step Group preferences and the Web navigations performed during recording. You can edit the script tree or Java code to customize the script.

General: The General group contains the following script types:

- **Block Scenario Script**: This option lets you create load scenarios in which they allocate a certain percentage of VUs to run dependent scripts in pre-determined sequences. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain a Run Scenario action that opens the script block details for the script blocks and child scripts that the scenario will group together and run. You can add child scripts as script assets and edit the block details XML tree using a shorcut menu to customize the block scenario. Block Scenarios can are then used in Oracle Load Testing to run the scenario.
- **Database**: This option lets you create the basic structure of a Database script for automated testing of SQL statements to test a database and run them in the Oracle Load Testing application. A basic script structure contains only the Initialize, Run, and Finish nodes. You can use the **Import Database Capture File** option on the **Tools** menu to import an Oracle Database Replay capture workload file, plain SQL and PL/SQL statement .SQL script file, or SQL statements captured and stored in an SQL Tuning Set (STS) into a script.

- **Java Code Script:** This option lets you create a new automated test script using your own custom Java code through the OpenScript Eclipse IDE. A basic script structure contains only the Initialize, Run, and Finish nodes. You can edit the script tree or Java code to develop your own custom script. Java Code scripts are typically used for function libraries.
- **Script from Template**: This option lets you create a new script from a script that has previously been saved as a template script. When you select this option, you select from a list of previously saved template scripts before specifying the name for the new script. The resulting script will contain the Initialize, Run, and Finish nodes and include any custom code that was added to the template script. You can edit the script tree or Java code to customize the script.
- **Web Services**: This option lets you create the basic structure of a Web Services script for automated testing of Web Services at the SOAP/HTTP protocol level. A Web Services script structure contains only the Initialize, Run, and Finish nodes. You can use the WSDL Manager to add WSDL files and edit the script tree or Java code to customize the script. If you have a Web Services client application written already that communicates over HTTP and which communicates through a proxy, you can record the traffic using the OpenScript HTTP recorder.

Load Testing (Protocol Automation): The Load Testing group contains the following script types:

- **Adobe Flex (AMF)**: This option lets you create a new script for load testing of Web applications that utilize HTTP and the Adobe Flex Action Message Format (AMF) protocols at the protocol level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded Flex AMF and HTTP protocol navigations based upon the defined Step Group preferences and the navigations protocol for actions performed during recording. You can edit the script tree or Java code to customize the script.
- **Oracle EBS/Forms**: This option lets you create a new script for load testing of Oracle E-Business Suite and other applications that utilize HTTP and Oracle Forms (NCA) protocols at the protocol level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded HTTP protocol navigations based upon the defined Step Group preferences and the navigations and Forms protocol for actions performed during recording. You can edit the script tree or Java code to customize the script.
- **Oracle Fusion/ADF**: This option lets you create a new script for load testing of Oracle Application Development Framework (ADF)-based applications and other applications that utilize HTTP and ADF protocols at the protocol level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded HTTP protocol navigations based upon the defined Step Group preferences and the navigations and ADF protocol for actions performed during recording. You can edit the script tree or Java code to customize the script.
- **Oracle Hyperion**: This option lets you create a new script for load testing of Oracle Hyperion-based applications and other applications that utilize HTTP and Hyperion correlation rules at the protocol level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded HTTP protocol navigations based upon the defined Step Group preferences and the navigations andHyperion correlation rules for actions performed during recording. You can edit the script tree or Java code to customize the script.

- **Oracle JD Edwards EnterpriseOne**: This option lets you create a new script for load testing of Oracle JD Edwards EnterpriseOne-based applications at the HTTP protocol level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded HTTP protocol navigations based upon the defined HTTP Step Group preferences and the JD Edwards Load Test Module correlation library. You can edit the script tree or Java code to customize the script.
- **Oracle PeopleSoft**: This option lets you create a new script for load testing of Oracle PeopleSoft-based applications and other applications that utilize HTTP and PeopleSoft correlation rules at the protocol level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded HTTP protocol navigations based upon the defined Step Group preferences and the navigations andPeopleSoft correlation rules for actions performed during recording. You can edit the script tree or Java code to customize the script.
- **Siebel** This option lets you create a Siebel script structure of a new OpenScript script project. A Siebel script lets you record Siebel Web navigations using a browser for load testing Siebel applications. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded HTTP protocol navigations based upon the defined Step Group preferences and the Web and Siebel navigations performed during recording. You can edit the script tree or Java code to customize the script.
- **Web/HTTP** This option lets you create a new script for load testing of Web Applications at the HTTP protocol level. The resulting script will contain the Initialize, Run, and Finish nodes. The Run node will contain recorded Web navigations based upon the defined Step Group preferences and the Web navigations performed during recording. You can edit the script tree or Java code to customize the script. This script type has two recording modes: Web and HTTP. The recording mode is specified in the HTTP module Recording preferences, which you should set before recording new scripts. See Section 2.5.2, "HTTP Preferences" for additional information about setting the recording mode.

Script Asset: The Script Asset group contains the following script asset types:

- **Databank**: This option lets you create a new databank or open an existing databank file. The new asset wizard lets you navigate to the databank file location of an existing databank file or enter the name of a new databank file. When you click Finish in the wizard, the existing or new databank file opens in a text editor view.
- **Object Library**: This option lets you create a new Object Library or open an existing Object Library. The new asset wizard lets you navigate to the Object Library file location of an existing Object Library file or enter the name of a new Object Library file. When you click Finish in the wizard, the existing or new Object Library file opens in the Object Library editor view.

3. Click **Next**.
4. Select the location where you want to store the script project. Scripts can be stored in repositories and workspaces. Load test scripts developed for use with Oracle Load Testing must be stored in a repository/workspace.
   - **Path**: Shows the file path of the selected repository/workspace.

- **My Repositories**: Specifies the repository where the script project will be saved. Select a repository and workspace from the tree. Repositories can be managed using **Manage Repositories** on the **Tools** menu.
- **[file list]**: List the names of the existing files or scripts in the selected repository/workspace.
- **Script**: Specify a name for the script project. The script name is required and must be unique.

5. Enter a script name.
6. Click **Finish** to create a script or select **Create as Function Library script** and click **Next** to create a function library.

   For Java Code Scripts, a basic script tree will be created in the script view. You can edit the Java code in the code view. For module scripts, a script tree will be created in the script view. After you record the script, the tree view will contain the navigations and actions depending upon the type script.

   For existing scripts, the file concurrency control prevents multiple users from editing the same script. If you try to open a script that is in use by another user, the script copy wizard opens and you will be asked if you want to make a copy of the script and additional files.
7. If you are creating a function library, enter a unique Package Name and Class Name to identify the library. The Package Name and Class Name must be unique among all function libraries used by a team. The Package Name and Class Name name should conform to the syntax of a Java class name, and must not contain Double-Byte Character Sets (DBCS). The suggested format for the Package name is "lib.orgName.groupName.subgroupName", for example: "lib.oracle.oats.dev.TestLibrary". The Class Name should be:
   - meaningful and provide context as to the purpose of the library,
   - clear and concise so it is easy to read and type in scripts,
   - something unique so it is not confused with other function libraries.
8. Click **Finish** to create the function library. If an existing script is converted to a function library, the initialize, run, and finish sections will appear in the function library, but they only can be called as any other method in the library from caller script.

   When creating function libraries, you can use the menu options to add functions to the library file. Select the **Script** menu and then select **Other** from the **Add** sub menu. The procedure is similar to adding custom functions to a script. See Section 3.3.6, "Using a Script as a Dedicated Function Library" for additional information about creating a function library.

3.2.1 Recording Scripts

OpenScript includes Record capabilities for recording user actions within the application-under-test. After creating a script project, you can use the Record toolbar button or **Record** menu option on the **Script** menu to start the recorder for the type of script project. You can specify which section of the script (Initialize, Run, Finish) in which to record using the **Set Record Section** menu option on the **Script** menu.

When you start recording, OpenScript launches a browser instance along with the OpenScript Browser Helper Object (BHO) for the type of script being recorded. You can then start the application-under-test and begin recording user actions. Recorded actions will appear in the script tree and in the Java code view. A floating toolbar

appears with toolbar buttons for common script recording actions such as adding tests, pause and resume, and stop recording.

Some basic script recording tips include:

- Non-Administrator users running OpenScript on Vista, Windows 7, Windows 2008, Windows 2008r2 with Internet Explorer browsers may encounter recording and playback problems due to account privileges.

  Users in the Administrators Group, but not the exact Administrator, can use "Run As Administrator" if any recording and playback problems are encountered.

  Users belonging to Non-Administrators Group may have more problems because of access privileges to the registry or file system. These users can try the following to work around recording and playback issues:

  – add "about:blank" and the applications/web sites under testing into Trusted sites.

  – turn off the "Protected Mode" settings of the related zone (Internet, Local intranet, Trusted sites, Restricted sites). In Internet Explorer, select **Internet Options** from the **Tools** menu, then select the **Security** tab and clear the **Enable Protected Mode** option.

  If these workarounds do not resolve recording and playback issues for Non-Administrator users, the users will need to get administrator privilege to "Run As Administrator" (Administrator password needed).

- The browser instance launched by OpenScript is index `0`. Additional tabs or browser windows are indexed incrementally as they are opened. The index value is one of the identifier values used in the script Java code for object identification.
- If you need to access another application while recording is paused or stopped, you should open a new browser window rather than use a new tab in the browser used for recording.
- Make sure the browser zoom setting is set to 100%. Record and playback of functional test scripts at other zoom settings is not supported and may not work correctly.

3.2.1.1 Recording Scripts from Another Machine

OpenScript can record requests originating from another machine or device (phone, tablet pc, etc.) which has the proxy in Internet Explorer configured to use the IP address of the OpenScript machine.

For example, with two machines as machine A and machine B:

On machine A,

- launch OpenScript,
- create an HTTP script,
- select **OpenScript Preferences** from the **View** menu,
- expand the Record section and select HTTP,
- select the **General** tab,
- clear **Only record requests originating from the local machine**,
- start recording.

On machine B,

- launch Internet Explorer browser,
- set the proxy in Internet Explorer to the IP address of machine A,
- open a web site in the browser and navigate pages.

OpenScript records HTTP requests that originate from machine B.

3.2.2 Opening Existing Scripts

The introduction of Script Assets (in Script Properties) requires pre-version 9.10 scripts to be migrated to the current version of 9.10 or higher. This section provides information about backwards compatibility of OpenScript scripts and upgrading OpenScript scripts.

Scripts created in older versions of OpenScript will always run in new versions of the product without modification from the command-line, Oracle Load Testing, and Oracle Test Manager.

Older OpenScript scripts may not be opened or played back in the newer version of the OpenScript User Interface without upgrading them first.

Previously published script API functions are supported in the latest release. Some published API may be marked as deprecated, but will still work in the new release in order to maintain backwards compatibility.

3.2.2.1 Opening Older Scripts in OpenScript

OpenScript requires that scripts be upgraded to the latest release in order to open them in the OpenScript User Interface. You are not required to upgrade a script to the new version unless you want to open the script in the OpenScript User Interface. Older versions of OpenScript scripts can be run without modification from the command-line, Oracle Load Testing, and Oracle Test Manager. However, for version 9.0x scripts, you must maintain the Repository/Workspace structure as `repositoryLocation/workspace!/script`. It is not possible to copy 9.0x scripts directly to a Version 9.1+ repository.

Caution: Version 9.10 and higher scripts cannot be played back in earlier versions of OpenScript, Oracle Load Testing, and Oracle Test Manager. If you want to maintain pre-version 9.10 scripts, you should make a back up copy of your scripts *before* opening and saving them in version 9.10 or higher. OpenScript automatically migrates any pre-version 9.10 scripts when the script is opened and saved in OpenScript version 9.10 or higher.

OpenScript automatically prompts you to upgrade older version scripts to the current version whenever the script is opened in the OpenScript User Interface. When opening an older script, you can choose not to open the script and the script will not be upgraded.

When prompted to upgrade a script, if the script depends on any child scripts or function libraries, OpenScript provides an option to upgrade the child scripts or function libraries to the new version also.

Once a script is upgraded to a new release, the script cannot be opened or run using older versions of Oracle Application Testing Suite (OpenScript, Oracle Load Testing, or Oracle Test Manager).

3.2.2.2 Migrating Older Scripts in OpenScript

If you wish to upgrade scripts without opening them individually in OpenScript, you can use the **Migrate Scripts** upgrade option on the **Tools** menu. The Migrate Scripts tool lets you migrate pre-version 9.10 scripts to the current version without having to open scripts individually.

The Migrate Scripts tool provides options for migrating top-level scripts and locating all dependent child scripts. The Migrate Scripts tool lets you select which scripts to migrate to the current version and find any child scripts that also need to be migrated.

Since version 9.10, scripts that will be run in Oracle Load Testing may not specify absolute paths for their repositories or script assets. However, if your pre-9.10 scripts use absolute paths, you may continue to run the same scripts, unmodified, in the current version of Oracle Load Testing. As soon as you upgrade the pre-9.10 scripts to the current version using either the OpenScript User Interface or the Migrate Script tool, the script will not playback in Oracle Load Testing until the absolute paths are changed to relative paths. The Migrate Scripts tool does not migrate absolute paths to relative paths or to repository paths. The absolute paths must be changed in the scripts manually.

3.2.2.3 Running Mixed Versions of Scripts

You are advised not to run mixed versions of "job" scripts where a parent script calls child scripts or function libraries. This may happen in cases where you may have 9.1x "parent" scripts that run 9.0x "child" scripts or function libraries. Although this configuration has been tested and is supported, the combination of mixed versions scripts may lead to unpredictable results and some confusion as to which scripts are the latest version. In addition, mixed version job scripts may not be able to take advantage of certain new version 9.10 improvements, such as:

- Version 9.10 provides an option to visually inspect and add child script functions into a parent script. If child scripts are not upgraded to 9.10, OpenScript will not display their available functions in the user interface options.
- Version 9.10 scripts no longer require that parent scripts add all child script databanks as their own databanks. If child scripts are not upgraded to 9.10, then parent scripts still must have child script databanks added as their own databanks.

3.2.2.4 Multiple Users Opening Scripts

For existing scripts, the file concurrency control prevents multiple users from editing the same script. If you try to open a script that is in use by another user, The script copy wizard opens and you will be asked if you want to make a copy of the script and additional files.

3.2.3 Exporting and Importing Scripts

The **Export Script** and **Import Script** options on the **File** menu provide a way to create a ZIP archive of a script and its resource files and move it to another machine running another instance of OpenScript.

3.2.3.1 Exporting Scripts

To export a script to ZIP archive file:

1. Open the script.
2. Select **Export Script** from the **File** menu.

This dialog box lets you export the currently open script to a ZIP archive file. The ZIP archive file can be used to move a script from one system to another as ZIP file that includes the selected script resources. The Export Script to ZIP Archive dialog box has the following options:

- **Filename**: Specifies the file name to use as the ZIP archive file. Enter a file name or click **Browse** to select an existing file to overwrite. By default, the ZIP archive file name is the same name as the OpenScript script.
- **Create self-contained zip file**: When selected, the zip archive file includes all necessary script resources required to import and play back the script on another machine. The Playback Settings and DataBank **Additional Files to export** options are automatically selected. Other **Additional Files to export** options can be set as desired. The script folder name in the ZIP archive files is converted to *`scriptname`*`.all` with subfolders for the script and resource files to prevent overwriting of an existing script with the same name when the ZIP archive is imported into an OpenScript workspace. When cleared, the ZIP archive file will contain the script files and only the selected **Additional Files to export**. The script files are stored in the `script` folder within the ZIP archive file.
- **Additional Files to export**: Specifies which script resource files to include in the ZIP archive file, as follows:
  - **Playback Settings**: When selected, the ZIP archive file includes the `playbackSettings.properties` file. This option is selected by default if the **Create self-contained zip file** option is selected.
  - **Recorded Data**: When selected the ZIP archive file includes a recordedData folder under the *`scriptname`*`.all` or `script` folder containing the recorded content and screen shots. When cleared, the recorded data is not included in the ZIP archive.
  - **Playback Results**: When selected the ZIP archive file includes a results folder containing `Session` results folders for each script playback. Each `Session` folder contains files for content source, request headers, content text, request text, and a data folder. The data folder contains log files, segment information, and service properties. When cleared, no playback results are included in the ZIP archive file.
  - **Databank**: When selected the ZIP archive file includes a databanks folder containing any databank files attached to the script as Script Assets. When cleared, no databanks are included in the ZIP archive file. This option is selected by default if the **Create self-contained zip file** option is selected.
  - **Error Log**: When selected the ZIP archive file includes any error logs for the script. When cleared, no error logs are included in the ZIP archive file.
- **Select All**: Selects all **Additional Files to export** options.
- **Deselect All**: Clears the **Additional Files to export** options. If **Create self-contained zip file** is selected, only Recorded Data, Playback Results, and Error Log options are cleared. When **Create self-contained zip file** is selected the Playback Settings and Databank options are selected by default and cannot be cleared.

3. Change the file name, leave the default name, or click **Browse** to select an existing file to overwrite.
4. Select or clear the **Create self-contained zip file** option.
5. Select or clear the **Additional files to export** options.

6. Click **OK**.

3.2.3.2 Importing Scripts

To import a previously created ZIP archive file into an OpenScript workspace:

1. Select **Import Script** from the **File** menu.

   This dialog box lets you select a previously exported ZIP archive file into an OpenScript workspace. The Import Script dialog box has the following options:

   - **Filename**: Specifies the file name of the ZIP archive file to import. Click **Browse** to select an existing file.
   - **Type**: Shows the type of the selected ZIP archive file to import: Single Script or Self-contained zip file.
   - **Path**: Shows the drive and directory location of the repository. ZIP archive files must be imported into an OpenScript repository and workspace. Click the Up one level button or select the workspace from the repository tree to where the ZIP archive will be imported.
   - **Script**: Shows the name that will be assigned to the script after import of the ZIP archive file. The default is the same name as the ZIP archive file. However, the name of the script within the ZIP archive file may be different.
   - **Subfolder**: Specifies the name of the subfolder where the script files will be stored after import from the ZIP archive file. The subfolder only applies to Self-contained zip file archive files. The default is *name*.All.

2. Click **Browse** to select an existing ZIP archive file.
3. Select the workspace where you want to import the archived script.
4. If the ZIP archive file is Self-contained zip file, optionally specify the name of the subfolder.
5. Click **OK**. The archived script and related files are extracted from the ZIP file and the script opens in the OpenScript workbench.

3.2.4 Migrating Scripts

To migrate scripts to the current version:

1. Select **Migrate Scripts** from the **Tools** menu.

   This dialog box lets you migrate pre-version 9.10 OpenScript scripts to the current version. The Prompt on the top of the dialog box shows if the selected script is current or should be migrated and prompts for the appropriate action. The Script Migration Manager has the following options:

   - **Path**: Shows the file path of the selected repository/workspace.
   - **My Repositories**: Specifies the repository for selecting scripts to migrate or search. Select a repository and workspace from the tree.
   - **[file list]**: List the names of the existing files or scripts in the selected repository/workspace.
   - **Script**: Specify a name of a script to migrate or search.
   - **Migrate**: When enabled, the selected script is a pre-version 9.10 script and can be migrated to the current version. When disabled, the script is already a version 9.10 or higher script and doe not require migration.

- **Find child scripts**: When enabled, the selected script is a pre-version 9.10 script and can be migrated to the current version. When disabled, the script is already a version 9.10 or higher script and does not require migration.

2. Expand the My Repositories tree to navigate to a workspace folder containing the script files.
3. Select the script.

   If **Migrate** is enabled, the script is a pre-version 9.10 and can be migrated. If **Migrate** is disabled, the script is already a version 9.10 or higher script and does not require migration.

   If **Find child scripts** is enabled, the script is already a version 9.10 or higher script and you can use the Find child scripts feature to locate any child scripts that may be assets for the currently selected script. If child scripts are located, you can use the Migrate Child Scripts options to migrate child scripts or search for additional child scripts. If **Find child scripts** is disabled, the script is a pre-version 9.10 script and must be migrated to the current version.
4. Click **Migrate** or **Find child scripts** as required for the selected script file.
5. Click **Close** when finished.

3.2.5 Creating New Scripts from Templates

You can create a new script from any other previously saved script. The resulting script will contain the Initialize, Run, and Finish nodes and include any custom code that was added to the saved script.

To create a new script from a saved script:

1. Select **New** from the **File** menu or click the toolbar button.
2. Expand the General section and select **Script from Template**.
3. Click **Next**.

> **Note:** If you have previously saved an existing script as a template script using an earlier version of OpenScript, you can also create a new script from a template script.

4. Select the script to use to create the new script and click **Next**.
5. Enter a script name for the new script and click **Finish**. If you are creating a new dedicated function library script from a saved dedicated function library script, you will also need to specify a new library class name for the new function library so there will not be duplicated function library classes.

3.2.6 Setting Script Properties

Script properties specify the property settings for a specific script. You can set script properties at any time when a script is open. Script properties include the following:

- Correlation properties for Load Testing (protocol automation)-type scripts.
- Module properties specifying which module services to include with a script.
- Step Group properties specifying how step groups are created during recording.

To set Script Properties:

1. Open or a create a script project.
2. Select **Script Properties** from the **Script** menu.
3. Select the property type in the left pane.
4. Use the options in the right pane to set specific properties.
5. Click **OK** when finished.

The script property panes are described in the following sections.

3.2.6.1 About

This dialog box shows the current script's location and history.

3.2.6.2 Correlation and Validation

This dialog box lets you specify correlation properties for Load Testing (protocol automation)-type scripts. The Correlation pane has the following options:

- **Module**: Specifies the module type the script will use for the correlation rules.
- **Selected Module's Settings**: Shows the current script's Correlation library and rules settings. Expand the tree view to view the selected libraries and rules.
  - **Edit**: Opens the correlation properties window for the specified module type.

3.2.6.3 Modules

This dialog box lets you specify which module services to include with a script. The Modules pane has the following options:

- **Modules**: Shows which module services are included with the current script. The Basic and Utilities modules are common to all script types. The Shared Data module can also be used with all script types. The HTTP module is common to all load testing (protocol automation)-type scripts. The Functional Test and Web Functional Test modules are common to functional test-type scripts. Other modules are specific to a script type.

3.2.6.4 Step Groups

This dialog box lets you specify Step Group properties for the current script. The Step Group pane has the following options:

- **Module:** Specifies the module type the script will use for the step group rules.
- **Selected Module's Settings**: Shows the current script's Step Group settings. The settings are specific to the script type.
  - **Edit**: Opens the Step Group properties window for the specified module type.

3.2.7 Importing Database Capture Files

You can import a Database Replay capture file, plain SQL and PL/SQL statements .SQL script file, or SQL statements captured and stored in an SQL Tuning Set (STS) to generate an OpenScript load testing script that connects to a database and executes the SQL statements.

To set the Database preferences:

1. Select **OpenScript Preferences** from the **View** menu.
2. Expand the OpenScript node.

3. Select the General node.
4. Select the Date Format. See Chapter 2.3.1, "General Preferences" for additional information.
5. Click **OK**.

To create a script from Database Capture file:

1. Select **New** from the **File** menu.
2. Expand the General group and select the Database script type.
3. Click **Next**.
4. Select the repository and workspace where you want to store the script.
5. Enter a script name and click **Finish**. A new basic script project is created in the Script tree.
6. Select **Import Database Capture File** from the **Tools** menu. The Import Database Capture File Wizard opens for specifying the file format type, source file, and SQL statement parameterization for imported database capture files or SQL statement files.
7. Select the Database file format to import:
   - **Database Replay Capture File**: When selected, an Oracle DBReplay capture workload file can be imported.
   - **SQL and PL/SQL Statement Script**: When selected, a plain SQL and PL/SQL statements .SQL script file can be imported.
   - **STS in Oracle Database**: When selected, SQL statements captured and stored in an SQL Tuning Set (STS) can be imported.
8. Specify the maximum number of items to import.
9. Click **Next**.
10. Enter the file path and name of the Database Replay capture file (.rec file extension), SQL and PL/SQL statement script file (.sql file extension), or SQL Tuning Set (STS), or click **Browse** to select the file.
11. Click **Next**.
12. Select the alias name of the database connection to use or click **New** to specify a new connection. If you select **New**, enter the Database Driver, URL, username, password, and alias information. The alias is the name used in the script to reference the database connection. Click **Test** to verify the connection and click **OK** to use the new connection.
13. Select or clear individual SQL statements to include in the script and set the parameterization for the SQL statements.
    - **Edit**: Opens a dialog box for editing a specific SQL query. You can edit the SQL query or use the edit options to Parameterize specific values of the query.
    - **Parameterize**: Creates SQL bindings for all literal parameters of the selected SQL query. This is used to make the database parse the SQL statements more efficiently. For example, an application that makes the following three database queries:

```
BEGIN DBMS_OUTPUT.GET_LINES(1, 100); END;
BEGIN DBMS_OUTPUT.GET_LINES(5, 100); END;
BEGIN DBMS_OUTPUT.GET_LINES(7, 100); END;
```

The above statements require the database to parse the entire query 3 times, even though the structure of the query does not change. It is more efficient for database parsing to pass parameter arguments to the queries. For example:

```
BEGIN DBMS_OUTPUT.GET_LINES(?, ?); END;
BEGIN DBMS_OUTPUT.GET_LINES(?, ?); END;
BEGIN DBMS_OUTPUT.GET_LINES(?, ?); END;
```

When executing the above SQL statements, the application would separately specify the literal parameter values to input to the database (that is: 1, 100, 5, 100, 7, 100).

If you insert a new SQL query manually using **Edit**, the **Parameterize** button parameterizes all literals and named parameters. For example, the following statement:

```
BEGIN DBMS_OUTPUT.GET_LINES(5, :NUMLINES); END;
```

will be parameterized as:

```
BEGIN DBMS_OUTPUT.GET_LINES(?, ?); END;
```

In certain cases, using the **Parameterize** button may produce unwanted parameterization of a literal. For example, in the following statement:

```
SELECT 1 FROM EMPLOYEES
```

will be parameterized as:

```
SELECT ? FROM EMPLOYEES
```

In the above example, if the you did not intend for the "1" to be parameterized, you would have to manually adjust the parameterization using **Revert**.

When importing a DBReplay file, or .SQL file, OpenScript will only parameterize things that are already parameterized in the file itself.

- **Revert**: Reverts the selected parameterized SQL query back to the original imported statement.
- **Check All**: Selects all imported SQL statements.
- **Uncheck All**: Clears the check marks from all imported SQL statements.
- **Automatically Add Row Count Tests**: When selected, a row count test is automatically inserted for each SQL statement added to the script.
- **Parameterize Checke**d: Creates SQL bindings for all literal parameters in all checked statements.
- **Revert Checked**: Reverts to the original imported SQL statement for all checked statements.

14. Click **Finish**.

The Database capture or SQL file import recorder parses the file and generates an OpenScript script using the specified **Parameterize** settings. The script creation time can vary depending upon the size of the capture file.

The resulting script contains the Java code required to connect to the database and execute the SQL queries selected in the import wizard. The script uses the OpenScript `utilities.getSQLService()` methods to execute the SQL statements.

3.2.8 Importing Oracle Real User Experience Insight (RUEI) Session Logs

You can import a RUEI captured user session log file to generate an HTTP-based OpenScript load testing script. The RUEI User Session log must be generated using RUEI version 6 or higher.

To create a script from RUEI user session log:

1. Select **New** from the **File** menu.
2. Expand the Load Testing (Protocol Automation) group and select the Web/HTTP script type.
3. Click **Next**.
4. Select the repository and workspace where you want to store the script.
5. Enter a script name and click **Finish**. A new HTTP protocol script project is created in the Script tree.
6. Select **Import Oracle Real User Experience Insight (RUEI) Session Log** from the **Tools** menu.
7. Enter the file path and name of the RUEI User Session log (.tab file extension) or click **Browse** to select the file.
8. Set the **Correlate script** and **Create step groups** options.
9. Click **OK**.

The RUEI Session log import recorder parses the log file and generates an HTTP-based OpenScript script using the specified **Correlate script** and **Create step groups** settings. The script creation time can vary depending upon the size of the log file and if the **Correlate script** and **Create step groups** settings are set or not. Generally, when **Correlate script** and **Create step group**s are set, the script creation time increases.

The RUEI User Session Log consists of the following files and folder:

- data.tab file: This file contains the url, host and port, method, postdata, etc.
- version.txt: This files contains the export version number which determines the version of the OpenScript RUEI User Session Log importer to use.
- content folder: This folder contains text content that corresponds to the entries in the data.tab file.

To get the necessary files to be exported from RUEI, URL prefix masking should be set to "Complete logging". Complete logging is not turned on by default in RUEI. In addition, the session exported from RUEI should not be older than the Full Session Replay (FSR) setting specified in the Collector data retention policy settings. Otherwise, no (or only partial) data will be available. See the *Oracle Real User Experience Insight User's Guide* for details about URL prefix masking and the Collector data retention policy settings.

3.2.9 Exporting Script Playback Settings

You can export the script playback settings to a properties file. The playback setting properties file lists all of the options and settings that were specified for the script. You can use the properties file for Command Line execution of the script using the `-propertiesPath` setting. See Appendix A, "Script Command Line Reference" for additional information about using the `-propertiesPath` setting.

To export script playback settings:

1. Open the script in OpenScript.

2. Select **Export Playback Settings** from the **Tools** menu.
3. Select the directory location and specify a file name.
4. Click **Save**. The file name is *filename*.properties.

3.3 Modifying Scripts

Once you have created a script project, you can customize the script for your specific testing purposes using the available menu options or editing your own code in the Java Code view.

3.3.1 Adding Step Groups to a Script

Step groups provide a way to group multiple procedures into a single reporting step.

To add a manual step group to a script:

1. Open or create a script project.
2. Select the script node where you want to add the step group.
3. Select the **Script** menu and then select **Step Group** from the **Add** sub menu.

   This dialog box lets you specify or modify a step group node in a script tree.
4. Enter a name for the Step Group.

   Title: Specify the title text of the step group. The title text will appear in the script tree.
5. Enter any think time delay to add to the Step Group.

   Think time: Specify the amount of time in milliseconds to use as a think time delay for the step group or select no delay.

   - **No delay**: When selected, no additional think time delay is added to the beginning of the step group.
   - **Delay [ ] sec**: When selected, the specified amount of think time delay is factored with the **VU Pacing (think time)** playback setting to specify the amount of delay to add to the beginning of the step group.
6. Click **OK**. The Step Group is added to the script tree.

To add a step groups to a script based upon preferences:

1. Open or create a script project.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node.
4. Expand the Record node.
5. Select the Step Groups node.
6. Specify the Step Group preferences and click **OK**.
7. Select **Create Step Groups** from the **Script** menu. The Step Groups will be automatically added to the script tree.
8. In the Java Code view, the step group consists of the code executed between `beginStep` and `endStep`:

```
beginStep("Step Group 1", 10);
{
```

```
        /**
         * Add code to be executed for the step group.
         */
       info("Step Group 1");
}
endStep();
```

3.3.2 Adding a Delay to a Script

To add a delay to a script:

1. Open or create a script project.
2. Select the script node where you want to add the delay.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the General node and select **Think Time**.

   This dialog box lets you specify or modify a delay time in seconds.
5. Enter a valid integer to use as the think time in seconds.
6. Click **OK**. The Think node is added to the script tree.
7. In the Java Code view, the `think(`*`time`*`;)` (the time is in seconds) statement will be added to the script code:

   ```
   think(10.0);
   ```

3.3.3 Adding a Log Message to a Script

To add a log message to a script:

1. Open or create a script project.
2. Select the script node where you want to add the log message.
3. Select the **Script** menu and then select **Message** from the **Add** sub menu.

   This dialog box lets you specify or modify a log message in a script tree.
4. Enter the message text.

   Message: Specify the text of the log message. The text will appear in the Console view on script playback.
5. Click **OK**. The log message node is added to the script tree.
6. In the Java Code view, the *`type`*`("`*`log message`*`")` method will be added to the script code:

   ```
   info("Message");
   warn("Message");
   fail("Message");
   ```

The log message text appears in the Console View when the script is played back. The `info()` method is similar to a `printf` command.

3.3.4 Adding a For Statement to a Script

To add a For statement to a script:

1. Open or create a script project.
2. Select the script node where you want to add the For statement.

3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Control Statements node and select **For**.

   This dialog box lets you specify or modify the For statement loop count.
5. Enter a valid integer to use as the loop count.

   Loop Count: Specify the number of times to loop though the For statement.
6. Click **OK**. The For node is added to the script tree.
7. In the Java Code view, the `for (int i=0; i < loop count;i++)` statement will be added to the script code:

```
for (int i=0; i < 10; i++)
```

3.3.5 Adding a Function to a Script

You can add your own custom functions to your script and specify the arguments to pass to the function. Custom functions can be in the current script or in another script or function library script that has been added to the current script's Scripts Assets Properties. See Section 3.3.6, "Using a Script as a Dedicated Function Library" for additional information about creating a dedicated function library script.

To add a function to a script:

1. Create a script project.
2. Record a complete script.
3. Select the **Run** node in the script tree.
4. Select the **Script** menu and then select **Other** from the **Add** sub menu.
5. Expand the General node and select **New Function**.

   This dialog box lets you specify a custom function name with multiple arguments.

   Name: Specifies the name of the custom function. Click **Add** to define the names and data type of an argument.

   Description: Specifies a user-defined description for the custom function.

   Argument: Lists the defined function arguments for the custom function.

   Type: Lists the data type for the defined argument for the custom function.

   Description: Lists the user-defined description for the argument defined for the custom function.

   Add: Opens a dialog box for defining a new argument for the custom function.

   Edit: Opens a dialog box for editing the selected argument.

   Delete: Removes the selected argument from the list.

   Up: Moves the selected argument up one place in the list.

   Down: Moves the selected argument down one place in the list.
6. Enter the function name.
7. Enter a description for the function.
8. Click **Add**.

   This dialog box lets you specify a custom function argument to use to pass data to the function.

Name: Specify a name of the custom function argument.

Type: Select the data type: String, Integer, Double, Long, Boolean, Select List, List<String>, or Map<String, String>.

Description: Specify a description for the argument (you may want to include the data type in the description so that it is indicated in the Substitute Variable list).

9. Enter an argument name.
10. Select the data type for the argument.
11. Click **OK**.
12. Click **Add** and add more arguments or click **OK** to add the function to the script. The *function name* node is added to the script tree.
13. In the Java Code view, the `public void` *`function name`* statement will be added to the script code followed by the arguments with the data types:

```
/**
 * My custom Function
 * @param argString Description of argString
 * @param argInt Description of argInt
 * @param argDouble Description of argDouble
 * @param argLong Description of argLong
 * @param argBool Description of argBool
*/
public void MyFunction(@Arg("argString") String argString,
      @Arg("argInt") int argInt,
      @Arg("argDouble") double argDouble,
      @Arg("argLong") long argLong,
      @Arg("argBool") boolean argBool)
   throws Exception {
```

14. Add items into the Function. You can use the Tree View drag/drop or cut/paste features to move Tree View items to the function. You can use the Script **Add** option to add variable items to the function. You can also use the Code View to add custom code to the function.

To pass arguments into a function:

Define the variables to use to pass values to the custom function arguments somewhere in the script before where the Call Function statement will be placed in the script:

1. Select the script node where you want to add variables.
2. Select the **Script** menu and then select **Other** from the **Add** sub menu.
3. Expand the Variables node and select **Set Variable**.

   This dialog box lets you define a variable in a script.
4. Enter the variable name and value.
   - **Name**: Specify the name of the variable.
   - **Value**: Specify the value to assign to the variable.
5. Enter a value or click the Substitute Variable icon to select a variable value to assign to the variable.
6. Click **OK**.

7. In the Java Code view, the `getVariables().set()` statement will be added to the script code followed by the variable name and value for each variable:

```
getVariables().set("MyString", "String");
getVariables().set("MyInt", "1");
getVariables().set("MyDouble", "12.34");
getVariables().set("MyLong", "1234560");
getVariables().set("MyBool", "True");
```

   The following is an example of a variable set to a Databank value:

```
getVariables().set("MyString", "{{db.customer.FirstName,String}}");
```

8. Select the Function node (your custom function name) in the script.
9. Select the **Script** menu and then select **Other** from the **Add** sub menu.
10. Expand the tree and select the item to add. For example **Message** under the General node or **Set Variable** under the Variables node.
11. Click the Substitute Variable icon to select a custom variable or function argument. The Select Variable tree lists the custom function with all of is defined arguments.
12. Select an argument for the custom function.
13. Click **OK**.
14. In the Java Code view, the message statement (`info`, `warn` or `fail`) or `getVariables().set()` statement will be added to the script code followed by the variable name and value for each variable:

```
public void MyFunction(@Arg("argString") String argString,
      @Arg("argInt") int argInt,
      @Arg("argDouble") double argDouble,
      @Arg("argLong") long argLong,
      @Arg("argBool") boolean argBool)
   throws Exception {
      info("{{arg.argString}}");
      getVariables().set("MyArgString", "{{arg.argString}}");
      getVariables().set("MyArgInt", "{{arg.argInt}}");
      getVariables().set("MyArgDouble", "{{arg.argDouble}}");
      getVariables().set("MyArgLong", "{{arg.argLong}}");
      getVariables().set("MyArgBool", "{{arg.argBool}}");
}
```

To call a custom function in a script:

1. Select the node in the script tree here you want to call the function.
2. Select the **Script** menu and then select **Other** from the **Add** sub menu.
3. Expand the Script Function node and the sub node where the custom function is located. Custom functions can be in the local (currently open) script or in another script added to the Assets (select Assets tab in the script view to add other scripts to the script Assets).
4. Select the function to call and click **OK**.

   This dialog box lets you specify a custom function to call and specify the argument values.
5. Enter the argument data to pass to the custom function or click the Substitute Variable icon to select a custom variable or databank variable.

- **Function**: Select the name of the custom function. The names of custom functions that were added to the script will appear in this list.
- ***Arguments***: A field for each custom function argument will appear for the selected function. Enter the argument value or click the Substitute Variable icon to select a custom variable or databank variable.

6. Click **OK**.
7. In the Java Code view, the `callFunction`, `getScript("`*`ScriptAlias`*`").callFunction()`, or *`className`*`.`*`function`*`()` statement will be added to the script code followed by the function name and arguments. If the function is in the same script, the `callFunction` statement is added:

```
callFunction("MyFunction", "MyStringArg");
```

To pass data types other than String, enclose a defined variable name in double curly braces as follows, "{{*VarName*}}".

```
callFunction("MyFunction", "{{MyString}}", "{{MyInt}}", "{{MyDouble}}",
"{{MyLong}}", "{{MyBool}}");
```

If the function is in a another script (a script containing functions that has been added to the current script as a script asset), the `getScript("MyScriptAlias").callFunction("MyCustomFunction", "args");` statement is used to call the function similar to the following example:

```
getScript("MyScriptAlias").callFunction("MyCustomFunction",
  "MyString", "true", "2", "10.5", "100", "list1",
  toList("ListString1", "ListSting2"), toMap("key1=value1"));
```

If the function is in a dedicated function library script (a script specifically created as a function library that has been added to the current script as a script asset), the *`className`*`.`*`function`*`("args");` statement is used to call the function similar to the following example:

```
myFunctionLibrary.MyLibraryFunction("MyString", true, 2, 10.5, 100,
  "list1", toList("ListString1", "ListString2"),
  toMap("key1=value1"));
```

See Section 3.3.6, "Using a Script as a Dedicated Function Library" for addition information about dedicated function libraries.

3.3.5.1 Adding Functions that Use Lists

The Type options of the Argument dialog box used when adding a function to a script includes a List<String> type that can be used to create a function that accepts a list of values as a parameter.

Follow steps 1-9 in Section 3.3.5, "Adding a Function to a Script" to add a new function to a script. Select List<String> as the argument type.

Enter the list of values into the Arguments **Values** field and click **OK**.

The following example code shows a function that uses List<String> for one of the parameter values:

```
/**
 * a function that uses List<String> parameter.
 * @param user a String specifying the user name.
 * @param password a String specifying the password.
 * @param urls a List<String> specifying the urls.
```

```
 */
public void myListFunction(@Arg("user") String user,
      @Arg("password") String password,
      @Arg("urls") java.util.List<String> urls)
   throws Exception{
   }
```

When the user calls the function using the Script menu options, the Call Function dialog box will include a multi-line text box that allows a list of values to be entered. In the Java Code view, the `callFunction()` code will include a `toList` parameter similar to the following example:

```
callFunction("myListFunction", "testuser", "testpwd", toList(
    "http://mysite.com/url1.html",
    "http://mysite.com/url2.html",
    "http://mysite.com/url3.html"));
```

If the function is in a function library script (a function library script added as a script asset), the *`ClassName.function()`* statement is used to call the function similar to the following example:

```
MyClassNameAlias.myListFunction("testuser", "testpwd", toList(
    "http://mysite.com/url1.html",
    "http://mysite.com/url2.html",
    "http://mysite.com/url3.html"));
```

3.3.5.2 Adding Functions that Use Maps

The Type options of the Argument dialog box used when adding a function to a script includes a Map<String, String> type that can be used to create a function that accepts key/value pair maps as a parameter.

Follow steps 1-9 in Section 3.3.5, "Adding a Function to a Script" to add a new function to a script. Select Map<String, String> as the argument type.

Enter the key=value pairs into the Arguments **Values** field and click **OK**.

The following example code shows a function that uses Map<String, String> for the parameter values:

```
/**
 * a function that uses Map<String, String> parameter.
 * @param keyandvalue a Map<String, String> specifying key/value pair maps.
 */
public void myMapFunction(
      @Arg("keyandvalue") java.util.Map<String, String> keyandvalue)
   throws Exception {
   }
```

When the user calls the function using the Script menu options, the Call Function dialog box will include a multi-line text box that allows a key/value pair maps to be entered. In the Java Code view, the `callFunction()` code will include a `toMap` parameter similar to the following example:

```
callFunction("myMapFunction", toMap(
    "key1=value1",
    "key2=value2",
    "key3=value3"));
```

If the function is in a function library script (a function library script added as a script asset), the *`ClassName.function()`* statement is used to call the function similar to the following example:

```
MyClassNameAlias.myMapFunction(toMap(
    "key1=value1",
    "key2=value2",
    "key3=value3"));
```

3.3.5.3 Adding Functions that use Enumerated Lists

The Type options of the Argument dialog box used when adding a function to a script includes a Select List type that can be used to create a function that accepts a value from a known list of values as parameters.

Follow steps 1-9 in Section 3.3.5, "Adding a Function to a Script" to add a new function to a script. Select Select List as the argument type.

Enter the values into the Arguments **Values** field and click **OK**.

The following example code shows a function that uses Select List for the parameter values:

```
/**
 * a function that uses Select List parameter.
 * @param color a Select List specifying enumerated list of color values.
 */
public void mySelectListFunction(
      @Arg("color") @Values( { "red", "blue", "green" }) String color)
   throws Exception {
   }
```

When the user calls the function using the Script menu options, the Call Function dialog box will include a select list box that allows a single value from a list of values to be selected. In the Java Code view, the `callFunction()` code will include the function name and selected parameter similar to the following example:

```
callFunction("mySelectListFunction", "blue")
```

If the function is in a function library script (a function library script added as a script asset), the *`ClassName`*`.`*`function`*`()` statement is used to call the function similar to the following example:

```
MyClassNameAlias.mySelectListFunction("blue");
```

3.3.5.4 Inputting Values from a File

The list and map functions can also specify one or more files as an alternate method for inputting list or map data to function arguments using the `@dataFile` variable. The `@dataFile` variable specifies the optional repository name, data file path, and optional file encoding using the following format:

```
{{@dataFile('repository','relativeFilePath')}}
{{@dataFile('repository','relativeFilePath','encoding')}}
```

Create data files containing the appropriate function argument data.

For list functions that use `toList()`, the data file might appear similar to the following example:

```
    "http://mysite.com/url1.html"
    "http://mysite.com/url2.html"
    "http://mysite.com/url3.html"
```

For map functions that use `toMap()`, the data file might appear similar to the following example:

```
"key1=value1"
"key2=value2"
"key3=value3"
```

The `toList()` and `toMap()` methods also parse all arguments for any newline delimiters. Any newline delimiters embedded in a value or file will be converted into new list entries. For example,

```
"value1\r\nvalue2\r\nvalue3"
```

is equivalent to:

```
"value1"
"value2"
"value3"
```

When the user calls a function that uses lists or maps as argument values using the Script menu options, the Call Function dialog box will include the **Substitute Variable** icon next to the Arguments values field.

Click the **Substitute Variables** icon next to the Arguments values field.

Expand the `@Functions` tree, select `@dataFile`, and click **Finish**.

The argument value field will contain the `{{@dataFile('repository', 'relativeFilePath')}}` transform variable. Edit the `repository` and `relativeFilePath` values to specify the repository name and relative path to the data file containing the data to pass to the function.

- `repository` - specifies the name of the repository where the data file is located. The repository argument can be left blank (using two single quotation marks `''`) if a relative path is specified for the data file.
- `relativeFilePath` - specifies the path and file name of the data file. The file path is relative to the script directory. If the `repository` is not specified (`''`), then `relativeFilePath` should begin with `../`.
- `encoding` - specifies the file encoding to use. Although not automatically added the in the `@dataFile()` default value, the file encoding can be added as a third String value in the `@dataFile()` variable. For example, UTF-8, cp1252 (ANSI), SHIFT-JIS, cp1250 (Central Europe), cp1251 (Eastern Europe).

When you click **OK**, the function will be added to the tree view similar to the following:

```
myMapFunction(toMap({{@dataFile('myRepo','files/datafile.txt','cp1252')}}))
```

In the Java Code view, the `callFunction()` code will include the function name and `@dataFile()` variable similar to the following example:

```
callFunction("myMapFunction", toMap(
  "{{@dataFile('myRepo','files/datafile.txt','cp1252')}}"));
```

If the function is in a function library script (a function library script added as a script asset), the *`ClassName.function`*`()` statement is used to call the function similar to the following examples:

```
MyClassNameAlias.myMapFunction(toMap("{{@datafile('','../files/datafile.txt'}}"));

MyClassNameAlias.myListFunction("user", "pwd",
   toList("{{@datafile('','../files/datafile.txt'}}"));
```

3.3.6 Using a Script as a Dedicated Function Library

You can create a script that can be used as a dedicated function library that can be used by other scripts as a script asset. The dedicated function library script can contain custom functions that can be called from other scripts. A dedicated function library provides a way for having code assistance in scripts calling functions from the library. When creating dedicated function library scripts you should make the library either a generic Java code script or the same type as the scripts (that is, Web, HTTP, Siebel, etc.) that will be calling functions from the library.

3.3.6.1 About Function Libraries

This section provides basic information and Frequently Asked Questions (FAQs) about function libraries.

What is a function library?

A function library is a means to make your code more reusable and modular. Any script can be used as a function library by creating a library of custom functions within the script code. The function library script can then be added to other scripts as a script asset. The other scripts can then call custom functions from the function library script.

What is the difference between a function library and a regular script?

In general, there is only a minor difference between a function library script and regular script. A function library is a script which you can record into and play back but also includes a unique package and class name. However, function library scripts are generally not intended to be played back. You use a function library script as an asset for other scripts that can call the custom functions contained in the library. Function library scripts include the Initialize, Run, and Finish methods the same as a regular script. The Initialize, Run, and Finish methods in a dedicated function library script are not intended to be called from other scripts. The Initialize, Run, and Finish methods are provided for debugging function library scripts before publishing them.

How do I create a function library?

Before OpenScript version 12.1.0.1, regular scripts were used as function libraries. Users simply added custom functions to a script which was designated as a function library. See Section 3.3.5, "Adding a Function to a Script" for additional information about adding functions to a script. OpenScript version 12.1.0.1 introduces dedicated function library scripts. See Section 3.3.6.2, "Creating a Dedicated Function Library Script" for additional information about creating a dedicated function library script.

What are the advantages of using a dedicated function library?

There are three main advantages to using dedicated function libraries compared with using a script containing functions:

- The primary advantage is provided for teams that widely use the Java Code view with custom code. The style of calling functions from a dedicated function library script is much more straight forward and clear. For example, with the dedicated function library you call a function a function `foo` in the library with alias `myLib` as follows:

  ```
  myLib.foo("myArg");
  ```

  With custom functions in a regular script, you call a function using the Reflection style, as follows:

  ```
  getScript("myLib").callFunction("foo", "MyArg")
  ```

- A second advantage of a dedicated function library is that the direct style of calls provides code assistance in the Java Code view for calling functions stored in a dedicated function library script. There is no code assistance in the Java Code view for calling functions stored in a regular script and called using the `callFunction` method.
- A third advantage of a dedicated function library is the ability to use custom classes in the library.

What are the disadvantages of using a dedicated function library, versus using a script with functions in it?

There are some advantages to using dedicated function libraries compared with using a script containing functions:

- A dedicated function library not only has a unique script location, but also a unique library class name. The library class name is rendered when creating the dedicated function library and cannot be changed. If for some reason you need or want to change function library class name (for example, a duplicated function library name was found or something else), you will need to create a new script by selecting **Save As** from the **File** menu or by creating a script from an existing script. You will need to enter a new name for the function library class and a new script will be created. The new script will have all of the functions declared in the old function library.
- Another minor disadvantage of using a dedicated function library is that it needs to be assigned as an asset to each script that uses it. In contrast, using `getScript("funcLib").callFunction()`, you only need to assign a function library as an asset with the alias "funcLib" to the top level script. Any child scripts will find the function library by its alias during runtime. However, this usage is blind as users have no Tree View or dialog support for inserting such code. By not assigning the dedicated function library directly to a script as an asset, you can only call its functions by typing code directly into the Java code view using the Reflection style syntax. Also, code assistance is not provided for Reflection style syntax.

How can I use a common function library in all scripts I create?

Particularly when creating many test scripts, it can be time consuming to attach commonly used function libraries to each one individually. There are two approaches to solve this problem.

1. Create Scripts from an Existing Script

   You can create an empty script and attach to it all the common function libraries you want your scripts to use. Each time you create a new script, select **New** from the **File** menu, select **Script from Template** in the General section, then select the script with your common functions to create the new script. Each new script will have the common function libraries attached to it.

   The one downside to attaching a function library to every script. If you physically move a function library to a different location on disk, then all existing scripts referencing it from the old location will break.

2. Attach Function Libraries to a Parent "Master" Script

   If you have a test setup where one parent "master" script calls several "child" scripts, you can attach the common function libraries only to the parent script and not attach it to any child scripts. Child scripts will be able to directly call the function libraries from Java Code using the reflexive code syntax. For example

   ```
   getScript("myLib").callFunction("foo", "MyArg");
   ```

The benefit to attaching the function library to the parent script, and not to the child scripts, is that if the function library is moved to a different location, you only need to update the parent script with the new function library location. All child scripts will continue to work. However, using this approach, child scripts cannot use the explicit code syntax when calling functions. For example:

```
myLib.foo("MyArg");
```

Child scripts will not be able to use any custom classes exposed by the function libraries or leverage any Java Code assistance features. The child scripts' Tree view will also not provide any UI for adding available functions. This approach is only recommended for teams not relying on the Tree view UI for adding function calls.

Will my existing pre-12.1 function libraries continue to work in new releases?

Yes. See Section 3.2.2, "Opening Existing Scripts" for additional information about backwards compatibility of scripts. Existing scripts and function libraries will continue to work, unmodified, in newer releases.

If you want to take advantage of new features such as code assistance, custom classes, and the explicit function calling syntax, then you would need to create a dedicated function library. However, it is optional if you want to convert your existing script-based function libraries into dedicated function libraries or leave them as regular scripts.

What happens if two different dedicated function library scripts used in a test suite have the same library class name?

Using two function libraries with the same library class name will cause your script to behave unpredictably. Most likely, the OpenScript script editor will display an error when it compiles the script, indicating that some particular method or class cannot be found. This is why, when you are creating a name for library class, it is better to use a long package path to differentiate your library class from other libraries. For example, `lib.TestLib` is a poor choice. A much better choice would similar to: `lib.myCompany.myTeam.framework2.UtilLib`.

Can I use custom classes as arguments and return values for functions in function libraries?

You can only use custom classes as arguments and return values for functions in dedicated function libraries. In order for custom classes to be visible to calling scripts they should reside in package "lib" that is automatically created for each dedicated function library. If the author of the function library needs some internal custom classes, then their place is in a sub-package of "lib" package. This way they will be hidden from other scripts. It is better not to put any additional classes in the package where library class itself resides.

Be cautious and do not overuse custom classes. The user of the function library will not get help from UI or code assistance for them. Almost any task can be achieved with the set of data types provided in the OpenScript UI for providing assistance, which includes all primitive types, Enum(selection), List<String>, and Map<String, String>.

I have a function in a function library that returns a custom type. How do I to save it for later use?

You will need to know what type is returned and save it in a script or java variable. Note that OpenScript does not support declaring custom functions (in regular scripts or dedicated function library scripts) that expect the `varargs` argument feature. That

is, arg type, `Something...`). For example, `Foo(String name, String... args);`.

Also, having an argument type of `List` is always preferable to argument of type `Array`. Thus `List<String>` is better than `Array[]`. The reason is creating functions with argument that has an `Array` type requires the users of the function resolve any ambiguity of casting `Array[]` vararg type, as it could be cast to `Object` or `Object[]`.

My function library needs to use third-party JAR files. How I can I do it?

A function library is a script. Assign the JAR file to the script as a generic JAR script asset. You will then have access to all public methods from the Java Code view. Code assistance will help you add the required import statements and call methods in the code view.

Note: Generic JAR files should not contain any code that uses OpenScript API or Eclipse API.

I would like to expose some methods in third-party JAR files to test scripts to be able to call them. What I should do?

You can assign this JAR file to each script that is going to use it as a script asset. However, it is a blind way for users to call functions in JAR file. A better way is to assign this JAR file to a function library script and wrap the methods in the JAR file that will be called by scripts within functions in the function library. You will be able to control access to the JAR file and, more importantly, add extensive comments on when and how to use these methods. This will benefit the other members in your group by making the use of the functions easier.

What help should I provide to users of my function library?

Function library developers should provide meticulous and extensive Javadoc comments for each function in the function library. The Javadoc comments are what the user sees about each function in the Tree View and as code assistance in the Java Code view.

3.3.6.2 Creating a Dedicated Function Library Script

To create a dedicated function library script:

1. Select **New** from the **File** menu.
2. Select the project type (typically Java Code Script) and click **Next**.
3. Enter a script name for the function library (for example, myScriptLib).
4. Select **Create script as a Function Library**.
5. Click **Next**. The script wizard opens the Create Function Library options:

   Package: Specifies a unique Package name for the function library. Package must be a valid Java Package name that matches A-Z, a-z, 0-9, _. It must not contain spaces or Double-Byte Character Sets (DBCS). The initial default value is `myCompany.myTeam`. Subsequently, the default value will be set to the last value specified.

   Class: Specifies a unique alias to use as the name (typically the Class name) for the function library script. **Class** must be a valid Java Class name that matches A-Z, a-z, 0-9, _. It must not contain spaces or Double-Byte Character Sets (DBCS). The Class Name should be:

- meaningful and provide context as to the purpose of the library,
- clear and concise so it is easy to read and type in scripts,
- something unique so it is not confused with other function libraries.

6. Enter a unique **Package** name for the function library in the form:

   ```
   orgName.groupName.subgroupName
   ```

   For example:

   ```
   oracle.oats.dev
   ```

7. Enter a unique alias to use as the **Class** name for the function library to identify the library. For example:

   ```
   WebFunctLib
   ```

8. Click **Finish**.
9. Select the script section (Initialize, Run, or Finish) where you want to add custom functions.
10. Select the **Script** menu and then select **Other** from the **Add** sub menu.
11. Expand the **Script Functions** and **Local Script** nodes.
12. Select **[New Function]** and click **OK**.
13. Enter a function name and description.
14. Click **Add** to add any arguments to the function.
    - Enter a name, specify the data type, and enter a description for the argument.
    - Click **OK** to add the argument.
15. Repeat step 14 for each argument to add to the function.
16. Click **OK** to add the function to the script.
17. Add your custom code to the function.
    - Use the script recorder to record steps.
    - Switch to the Java Code view and edit the code in the function.
18. Repeat steps 10 through 17 to add additional functions to the library script. See also Section 3.3.5, "Adding a Function to a Script" for additional information about adding functions to a script.
19. Save the library script.

3.3.6.3 Calling Functions from a Function Library Script

To call functions from the library script:

1. Create a new script project (for example, masterScript).
2. Select the **Script** menu and then select **Other** from the **Add** sub menu.
3. Expand the **Script Functions**.
4. Select **[New Script]**.
5. Select the repository and library script.
6. Specify a script alias or use the default alias. The script alias must be unique for each library script added to a script.

7. Click **OK**.
8. Expand the **Script:** ***scriptLibraryAlias*** node.
9. Select the function name from the library script and click **OK**.
10. If the Call Function dialog box appears, enter the function arguments and click **OK**. The library script is automatically added to the current script as a script asset (click the Assets tab in the Script view to view script assets).

    The function name appears in the script tree as *scriptLibraryAlias.functionName*(*args*, [...])

    In the Java Code view, the `@FunctionLibrary` assignment will be added to the script code followed by the library Package Name and function library alias entered when the function library was created, as follows:

    ```
    @FunctionLibrary("assetAlias") lib.package.assetAlias scriptAlias;
    ```

    For example

    ```
    @FunctionLibrary("Web_func_lib") lib.oracle.oats.dev.Web_func_lib webFunctLib;
    ```

    Functions called from the function library appear in the Java Code view using the function library alias and called function name and arguments, as follows:

    ```
    scriptAlias.functionName("args", "[...]")
    ```

    For example:

    ```
    webFunctLib.selectColor("red");
    ```

    When you type the function library alias followed by a period in the code view, the code assistance view opens listing the functions available in the function library with the required arguments.

    In the Java Code view, the `getScript().callFunction()` statement can also be used with the script code followed by the function name and arguments, as follows:

    ```
    getScript("scriptLibraryAlias").callFunction("functionName", "args", "[...]")
    ```

    However, code assistance is not available for functions called using the `getScript().callFunction()` statement.
11. Save the master script and play it back to execute the custom functions. See Section 3.3.5, "Adding a Function to a Script" for additional information about passing arguments to functions.

> **Note:** In the master script, be sure to add a "Launch Browser" command to the Initialize section if it is not in the first function called from the master script.

3.3.7 Converting a Script to a Dedicated Function Library

You can convert existing scripts to a dedicated function library script.

To convert an existing script to a dedicated function library script:

1. Open the script to convert to a dedicated function library script.
2. Select **Convert to Function Library** from the **Tools** menu.

3. Specify a unique package name for the function library script.
4. Specify a unique alias as the Class name for the function library script. See Section 3.3.6, "Using a Script as a Dedicated Function Library" for additional information about package and class names.
5. Click **OK**. The converted script is saved and reopened.

3.3.8 Adding Script Assets

You can add assets to a script such as, databanks, generic JAR files, object libraries, or other scripts containing recorded steps or custom functions. The asset must exist before it can be added. Select **New** from the **File** menu to record scripts or create databanks and object libraries. You also use the Add option in the Script View Assets tab to create databanks and object libraries.

To add assets to a script:

1. Open or create a script project.
2. Select the **Assets** tab in the script view . The Assets view has the following options:
   - **Databanks**: Lists the Databank assets added to a script.
   - **Object Libraries**: Lists the Object Library assets added to a script.
   - **JAR files**: Lists generic JAR file assets added to a script.
   - **Script**: Lists the child script assets added to a script.
   - **Add**: Opens a file selection dialog box for selecting the file to add as an asset. Expand the My Repositories tree to navigate to a workspace folder containing the file. For Databanks, a submenu opens for selecting CSV file or database-type databanks. For CSV file databanks, you select the file. For database-type databanks, you specify the database driver and connection information.

     Note: Any scripts you plan to run, along with any associated assets, in the Oracle Load Testing application must be stored in a repository/workspace that can be accessed by the Oracle Load Testing Controller. If you create new repositories in OpenScript, you should also add the new repositories in Oracle Load Testing.

   - **Edit**: Opens a file selection dialog box for changing which file is added as an asset.
   - **Open**: Opens the selected asset file in the appropriate editor.
   - **Remove**: Removes the selected asset file from the Assets tree. The file still exists in repository/workspace.
3. Select the type of asset to add and click **Add**.
4. Select the Databank, Object Libraries, JAR File or Script. For Databanks, select CSV file or Database. If you select Database, specifiy the database connection parameters.
5. Select the asset to add from a repository.
6. Set the **Relative to** option. The **Relative to current script** and **Relative to a repository** options specify how the current script will locate the specified script

asset. The **Relative to a repository** option locates the script asset by a repository path such as, `[Repository: Default] Default!/WebTutor`, if the asset is selected from a repository. The **Relative to current script** option locates the script asset by a relative path such as `../WebTutor`. Selecting the The **Relative to current script** option is not recommended as script-relative paths are more brittle than repository-relative paths if scripts are moved or shared.

The following are guidelines when using script assets in a team or distributed environment:

- Do not use Absolute Paths when referring to assets or saving assets. Oracle Load Testing does not support absolute paths.
- OpenScript, Oracle Test Manager, Oracle Load Testing, and all command-line agents should all use the same shared repository names and paths.
- Do not refer to an asset in another repository by a relative path.

7. Click **OK** to add the asset to the script properties.
8. Click **OK** when finished adding script assets to close the script properties.

Script asset information is stored in the assets.xml file located in the script project directory.

3.3.9 Adding a Script to Run from a Script

To add a script to run to a script:

1. Open or create a script project.
2. Select the script node where you want to add the script to run.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the General node and select **Run Script**.
5. Click **OK**.

   This dialog box lets you specify the script to run from within another script.

   Script: Specifies the OpenScript script to run.

   New: Opens the script properties for selecting the script asset to run.

   Sections of script to run: Specifies which script sections to run during playback.

   - **Initialize Section**: When selected, the code in the Initialize section of the selected script to run is executed during playback. When cleared, the code in the Initialize section is skipped.
   - **Run Section**: When selected, the code in the Run section of the selected script to run is executed during playback. When cleared, the code in the Run section is skipped.
   - **Finish Section**: When selected, the code in the Finish section of the selected script to run is executed during playback. When cleared, the code in the Finish section is skipped.

   Iterations: Specify the number of script iterations to run.
6. Select the script using **New** next to the **Script** field.
7. Select the next script to run from the available script assets in the Script properties. Use the Add button to add scripts to the script assets properties.
8. Select or clear the **Sections of script to run** option.

9. Set the iteration count.
10. Click **OK**. The *script name* node is added to the script tree.
11. In the Java Code view, the `getScript().run();` statement will be added to the script code, as follows:

    ```
    getScript(alias=String).run(interation count = int, initialize = true|false,
    run = true|false, finish = true|false);
    ```

    Example

    ```
    getScript("Web1").run(1, true, true, true);
    ```

3.3.10 Adding a Synchronization Point to a Script

A sync point allows multiple scripts being run as virtual users in Oracle Load Testing to synchronize their actions and interactions with the application under test. Sync points provide the ability to create realistic multi-user situations that may expose resource conflicts such as deadlocks. When you specify a sync point, multiple virtual users executing the script will reach this sync point at various times depending on a number of factors (for example, the speed of the machine).

Sync points cause each virtual user to wait until all virtual users have reached that sync point. Each of the virtual users notifies the master upon reaching the sync point. The master waits for all of the virtual users to notify it and then issues the go-ahead for all the virtual users to continue past that sync point.

Sync points are added to individual scripts (parent or child scripts) when they are created in OpenScript. The execution parameters for sync points are defined in the Oracle Load Testing application.

To add a sync point to an OpenScript script:

1. Create or open a script in OpenScript.
2. Select the script node where you want to add the sync point.
3. Select **Add** from the **Script** menu then select **Other**.
4. Select the **Synchronization Point** node and click **OK**.

   This dialog box lets you specify the name to use for the sync point.

   Name: Specifies the name used to reference the sync point in the Oracle Load Testing application.
5. Enter a name for the synchronization point and click **OK**.
6. In the Java Code view, the `syncPointWait("name");` method will be added to the script code, as follows:

   ```
   syncPointWait("MySyncPoint");
   ```

7. Save the script in OpenScript.
8. Load the script into the Oracle Load Testing application and specify the execution parameters for the sync point(s) in the load test scenario. See the *Oracle Load Testing User's Guide* for additional information about specifying the sync point execution parameters.

3.3.11 Adding a Set Variable to a Script

To add a Set Variable to a script:

1. Open or create a script project.
2. Select the script node where you want to add the set variable.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Variable node and select **Set Variable**.

   This dialog box lets you set a variable value in a script.
5. Enter the variable name and value.
   - **Name**: Specify the name of the variable.
   - **Value**: Specify the value to assign to the variable.
6. Click **OK**. The Set *variable* = *value* node is added to the script tree.
7. In the Java Code view, the `getVariables().set("variable name", "value");` method will be added to the script code:

   ```
   getVariables().set("sVar_MyVar", "My_Value");
   ```

   If you want to set the variable with a value from an OpenScript transform variable (i.e. a variable value contained in {{}} syntax), use the Transforms.transform method with the getVariables().set, as follows (requires HTTP module):

   ```
   http.solve("varTitle", "<TITLE>(.+)</TITLE>", "Page Title Error", false,
   Source.Html, 0);

   getVariables().set("sVar_MyVar", Transforms.transform("{{varTitle}}",
   getVariables()));
   ```

3.3.11.1 Variables with Scope

You can use the `variables` method in the Java code to get or set variable with scope. For example:

```
variables.set(String name, String Value, Variables.Scope.scope)
variables.get(String name, Variables.Scope.scope)
```

Script variables are global for all scripts and functions for a *single* Oracle Load Testing Virtual User.

- Each Virtual User keeps its own map of script variables.
- One Virtual User cannot read/write another Virtual User's script variables. The exception is that a Child Virtual User (i.e. a Virtual User as a child script) has access to all variables in its parent Virtual User.
- All scripts and functions that a Virtual User runs will have read/write access to all the Virtual User's variables.
- In Functional Testing, a script typically represents only one Virtual User. In Functional Testing, script variables are generally global variables.

There are three scopes:

- `Local` - all variables that the current script explicitly defines as local variables.
- `Parent` - all variables that the parent (calling) script defines as its own local variables.
- `Global` - all other variables not defined in an explicit scope. This is the default scope when no scope is specified. The Global scope is the parent scope of top-level

VUser (top-level script). So a top-level script will have two scopes (Global and Parent) that coincide.

Scope can be used to avoid confusion. If an author of a child script wants to change variables in global or parent scope, the script author should do it explicitly. If the author of a child script wants to change local script-level variables, then Scope `Local` should be used.

Child scripts inherit its entire parent script variables. It is not a copy of the variables, it is a reference to the same variables, i.e. `getParentVUSer().variables`.

The following examples show uses of the Variable Scope:

```
//local variable
variables.set("user", "rich", Variables.Scope.Local);

//global variable (same as set("myData", "globalData", Variables.Scope.Global);)
variables.set("myData", "globalData");

//parent variable
variables.set("anotherData", "parentData", Variable.Scope.Parent);

getScript("Script2").run();

// "globalData" as parent and global scope for top-level script coincide.
variables.get("myData", Variables.Scope.Parent)
```

3.3.12 Removing Unchanging Variables

You can select and remove script variables that it is known will never change. Removing unchanging variables can improve script playback performance because unchanging variables will not need to be evaluated during script playback. The script will use the value captured during recording as a fixed value rather than a variable.

To remove unchanging variable from a script:

1. Record a script.
2. Play back the script at least once to compare the recorded values to the playback values to determine which variables are unchanged.
3. Select **Remove Unchanging Variables** from the **Tools** menu.

   This dialog box lets you specify the variables to remove from the script:

   Variable Name - shows the name of the script variable.

   Recorded Value - shows the variable value set during recording.

   Playback Value - shows the variable value set during playback.

   Check All - selects all of the variables in the table.

   Check Unchanged - selects only the unchanged variables in the table.

   Uncheck All - unselects all of the variables in the table.
4. Select the variables to remove using the **Check**/**Uncheck** buttons.
5. Click **OK** when finished.

3.3.13 Parameterizing URLs

You can create variables to use for URLs in a script. In cases where you need to change the base URL of a script, paramterizing the URLs provides a quick way to re-baseline a script to use a new URL. The URL will only need to be changed in one place.

To parameterize URLs:

1. Record a script.
2. Select **Parameterize URLs** from the **Tools** menu.

   This wizard lets you create variable names to use for URLs contained in the script. The Enter URL panel lets you specify the URL and variable name to parameterize:

   URL - specifies the URL to parameterize. Use the dropdown selector to select from URLs that have been recorded to the current script.

   Variable Name - specifies the name to use as the script variable.
3. Select the URL to parameterize.
4. Enter a variable name to use for the URL.
5. Click **Next**.

   The Verify Changes panel lets you verify and select the which instances of the URL in the script will be changed:

   Tree view - shows the script nodes with the URL instances.

   Check All - selects all of the URL nodes in the tree.

   Uncheck All - unselects all of the URL nodes in the tree.

   Original URL - shows the original value of the URL before parameterizing as a script variable.

   New URL - shows the value of the URL after parameterizing as a script variable. For example, if you parameterize the URL `http://myServer.com` as the variable name `myServerVar`, the new URL will be the parameterized script variable `{{myServerVar,http://myServer.com}}`.
6. Select which instances of the URL in the script will be changed by selecting or clearing the check boxes in the Tree view or using the **Check/Uncheck** buttons.
7. Click **Finish**.
8. In the Java Code view, the `getVariables().set("variable name", "value",scope);` method will be added to the script code in the `initialize()` section:

   ```
   getVariables().set("myServerVar", "http://myServer.com",
      Variables.Scope.GLOBAL);
   ```
9. Repeat steps 2-7 to parameterize other URLs in the script.

3.3.14 Adding Comments to Script Results

To add comments to script results:

1. Open or create a script.
2. Click the Code view tab.
3. Add comments or warnings using one of the following code examples:

- Using a step group:

```
beginStep("Any comment string", 0);
{
//The comment string appears in the Name column of the Results view.
}
endStep();
```

- Using the `getStepResult().addComment` method:

```
//The comment string appears in the Summary column of the Results view
getStepResult().addComment("Any comment string");
```

- Using the `getStepResult().addWarning` method:

```
//The warning string appears in the Summary column of the Results view.
//addWarning overides addcomment.
getStepResult().addWarning("Any warning string");
```

3.3.15 Adding Error Recovery to a Script

To add error recovery to a script:

1. Open or create a script project.
2. Select the script node where you want to add the log message.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the General node and select **Error Recovery Action**.

   Exception: Select the type of exception error. The list will vary depending upon the script type.

   Action: Select the error recovery action: Fail, Warn, Ignore, Report , or P:ause as follows:

   - Fail: Report the error as failure and stop script execution.
   - Warn: Report the error as a warning and continue script execution.
   - Ignore: Ignore the error and continue script execution.
   - ReportErrorAndContinue: Report the error to the results log and continue script execution.
   - Pause: Pause playback and wait for user's decision to continue or abort script execution.
5. Click **OK**. The log message node is added to the script tree.
6. In the Java Code view, the `setErrorRecovery(`*`scriptType`*`.`*`constant`*`, ErrorRecoveryAction.`*`action`*`);` method will be added to the script code:

```
setErrorRecovery(BasicErrorRecovery.ERR_VARIABLE_NOT_FOUND,
ErrorRecoveryAction.Fail);
```

3.3.15.1 Script Types

The following are the possible values for *`scriptType`* in the Java code statements:

`BasicErrorRecovery` (Basic module)

`FormsErrorRecovery` (EBS/Forms Functional module)

`FTErrorRecovery` (Generic Functional module)

`HttpErrorRecovery` (HTTP module)

`NcaErrorRecovery` (EBS/Forms Load module)

`UtilitiesErrorRecovery` (Generic Utilities)

`WebErrorRecovery` (Web Functional module)

3.3.15.2 Constants

The following are the possible values for *`constant`* in the Java code statements:

`BasicErrorRecovery` (Basic module)

```
ERR_VARIABLE_NOT_FOUND
ERR_CREATE_VARIABLE_ERRORCODE
ERR_FILE_NOT_FOUND
ERR_SEGMENT_PARSER_ERROR
ERR_BINARY_DECODE
ERR_ENCRYPTION_SERVICE_NOT_INITIALIZED
ERR_GENERIC_ERROR_CODE
```

`FormsErrorRecovery` (EBS/Forms Functional module)

```
ERR_FORMS_FT_ERROR
STATUSBAR_TEST_ERROR
```

`FTErrorRecovery` (Generic Functional Module)

```
ERR_FT_MATCH_ERROR
ERR_OBJECT_TEST_ERROR
ERR_TABLE_TEST_ERROR
```

`HttpErrorRecovery` (HTTP Module)

```
ERR_ZERO_LENGTH_DOWNLOAD
ERR_MATCH_ERROR
ERR_RESPONSE_TIME_ERROR
ERR_SOLVE_ERROR
ERR_HTML_PARSING_ERROR
ERR_INTERNET_INVALID_URL
ERR_INVALID_HTTP_RESPONSE_CODE
ERR_KEYSTORE_LOAD_ERROR
```

`NcaErrorRecovery` (EBS/Forms Load Module)

```
CONNECT_ERROR
MESSAGE_IO_ERROR
CONTROL_INITIALIZE_ERROR
```

`UtilitiesErrorRecovery` (Generic Utilities)

```
ERR_SQL_EXECUTE_ERROR
ERR_XML_PARSING_ERROR
ERR_CSV_LOADING_ERROR
```

`WebErrorRecovery` (Web Functional module)

```
ERR_RESPONSE_TIME_ERROR
ERR_WEBDOM_SOLVE_ERROR
ERR_WAIT_FOR_PAGE_TIMEOUT_ERROR
```

3.3.15.3 Actions

The following are the possible values for *`action`* in the Java code statements:

```
Fail
Ignore
Warn
ReportErrorAndContinue
Pause
```

3.3.16 Verifying Script Actions

You can verify script actions to check the result of a script action and adjust the behavior of the script based on the result of the action.

The basic process to use verify script actions is as follows:

1. Add an Error Recovery Action before the script node where you want to verify the result code. You can add the Error Recovery Action from the script **Add** sub menu or in the Java Code view. Set the error recovery action to Warn or Ignore to ensure that the Has Error block gets executed. This allows script execution to continue past the code where an exception occurred to the next statement in the script code.
2. Add a 'Has Error' Control Statement after the script node where you want to verify the result code. You can add the Has Error Control Statement from the script **Add** sub menu or in the Java Code view. The `if(hasLastError())` block is added to the script code directly after the script node where you want to verify the result code.
3. Add your custom code into the `if(hasLastError())` block in the Java Code view.
4. Add Results Object messages to return the result values. The Result Code Verification features provide access to a Results object. The Result Object provides Result Code, Summary, Error Message, and Duration information.

The following sections explain the steps in more detail.

3.3.16.1 Adding an Error Recovery Action

To add an Error Recovery Action:

1. Select the script node before the script node where you want to verify the result code.
2. Select the **Script** menu and then select **Other** from the **Add** sub menu.
3. Expand the General node.
4. Select Error Recovery Action and click **OK**.
5. Select the Exception type. See Section 3.3.15, "Adding Error Recovery to a Script" for additional information.
6. Select the **Action** type and click **OK.**
7. Add the Has Error condition to the script. See Section 3.3.16.2, "Adding a Has Error Control Statement" for additional information.

3.3.16.2 Adding a Has Error Control Statement

The Has Error Control Statement can be added to the script using the Tree view. However, the conditional behavior must be specified in the Java Code view.

To add a Has Error condition:

1. Select the script node after the script node where you want to verify the result code.

2. Select the **Script** menu and then select **Other** from the **Add** sub menu.
3. Expand the Control Statements node.
4. Select Has Error? and click **OK**. The `if (hasLastError())` node is added to the script tree.
5. Add a Result Object or add your custom code in the `if (hasLastError())` block in the Java Code view. See Section 3.3.16.3, "Adding a Result Object Message" for additional information.

3.3.16.3 Adding a Result Object Message

The Result Object can be used to return result values.

To add a Result Object message:

1. Select the `if (hasLastError())` node in the script tree.
2. Right-click the `if (hasLastError())` node and then select **Other** from the **Add** sub menu.
3. Expand the General node.
4. Select Message and click **OK**.
5. Select Info or Warn as the **Message Type**.
6. Click Substitute Variable.
7. Expand Last Result.
8. Expand All Actions or assertions and Verifications.
9. Select the Result to add to the message and click **Finish**.
10. Click **OK** to add the message to the script.

    In the Java Code view, the message code with the result type is added to the `if (hasLastError())` block:

    ```
    info("{{result.summary}}");
    ```

    You can customize the message string in the Java Code view. For example:

    ```
    info("Summary of last action: {{result.summary}}");
    ```

11. If necessary drag the message node into the `if (hasLastError())` node so the message is a child node of the `if (hasLastError())` block. For example:

    ```
    if (hasLastError()) {
        info("Summary of last action: {{result.summary}}");
    }
    ```

3.3.16.4 Actions That Can Be Verified

Only specific OpenScript actions provide the ability to verify their results. In general, all actions that are available for adding from the tree view UI, including all verifications and assertions, support verification.

The following types of actions typically do *not* support verification:

- Java methods that are only available from code and not from the UI
- Deprecated methods
- "Get" methods

- Methods that interact with OpenScript internal code such as Logger, VUDisplay, Settings, Counters
- Methods that don't throw any exceptions, such as `http.removeCookie`

3.3.17 Chaining Multiple Scripts

You can run multiple scripts from within a single script to chain playback of scripts together.

The procedure involves the following major steps:

- Setting the browser preferences
- Recording scripts
- Creating a shell script

3.3.17.1 Setting the Browser Preferences

The browser preferences specify if a new browser will launch when recording a different script. Because the navigation sequence between multiple scripts is important, the same instance of the browser should run all scripts if the scripts are a continuation of each other. If each script is self-contained and there is no navigation between scripts, each script can launch its own browser and you can skip the Browser Preferences steps.

1. Select **Preferences** from the **View** menu.
2. Expand the General category and select **Browsers**.
3. Clear the **Always launch a new browser when recording a different script** option.
4. Click **OK**.

3.3.17.2 Recording Scripts

When recording scripts for chained playback, it is important to plan the start and stop points between scripts. This is especially true if session state needs to be maintained between scripts. All of the scripts must be of the same type.

1. Create and record the first script, for example a Web Functional test log in script.
2. Stop the recording but do not close the browser.
3. Save the script.
4. Create and record the next script. The navigation in this script should start from the point in the browser where the first script stopped.
5. Stop the recording and save the script.
6. Create and record any additional scripts to chain. The navigation in these script should start from the point in the browser where the previous script stopped.

3.3.17.3 Creating a Shell Script

The shell script is used to run the previously recorded scripts in sequence.

1. Create a new script to use as the shell script.
2. Select the script node where you want to add the first script. This could be either the Initialize or Run nodes.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.

4. Expand the **General** node and select **Run Script**.
5. Click **OK**.
6. Click **New**.
7. Select the script to run from the available script assets in the Script properties. Use the **Add** button to add scripts to the script assets properties.
8. Click **OK**.
9. Select or clear the script sections to run and the iteration count.
10. Click **OK**.
11. Select the script node where you want to add the next script. This could be either the Initialize, Run, or Finish nodes.
12. Select the **Script** menu and then select **Other** from the **Add** sub menu.
13. Expand the **General** node and select **Run Script**.
14. Click **OK**.
15. Click **New**.
16. Select the next script to run from the available script assets in the Script properties. Use the **Add** button to add scripts to the script assets properties.
17. Click **OK**.
18. Select or clear the script sections to run and the iteration count.
19. Click **OK**.
20. Repeat the Add script steps for each additional script to run.
21. Save and playback the shell script to verify the script navigations work together correctly.
22. In the Java Code view, the `getScript().run()` methods will be added to the script code:

```
getScript("Web1").run(1, true, true, true);
getScript("Web2").run(1, true, true, true);
```

3.3.18 Moving Nodes in a Script

You can click and drag a node in the script tree view to move the node to another location in the script tree. For example, you move a step group node from the Run section to the Initialize section or move a navigation node.

To move script nodes in the script tree:

1. Open or create a script project.
2. Select the script node to move in the Tree View tab of the Script View.
3. Click and drag the mouse to move the node in the script tree. The script tree shows an indicator line that points to the location in the script tree where the node will be moved.
4. Release the mouse button when the indicator line is at the location where you want to move the script node.

When moving Step Groups between script sections (i.e. between Run and Initialize, etc.) you may need to move the node to the section node.

You can also switch to the code view and move lines of code manually.

3.3.19 Aborting and Resuming a Script Programmatically

The Application Programming Interface (API) includes methods for aborting and resuming a script programmatically.

The `abort()` method immediately aborts the Virtual User as soon as possible. The remaining actions in the iteration, including any parent scripts and functions, will not complete. The `Finish` section will not be run. The currently running script will return a failed result indicating that the Virtual User was aborted.

If a Virtual User is aborted, it is possible to resume the script by catching the abort user exception and calling `resume()`.

The `resume()` method allows the caller to reset a previously fired `abort()` request so that script execution can successfully continue from that point on. The caller must first catch the `StopScriptRuntimeException` that gets thrown by the `abort()` request and then call `resume()`.

The `resume()` method works together with the `abort()` method. Calling `resume()` will only recover from a previously called `abort()`.

The following examples show how to use the `abort()` and `resume()` methods.

```
//Example Use Case 1 - Abort a script at any point
info("Running a script...");
abort();
info("This line will not be run.");

//Example Use Case 2 - Abort a script and resume
try {
  info("Perform any steps inside a try-catch block...");
  abort();
  info("This line will not be run.");
}
catch (StopScriptRuntimeException e) {
   // optionally take any corrective action and optionally resume script
  resume();
}

//Example Use Case 3 - Abort Script after 5 Minutes and Execute the Finish Section
public void initialize() throws Exception {
  abortAfter(5*60); // Abort after 5 minutes
  try {
     // Insert script Initialize section here
     info("initializing resources");
  }
  catch (StopScriptRuntimeException e) {
     // ignore this and continue to run()
  }
}

public void run() throws Exception {
  try {
     // Insert script Run section here
     http.get(null, "http://myServer/longrunningtask?length=10min");
  }
  catch (StopScriptRuntimeException e) {
     // ignore the exception; OpenScript will run the finish()
  }
```

```
}

public void finish() throws Exception {
  resume();
  info("cleanup resources");
  // Insert real customer script Finish section here
}

//Example Use Case 4 - abort() after waiting for the specified amount of time
// Usage Example:
//abortAfter(5); to abort after 5 seconds
//abortAfter(5*60); to abort after 5 minutes
private void abortAfter(int seconds) {
  new Timer().schedule(new TimerTask() {
     public void run() {
       try {
          abort();
       }
       catch (StopScriptRuntimeException e) { /* ignore exception thrown */ }
     }
  }, seconds*1000);
}
```

3.4 Changing Text File Encoding

Before recording sites with international characters on an English OS, users should change the default character set encoding to a character set that supports the desired character set.

To change the text file encoding:

1. Start OpenScript and select **Developer Perspective** from the **View** menu.
2. Select **Preferences** from the **Windows** menu.
3. Expand the General node.
4. Select the Workspace node.
5. Select the **Other** option under **Text file encoding**.
6. Select desired text file encoding (i.e. UTF-8 for Japanese language Web sites, etc.).
7. Click **Close** when finished.

3.5 Debugging Scripts

You can use features of the Eclipse IDE to debug scripts. For debugging purposes, it is not always necessary to switch to the Debug Perspective to debug a script. You can add views to the Tester Perspective, such as Breakpoints, Debug, and Expressions views. In most cases, you do not need to use the Outline, Variables, and Tasks views. This section provides tips for basic script debugging techniques.

3.5.1 Adding Views to the Tester Perspective

In some cases, you may want to add additional views to the Tester Perspective for debugging purposes. You select the view to open using the shortcut keys to get to the Show View window.

To open the Show View window:

1. Press and hold the Shift and Alt keys, then press the Q key (Shift+Alt+Q).
2. Press the Q key again. The Show View window opens.
3. If necessary, expand the Debug tree.
4. Select the View(s) you want to open:
   - Press and hold the Shift key and click to select multiple contiguous view names.
   - Press and hold the Ctrl key and click to select multiple non-contiguous view names.
5. Click **OK**.

The selected views open in the Tester Perspective.

Note: If you are in the Developer Perspective, you can add a view by selecting **Show View** from the **Window** menu and then selecting **Other**.

3.5.2 Adding Breakpoints to a Script

You can add breakpoints to the script tree view or in the Java Code to halt script execution at a specific line of code in the script.

To add a breakpoint to the script tree view:

1. Create a script project.
2. Record the script.
3. In the Script view, click the Tree View tab.
4. Expand the script tree and select the node where you want to add a breakpoint.
5. Click the right mouse button and select **Add Breakpoint** from the shortcut menu. The "[Breakpoint]" indicator appears at the end of the script node text.
6. Play back the script.

   When you play back the script, code execution will stop at the breakpoint. The **Confirm Perspective Switch** message appears the when the code execution arrives at a breakpoint. Select the **Remember my decision** option if you do not want the message to appear again during future script playback.
7. Click **No** to stay in the Tester perspective or **Yes** to switch to the Debug Perspective. You can use the following Script menu options to debug scripts:
   - **Step** - runs the currently selected node and moves the execution pointer to the next sibling node. If the selected node has a child node, the execution pointer is moved to the first child node. This option is only active during script playback and script execution is suspended while stepping through the script code.
   - **Step Into** - steps into the function or sub procedure. This option is only active during script playback and script execution is suspended while stepping through the script code. The execution pointer is moved into the beginning of the function.
   - **Pause/Resume** - pauses and resumes script playback. These options are only active during script playback.

8. You can use the following right-click shortcut menu options to debug scripts
   - **Skip/Unskip** - set the code to skip or unskip.
   - **Playback to Here** - starts playback from the beginning of the script and halts playback at selected node in the script tree.
   - **Playback from Here** - starts playback from the selected node in the script tree and plays to the end or the next breakpoint.
   - **Add Breakpoint/Remove Breakpoint** - adds or removes a breakpoint in the script tree view. Script tree nodes with a breakpoint set show the "[Breakpoint]" indicator at the end of the script node text.
   - **Execute** - executes the code for the selected node in the script tree. This option is only active during script playback and the script is paused. Execute compiles the highlighted code in the editor or tree view and runs it in the currently paused thread. However, the original running program code isn't changed and the current execution pointer does not move when using Execute. You can use Execute to test changes to a script while debugging. For changes to be permanent, you must save the script which recompiles the code and returns the execution pointer back to the beginning of the `run()` section.
   - **Step** - runs the currently selected node and moves the execution pointer to the next sibling node. If the selected node has a child node, the execution pointer is moved to the first child node. This option is only active during script playback and script execution is suspended while stepping through the script code.
   - **Step Into** - steps into the function or sub procedure. This option is only active during script playback and script execution is suspended while stepping through the script code. The execution pointer is moved into the beginning of the function.

To add a breakpoint to the script code:

1. Create a script project.
2. Record the script.
3. In the Script view, click the Java Code tab.
4. Double-click in the right-most column of the code view frame next to the code line where you want to add a breakpoint. The breakpoint indicator appears as a round dot in the frame. You can add as many breakpoints as needed.
5. Play back the script.

   When you play back the script, code execution will stop at the breakpoint. The **Confirm Perspective Switch** message appears the when the code execution arrives at a breakpoint. Select the **Remember my decision** option if you do not want the message to appear again during future script playback.
6. Click **No** to stay in the Tester perspective or **Yes** to switch to the Debug Perspective. You can use the following keyboard debug features to execute code while in debugging mode:
   - Single-step (F6) - executes the next line of code.
   - Step-into (F5) - opens the method/function class file.

Note: Source code for the JRE or for the Eclipse IDE is not included with the product. When stepping into code, an editor may appear that does not contain source code. In this case, close the editor and resume script playback. You can use the Step-into feature to step into your own custom functions that you have added to a script.

- Resume (F8) - resumes code execution to the script end or to the next breakpoint.

3.5.3 Adding a Java Exception Breakpoint

You can pause a script when any error occurs by adding a "Java Exception Breakpoint" to the Breakpoints list.

To add a Java Exception Breakpoint:

1. Create a script project.
2. Record the script.
3. Open the Breakpoints view.
4. Click the **Add Java Exception Breakpoint** icon on the Breakpoints View toolbar.
5. Type "AbstractScriptException" and click **OK** to add this exception to the breakpoint list.
6. Right-click on the breakpoint in the Breakpoints view and select **Breakpoint Properties**.
7. Select the **Suspend on Subclasses of this Exception** option in the breakpoint properties and click **OK**.

During script playback, if an exception occurs, you can correct the problem and then continue script playback.

3.5.4 Pausing and Resuming Script Playback in Debug Mode

You can pause and resume script playback using the Tree view or the Debug view.

To pause and resume play back in the Tree view:

1. Create a script project.
2. Record the script.
3. Play back the script.
4. Click the Pause toolbar button to pause playback.
5. Click the Resume toolbar button to resume playback of a paused script.

To pause and resume play back in Debug mode:

1. Create a script project.
2. Record the script.
3. In the Script view, click the Java code tab.
4. If necessary, add a Debug view to the Tester Perspective. If the Developer Perspective is open the Debug view should already be open.
5. Play back the script.

6. In the Debug view tree, select the `Thread [Iterating Agent 1]` thread and click the **Pause** toolbar button. The `Thread [Iterating Agent 1]` thread is the Virtual User's thread. You can ignore the others.
7. In the Debug view tree, select `script.run()` and click the **Resume** toolbar button to resume playback.

   If you want to resume from a specific point in a script, comment out all lines before the current one, save the script, and then resume.

You can also execute portions of the script without having to comment out lines and restart the playback.

1. Insert a breakpoint at the first line of the `run()` section.
2. Playback the script. You can execute or inspect any line when playback halts a the breakpoint.
3. Select the specific line(s) of code you want to playback, right-click and select **Execute**. You can modify the code and re-execute without having to save the script.
4. Repeat the select code, right-click, Execute process until the script works the way you want it to work.
5. Stop playback, or select the **Resume** button on the Debug view to replay from the breakpoint.

3.5.5 Inspecting and Changing Script Variable Values

You can inspect or watch script variable values to debug scripts. The script must be running or stopped at a breakpoint.

There is a difference between Java local variables and script variables. Java local variables are declared using standard Java syntax, such as `String x` or `int i`. Script variables are set using the OpenScript API methods, such as `getVariables().set("someVariable", "123"` or `http.solve()`.

To inspect the value of a script variable:

1. Create a script project.
2. Record the script.
3. Add a breakpoint to the script.
4. Play back the script.
5. At the breakpoint highlight the script code containing the variable or type the following code and highlight the code:

   ```
   getVariables().get("someVariable")
   ```

6. Right-click and select **Inspect** or **Watch**.
   - **Inspect** opens a pane that shows the variable (Shift-Alt-I adds the variable to the Expressions view).
   - **Watch** copies the variable to the Expressions view.
7. To change the value of a script variable, type the following code:

   ```
   getVariables().set("someVariable", "newValue")
   ```

8. Highlight the code.

9. Right-click and select **Execute**.

> **Note:** You can also test individual web actions by pausing the script, selecting the code for the action to test, then right-clicking and selecting **Execute** (or pressing Ctrl+U).

3.6 Enabling Debug Logging

OpenScript provides debug logging capability using Jakarta Log4j.

To enable debug logging:

1. Close OpenScript.
2. Open the file `log4j.xml` located in `C:\OracleATS\OpenScript`.
3. Locate the following section at the end of the file:

```
<!-- ======================= -->
  <!-- Setup the Root category -->
  <!-- ======================= -->

        <!-- For production  -->
  <root>
   <priority value="WARN"/>
   <appender-ref ref="AGENTFILE" />
  </root>

        <!-- For debugging
  <root>
   <priority value="DEBUG"/>
   <appender-ref ref="AGENTFILE" />
   <appender-ref ref="CONSOLE" />
  </root>
        -->
```

4. Move the ending comment brackets from:

```
<!-- For production  -->
  <root>
   <priority value="WARN"/>
   <appender-ref ref="AGENTFILE" />
  </root>

        <!-- For debugging
  <root>
   <priority value="DEBUG"/>
   <appender-ref ref="AGENTFILE" />
   <appender-ref ref="CONSOLE" />
  </root>
        -->
```

to:

```
<!-- For production
  <root>
   <priority value="WARN"/>
   <appender-ref ref="AGENTFILE" />
  </root>
        -->
```

```
      <!-- For debugging -->
<root>
 <priority value="DEBUG"/>
 <appender-ref ref="AGENTFILE" />
 <appender-ref ref="CONSOLE" />
</root>
```

5. Save the file `log4j.xml` and restart OpenScript.
6. Run scripts.

The debug messages are stored in the file `OpenScript.log` located in `<installdir>\OpenScript`.

To turn off debugging, move the ending comment braces back to the original locations.

4

Using Data Parameterization

OpenScript allows users to parameterize script data inputs to perform data driven testing. OpenScript uses the following types of data sources:

- Databank - one or more external comma-separated value (CSV) or text (TXT) files that provides inputs to script parameters. Multiple Databank files can be attached to a single script and users can specify how OpenScript assigns data during script playback. Script playback iterations can cycle through the Databank sequentially, randomly, and by shuffling the data. Databanks can be used with functional and load test scripts. See Section 4.2, "Using Script Databanks" for additional information.
- Database - a SQL query that extracts data from an Oracle Database in the same format as a Databank CSV or TXT file. See Section 4.2, "Using Script Databanks" for additional information.
- Data Table - a spreadsheet table that specifies the data by row and column. The data in the table can be entered manually or imported from an Excel spreadsheet file. The Data Table API provides methods for accessing the data in the table programmatically within functional test scripts. Data Tables are used with functional test scripts. See Section 4.3, "Using Data Tables" for additional information.
- Shared Data Service - a Shared Data Module API that provides methods for sharing data between Virtual User agents using a shared data queue or hash map. The Shared Data Service is used with load testing scripts. See Chapter 21, "Using the Shared Data Module" for additional information about using the Shared Data Service.

4.1 Understanding Data Driven Testing (Parameterization)

Data Driven Testing, or parameterization, allows you to quickly and efficiently create automated data-driven tests.

The OpenScript Modules record parameters defined by each page of the application-under-test to a script. Data sources are used to hold input data that can be automatically fed as parameters into your application when the script is run. You can use the OpenScript Data Parameterization features to define variable values in script parameters and substitute values from Databank files, Databases, or Data Tables for the variable values.

During playback, the parameters in the application are filled with values from the Data source. Databank files can be easily created or modified using any simple text editor, spreadsheet, word processor, or database application. Users can create

sophisticated unattended regression tests to thoroughly exercise the application by using varied input data.

Data Input Parameterization enables users to parameterize recorded script inputs to perform data driven testing in either the script GUI view or code view. These inputs could be form field inputs for Web applications but could also be other types of script inputs that users may parameterize. Types of inputs users may parameterize include:

- Any user entered input data (i.e. parameterize the data I entered for the search field)
- Test case values (i.e. parameterize a text string for validation so I can use different inputs for comparison during playback)
- Recorded navigations (i.e. parameterize a starting navigation so I can navigate to different host servers during playback)
- Recorded user actions/object identified (i.e. parameterize a link object path so I can click on different links during playback)

Data Input Sources enables users to drive input values from an external CSV file, Data Table/Excel file or other external data source such as a database (i.e. using a database query to pull inputs from a database table).

Data Parameterization GUI View enables users to configure the inputs they want to parameterize and the data source they want to drive the inputs from through a substitute variable GUI interface. For example, the "ticker" query string parameter for "Page [4] Ticker List" in the Tree View is set to the variable value "{{fmstocks_data.ticker,orcl}}".

Within the variable "{{fmstocks_data.ticker,orcl}}", "fmstocks_data" is the name of the Databank file, ".ticker" identifies the field name within the Databank file, and "orcl" is the recorded value.

Data Parameterization Code View Commands enable users to specify the inputs they want to parameterize and the data source they want to drive the inputs from through data parameterization in the code view.

For example, the "ticker" query string parameter for "Page [4] Ticker List" in the Tree View appears as `http.querystring(http.param("ticker", "{{fmstocks_data.ticker,orcl}}")` in the `http.get` method code in the Code View, as follows (line breaks and spacing added for clarity):

```
beginStep("[4] Ticker List", 3422);
{
   http.get(6, "http://testserver2/fmstocks/{{LINK_1_3,TickerList.asp}}",
        http.querystring(http.param("ticker", "{{fmstocks_data.ticker,orcl}}"),
             http.param("company", "")),  null, true, "ASCII", "ASCII");
}
endStep()
```

4.2 Using Script Databanks

Databanks are used to hold unlimited amounts of input data that can be automatically fed into your Web application. During playback, the parameters in the Web page are filled with values from the Databank file. The Databank and script parameter shortcut menu options allow you to map parameters in a script to fields in a Databank file as variable names.

Scripts must be configured to use Databanks. Use the options on the **Assets** tab of the script view to specify the Databank file(s) to use with a script. Scripts can be configured to use more than one Databank file.

When you record a script that has a navigation that uses parameters, the parameter nodes appear under the Query String node:

In the Code View, the parameters appear in the `http.param` parameters of the `http.querystring` parameter.

When you configure the Databank(s) to use with the script, the Get next Databank record from *databank name* node and Java code are added to the script.

Select the script parameter node to map to a Databank and use the **Substitute Variable** option on the right-click shortcut menu to select the Databank field name to map to the parameter. The Databank file and field name appear in the parameter node of the script tree.

The variable appears in the Code view in the `http.param` parameters of the `http.querystring` parameter.

Use the Playback iterations to playback using the records in the Databank. You can also use custom code to loop through Databank records and assign values to variables.

4.2.1 Configuring Databanks

You must configure the Databank to use with a script before you can get records from the Databank to use in a script.

To configure Databanks to use with a script:

1. Open or create a Script project.
2. Select the **Assets** tab in the script view. See Section 3.3.8, "Adding Script Assets" for additional information about adding script assets.
3. Select **Databanks**.
4. Click **Add**.
5. Select **Databank**.
6. Select **CSV file** or **Database**. Once a databank is defined as CSV or Database (SQL), the databank type cannot be changed to the other type.
7. For CSV files:
   a. Select the Repository from the **My Repositories** tree.
   b. Select the Databank file from the repository.
   c. Set the **Type** to Databanks (*.csv, *.txt).

      Type: Specifies the type of databank file to add to the script (*.csv, *.txt).
   d. Select the **Charset** to use.

      Charset - specifies the character set encoding used for the databank file. The suggested charset encoding of the databank .csv file is the native charset of user machine. For US machines the suggested encoding is cp1252. For East European machines, the suggested charset is cp1251. If a databank file was saved with UTF-8 encoding with Unicode byte order mark (BOM), OpenScript detects it, and sets encoding to UTF-8. If the charset used for the databank file is different from the charset of the user machine or UTF-8 with BOM, then you must the correct charset. Otherwise, the databank will not be read correctly

and may cause a script failure. You can select the correct charset from the Charset list or enter the correct Charset in the field.

During playback, the Agent will define the charset for reading the databank file in following order:

- If file has UTF-8 encoding with BOM, then it will be used.
- The Charset specified during asset configuration will be used.
- If no charset specified (case of 9.1 scripts), then UTF-8 encoding will be used.

e. Enter an alias name to use for the Databank or leave the default alias name. The default alias name is the name of the .CSV Databank file.

Alias: Specifies an alias name to use for the Databank. The Databank file name is the default. The Databank alias name is the name that appears when you add a Databank record retrieval node to a script tree.

f. Set the **Relative to** option. The **Relative to current script** and **Relative to a repository** options specify how the current script will locate the specified script asset. The **Relative to a repository** option locates the script asset by a repository path such as, `[Repository: Default] Default!/WebTutor`, if the asset is selected from a repository. The **Relative to current script** option locates the script asset by a relative path such as `../WebTutor`. Selecting the The **Relative to current script** option is not recommended as script-relative paths are more brittle than repository-relative paths if scripts are moved or shared.

The following are guidelines when using script assets in a team or distributed environment:

- Do not use Absolute Paths when referring to assets or saving assets. Oracle Load Testing does not support absolute paths.
- OpenScript, Oracle Test Manager, Oracle Load Testing, and all command-line agents should all use the same shared repository names and paths.
- Do not refer to an asset in another repository by a relative path.

g. Click **OK**.

h. Click **OK** to add the Databank file.

8. For Databases, a Databanks Database Assets dialog box appears. This dialog box lets you specify the database and query to use as a databank. Contact your Database Administrator for the appropriate settings for your database. The following options are available:

a. Specify the Database Driver.

Oracle Thin - This driver option applies to Oracle databases.

- **Hostname** - Specify the host name of the machine running the database. This is not required for a JDBC:ODBC or Custom driver setting.
- **Port** - Specify the port for the driver you selected. For example, the default port for an Oracle Thin JDBC driver is 1521. Modify the port number if necessary. This is not required for a JDBC:ODBC or Custom driver setting.
- **SID** - Specify the database or server ID.
- **Service name** - Enter the Service name used for the Oracle database.

ODBC - This driver option is available as an option for SQL and Oracle databases and any other database for which you have a JDBC:ODBC Bridge driver.

- **Data Source** - Specifies the data source for the ODBC driver.

b. Specify the URL, username, password, query string, and databank alias.

- **URL** - Specifies the URL to use to connect to the database.
- **Username** - Enter the username for connecting to the database, if required for authentication.
- **Password** - Enter the password for connecting to the database, if required for authentication.
- **Query** - Specify a single SQL query that returns all the rows needed as databank values. The SQL query cannot contain PL/SQL or SQL*Plus code. Only pure SQL is supported. You must ensure that the query returns the column names (i.e. databank fields) that the script expects.

  If you have a large database-backed databank, but will only use a small portion of the records in the test, then use the "Where" clause in the SQL query to minimize the amount of records retrieved from the database. For example, if you have database with 200000 records and only need to have 100 iterations retrieving records 201 through 301 sequentially starting from record 201. use a query such as `Select * From LargeTable Where id > 200 AND id < 302`. The start record will be 1 and no is range set. This reduces the databank preparation time and minimizes the amount of records retrieved from the database.

  Use the "Order By" clause in the query to make sure the results returned from the database are ordered as intended. For example, if you have a database table with Columns `id`, `firstName`, and `lastName` that is populated it with the following data:

  ```
  1, John, Smith
  2, Jane, Doe
  [...]
  400, Maria, Sanchez
  [...]
  200000, Sachin, Rajaram
  ```

  If you use the query `Select * From users`, the results of query could be unordered, meaning the first record in databank could be 400, Maria, Sanchez instead of 1, John, Smith. Using the query `Select * From users Order By id` would order the first databank record as 1, John, Smith as expected.

- **Alias**: Specifies an alias name to use for the Databank. The Databank alias name is the name that appears when you add a Databank record retrieval node to a script tree.

c. Click **Test** to verify the connection to the database.

d. Click **OK**.

e. Click **OK** to add the Databank file. For Databases used as databanks, a copy of all data is retrieved and indexed before the start of the test. The data is not read live during the test. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about playing back scripts with databanks.

4.2.2 Creating or Editing Databank Files

Databank files are comma-separated value (".csv" or ".txt") files with the addition of formatting rules specific for databanks and rules derived from Excel formatting. Databanks can also be data retrieved from a database using an appropriate query to that generates data that conforms to the .csv databank file formatting rules.

When you open a Databank file from the Assets tab of the Script view, the Databank file opens in a text editor view. You can edit the Databank file directly in the text editor view. You can also create or edit databank files using another text editor or spreadsheet that can export to .csv formatted text files.

The general databank file formatting rules are as follows:

- The first line of the databank defines the field headers (column titles). A comma is used as the field header delimiter (no spaces). The field header names are user defined. For example, `FirstName,LastName,Mail,Phone` defines four field headers for a databank file. The field headers can be referenced in the script code to specify valid databank variables. For example, if the first line of a databank with the alias name "myDB" contains the field headers `user` and `password`, the following databank variables are valid in a script configured to use the "myDB" databank: `db.myDB.user` and `db.myDB.password`.
- Each line in the file following the field headers defines a databank record.
- A line can end with a Line Feed (LF) character or Carriage Return/Line Feed (CR LF) characters.
- Each databank record consists of field data (columns). A comma is used as the field delimiter (different line for each record, no spaces around commas). For example, `John,Smith,JohnS@company.com,x993` defines the field data for a databank record corresponding to the field headers `FirstName,Lastame,Mail,Phone`.
- Each databank record *must* have the same number of fields as the number of field headers. For example, if a databank file has four field headers in line 1 as `FirstName,LastName,Mail,Phone`, each databank record on lines 2 through *n* must have four field data columns in each record. The databank field data record `john,smith,JohnS@company.com,x993` is correct. However, `john,smith,JohnS@company.com` is incorrect as this record contains only three fields. Insert an extra comma to leave a field column blank. For example `Sachin,Bhat,,x783`. As follows:

  ```
  FirstName,LastName,Mail,Phone
  John,Smith,JohnS@company.com,x993
  Mary,Ellen,MaryE@company.com,x742
  Sachin,Bhat,,x783
  ```

- A quotation mark (`"`) is used as an escape character for embedding new line characters (LF, CR) or comma (,) inside of a databank record. All escaped records, regardless if it has embedded LF, CR, or comma or not, should start with a quotation mark and end with a quotation mark followed by comma or CR/LF. For example, if a data value contains a comma, place quotation marks around the value, as follows:

  ```
  John,Smith,"Anytown, MA",(603) 993-0000
  ```

  New lines may be embedded inside of quotation marks, as follows:

  ```
  field1,"field2 contains two lines: Line one.
  Line two.",field3
  ```

To use a quotation mark as itself not as an escaped character, escape the quotation mark. The correct format is `""`. Quotation marks in the middle of a record should be escaped always. For example the following record,

```
THIS IS BEGINNING AND ""THIS IS END""
```

is formatted correctly. The following record,

```
THIS IS BEGINNING AND "THIS IS END"
```

is not formatted correctly.

- Blank lines are stripped out and ignored.

The character encoding of the CSV file is determined by an (optional) byte-order-mark (BOM) at the beginning of the file. Programs such as Notepad++ or Excel set this byte-order mark when users save a text document with a specific encoding character set like UTF-8. If no byte-order mark is specified, the CSV reader uses character set assigned to a databank asset, when the user adds the databank asset to a script, or uses the current platform's default character set to read the file (for example, cp1252 on most Windows English installations) for legacy databanks prior to Version 9.2.

4.2.3 Getting Databank Records

To get Databank records to use with a script:

1. Open or create a script project.
2. Configure the Databank to use with a script in the Script Assets Properties.
3. Select the script node where you want to use the Databank record.
4. Select the **Script** menu and then select **Other** from the **Add** sub menu.
5. Expand the **General** node and select **Get Next Databank Record**.
6. Select the databank or click **New** to add a new databank.
7. Select the Record.
8. Click **OK**.
9. Select the Databank alias to specify the Databank file to get the record from.
10. Click **OK**. A GetNextDatabankRecord: *databank alias* node will be added to the script.

    In the Java Code view, a `getDatabank("`*`databank alias`*`").`*`method`*`();` will be added to the script code depending upon the type of record selected:

    ```
    getDatabank("customer").getNextDatabankRecord();
    getDatabank("customer").getFirstRecord();
    getDatabank("customer").getLastRecord();
    getDatabank("customer").getRecord(5);
    ```

11. Right click the parameter node in the script tree that you want to substitute with a Databank variable and select **Substitute Variable**.
12. If necessary, expand the Databanks node and select the Databank field you want to use as the input parameter data.
13. Click **Finish**.
14. The script node name/value pair changes to show the Databank alias name, field name, and recorded value as a variable value. For example:

```
login = {{db.customer,login,ta906}}
```

In the Java Code view, the parameter code changes to show the Databank alias name, field name, and recorded value as a variable value. For example:

```
http.postdata(http.param("login", "{{db.customer,login,ta906}}")
```

15. Click the Playback toolbar button to playback the script once to verify the it plays back correctly.

4.2.3.1 Getting Databank Records Using the API

You can use the additional API methods available with `getDatabank("databank alias")` in the Java Code view to retrieve specific records from the databank. This section provides examples of the available methods.

4.2.3.1.1 Databank API Usage Notes

These API methods are not compatible with databank's iteration settings **Randomly** or **Shuffle Records**. The Databank Exception "incompatible with db setting" will be thrown if these methods are used with the **Randomly** or **Shuffle Records** iteration settings.

The records obtained through these API calls are not counted against the usage count of all records. It is possible for an infinite script loop to occur if the **When Out of Records** iteration setting is set to **Stop the User**, but the script only uses these API calls to read records.

4.2.3.1.2 Loading a Databank

The following example uses the `load()` method to get a new databank that will override a statically defined databank in OpenScript:

```
getDatabank("alias").load("repository", "dbPathRelToRepository", "settings");
```

The `load()` method is used to programmatically override static databanks in a script. The `load()` method is used with functional testing scripts only. The parameters for the `load()` method are as follows:

`repository` - a String specifying the repository to look inside to locate the databank file. Valid repositories are mapped using the **Manage Repositories** options on the **Tools** menu. An example repository could be named "Default" and map to C:\OracleATS\OFT.

`dbPathRelToRepository` - a String specifying the path to the databank file within the named OpenScript repository. The file extension is not required. For example, it can be specified as "databank1", "databank1.csv", or "databank1.txt". Any leading file separator on the path, such as `/` or `\` is ignored. The *`dbPathRelToRepository`* parameter cannot be `null`.

`settings` - a String specifying the databank settings to apply to the loaded databank. For example `"startIndex=10:select=SEQUENTIAL:whenOut=LOOP_FOREVER"`. See the `-dbopts` settings in the General Section of Appendix A.1, "Specifying Command Line Settings" for the databank settings. If `null`, the settings revert to the default databank settings.

The following example shows how to load a databank file "euroCustomer" dynamically from the Databanks/files folder in the default repository and use the default settings.

```
public void run(){
```

```
 getDatabank("customer").load("default", "Databanks/files/euroCustomer", null);
 getDatabank("customer").getNextDatabankRecord();
 web.textbox("....").setText("{{db.customer.column1}}");
}
```

The following example shows how to load a databank file "euroCustomer.csv" to dynamically override the statically mapped databank file "customer.csv" from the Databanks/files folder in the default repository and use the default settings.

```
public void run(){
 //Alias "customer" is defined as a static Databank Asset customer.csv.
 getDatabank("customer").getNextDatabankRecord();
 web.textbox("....").setText("{{db.customer.accountNumber}}");

 //Override Alias "customer" with databank "euroCustomer.csv"
 getDatabank("customer").load("default", "Databanks/files/euroCustomer", null);
 getDatabank("customer").getNextDatabankRecord();
 web.textbox("....").setText("{{db.customer.accountNumber}}");
}
```

4.2.3.1.3 Getting a Record Count

The following example uses the `getDatabankRecordCount()` method to get the record count from the "customer" databank and prints the value to the Results view:

```
int recordCount = getDatabank("customer").getDatabankRecordCount();
info("Record Count = " + recordCount );
```

4.2.3.1.4 Getting a Specific Record

The following example uses the `getRecord(`*`n`*`)` method to get a specific record from the "customer" databank and prints the value to the Results view:

```
getDatabank("customer").getRecord(5);
```

The following code example use a `For` statement to loop through all records in the databank:

```
int recordCount = getDatabank("customer").getDatabankRecordCount();
for (int i=1; i<=recordCount; i++) {
   info("Record count = " + i + " of " + recordCount);
   getDatabank("customer").getRecord(i);
}
```

4.2.3.1.5 Getting the First Record

The following example uses the `getFirstRecord()` method to get the first record in the "customer" databank:

```
getDatabank("customer").getFirstRecord();
```

4.2.3.1.6 Getting the Last Record

The following example uses the `getLastRecord()` method to get the last record in the "customer" databank:

```
getDatabank("customer").getLastRecord();
```

4.2.4 Playing Back Scripts With Iterations

OpenScript allows repetitive playback of navigations in a script. The iterations can be performed with or without databanks.

1. Start OpenScript.
2. Open the script to play back.
3. Configure the script to use a databank as described in Section 4.2.1, "Configuring Databanks".
4. Select **Iterate** from the **Script** menu or click the toolbar button. The resulting dialog box has the following options:

   Use Databanks: When selected, databanks will be used for script playback. Databanks configured for the script show the following settings:

   - **Name:** Lists the alias name(s) for the databank file(s).
   - **Range:** Lists the range of databank records to use for script playback. This list corresponds to the **Range** option selected for each databank file.
   - **Start:** Lists the starting databank record to use for script playback. This list corresponds to the **Starting Record** specified for each databank file.
   - **Select Record:** Lists the how databank records are selected for script playback. This list corresponds to the **Select Next Record** setting selected for each databank file.
   - **When Out of Records:** Lists the action to take when the databank file is out of records during script playback. This list corresponds to the **When Out of Records** setting selected for each databank file.
   - **Data:** Lists the data in the Starting Record of each databank file.

   Databank Source: This section shows the following information about the selected databank:

   - **Alias:** Shows the alias name of the selected databank file and the number of rows in the file.
   - **Type:** Shows the type of the selected databank file. Databanks can be CSV text files or databases.
   - **Source:** Shows the path and filename of CSV text files or the database query used for database databanks. While there is not a maximum file size, the recommended maximum sizes is 200 MB. The only limitation is how long it takes to generate the index. The databank must be indexable within 30 seconds, by default. This setting is configurable in the "Databank Setup Timeout" setting in the General Preferences.

   Databank Settings: This section specifies the settings to use for the selected databank:

   - **Advance to Next Record:** Specifies when the virtual user should advance to the next databank record during script playback. The master script being played is always the script that triggers when an iteration occurs. The following options are available:
     - **When Script Requests a Record:** The databank record advances every time a script explicitly requests a record during script playback. A record request corresponds to the script Java code calling the `getDatabank(alias).getNextRecord()` method. This is the default behavior.
     - **Each Occurance:** The databank record advances when a script refers to a databank column (i.e. databank field) in the script. A record request corresponds to the script Java code evaluating a parameterized value such

as `{{db.fmstocks_data.ticker}}`. You can specify that any column advances to the next record or specify a particular databank column advances to the next record. For example, if you have an employee databank with a `firstName` field and the `firstName` column is specified as the **Column** value, the databank record advances only when the `{{db.employees.firstName}}` value in the script Java code is evaluated on script playback. Select the databank field name as the **Column** value or select <Any> to allow any field to advance the databank record.

  - **Each Iteration of Script:** The databank record advances before a script containing the databank starts another playback iteration.

- **Select Next Record:** Specifies how a new record is selected from the databank when the databank record advances. The following options are available:
  - **Sequentially:** The databank records increment one by one in sequential order from the start of the specified range. When multiple virtual users are running, records are distributed in sequential order across all virtual users.
  - **Randomly:** The databank records are selected at random from the databank. The same record may be used multiple times before all records are exhausted. Random record selection is only provided for databanks that can be indexed. When configuring databank settings, if the databank file is too large to index, the **Randomly** or **Shuffle** record options may not be available. The **When Out of Records** setting does not apply when Random is selected.
  - **By Shuffling:** The databank records are selected at random from the databank ensuring that once a record is selected, it is never selected again. The setting works similar to selecting a random card from a deck until no cards are left. Shuffle mode only supports databanks containing fewer than 200,000 records. For databanks containing more than 200,000 records, you can shuffle the values in the actual data file or you should use the **Randomly** mode.
  - **Use Seed:** Specifies a randomization seed value to use when using the **Randomly** or **Shuffle** modes. Use the same seed across multiple tests to create the same sequence of random numbers for all tests. If 0 or not specified, a seed is generated automatically based on the current time.
- **When Out of Records:** Specifies the action the virtual user takes if all databank records in the specified range have been used and a new record is requested. The following options are available:
  - **Loop Over Range:** Loops back to the first record in the range after all records in the range are used and continues distributing records. Use the **Maximum Iterations** settings to prevent the virtual user from running forever.
  - **Keep the Same Record:** Continues to use the last record requested after all records in the range are used. No additional records are requested from the databank. Any calls in the Java code to `getNextDatabankRecord()` are ignored after all records are used. Custom Java code may be used in the script to have Virtual users request an individual record using `getRecord(`*`n`*`)`, `getLastRecord()`, or `getFirstRecord()`.
  - **Stop the User:** The virtual user immediately stops running the next time a record is requested from the databank after all records in the range are

used. The virtual user will stop regardless of how many iterations are specified by the **Maximum Iterations** settings.

- **Range:** Specifies the range of records to use. The following options are available:
  - **All Records:** When selected, the virtual user uses all records in the databank. The first record is 1.
  - **Specific Records:** When selected, the virtual user uses a subset of records in the databank. Specify the first and last records to use for the range. The range includes both the starting and ending record in the specified range.
- **Starting Record:** Specifies which databank record to use first. The first record in a databank is 1. The starting record must be within the specified range of records. For example if you select **Specific Records** and set the range to 5:10, the starting record must be at least 5, but not more than 10.

Maximum Iterations: This section specifies the maximum number of iterations of a main script's `run()` section to complete:

- **Run no more than [ ] iterations:** Specifies the maximum number of iterations. If a databank exhausts all records and **When Out of Records** specifies **Stop the User,** the virtual user will always stop running, even if the specified number of iterations has not completed.

5. Select **Use Databanks**.
6. Select which databank file to specify the settings for if more than one database is configured for the script.
7. Specify the settings for the databank file.
8. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations.
9. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

4.2.4.1 Notes and Limitations

Certain setting combinations are not allowed, or may cause exceptions when the script is run. The following are situations to be aware of when using iteration options.

1. The **When Out of Records** option is not available when **Select Next Record** is set to **Randomly**. When random is selected, an infinite supply of random records exists.
2. Virtual users may still request an individual record using `getRecord(n)` after all records are used up, and **When Out of Records** is set to **Keep the Same Record**.
3. The `getRecord(n)`, `getFirstRecord()`, and `getLastRecord()` Java code methods do not advance the record cursor used by `getNextDatabankRecord()`. Therefore:

   ```
   getNextDatabankRecord();// returns 1
   getRecord(7);// returns 7
   getNextDatabankRecord();//returns 2, not 7
   ```

4. The `getRecord(n)`, `getFirstRecord()`, and `getLastRecord()` Java code methods throw an exception when they are invoked if **Select Next Record** is set to **Shuffle Records** or **Randomly**.

5. The `getRecord(n)`, `getFirstRecord()`, and `getLastRecord()` Java code methods throw an exception if they are invoked and the databank is not indexed.
6. **Use Seed** is only available when **Select Next Record** is set to **Shuffle Records** or **Randomly**.
7. A specific databank range and starting index may not be set if the databank cannot be indexed.
8. The **Select Next Record**: **Shuffle Records** and **Randomly** options are only allowed when the databank can be indexed.
9. The **Select Next Record**: **Shuffle Records** is only allowed when the databank can be indexed and when there are fewer than 200,000 records.

4.2.4.2 Using Very Large Databanks

If you want to use an extremely large databank (for example, records in the millions), use the follow procedure:

1. Make sure the script does not use these databanking methods in the script code:

```
getRecordNumber()
getFirst()
getLast()
getRecord(n)
```

2. Set the Databank Setup Timeout to a very *small* value, for example, 1 second.
   a. Select **OpenScript Preferences** from the **View** menu.
   b. Select **General** in the preferences.
   c. Set **Databank Setup Timeout** to 1 (sec).
3. Set the databank setting to Sequential mode.
   a. Select **Iterate** from the **Script** menu.
   b. Make sure **Select Next Record** is set to "Sequentially".
4. Save the script and playback in Oracle Load Testing. The index preparation will timeout (expected). Although the index will not be generated, Oracle Load Testing will still be able to run with the databank using Sequential mode and deliver records to Virtual users.

4.3 Using Data Tables

Data Table is a script service that allows users to use an Excel file as a data input and output for scripts. The Data Table is a spreadsheet-like data table for Functional testing scripts. As a script service, a Data Table exists only during script playback. If you need to persistant data from the Data Table, the data can be saved to an Excel file. While Excel file data can be imported and exported to and from a Data Table, the OpenScript Data Table is not as robust as Excel in handling custom formats. The Data Table is primarily used for data manipulation. To avoid format incompatibilities you should use only basic formats like General and String. Use Numeric and Date format only if absolutely needed.

There are two versions of the Data Table: the view/edit Data Table and the runtime Data Table. The view/edit Data Table can be accessed using the **Data Table** option on the **View** menu. The view/edit **Data Table** tab appears in the tab views of the OpenScript main window. The view/edit Data Table content can be changed by manually inputting data into cells or by importing an Excel file before playback.

The Data Table API provides methods for accessing the data in the Data Table programmatically within functional test scripts during playback of a script.

When you play back a script and select the results in the Results view, the Details view includes a Result Data Table tab that shows the runtime Data Table resulting from the playback of the script. Changes made to the runtime Data Table during script playback do not appear in the view/edit Data Table. The Result Data Table can be exported to an Excel file to view the changes.

The following sections explain how to use Data Tables within functional test scripts.

4.3.1 Enabling the Data Table Service

The Data Table Service provides programmatic access to data stored in a Data Table using the Data Table API. The Data Table service must be enabled to provide access to the Data Table view and the runtime Data Table.

To enable the Data Table Service:

1. Record a functional test script.
2. Select **Script Properties** from the **Script** menu.
3. Select **Modules**.
4. Select the **Data Table** option and click **OK**.
5. Select **Data Table** from the **View** menu to show the view/edit Data Table tab view. The runtime Data Table is accessed usingthe Data Table API.

4.3.2 Setting the First Row Policy

The First Row policy specifies how the data in the first row of the Data Table will be used. The first row policy should be set before any data is manipulated within a Data Table sheet. The first row policy is important because Data Tables and Excel differ in the use of data as column headers. Data Tables can use the first row of data as column headers. Excel files use fixed indexes (for example, A, B, C) as the column headers. If your script manipulates data and columns in a Data Table during script playback, how the columns are referenced is important. For example, if you add a column in an Excel file, the new column inserts into the sheet essentially moving the current data to be under a different column header. If a new column is added before column C, the new column is inserted as column C and the data that was under column C is now under column D. If you add a column in a Data Table and the first row policy is not to use the first row of data as the column header, the Data Table will perform the same as an Excel file. However, if the first row policy is set to use the first row of data as the column header, the inserted column will have a new column header inserted, and the existing data will still be under the same column header as before the new column was added. If your script imports and exports the manipulated data from a Data Table to and from an Excel file, the first row policy will affect how the data is referenced.

To set the first row policy, right-click in the Data Table view and select **Use First Row as Column Header**. The data values in the first row of the table move up into the table header as column names.

In the Java Code view, the first row policy for each Data Table sheet is set using the `useFirstRowAsColumnHeader()` method, as follows:

```
datatable.useFirstRowAsColumnHeader("Sheet1");
```

4.3.3 Entering Data Manually

Data can be entered into the view/edit Data Table manually by editing individual cell contents. You can right-click on a data table cell to open the shortcut **Edit** menu.

To enter data into the view/edit Data Table manually:

1. Enable the Data Table service for the functional test script and show the Data Table view.
2. Right-click on the table cell and select from the shortcut menu. The Data Table right-click edit menu can be used to add data to the Data Table manually. The following options are available on the shortcut menu:
   - **Edit**: Changes the selected cell to text edit mode. Type data into the cell and press Enter.
   - **Cut**: Cuts the data from the selected cell.
   - **Copy**: Copies the text for the selected cell to the clipboard.
   - **Paste**: Pastes text from the clipboard to the selected cell.
   - **Delete**: Deletes the text from the selected cell.
   - **Insert Row Before**: Inserts a new row into the table before the selected row.
   - **Insert Row After**: Inserts a new row into the table after the selected row.
   - **Delete Row**: Deletes the selected row from the table.
   - **Use First Row as Column Header**: Sets the policy for how the first row of data in each Data Table sheet is used. When selected, the first row of data in the sheet is used as the column header value. When not selected, the first row of data is not used as the column header. The first row policy should be set before any data is manipulated within a Data Table sheet. See Section 4.3.2, "Setting the First Row Policy" for additional information.
   - **Insert Column Before**: Inserts a new column into the table before the selected column.
   - **Insert Column After**: Inserts a new column into the table after the selected column.
   - **Rename Column**: Opens a dialog box for specifying a new heading name for the selected column.
   - **Delete Column**: Deletes the selected column from the table.
3. Type the data into the table cell and press Enter.

Additional worksheets can be added to the Data Table or removed from the Data Table using the options on the worksheet shortcut menu.

To add worksheets to or remove worksheets from a Data Table:

1. Right-click on a worksheet tab in the Data Table view to open the worksheet shortcut menu. The Data Table right-click worksheet menu can be used to add worksheets to and remove worksheets from the Data Table manually. The worksheet shortcut menu has the following options:
   - **Insert Sheet Before**: Inserts a worksheet tab into the Data Table before the selected worksheet tab.
   - **Insert Sheet After**: Inserts a worksheet tab into the Data Table after the selected worksheet tab.

- **Rename Sheet**: Opens a dialog box for specifying a new name for the selected worksheet tab.

  Note: Excel restricts sheet names to 31 characters. Other spreadsheet applications may allow longer sheet names. The Data Table conforms to the Excel restrictions. If a sheet name is longer than the 31 character restriction, then the Data Table truncates the length of name to 31 characters. Any illegal characters, such as colon (:), backslash (\), asterisk (*), question mark (?), forward slash (/) ,opening square bracket ([) losing square bracket (]), will be substituted with the underscore (_) character.

- **Delete Sheet**: Deletes the selected worksheet from the Data Table. A confirmation dialog box appears to confirm the deletion.

2. Use the worksheet shortcut menu options as needed to add or remove worksheets in the Data Table.

4.3.4 Importing Data from a Spreadsheet File

Data can be loaded into a Data Table from an Excel spreadsheet file.

Note: Excel restricts sheet names to 31 characters. Other spreadsheet applications may allow longer sheet names. The Data Table conforms to the Excel restrictions. If a sheet name is longer than the 31 character restriction, then the Data Table truncates the length of name to 31 characters. Any illegal characters, such as colon (:), backslash (\), asterisk (*), question mark (?), forward slash (/) ,opening square bracket ([) losing square bracket (]), will be substituted with the underscore (_) character.s

To load data from an Excel spreadsheet file:

1. Enable the Data Table service for the functional test script and show the Data Table view.
2. Click the **Import** toolbar button in the Data Table View. The Data Table Import dialog box has the following options:

   File: Specifies the name of the file to import. Click the **Browse** button to select a file.

   Sheets: Lists the worksheets in the selected spreadsheet file. Select or clear the check boxes to specify which worksheets to import.

   Overwrite existing sheet: When selected, existing worksheets with the same name will be overwritten in the Data Table.

   Use first row as column name: When selected the first row of the imported spreadsheet file is used as the column name, replacing the default column names.
3. Enter or select the file to import.
4. Select the worksheets to import.
5. Select or clear the Overwrite and Use first row options.
6. Click **OK**.

4.3.5 Exporting Data to a Spreadsheet File

Data in a Data Table can be exported to an Excel spreadsheet file.

To Export data from a Data Table to an spreadsheet file:

1. Enable the Data Table service for the functional test script and show the Data Table view.
2. Click the **Export** toolbar button in the Data Table View. The Export Data Table to Excel document dialog box has the following options:

   Sheets: Lists the worksheets in the Data Table. Select or clear the check boxes to specify which worksheets to export.

   File: Specifies the name of the file to export. Click the **Browse** button to specify the name and location to save the file.

   Full path: Shows the full path and file name where the Data Table data will be exported.
3. Specify a file name or use the **Browse** button to save a new file.
4. Click **OK**.

4.3.6 Changing Data During Script Playback

The runtime Data Table content can be changed during script playback in the following ways:

- manually when playback is paused by a breakpoint
- manually when playback is paused by an exception
- manually when playback is paused using the Pause toolbar button
- programmatically at runtime using the `datatable` API

Changes to the runtime Data Table can be saved can be saved to the script's session result folder during script playback using the `datatable.save` method. The view/edit Data Table and Result Data Table can be exported to an Excel file.

The following sections provide examples of how to change data in the runtime Data Table programmatically using the `datatable` API.

4.3.6.1 Getting and Setting Cell Values

You can use the `datatable` API to get and set Data Table values programmatically during playback of a script. The following examples show how to get and set values using the `datatable` API `getValue()` and `setValue()` methods. Data table rows are 0 based. Cell A1 in the data table is accessed by `datatable.getValue(0, "A")`.

4.3.6.1.1 Getting Data by Row and Column Value

The following example script code retrieves the value in the cell at row 1, column A of the current worksheet and prints the value to the Results view:

```
String cellValue1 = datatable.getValue(0, "A").toString();
info("cell value = " + cellValue1);
```

4.3.6.1.2 Getting Data by Sheet, Row, and Column Value

The following example script code retrieves the value in the cell at row 1, column A of the worksheet named "Sheet1" and prints the value to the Results view:

```
String cellValue2 = datatable.getValue("Sheet1", 0, "A").toString();
info("cell value = " + cellValue2);
```

4.3.6.1.3 Setting Data by Row and Column Value

The following example script code sets the value in the cell at row 1, column A to a boolean value of `true`:

```
datatable.setValue(0, "A", true);
```

The following example script code sets the value in the cell at row 1, column A to a double value of `10.5`:

```
datatable.setValue(0, "A", 10.5);
```

The following example script code sets the value in the cell at row 1, column A to a String value of `myString`:

```
datatable.setValue(0, "A", "myString");
```

4.3.6.1.4 Setting Data by Sheet, Row, and Column Value

The following example script code sets the value in the cell at row 1, column A of "Sheet1" to a boolean value of `true`:

```
datatable.setValue("Sheet1", 0, "A", true);
```

The following example script code sets the value in the cell at row 1, column A of "Sheet1" to a double value of `10.5`:

```
datatable.setValue("Sheet1", 0, "A", 10.5);
```

The following example script code sets the value in the cell at row 1, column A of "Sheet1" to a String value of `myString`:

```
datatable.setValue("Sheet1", 0, "A", "myString");
```

4.3.6.2 Adding and Deleting Rows and Columns

You can use the `datatable` API to add and delete Data Table rows and columns programmatically during playback of a script. The following examples show how to add and delete rows and columns using the `datatable` API `addColumn()`, `deleteColumn()`, `insertRow()`, and `deleteRow()` methods.

4.3.6.2.1 Adding Columns

The following example script code adds a new column named `New Column` to the current worksheet after the last column in the worksheet:

```
datatable.addColumn("New Column");
```

The following example script code adds a new column named `New Column` to the current worksheet before the column with an index value of `0`:

```
datatable.addColumn("New Column", 0);
```

The following example script code adds a new column named `New Column` to the worksheet named "Sheet1" after the last column in the worksheet:

```
datatable.addColumn("Sheet1", "New Column");
```

The following example script code adds a new column named `New Column` to the worksheet named "Sheet1" before the column with an index value of `0`:

```
datatable.addColumn("Sheet1", "New Column", 0);
```

4.3.6.2.2 Deleting Columns

The following example script code deletes the column named `A` from the current worksheet:

```
datatable.deleteColumn("A");
```

The following example script code deletes the column named `A` from the current worksheet named "Sheet1":

```
datatable.deleteColumn("Sheet1", "A");
```

4.3.6.2.3 Adding Rows

The following example script code adds a new row to the current worksheet before the row with an index value of `0`:

```
datatable.insertRow(0);
```

The following example script code adds a new row to the worksheet named "Sheet1" before the row with an index value of `0`:

```
datatable.insertRow("Sheet1", 0);
```

4.3.6.2.4 Deleting Rows

The following example script code deletes the row before the row with an index value of `1` from the worksheet named "Sheet1":

```
datatable.deleteRow("Sheet1", 1);
```

4.3.6.3 Adding and Deleting Worksheets

You can use the `datatable` API to add and delete worksheets programmatically during playback of a script. The following examples show how to add and delete worksheets using the `datatable` API `addSheet()` and `deleteSheet()` methods.

4.3.6.3.1 Adding Worksheets

The following example script code adds a new worksheet named "Sheet1" to the Data Table:

```
datatable.addSheet("Sheet1");
```

The following example script code adds a new worksheet named "Sheet1" to the Data Table before "Sheet2":

```
datatable.addSheet("Sheet1", "Sheet2");
```

4.3.6.3.2 Deleting Worksheets

The following example script code deletes the worksheet named "Sheet1" from the Data Table:

```
datatable.deleteSheet("Sheet1");
```

4.3.6.4 Getting Worksheet, Row, and Column Counts

You can use the `datatable` API to get sheet, row, and column counts programmatically during playback of a script. The following examples show how to get sheet, row, and column counts using the `datatable` API `getSheetCount()`, `getRowCount()`, and `getColumnCount()` methods.

4.3.6.4.1 Getting Worksheet Counts

The following example script code gets the sheet count from the Data Table and prints the value to the Results view:

```
int sheetCount = datatable.getSheetCount();
info("Sheet count = " + sheetCount);
```

4.3.6.4.2 Getting Row Counts

The following example script code gets the row count from the current worksheet and prints the value to the Results view:

```
int rowCount = datatable.getRowCount();
info("row count = " + rowCount);
```

The following example script code gets the row count from the worksheet named "Sheet1" and prints the value to the Results view:

```
int rowCount1 = datatable.getRowCount("Sheet1");
info("row count Sheet1 = " + rowCount1);
```

4.3.6.4.3 Getting Column Counts

The following example script code gets the column count from the worksheet named "Sheet1" and prints the value to the Results view:

```
int columnCount = datatable.getColumnCount("Sheet1");
info("column count Sheet1 = " +columnCount);
```

The following example script code gets the column count from the worksheet with an index of `0` and prints the value to the Results view:

```
int columCount0 = datatable.getColumnCount(0);
info("column count Sheet index 0 = " + columnCount);
```

4.3.6.5 Getting the Current Sheet and Row

You can use the `datatable` API to get the current sheet, row, and column programmatically during playback of a script. The following examples show how to get the current sheet, row, and column using the `datatable` API `getCurrentSheet()` and `getCurrentRow()` methods.

4.3.6.5.1 Getting the Current Sheet

The following example script code gets the name of the current sheet from the Data Table and prints the value to the Results view:

```
String currentSheet = datatable.getCurrentSheet();
info("Current Sheet = " + currentSheet)
```

4.3.6.5.2 Getting the Current Row

The following example script code gets the current row from the current worksheet and prints the value to the Results view:

```
int currentRow = datatable.getCurrentRow();
info("Current row = " + currentRow);
```

4.3.6.6 Setting Next and Previous Rows

You can use the `datatable` API to set the next and previous row programmatically during playback of a script. The following examples show how to set the next and

previous rows using the `datatable` API `setNextRow()` and `getCurrentRow()` methods.

4.3.6.6.1 Setting the Next Row

The following example script code sets the next row of the current sheet in the Data Table:

```
datatable.setNextRow();
```

4.3.6.6.2 Setting the Previous Row

The following example script code sets the previous row of the current sheet in the Data Table:

```
datatable.setPreviousRow();
```

4.3.6.7 Importing and Exporting Documents and Sheets

You can use the `datatable` API to get the import and export spreadsheet documents and worksheets programmatically during playback of a script. The following examples show how to import and export spreadsheet files and worksheets using the `datatable` API `importExcel()`, `importSheet()`, `exportToExcel()` and `exportSheet()` methods.

4.3.6.7.1 Importing an Excel Spreadsheet Document

The following example script code imports the `myXls.xls` Excel spreadsheet document into the Data Table:

```
datatable.importExcel("c:\\myXls.xls");
```

4.3.6.7.2 Importing Worksheets

The following example script code imports the single worksheet named "SourceSheet" from the `myXls.xls` Excel spreadsheet document and adds it to the Data Table with the name "DestinationSheet":

```
datatable.importSheet("c:\\myXls.xls", "SourceSheet", "DestinationSheet");
```

The following example script code imports all worksheet from the `myXls.xls` Excel spreadsheet document and adds them to the Data Table overwriting any sheets with the same names:

```
datatable.importSheets("c:\\myXls.xls", true);
```

The following example script code imports the specified list of worksheets from the `myXls.xls` Excel spreadsheet document and adds them to the Data Table overwriting any sheets with the same names:

```
import java.util.List;
import java.util.ArrayList;
//[...]
List<String> sheetList = new ArrayList<String>();
sheetList.add("Sheet1");
sheetList.add("Sheet2");
datatable.importSheets("c:\\myXls.xls", sheetList, true);
```

The following example script code imports the specified list of worksheets from the `myXls.xls` Excel spreadsheet document and adds them to the Data Table overwriting any sheets with the same names using the first row:

```
import java.util.List;
import java.util.ArrayList;
//[...]
List<String> sheetList = new ArrayList<String>();
sheetList.add("Sheet1");
sheetList.add("Sheet2");
datatable.importSheets("c:\\myXls.xls", sheetList, true, true);
```

4.3.6.7.3 Exporting an Excel Spreadsheet Document

The following example script code exports the `myXls.xls` Excel spreadsheet document from the Data Table to a file:

```
datatable.exportToExcel("c:\\myXls.xls");
```

4.3.6.7.4 Exporting Worksheets

The following example script code exports the single worksheet named "SourceSheet" from the Data Table to the `myXls.xls` Excel spreadsheet document with the name "DestinationSheet":

```
datatable.exportSheet("c:\\myXls.xls", "SourceSheet", "DestinationSheet");
```

The following example script code imports all worksheet from the `myXls.xls` Excel spreadsheet document and adds them to the Data Table overwriting any sheets with the same names:

```
datatable.importSheets("c:\\myXls.xls", true);
```

The following example script code exports the specified list of worksheets from the Data Table to the `myXls.xls` Excel spreadsheet document:

```
import java.util.List;
import java.util.ArrayList;
//[...]
List<String> sheetList = new ArrayList<String>();
sheetList.add("Sheet1");
sheetList.add("Sheet2");
datatable.exportSheets("c:\\myXls.xls", sheetList);
```

4.3.6.8 Using Data Tables with Parent and Child Scripts

You can use the `datatable` API to change data in the Data Table of a parent script that runs a child script programmatically during playback of parent and child scripts. The following examples show how to change data in parent scripts from child scripts using the `datatable` API `getParentDatatable()` and `getGlobalDatatable()` methods.

4.3.6.8.1 Accessing the Parent Data Table from a Child Script

The following example script code shows how a child script can access and change data in the Data Table of the parent script. Both the parent and child scripts must have the Data Table service enabled. It is important that it always should be verified that the return value is not `null`. In the child script, make sure the Data Table service is enabled and create a parent Data Table instance:

```
import oracle.oats.scripting.modules.datatable.api.DataTableService;
//[...]
DataTableService parentDatatable = datatable.getParentDatatable();
  if(parentDatatable != null)
   {
      info("set parent datatable value");
```

```
    parentDatatable.setValue(0,"A", 30);
    parentDatatable.save();
}
```

In the parent script, run the child script using the `getScript("alias").run()` method:

```
getScript("childScript").run(1);
```

The Data Table results appear in the parent script Results and the Results Data Table of the Details view.

4.3.6.8.2 Accessing the Top-Most Data Table in Chain of Parent Scripts

The following example script code shows how a child script can access and change data in the top-most Data Table in a chain of parent scripts. All scripts in the chain must have the Data Table service enabled. In the child script, make sure the Data Table service is enabled and create a global Data Table instance:

```
import oracle.oats.scripting.modules.datatable.api.DataTableService;
//[...]
DataTableService globalDatatable = datatable.getGlobalDatatable();
info("set global datatable value");
globalDatatable.setValue(0, "B", 20);
globalDatatable.save();
```

If the script containing this code is run as a stand alone script the return value is a datatable of the script itself.

5
Using the Web Functional Test Module

This chapter provides instructions on configuring and using the OpenScript Web Functional Test Module, which tests Web-based applications by accessing objects through the Document Object Model (DOM) of the Web browser.

5.1 About the Web Functional Test Module

The OpenScript Web Functional Test Module is an application module that supports functional testing of Web-based applications that uses the Web Document Object Model (DOM). OpenScript provides a flexible and easy-to-use scripting interface for both Technical Testers and Non-Technical Testers. The OpenScript Functional Test Module enables script creation from both the code view and GUI view scripting interfaces.

The Functional Test Module extends the OpenScript platform with Document Object Model (DOM) recording and playback capabilities. The DOM recorder automatically captures Web page objects, actions, and navigations and records them as tree view nodes (with the underlying code in the Code View) in the script. The Functional Test Module also provides additional GUI script modification options. Web Functional Test Scripts differ from HTTP Scripts. Even though both are used to test web applications, HTTP scripts automate the underlying HTTP network protocol, whereas Web Functional Test scripts automate the browser UI.

The Web Functional Test Module is an extension module to the Oracle OpenScript platform that extends the platform with Web Functional Test recording and playback capabilities. The Web Functional Test Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager.

The Web Functional Test recorder displays commands in the Tree View in easy-to-understand commands. By default, script commands are grouped into Steps Groups by the Web page on which they were performed. Each Step Group contains one or more script commands corresponding to recorded actions that were performed on the page. The default name for the Step Group is the Web page Title (as specified in the "Title" tag).

OpenScript shows the results of Web Functional Test script playback in the Results view. The Results view shows results for each script command (including duration and summary for failures). The Results Report compiles the same information into an HTML Results Report. Results can be exported from the OpenScript GUI in standard format (CSV / HTML). Results are also generated for unattended playback through the command line.

The Web Functional Test Module API includes a "Web" class that provides additional programming functionality.

5.1.1 Key Features of the Web Functional Test Module

The Web Functional Test Module provides the following functionality:

- Records Document Object Model objects and actions for playback automation. The objects and actions can be generated by a Web browser (i.e. IE or Firefox).
- Plays back functional testing scripts to validate proper functionality. Playback runs interactively in the OpenScript user interface and is also supported in Oracle Test Manager.
- Provides full script code view integration to support script generation for the Web Functional Test Module. The Web Functional Test Module includes an additional API to support Web Functional Test protocol code scripting.
- Allows users to parameterize user inputs to Functional Test scripts and drive those inputs from an external data file (Databank).
- Allows users to insert test cases for validation of objects and actions.
- Provides additional options/settings that are specific to Functional Test scripts within the Functional Test categories in the preferences interface.
- Reports playback results for Functional Test scripts in the Results and Console views.

5.2 Recording Web Functional Tests

The Web Functional Test Module records Document Object Model objects and actions in a Web browser for playback automation. The Recorder creates functional and regression test scripts for automating GUI applications in a browser.

OpenScript records standard Web DOM objects (links, images, forms, form elements, etc.) and events (click, mousedown, focus, etc.) for playback. Web DOM objects are identified by one or more attribute as configured through the Web Functional Test Record Preferences under Object Identification. Object Identification attributes can later be modified by users through the Preferences global settings for new scripts or for already recorded commands in the tree view or code view. All Web DOM objects, events and attributes should be supported. Recording can be configured through Internet Explorer or Firefox. You can set the browser type in the Preferences.

The Web Functional Test Module provides a record toolbar button that lets you initiate the Web DOM recorder and capture Web page actions to the script view. The record toolbar includes start and stop recording toolbar buttons. OpenScript recorders also open a floating toolbar that can be used while recording without having to switch between the browser and OpenScript.

5.2.1 Setting Web Functional Test Record Preferences

Before recording Web Functional Test scripts, set Web record preferences.

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Record category.
4. Select **Web Functional Test**.
5. Click the tabs and set the preferences. See Section 2.5.8, "Web Functional Test Preferences" for descriptions of the Record Preferences settings.
6. Click **OK** when finished.

5.2.2 Adding/Editing Object Identifiers

The Web Functional Module uses object identification to specify attributes used to identify Web objects. The Web Functional Test Module includes predefined path attributes for common Web objects. Object paths are specified in XPath format. For example, for web objects, the object identification path appears as follows in Java code commands:

```
/web:window[@index='0']
   /web:document[@index='0']
   /web:form[@index='0']
   /web:input_text[@id='ticker' or @name='ticker' or @index='0']
```

You can set the default object attributes in the Web Functional Test Module Record Preferences. You can also edit object attributes in recorded scripts in the tree view or the code view.

In addition to the predefined object identification, you can add an Object Library to the script to record paths into a library file. Object Library files may be shared and reused across other scripts. The Object Library files provide a more convenient "short name" for objects to provide for more convenient programming. See Section 5.5, "Editing Object Libraries" and Section 5.4.4, "Adding Object Libraries to a Script" for additional information.

The Web Functional Test Module includes object identifiers that specify how the recorder identifies Browser objects. You can add object identifiers or edit the existing object identifiers in the Record preferences.

To add or edit an object identifier:

1. Select the **OpenScript Preferences** from the **View** menu.
2. Expand the Record node and select Web Functional Test.
3. Click the **Object Identification** tab.
4. Click **Add** or select an existing object identifier and click **Edit**.

   The Object Element dialog box lets you define an object identifier for a Web Functional Test recorder. The object identifier specifies how the OpenScript Web Functional Test recorder module identifies web objects in the Document Object Model on a Web page.

   Active Profile: Specifies which object identification profile to use as the active profile when recording scripts. Profiles define a specific set of object identifiers to use when recording Web Functional tests. Use the **Add Profile** option to create a new custom profile. Once you have created a profile, select the profile name in the **Name** column and use **Add Object** to define custom objects and attributes in the custom profile.

   Name: Shows the name(s) of the defined Web object identifiers.

   Attributes: Shows the pattern(s) specified for the defined Web object identifiers.

   Add Profile: Opens a dialog box for specifying a new Web object identifier profile.

   Add Object: Opens a dialog box for specifying a new Web object identifier.

   Edit: Opens a dialog box for editing the selected Web object identifier or profile.

   Delete: Deletes the selected Web object identifier or profile. The default profile cannot be deleted.

Export: Opens a dialog box for exporting the currently selected Web object identifier profile to an XML file. Select the profile name in the **Name** column to activate the export option.

Import: Opens a dialog box for importing the currently selected Web object identifier profile to an XML file. Select a profile name in the **Name** column to activate the import option.

Revert: Reverts the default Web object identification profile to the default profile. Any changes to the default profile are removed. Select the default profile name in the **Name** column to activate the revert option.

For each object element, you specify a name (typically a Web object attribute), an operator, a value and a value type. As you add object elements, OpenScript builds the object identifier using logical OR between each object identifier element. Click **Edit** to change between logical OR and AND.

5. If adding a new object identifier, click **Add Object** and specify the path segments for the object identifier.

   The Path Segment dialog box lets you define a segment of an object identifier for a Web Functional Test recorder. The object identifier specifies how the OpenScript Web Functional Test recorder module identifies web objects in the Document Object Model on a Web page.

   Attribute: When selected, specify the name, operator, value, and value type for the object identifier.

   - **Name**: Specify the name of the Web object attribute to use to identify the object.
   - **Operator**: Specify the logical operator to use to identify the object value.
   - **Value**: Specify the value of the Web object attribute to use to identify the object.
   - **Value Type**: Specify the value type to use to identify the object. The value type can be a string, a number, or by variable reference. Set the **Value type** to match the specified value.
     - **String**: The value in the **Value** field will be matched as a text string to identify the Web object.
     - **Number**: The value in the **Value** field will be matched as a numeric value to identify the Web object.
     - **Reference**: The value in the **Value** field will be matched as a variable name to identify the Web object.

   Group: When selected the Web object identifier can be specified as a string representing a logical group of Names, Operators, and Values.

6. If editing an existing segment, click **Edit** and specify the path attributes for the object identifier.

   The Path Attribute dialog box lets you edit an element of an object identifier.

   Name: Specify the name of the Web object attribute to use to identify the object.

   Operator: Specify the logical operator to use to identify the object value.

   Value: Specify the value of the Web object attribute to use to identify the object.

Value Type: Specify the value type to use to identify the object. The value type can be a string, a number, or by variable reference. Set the **Value type** to match the specified value.

- **String**: The value in the **Value** field will be matched as a text string to identify the Web object.
- **Number**: The value in the **Value** field will be matched as a numeric value to identify the Web object.
- **Reference**: The value in the **Value** field will be matched as a variable name to identify the Web object.

And/Or: specifies logical OR or AND between object elements. This option does not appear if the object element being edited is the last object element in the object identifier group.

7. If editing an existing path, click **Edit** and specify the path attributes for the object identifier.

   The Path Attribute Group dialog box lets you edit an object identifier group as a string value.

   Group: A text string of the object identifier group. The object identifier group syntax follows the `@name operator value/type` format. Values by reference are enclosed in double curly braces `{{}}`. String values are enclosed in single quotation marks. Numeric values are not enclosed. Logical OR or AND are used between object elements. Parenthesis are used for logical grouping of multiple object elements. Object identifiers can use the wildcard characters `*` (asterisk) for multiple characters and `?` (question mark) for single characters. For example, `@text="Login*"`. The following are examples of object identifier group syntax:

   ```
   @index={{index}} or @title='title' or @number=5

   @text={{text}} or @href={{href}} or @index={{index}}

   (@id={{id}} or @name={{name}} or @index={{index}}) and multiple mod
   {{multiple}}
   ```

8. Click **OK** when finished adding or editing segments or attributes. The object identifier is added to the record preferences.
9. If you have the browser open when adding or editing object identifiers, close and restart the browser.

5.2.2.1 Available Attributes for Web DOM Elements

The following table lists the attributes available for Web Document Object Model objects.

Element	Attributes
Common attributes for all web DOM elements	tag, id, index, title, style, class, html, _ adftrueval, rn, un, ot
web:window	index, title
web:document	index, url
web:a	disabled, text, href
web:button	type, name, value, disabled, text
web:input_button	type, name, value, disabled, text

Element	Attributes
web:input_file	type, name, value, index, disabled, size
web:input_hidden	type, name, value, index, disabled
web:input_image	type, name, disabled, alt, align, border, height, hspace, src, lowsrc
web:input_submit	type, name, value, disabled, text
web:input_text	type, name, value, defaultValue, disabled, maxlength, readOnly, size
web:input_check	type, name, value, disabled, checked, defaultChecked, readOnly, size
web:input_radio	type, name, value, disabled, checked, defaultChecked, readOnly, size
web:img	alt, height, longdesc, name, src, width
web:option	value, text, optionIndex(index in the select), defaultSelected, selected
web:select	name, disabled, value, selectedIndex
web:textarea	name, disabled, value, defaultValue, readonly, cols, rows

5.2.3 Recording Web Functional Test Scripts

To record Web Functional Test scripts:

1. Start OpenScript.
2. Set the Web Functional Test Recording preferences.
3. Select **New** from the **File** menu.
4. Expand the Functional Testing group.
5. Select **Web**.
6. Click **Next**.
7. Select the Repository and Workspace.
8. Enter a script name.
9. Click **Finish**. A new Script tree is created in the Script View.
10. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.

> **Note:** Make sure the browser zoom setting is set to 100%. Record and playback of functional test scripts at other zoom settings is not supported and may not work correctly.

11. Load the web page where you want to start recording into the browser.
12. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
13. When finished navigating pages, close the browser.

14. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
15. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

5.3 Playing Back Scripts

OpenScript plays back recorded Web actions/commands which consist of an event plus an object identified by its attributes (for example: `click link(text="Home")`). The actions used for playback will either be those that are recorded or are specified manually in the Java Code view. Playback can be configured through Internet Explorer or Firefox. Unattended playback is supported through Oracle Test Manager or third-party tools using OpenScript's command line interface. Web Functional Test scripts do not play in Oracle Load Testing.

The Web Functional Test Module provides playback and iterate toolbar buttons that allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for Web Functional scripts can be viewed in the Results and Console views.

5.3.1 Setting Web Functional Test Playback Preferences

To set Web Functional Test Playback preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Playback category.
4. Select **Web Functional Test**.
5. Expand the groups and set the preferences. See Section 2.4.7, "Web Functional Test Preferences" for descriptions of the Playback Preferences settings.
6. Click **OK** when finished.

5.3.2 Playing Back Web Functional Scripts

To play back Web Functional scripts:

1. Start OpenScript.
2. Open the Web Functional script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

5.3.3 Playing Back Web Functional Scripts with Iterations

To play back Web Functional scripts with iterations:

1. Start OpenScript.
2. Open the Web Functional script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.
5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.
7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

5.4 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

5.4.1 Path Editor Toolbar

Several dialog boxes used to add tests use a common toolbar for editing object paths. See Section 5.2.2, "Adding/Editing Object Identifiers" for additional information about adding or editing object identification paths.The toolbar options are as follows:

Path/Window: Specifies the path to the object to test. The options available on the object path toolbar are as follows:

- **Toggle between tree and text**: Toggles the path between a tree view and a text view. In the tree view, you can add, edit or delete segments of the object path using the toolbar or use the Up and Down arrows to reorder segments of the object path. In the text view, you can edit the object path directly in the text box.
- **Add**: Opens a dialog box for adding a new path segment to the object path. This toolbar button is only available when the path is in tree view.
- **Edit**: Opens a dialog box for editing the attributes of the currently selected path segment in the object path. This toolbar button is only available when the path is in tree view.
- **Delete**: Deletes the currently selected path segment from the object path. This toolbar button is only available when the path is in tree view.
- **Up**: Moves the currently selected path segment up one level in the tree. This toolbar button is only available when the path is in tree view and the selected path segment is not the highest level segment.
- **Down**: Moves the currently selected path segment up one level in the tree. This toolbar button is only available when the path is in tree view and the selected path segment is not the lowest level segment.
- **Capture object in browser**: Opens a capture object dialog box and starts the capture mode. Navigate to the object to capture and use options in the capture dialog box to capture the object path.

- **View object path from Object Library**: Opens a dialog box for viewing both the tree view and text view of the currently selected object path.
- **Select from Object Library**: Opens a dialog box for selecting an object from a saved library.
- **Save to Object Library**: Opens a dialog box for saving the current object to a library.

5.4.2 Adding Browser Navigation to a Script

To add a Browser Navigation to a Script:

1. Record a Web Functional Test script.
2. Select the script node where you want to add the navigation.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Web Actions node.
5. Expand the Browser node and select Navigate.
6. Click **OK**.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar.
8. Enter the URL.
9. Click **OK**. The navigate node is added to the script tree.

   In the Java Code view, the `web.window(`*`objectId`*`).navigate` method will be added to the script code:

```
web.window(1,
"/web:window[@index='0']").navigate("http://testserver2/fmstocks");
```

   The Browser web actions includes additional options for browser navigation such as Back, Forward, Close, Refresh, Wait for Page, etc. Additional browser actions have corresponding Java code methods:

```
web.window(1,
"/web:window[@index='0']").waitForPage("http://testserver2/fmstocks/",
"Stocks", true, null)
web.window("/web:window[@index='0']").close();
web.window("/web:window[@index='0']").back();
web.window("/web:window[@index='0']").forward();
web.window("/web:window[@index='0']").refresh();
web.window("/web:window[@index='0']").capturePage();
web.window("/web:window[@index='0']").solve("MyVariable", "(.+?)", true, 1);
web.window("/web:window[@index='0']").storeResponseTime("MyRespTime");
web.window("/web:window[@index='0']").waitFor(10);
web.window("/web:window[@index='0']").storeAttribute("MyAttribVar",
"MyAttribute");
```

5.4.3 Adding Web Actions on Browser Objects

The Web Functional Test Module includes actions for Browser objects that can be added to a script.

To add Web actions on Browser objects to a script:

1. Record a Web Functional Test script.

2. Select the script node where you want to add the action.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Web Actions node.
5. Expand an action node and select the action.
6. Click **OK**.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar.
8. Click **OK**. The action node is added to the script tree.

   In the Java Code view, a `web.object(objectId).action()` method will be added to the script code:

```
web.button("/web:window[@index='0']
   /web:document[@index='0']
   /web:form[@index='0'
     or @id='loginform'
     or @name='loginform']
   /web:input_submit[@name='LoginButton'
     or @value='Login'
     or @index='0']").click()
```

   The Web Actions node includes actions for objects such as buttons, check boxes, dialogs, images, links, tables, etc. Other object actions have corresponding Java code methods:

```
web.link("/web:window[@index='0']
   /web:document[@index='0']
   /web:a[@text='Open a new account.'
     or @href='http://testserver2/fmstocks/_NewAccount.asp'
     or @index='0']").click()
```

   or

```
web.select("/web:window[@index='0']
   /web:document[@index='0']
   /web:form[@index='0']
   /web:select[(@id='namespace'
     or @name='namespace'
     or @index='0')
     and multiple mod 'False']").selectOptionByText("[select product]");
```

   or

```
web.clearSessionCookies();
web.clearAllPersistentCookies();
web.clearPersistentCookies("domain");
web.clearCache("domain");
```

5.4.4 Adding Object Libraries to a Script

To add Object Libraries to a script:

1. Open or create a Web Functional Test script project.
2. Select the Assets tab in the Script view and select Object Libraries.
3. Click **Add**.

4. Specify the library file and alias:
   - If you have already created an Object library file, select the file in a repository.

     Note: Any scripts you plan to run, along with any associated assets, in the Oracle Load Testing application must be stored in a repository/workspace that can be accessed by the Oracle Load Testing Controller. If you create new repositories in OpenScript, you should also add the new repositories in Oracle Load Testing.

   - If you have not created a library file, enter a file name, and click **OK**. A new file will be created. When you record a Web Functional Test script, the browser actions will be automatically added to the object library file.
5. Click **OK**. The Object Library file is added as a script test asset under the Object Libraries tree. Script test assets are referenced in the assets.xml file located in the script directory of the repository.
6. Click **OK**.
7. Record a Web Functional Test script to use the object library.

   The object identification paths for browser objects will be automatically added to the object library file during recording. You can open and edit the object library file using the Object Library Editor. Select **Open Object Library** from the **File** menu, select the Object Library `.properties` file and click **Open**. The Object Library editor view opens in the Workbench with the objects and details included in the selected Object Library. You can add objects to and delete objects from the Object list. You edit the object attributes in the object string or in the tree hierarchy of the Details section. The tree hierarchy lets you move attributes up or down in the priority order.
8. Save the object library and script when finished.

5.4.5 Adding a Server Response Test

You can use Server Response Tests to report an error and/or abort the script if a Web page does not return back to the client within a specified time range.

To add a Server Response Test to a script:

1. Record a Web Functional Test script.
2. Expand the **Run** node.
3. Select the script node where you want to add the Server Response test.
4. Select the **Script** menu and then select **Other** from the **Add** sub menu.
5. Expand the Web Tests group.
6. Select **Server Response Test** and click **OK**.
7. Enter the object identification path for the window. You can use the **Capture** or **Select** menu options to capture or select an object path. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar.
8. Enter a name for the test.
9. Enter the minimum and maximum time values.
10. Set the **Verify only, never fail** option.

11. Click **OK** to add the Server Response node to the script tree.

    In the Java Code view, the Server Response Test consists of the code executed in the `web.window(objectId).verifyResponseTime()` or `web.window(objectId).assertResponseTime()` method:

    ```
    web.window("/web:window[@index='0']").verifyResponseTime("MyserverResp", 10.0,
    20.0);
    ```

    or

    ```
    web.window("/web:window[@index='0']").assertResponseTime("MyserverResp2", 10.0,
    50.0);
    ```

    In the above code examples, `verify` means "do not stop on failure" and `assert` means "stop on failure".

5.4.6 Adding Text Matching Tests to a Script

To add a Text Matching Test to a Script:

1. Open or create a Web Functional Test script project.
2. Select the script node where you want to add the Text Matching Test.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Web Tests node.
5. Select **Text Matching Test**.
6. Click **OK**.
7. Select the **Look in** location:

   All Browsers: When selected, the Text Matching Test looks for the text to match in all open browsers.

   Specified Browser: When selected, the Text Matching Test looks for the text to match by the browser ID specified in the **Path**.

   Specified Document: When selected, the Text Matching Test looks for the text to match in the document specified by the object ID in the **Path**.
8. If you selected **Specified Browser** or **Specified Document**, enter the object identification path for the browser, document, or frame in the **Path**. You can use the **Capture** or **Select** buttons to capture or select an object path. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar
9. Enter a name for the test.
10. Enter the text string or Regular Expression to match or click the Substitute Variable icon to select a databank or script variable to find in the source.

    Note: The pound (`#`) character and double brace (`{{` and `}}`) character sequences need to be escaped with a preceding pound (`#`) character if used in the Text Matching Test as a literal string (not a string specifying an OpenScript databank or script variable). For example, the pound character should be doubled (`##`) and double braces should be preceded by a pound character (`#{{` and `#}}`).

11. Select the source location that will be searched for the matching text:

HTML Display Contents: Search the browser rendered text of the page.

Raw HTML: Search HTML source of the page.

12. Select the **Pass when** setting.

    Selected text is present: The test case passes if the **Text to Match** string is found in the selected source.

    Selected text is absent: The test case passes if the **Text to Match** string is not found in the selected source.

13. Select the **Match** type.

    Exact: Matches the **Text to Match** string exactly.

    Regular Expression: Matches using the Regular Expression specified in **Text to Match**.

    Wildcard: Matches using the wildcard characters specified in **Text to Match**.

14. Set the **Verify only, never fail** option.

15. Click **OK**. The Text Matching Test node is added to the script tree.

    In the Java Code view, the `web.assertText` or `web.document(`*`specifiedDoc`*`).assertText` method will be added to the script code if the **Verify only, never fail** option is not selected:

```
web.assertText("MyTextMatchTest", "Home", TextPresence.PassIfPresent,
MatchOption.Exact)

web.document("Main").assertText("MyTextMatchTest2", "Home",
TextPresence.PassIfPresent, MatchOption.RegEx);
```

    In the Java Code view, the `web.verifyText` or `web.document(`*`specifiedDoc`*`).verifyText` method will be added to the script code if the **Verify only, never fail** option is selected:

```
web.verifyText("MyTextMatchTest", "Home", TextPresence.PassIfPresent,
MatchOption.Exact)

web.document("Main").verifyText("MyTextMatchTest2", "Home",
TextPresence.PassIfPresent, MatchOption.RegEx);
```

5.4.7 Adding Object Tests

The Web Functional Test Module includes an object test case for Browser objects that can be added to a script.

To add an object test to a script:

1. Record a Web Functional Test script.
2. Select the script node where you want to add the object test.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Web Tests node and select Object Test.
5. Click **OK**.
6. If necessary, select the Object Type.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar.

8. Enter a test name.
9. Select or clear the **Verify only, never fail** option.
10. Select an Attribute/Value pair and specify the test details.

    Attributes: Shows the attributes of the selected object. Select an attribute and specify the test details.

    - **Attribute**: Shows the name of the attribute. When selected, the attribute is included in the object test. When cleared, the attribute is not included in the object test.
    - **Value**: Shows either the recorded value or the test criteria for the attribute depending upon which **Display** option is selected.
    - **Enable All**: Enables all properties for inclusion in the test criteria.
    - **Disable All**: Disables all properties for exclusion from the test criteria.
    - **Add Row**: Adds a new row to the table.
    - **Display**: Selects which values appear in the properties list.
      - **Tests**: When selected, the **Value** list shows the test criteria defined for each attribute.
      - **Recorded Values**: When selected, the **Value** list shows the recorded value for each attribute.

    Test Details: Specifies the test criteria for the selected attribute.

    - **Recorded Value**: Shows the recorded value of the attribute.
    - **Enable**: When selected, the attribute is included in the object test. When cleared, the attribute is not included in the object test.
    - **Value Type**: Specifies the data type to use as the test criteria for the attribute. The available options in the **Operator** list depend upon the selected value type to test: String, Boolean, Date, or Numeric.
    - **Operator**: Specifies the operator to use for the playback test criteria. The list of operators changes depending upon the selected **Value Type**.

      The following options are available for String values:

      - **Exact**: The test passes if the text or attribute value matches exactly the current value of the text string during playback of the script.
      - **Wildcard**: The test passes if the text or attribute value matches the current value of the Like operator pattern during playback of the script.
      - **Regular Expression**: The test passes if the text or attribute value matches the current value of the Regular Expression pattern during playback of the script.

      The following options are available for Numeric and Date values:

      - [Relational Operators]: Lists the relational operators. Select the type of comparison to use during playback of the script. If you select the Range operator, the Object Test adds a field for specifying the range. Specify the from-to range.

      The following option is available for Boolean values:

      - **Equals**: The test passes if the text or attribute value matches the True or False value specified in the **Value** field during playback of the script.

- **Value**: Specifies the value to use for the test criteria for the attribute.
- [Substitute Variable]: Opens a window for selecting a databank variable to substitute as the value to use for the test criteria for the attribute.

11. Specify the test details for each Attribute/Value pair as required for the test.
12. Click **OK**. The object test node is added to the script tree.

    In the Java Code view, a `web.element(`*`objectId`*`).verifyAttributes` method will be added to the script code:

```
web.element(15, "/web:window[@index='0']
   /web:document[@index='0']
   /web:form[@index='0'
     or @id='loginform'
     or @name='loginform']
   /web:input_password[@name='password' or @index='0']").verifyAttributes(
     "MyObjectTest", web.attributes(
     web.attribute("index", "0", TestOperator.StringExact),
     web.attribute("tag", "INPUT", TestOperator.StringExact),
     web.attribute("name", "password", TestOperator.StringExact),
     web.attribute("value", "ta", TestOperator.StringExact),
     web.attribute("type", "password", TestOperator.StringExact),
     web.attribute("checked", "False", TestOperator.StringExact),
     web.attribute("disabled", "False", TestOperator.StringExact)));
```

5.4.8 Adding Table Tests

The Web Functional Test Module includes a table test case for HTML tables that can be added to a script.

To add a table test to a script:

1. Record a Web Functional Test script.
2. Select the script node where you want to add the table test.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Web Tests node select Table Test.
5. Click **OK**.
6. If necessary, select the Object Type.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar.
8. Enter a test name.
9. Select or clear the **Verify only, never fail** option.
10. Select a row and column cell and specify the test details.

    Test Details: Specifies the test to perform for each value. Select a value in the row and column list to set the details for that specific value.

    - **Row**: Shows the row number of the selected table value.
    - **Column**: Shows the column number of the selected table value.
    - **Enable**: When selected, the table value is included in the comparison test.
    - **Recorded value**: Shows the selected attribute's recorded value.

- **Value Type**: Specifies the data type of the selected value: String, Numeric, Date, or Boolean.
- **Operator**: Specifies how to compare the value. The **Operator** options change depending upon the selected value type. For relational operators, if you select the Range operator, a field is added for specifying the numeric or date range. Specify the from-to range.
- **Value**: Specifies the value to test.
- [Substitute Variable] - opens a dialog box for selecting the variable to use as the value to test.

11. Specify the test details for each table cell as required for the test.
12. Click **OK**. The table test node is added to the script tree.

    In the Java Code view, a `web.table(`*`objectId`*`).assertCells` or a `web.table(`*`objectId`*`).verifyCells` method will be added to the script code depending upon the **Verify only, never fail** setting:

```
web.table(24, "/web:window[@index='0']
   /web:document[@index='0']
   /web:table[@index='6']").verifyCells("MyTableTest", web.cells(
     web.cell(1, 1, "Ticker ", TestOperator.StringExact),
     web.cell(1, 2, "Company ", TestOperator.StringExact),
     web.cell(2, 1, "ORCL ", TestOperator.StringExact),
     web.cell(2, 2, "Oracle Corporation ", TestOperator.StringExact)))
```

    or

```
web.table(24, "/web:window[@index='0']
   /web:document[@index='0']
   /web:table[@index='6']").assertCells("MyTableTest", web.cells(
     web.cell(1, 1, "Ticker ", TestOperator.StringExact),
     web.cell(1, 2, "Company ", TestOperator.StringExact),
     web.cell(2, 1, "ORCL ", TestOperator.StringExact),
     web.cell(2, 2, "Oracle Corporation ", TestOperator.StringExact)))
```

5.4.8.1 Testing Images in Tables

You can use the Table test to create tests that check for the existence of one or more images in a Web page table.

To add an image existence test to a script using a table test:

1. Record a Web Functional Test script that has a Web page table with one or more images in the table.
2. When the Web page table with the image is displayed, select the **Script** menu and then select **Other** from the **Add** sub menu.
3. Expand the Web Tests node and select Table Test.
4. Click **OK**.
5. Click the **Capture object in browser** toolbar button. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar.
6. Select the image in the Web page table and press the F10 key to capture the object path.
7. Click **OK**. The Table Test dialog box opens with a **web.table** Object Type.
8. Enter a test name.

9. Select or clear the **Verify only, never fail** option.
10. Select an table cell and specify the test details. The Table Test uses the text of the SRC=http://*path/imageName* for the image existence test.
11. Specify the test details for each table cell as required for the test. For images, the value specifies the path and name of the image file. Other table cell values for the table may also be included depending upon the Web page source.
12. Click **OK**. The table test node is added to the script tree.

    In the Java Code view, a `web.table(`*`objectId`*`).assertCells` or a `web.table(`*`objectId`*`).verifyCells` method will be added to the script code depending upon the **Verify only, never fail** setting:

```
web.table(7,"/web:window[@index='0' or @title='MyImageTable']
 /web:document[@index='0']
 /web:table[@firstTableCell='http://my.com/img1.gif' or @index='28']")
 .verifyCells("ImageTableTest",
   web.cells(web.cell(1, 1,"http://my.com/img2.gif", TestOperator.StringExact),
    web.cell(2, 1, "http://my.com/img3.gif", TestOperator.StringExact),
    web.cell(3, 1, "http://my.com/img4.gif", TestOperator.StringExact),
    web.cell(4 ,1, "http://my.com/img5.gif", TestOperator.StringExact)));
```

    or

```
web.table(7,"/web:window[@index='0' or @title='MyImageTable']
 /web:document[@index='0']
 /web:table[@firstTableCell='http://my.com/img1.gif' or @index='28']")
 .assertCells("ImageTableTest",
   web.cells(web.cell(1, 1,"http://my.com/img2.gif", TestOperator.StringExact),
    web.cell(2, 1, "http://my.com/img3.gif", TestOperator.StringExact),
    web.cell(3, 1, "http://my.com/img4.gif", TestOperator.StringExact),
    web.cell(4 ,1, "http://my.com/img5.gif", TestOperator.StringExact)));
```

13. Click the Stop toolbar button when finished recording.

5.4.9 Adding a Page Title Test

You can use Title Tests to report a page title error and/or abort the script if a Web page does not return back to the client a page with the expected title. The page title test compares the recorded page title to the title received on playback of the script.

To add a Title Test to a script:

1. Record a Web Functional Test script.
2. Expand the **Run** node.
3. Select the script node where you want to add the Title test. Selecting the WaitForPage node will add the object identification path ID to the test automatically.
4. Select the **Script** menu and then select **Other** from the **Add** sub menu.
5. Expand the Web Tests group and select Title Test.
6. Click **OK**.
7. Select the **Add to** option:
   - **Specific Page**: When selected, the page title test will be added to the specified or currently selected page in the script.

- **All Pages in Script**: When selected, a page title test will be added to each page in the script specifying the recorded title as the title to compare on playback.

8. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar.
9. Enter a name for the test.
10. Enter the page title. If you selected the WaitForPage node the page title is added to the test automatically.
11. Set the **Verify Only, never fail** option.
12. Click **OK** to add the Page Title Test node to the script tree.

    In the Java Code view, the Page Title Test consists of the code executed in the `web.window(objectId).verifyTitle()` or `web.window(objectId).assertTitle()` method:

```
web.window("/web:window[@index='0' or @title='Stocks']")
  .verifyTitle("MyTitleTest", "Home");
```

    or

```
web.window("/web:window[@index='0' or @title='Stocks']")
  .assertTitle("MyTitleTest2", "Home");
```

    In the above code examples, `verify` means "do not stop on failure" and `assert` means "stop on failure".

5.4.10 Adding an HTML Test

You can use HTML Tests to report an HTML error and/or abort the script if a Web page does not return back to the client a page with the expected HTML. The HTML test compares the recorded HTML to the HTML received on playback of the script.

To add a HTML test to a script:

1. Record a Web Functional Test script.
2. Expand the **Run** node.
3. Select the script node where you want to add the HTML test. Selecting the WaitForPage node will add the object identification path ID to the test automatically.
4. Select the **Script** menu and then select **Other** from the **Add** sub menu.
5. Expand the Web Tests group and select HTML Test.
6. Click **OK**.
7. Select the **Add to** option:
   - **Specific Document**: When selected, the HTML test will be added to the specified or currently selected HTML document in the script.
   - **All Documents in Script**: When selected, am HTML test will be added to each HTML document in the script.
8. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar.
9. Enter a name for the test.

10. Set the **Verify Only, never fail** option.
11. Click **OK** to add the HTML Test node to the script tree.

    In the Java Code view, the HTML Test consists of the code executed in the `web.document(`*`objectId`*`).verifyHTML()` or `web.document(`*`objectId`*`).assertHTML()` method:

```
web.document("/web:window[@index='0' or @title='Stocks']
  /web:document[@index='0']")
   .verifyHTML("MyHTMLTest","e6306c5cf8f6697d9ba7bb2110888a50");
```

    or

```
web.document("/web:window[@index='0' or @title='Stocks']
  /web:document[@index='0']")
   .assertHTML("MyHTMLTest2","e6306c5cf8f6697d9ba7bb2110888a50");
```

    In the above code examples, `verify` means "do not stop on failure" and `assert` means "stop on failure".

If an HTML test fails on playback, you can select the HTML test result in the **Results** view and view the differences in the **Comparison** tab of the **Details** view.

5.4.11 Adding an XML Test

You can use XML Tests to report an XML error and/or abort the script if an XML file does not return back to the client a file with the expected XML. The XML test compares the recorded XML to the XML received on playback of the script.

To add an XML test to a script:

1. Record a Web Functional Test script.
2. Navigate to a URL for an XML file.
3. Expand the **Run** node.
4. Select the script node where you want to add the XML test. Selecting the WaitForPage node will show the file in the Details view with tabs for Screenshot XML, XML tree, and DOM Tree. The XML tab shows the content of the XML file. The XML Tree tab shows the content of the XML file in tree format with the attributes and elements for each node. The DOM Tree tab show the HTML representation of the XML.
5. Select the **Script** menu and then select **Other** from the **Add** sub menu.
6. Expand the Web Tests group and select XML Test.
7. Click **OK**.
8. Enter a name for the test.
9. Set the **Verify Only, never fail** option.
10. If necessary, select the **Object Type**: `web.XMLDocument`.
11. Enter the window path. You can use the **Capture** or **Select** menu options to capture or select an object path. See Section 5.4.1, "Path Editor Toolbar" for additional information about the Path Editing Toolbar.
12. Select an XML node to test in the **XML** tree and enable or disable the test for the node by selecting or clearing the check box. The XML tree has the following navigation toolbar options:

- **Filter disabled items**: Clears unselected items from view in the XML test dialog box.
- **Next enabled node**: Moves the selection pointer to the next enabled node in the XML tree.
- **Previous enabled node**: Moves the selection pointer to the previous enabled node in the XML tree.

13. Select an XML node to test and select the **Tests** attributes for each selected XML node.

    Tests: Shows the attributes of the selected XML node. Select an attribute and specify the test details.

    - **Attribute**: Shows the name of the attribute. When selected, the attribute is included in the XML test. When cleared, the attribute is not included in the XML test.
    - **Expected Value**: Shows either the recorded value or the test criteria for the attribute depending upon which **Display** option is selected.
    - **Enable All**: Opens a menu for selecting which types of XML or attributes to enable all.
    - **Disable All**: Opens a menu for selecting which types of XML or attributes to disable all.
    - **Display**: Selects which values appear in the Test Detail section.
      - **Tests**: When selected, the **Value** list shows the test criteria defined for each attribute.
      - **Recorded Values**: When selected, the **Value** list shows the recorded value for each attribute.

14. If necessary, expand the **Test Details** section and edit the details for the selected XML node.

    Test Details: Specifies the test details for the select XML node in the tree:

    - **Recorded Value**: Shows the recorded value for the selected XML node.
    - **Attribute**: Shows the attribute name of the selected XML node.
    - **Value Type**: Specifies the data value type of the selected XML node.
    - **Operator**: Specifies the test operator to use for the selected XML node.
    - **Expected Value**: Specifies the value to compare to the recorded value for the selected XML node.

15. Click **OK** to add the XML Test node to the script tree.

    In the Java Code view, the XML Test consists of the code executed in the `web.xmlDocument(`*`objectId`*`).verifyXML()` or `web.xmlDocument(`*`objectId`*`).assertXML()` method:

```
web.xmlDocument(5, "/web:window[@index='0' or
  @title='http://myServer/myFile.xml']
  "/web:document[@index='0']")
  .verifyXML("XMLTest", web.nodeTests());
```

    or

```
web.xmlDocument(5, "path").verifyXML("testXML",
  web.nodeTests(web.node("/root").attrCountTest("0", TestOperator.IntEqual),
```

```
   web.node("/root").childCountTest("2", TestOperator.IntEqual),
   web.node("/root/testNode[2]").attrTest("id", "testNode2",
     TestOperator.StringExact),
   web.node("/root/testNode[2]").attrCountTest("1", TestOperator.IntEqual),
   web.node("/root/testNode[2]").childCountTest("2", TestOperator.IntEqual),
   web.node("/root/testNode[2]/testSubNode[1]").attrTest("id","testSubNode1",
     TestOperator.StringExact),
   web.node("/root/testNode[2]/testSubNode[1]").attrCountTest("1",
     TestOperator.IntEqual),
   web.node("/root/testNode[2]/testSubNode[1]");.childCountTest("0",
     TestOperator.IntEqual)));
```

or

```
web.xmlDocument(5, "/web:window[@index='0' or
  @title='http://myServer/myFile.xml']
  "/web:document[@index='0']")
  .assertXML("XMLTest", web.nodeTests());
```

or

```
web.xmlDocument(5, "path").assertXML("testXML",
  web.nodeTests(web.node("/root").attrCountTest("0", TestOperator.IntEqual),
   web.node("/root").childCountTest("2", TestOperator.IntEqual),
   web.node("/root/testNode[2]").attrTest("id", "testNode2",
     TestOperator.StringExact),
   web.node("/root/testNode[2]").attrCountTest("1", TestOperator.IntEqual),
   web.node("/root/testNode[2]").childCountTest("2", TestOperator.IntEqual),
   web.node("/root/testNode[2]/testSubNode[1]").attrTest("id","testSubNode1",
     TestOperator.StringExact),
   web.node("/root/testNode[2]/testSubNode[1]").attrCountTest("1",
     TestOperator.IntEqual),
   web.node("/root/testNode[2]/testSubNode[1]");.childCountTest("0",
     TestOperator.IntEqual)));
```

In the above code examples, `verify` means "do not stop on failure" and `assert` means "stop on failure".

If an XML test fails on playback, you can select the XML test result in the **Results** view and view the differences in the **Comparison** tab of the **Details** view.

5.4.12 Adding a Wait for Page

You can use the Wait for Page browser option to cause the script playback to wait until a page is returned by the server before continuing playback.

To add a Wait for Page command to a script:

1. Record a Web Functional Test script.
2. Expand the **Run** node.
3. Select the script node where you want to add the Wait for Page node.
4. Select the **Script** menu and then select **Other** from the **Add** sub menu.
5. Expand the Web Actions group.
6. Expand the Browser group.
7. Select **Wait For Page** and click **OK**.
8. Enter the document ID in XPath format.

9. Select or clear the **Wait for any page** option.
10. If you clear the **Wait for any page** option, specify the URL and Match option.
11. Set the **Timeout** value.
12. Click **OK** to add the Server Response node to the script tree.

    In the Java Code view, a `web.window(objectId).waitForPage()` method will be added to the script code:

```
web.window(6,
"/web:window[@index='0']").waitForPage("http://testserver2/fmstocks/home.asp",
null, null)
```

5.4.13 Inspecting Object Paths

To inspect a Web object path:

1. Record a Web Functional Test script.
2. Select the **Script** menu and then select **Inspect Path**. OpenScript opens the capture mode. This dialog box lets you capture the object path for an object on a Web page.

   Path: Shows the Object Identification path of the object highlighted with the mouse cursor.

   The following options are available on the pulldown menu after you press F10 to capture an object path:

   - **Capture Object:** Opens a capture object dialog box and starts the capture mode. Navigate to the object to capture and use options in the capture dialog box to capture the object path.
   - **View Object Path**: Opens a dialog box for viewing the currently selected object path.
   - **Save to Object Library**: Opens a dialog box for specifying the Object Library in which to save the object path.

   Position: Shows the x, y coordinates for the entire screen. The upper left coordinate of entire screen is {0,0}. This position coordinate is used with Functional tests that include object types, such as Flash, where screen positions are required to perform mouse actions on the object. The position coordinates are used with the functional test methods, `ft.mouseClick()`, `ft.mouseDown()`, and `ft.mouseUp()`.

   Offset: Shows the x, y coordinates relative to the currently highlighted object. The upper left coordinate of highlighted object is {0,0}. This offset coordinate is used with Web Functional tests that include object types, such as Flash, where screen positions are required to perform a mouse click on the object. The position coordinates are used with the Web functional test method, `web.element().mouseClick()`.
3. Navigate to the Web object and place the mouse cursor on the object.
4. Press F10 to capture the path.
5. Highlight the path with the mouse cursor and press the Ctrl+C keys to copy the path to the clipboard.
6. Click **OK** when finished.

5.4.14 Using the Object Details View

The Object Details view provides options for viewing the attributes and values for the objects selected in the browser connected to the view. The Object Details view is available for Functional tests.

To view object details:

1. Create a Functional Test script.
2. Select the **View** menu and then select **Object Details**. OpenScript opens the Object Details view for capturing and viewing objects in the Document Object Model. This dialog box lets you connect to a browser, capture the object path for an object on a Web page.

   The following toolbar buttons are available in the Object Details view:

   - **Refresh Tree** - refreshes the Object Details tree pane.
   - **Find a node to inspect by selecting in browser** - starts the capture mode for selecting the Web page object in the browser. Highlight the object in the browser and press F10 to select it and show the attributes in the Object Details View.
   - **Connect to browser/Disconnect from browser** - connects or disconnects the Object Details View to the browser. The Object Details View must be connected to the browser to capture objects.
3. Click the **Connect to Browser** toolbar button. A new browser opens.
4. Navigate to the page containing the object you wish to view. The tree view loads the DOM tree for the page in the browser.

   The following options are available in the Object Details view:

   - **Module:** Selects the type of OpenScript module. The objects in the tree view change to the specific module type. For example, the Web module shows the HTML DOM tree. The ADF module shows the ADF object tree.
   - **Find**: Provides search capabilities to locate specific text in the Object Details. Type the text to find and click **Next** or **Previous** to locate the attributes and values within the tree.
   - **Partial Match**: When selected the **Next** or **Previous** search will match partial text strings specified in **Find**. When cleared, the **Next** or **Previous** search will match the entire **Find** string.
   - **Next**: Searches down the tree for the next object that matches the **Find** string.
   - **Previous**: Searches up the tree for the previous object that matches the **Find** string.
   - **Tree pane**: Shows the Document Object Model (DOM) tree of a Web application, the ADFUIComponents tree of an ADF page, and the Forms element tree of a forms application. You can use the **Find/Next/Previous** options to locate objects in the tree or use the Find a node to inspect by selecting in browser toolbar button to locate the object by selecting it in the connected browser. The right-click shortcut menu includes the following options for working with the object selected in the tree:
     - **View Object Path**: Opens a dialog box showing the full path of the object.
     - **Add Object Test**: Opens the Object Test dialog for defining an object test for the object selected in the tree.

  - **Add Table Test**: Opens the Table test dialog box for defining a table test for the table object selected in the tree. This option is only available for table objects.
  - **Save to Object Library:** Opens the Save to Object Library dialog box for saving the object path to an object library.
- **Attribute:** Shows the attribute name of the object selected in the tree.
- **Value:** Shows the value of the object attribute selected in the tree.

5. Use the **Find/Next/Previous** options to locate objects in the tree or use the Find a node to inspect by selecting in browser toolbar button to locate the object by selecting it in the connected browser.

   If you use the **Find a node to inspect by selecting in browser** toolbar button, highlight the object you wish to view in the browser and press F10. The tree view expands and selects the object.

6. Use the tree view shortcut menu to view the object path, add tests, or save the object path to an Object Library.

7. Select the **Disconnect from Browser** toolbar button when finished.

5.4.14.1 Viewing the Object Path

To view the path for an object:

1. Open the Object Details view and select an object in the tree.

2. Right-click on the object and select **View Object Details**. The View Object Details dialog box has the following option:

   - **Path**: Shows the path of the object. Click the toolbar button to switch between the tree view and the XPath view.

5.4.14.2 Adding an Object Test

To add an object test:

1. Open the Object Details view and select an object in the tree.

2. Right-click on the object and select **Add Object Test**. The Object Test dialog box opens for defining an object test for the object selected in the tree. See Section 5.4.7, "Adding Object Tests" for additional information about defining an Object Test.

5.4.14.3 Adding a Table Test

To add a table test:

1. Open the Object Details view and select a Table object in the tree.

2. Right-click on the object and select **Add Table Test**. The Table Test dialog box opens for defining a table test for the table object selected in the tree. This option is only available for table objects. See Section 5.4.8, "Adding Table Tests" for additional information about defining a Table Test.

5.4.14.4 Saving an Object Path to an Object Library

To save an object path to an object library:

1. Open the Object Details view and select an object in the tree.

2. Right-click on the object and select **Save to Object Library**. The Save to Object Library dialog box opens for saving the object path to an object library. The Save to Object Library dialog box has the following options:
   - **Object Library**: Lists the saved object libraries where the selected path can be saved.
   - **New**: Opens a dialog for specifying a new Object Library file.
   - **Name**: Specifies a name for the object path.
   - **Description**: Specifies a description for the object path.
3. Select the library where you want to save the path or create a new library.
4. Enter a new name and description.
5. Click **OK**.

5.4.15 Setting Script Properties

To set script properties:

1. Record a Web Functional Test script.
2. Select the **Script** menu and then select **Script Properties**.
3. Select the properties category.
4. Specify the properties for the category.
5. Click **OK** when finished.

5.4.16 Substituting Databank Variables

To substitute a databank variable for a query string parameter in a script:

1. Record a Web Functional test script that has text string parameters.
2. Expand the **Run** node.
3. Expand the node containing text parameters.
4. Right-click a text parameter node and select **Properties** from the shortcut menu.
5. If you have already configured the script with one or more databanks, select the click the [Substitute Variable] icon and select the databank field to substitute for the text parameter from the desired databank file and click **Finish**.
6. If you have not already configured the script with a databank, select **Add new databank** and click **Next**.
7. Select the database type and click **Next**.

   For CSV File databanks:

   a. Select the repository and folder then select the databank file to use.
   b. Click **Next**.
   c. Select the column (field name) to substitute for the text parameter and click **Finish**.

   For database databanks:

   a. Select the Database Driver to use and specify the database connection information. The **URL** will be populated with the connection information.

b. Specify the **Username** and**Password** authentication for the database.

c. Specify the database **Query** to use to retrieve the databank columns and data from the database.

d. Specify the alias name to use for the databank.

e. Click **Test** to verify the connection to the database.

f. Click **Next**.

g. Select the column (field name) to substitute for the text parameter and click **Finish**.

In the Tree View, the databank variable appears in place of the recorded value as {{db.*databankFileName.field*,*recordedValue*}}.

In the Java Code view, the databank variable appears as `{{db.`*`databankFileName`*`.`*`field`*`,`*`recordedValue`*`}}` in the `web.text(`*`objectId`*`).setText` method:

```
web.text(1, "/web:window[@index='0']
   /web:document[@index='0']
   /web:form[@index='0']
   /web:input_text[@id='ticker'
     or @name='ticker'
     or @index='0']").setText("{{db.fmstocks_data.ticker,orcl}}")
```

5.4.17 Using the Web Functional Test Module API

The Web Functional Module includes a script Application Programming Interface (API) specific to Web DOM functional testing. The Web Functional Test Module recorder creates the Java code that corresponds to the Tree View and displays the Web Functional Test commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the Web Functional Test API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Web Functional Testing Module are part of the "web" class. Additional functional test methods are available in the "ft" class. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the Web Testing Module API include:

- Adding control statements
- Adding think time
- Launching and closing the Browser
- Navigating to a URL
- Performing actions on Web objects (click, double click, etc.)
- Setting text fields
- Waiting for a page to load

Many API methods can be added using the Web Functional Test Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures and code examples. See the API Reference in the OpenScript help for additional programming information.

5.5 Editing Object Libraries

To edit an object library:

1. Create a Web Functional Test script project.
2. Add an Object Library to the script and record the Web application under test.
3. Select **Open Object Library** from the **File** menu.
4. Select the *libraryName*.properties file and click **Open**.
5. If necessary, click the Object Library tab.
6. Select an object in the Objects list. The Objects list section has the following toolbar buttons:

   Sort the paths alphabetically - sorts the object XPaths alphabetically.

   Add: Adds a new object to the list. After adding a new object, specify the name and object identification path in the Details section.

   Delete: Deletes an object from the object list. A confirmation message appears.

   Inspect Path: Starts the object capture mode and opens a browser for selecting the object path to capture.

7. Edit the object attributes in the Details section.

   You can add objects to and delete objects from the Object list. You edit the object attributes in the object string or in the tree hierarchy of the Details section. The tree hierarchy lets you move attributes up or down in the priority order.

   Details: Specifies the name and object identification path segments and attributes for the selected object in the object list.

   - **Name**: Specifies the name of an object. A name is required for each object. The name is used in the script code to reference the object path in the object library. For example, an object in the Object Library named `web_window_0` may have a path defined as follows:

     ```
     /web:window[@index='0' or @title='Home']"
     ```

     In the script code, the object path in an Object Library is referenced using the script variable format `{{obj.libraryName.objectName}}`. For example, a script navigation command references an object path in an Object Library as follows:

     ```
     web.window(44, "{{obj.MyObjLib.web_window_0}}")
       .navigate("http://myServer/home");
     ```

   - **Description**: Specifies any description for the object. A description is optional.
   - **Path**: Specifies the full object identification path segments and attributes that the Web Functional Test Module uses to identify an object on a Web page during play back of a script. The following toolbar buttons are available for working with object paths:
     - **Toggle between tree and text**: Toggles the Path editor between tree view mode and text mode. In tree view mode, the object path is represented as tree hierarchy. Use the toolbar buttons to add, edit, or delete path attributes in the tree hierarchy or the change the priority order of the attributes. Click the arrows next to the path segments to expand and collapse specific tree segments. When fully collapsed, a path tree may appear as follows:

```
web:window
web:document
web:form
web:input_submit
```

When full expanded, a path tree may appear as follows:

```
web:window
  @index='0'            or
  @title='Stocks'
web:document
  @index='0'
web:form
  @id='loginform'       or
  @name='loginform'     or
  @index='0'
web:input_submit
  @name='LoginButton'   or
  @value='Login'        or
@index='0'
```

Select a path segment or attribute in the tree and click the **Edit** toolbar button or double-click on a path segment to open the edit dialog box.

In text mode, the object path is represented as an XPath text string as follows:

```
/segment[@attr1='value' and|or @attr2='value' and|or @attrN='value']
/nextsegment[...]/segmentN[...]
```

The following example XPath shows the full path to a Submit Input button of a Web form:

```
/web:window[@index='0' or @title='Stocks']/web:document
[@index='0']/web:form[@id='loginform' or @name='loginform' or
@index='0']/web:input_submit[@name='LoginButton' or @value='Login'
or @index='0']
```

You can edit the object XPath text directly in the edit box. The edit box includes an auto-complete feature that lists available options to select when editing an object path. Typing the / (forward slash) character opens the list of path segments available for selection. Typing the @ character opens the list of path attributes available for selection. Typing additional text narrows down the list of available selections to only those matching the text entered.

- **Add**: Opens a dialog box for adding a path segment or attribute to the object identification tree.
- **Edit**: Opens a dialog box for changing the value of the object path segment or attribute selected in the tree.
- **Delete**: Deletes the selected object path segment or attribute from the tree.
- **Up**: Moves the selected object path segment or attribute up in the tree hierarchy. This button is only active if the object path segment or attribute is able to be moved up in the tree hierarchy.
- **Down**: Moves the selected object path segment or attribute down in the tree hierarchy. This button is only active if the object path segment or attribute is able to be moved down in the tree hierarchy.

  - **View Full Path**: Opens a dialog box for viewing the full object path.

- **Use a different path for identifying the object during record time**: When selected, a secondary path editing box opens for specifying the object path to use when recording a script. The object path specified is used only when recording a script and can include Regular Expression, Wildcard and Exact matching for specific path segments and attributes.
- **Record Path**: Specifies the full object identification path and attributes that the Web Functional Test Module recorder uses to identify an object on a Web page. The Recorded Path toolbar buttons are the same as the **Path** toolbar button except for **View Full Path**. The object path specified is used only when recording a script and can include Regular Expression, Wildcard, and Exact matching for specific path segments and attributes. Specify the **Match** type to use for the **Record Path** path segments and attributes.
- **Match**: Specifies the matching type to use the record time object identification path segments and attributes. Regular Expression and Wildcard matching can only be used for Record object identification paths.
  - **Exact**: When selected, the object identification matches the full path exactly as specified.
  - **RegEx**: When selected, the object identification matches using Regular Expression matching. Edit the path segments and attributes to include a Regular Expression where you want to use Regular Expression matching.
  - **Wildcard**: When selected, the object identification matches using Wildcard matching. Use `?` (question mark) to match single characters or `*` (asterisk) to match multiple characters. Edit the path segments and attributes to include a Wildcard where you want to use Wildcard matching.

You can click the *libraryName*.properties tab to view or edit the object/object attribute text strings in the object library file source. The basic format of a object/object attribute text string is as follows:

```
objectName=path
```

objectName can be any user-defined name. This name corresponds to the **Name** specified in the **Details** section of an object in the library. *objectName* can also include names that specify the match option and recorded path strings for specific objects.

path is an XPath formatted string. *path* can also include the match option and recorded path strings for specific objects.

8. Select **Save** from the **File** menu or click the Save toolbar button to save changes to the object library file.

6

Using the HTTP Module

This chapter provides instructions on configuring and using the OpenScript HTTP Module, which tests Web-based applications by automating the underlying HTTP protocol traffic.

6.1 About the HTTP Module

The OpenScript HTTP Module is an application/protocol module that supports load testing of Web-based applications that communicate via http(s) protocol. OpenScript provides a flexible and easy-to-use scripting interface for both Technical Testers and Non-Technical Testers. The OpenScript HTTP Module enables script creation from both the code view and GUI view scripting interfaces.

The HTTP Module extends the OpenScript platform with HTTP Proxy recording and playback capabilities. The proxy recorder automatically captures Web page navigations and records them as tree view nodes (with the underlying code in the Code View) in the script. The HTTP Module also provides additional GUI script modification options for HTTP navigation.

The HTTP Module provides two recording modes which can be specified in the HTTP recording preferences.

- **Web**: When selected, the script recorder generates the Web mode HTTP script Java code for the requests. This Java code is less verbose than the HTTP mode to simplify Java coding of the scripts. The advantage of the Web mode compared to the HTTP mode is that it simplifies script creation and makes the script easier to read when testing Web browser applications. The Web mode can be used for any Web browser application that communicates via HTTP.
- **HTTP**: When selected, the script recorder generates the verbose HTTP script Java code with detailed GET and POST requests. This can be used for any HTTP application including Web browser applications and other applications that communicate via HTTP. This is the record mode used for HTTP scripts prior to version 9.20 of OpenScript.

6.1.1 Key Features of the HTTP Module

The HTTP module provides the following functionality:

- Records HTTP protocol requests for playback automation. The requests can be generated by a Web browser (i.e. IE) or by a plug-in (i.e. AJAX XMLHTTP plug-in).

- Plays back HTTP scripts to validate proper functionality. Playback runs interactively in the OpenScript user interface and is also supported in the Oracle Load Testing Agents (i.e. Java Agent).
- Provides full script code view integration to support script generation for the HTTP Module. The commands include (but are not limited to) methods to generate GET requests, POST requests, correlation substitutions, validation, etc. The HTTP Module includes an additional API to support HTTP protocol code scripting.
- Allows users to parameterize user inputs to HTTP scripts and drive those inputs from an external data file (Databank).
- Provides additional options/settings that are specific to HTTP scripts within the HTTP categories in the preferences interface.
- Reports playback results for HTTP scripts in the Results and Console views.

6.2 Navigation Editing (Correlation)

The HTTP Module enables users to view and edit all recorded navigations and related parameters (headers, post data, etc.) in either the script GUI Tree view or Code View. It also enables them to view and edit any default correlation/parameterization of dynamic navigations and apply their own correlation to handle dynamic navigations.

Navigation Editing GUI View: Configures the navigations they want to parameterize and the correlation rules they want to apply through a navigation editing GUI view interface. The GUI allows viewing and editing properties for different types of navigations (for both Web/HTTP applications and non-Web/HTTP applications) and data inputs. This GUI View includes:

- **Display & Editing for Recorded Navigations**: Includes recorded navigations and any navigation parameters like headers, etc. and a mechanism for users to edit/add/delete navigations including dynamic parameter sources/targets.
- **Display & Editing for Correlation Rules Library & Editing**: Includes a list of all default correlation rules included in the module(s) (listed by application type) and a mechanism for users to add/edit/delete correlation rules.

Navigation Editing Code View Commands: Users are able to specify the navigations they want to parameterize and the data source they want to drive the inputs from through navigation editing commands in the code view. These commands map to the navigation editing GUI view.

```
beginStep("[2] Home", 3266);
{
   http.post(4, "http://testserver2/fmstocks/{{FORMACTION_0,default.asp}}",
      null,
      http.postdata(http.param("login", "{{INPUT_0,ta616}}"),
         http.param("password", "{{INPUT_1,ta}}"),
         http.param("LoginButton", "{{INPUT_2,Login}}")),
      null, true, "ASCII", "ASCII");
         {
         http.solveXpath("LINK_1_3", ".//A[text()='research a company']
             /@href", "TickerList.asp", 0);
         http.solveXpath("LINK_1_2", ".//A[text()='Logout']
             /@href", "logout.asp", 0);
         }
}
endStep();
```

6.2.1 Setting Correlation Preferences

To set correlation preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Correlation and Verification category.
4. Select **HTTP**.
5. Select or clear the check boxes for defined rules.
6. Select **Add Library**, **Add Rule**, or **Edit Rule** to define custom correlation rules.
7. Click **OK**.

6.2.2 Adding Correlation Libraries

Selecting **Add Library** in the Correlation Rules Preferences opens the **Add Library** dialog box. This dialog box lets you specify a new correlation library for transforming dynamic data in recorded script URLs and related parameters (headers, post data, etc.) to variable names that will be recognized by the script playback engine (OpenScript or Oracle Load Testing). The dialog box has the following options:

Name: Specifies the name of the correlation library. After you define a library you can use the **Add Rule** button to specify the rules to include in the library. The name is required. You can also select **Copy rules** to copy correlation rules from an existing library.

Copy rules from existing library: Lets you copy correlation rules from an existing library to a new library.

- **Copy Rules**: When selected, a list of existing correlation rule libraries will be enabled for copying.
- **Library**: Lists the correlation rule libraries available for copying.

6.2.3 Adding and Editing Correlation Rules

Selecting **Add Rule** or **Edit Rule** in the Correlation Rules Preferences opens the **Add Rule** or **Edit Rule** dialog box. This dialog box lets you specify or edit a correlation rule for transforming dynamic data in recorded script URLs and related parameters (headers, post data, etc.) to variable names that will be recognized by the script playback engine (OpenScript or Oracle Load Testing). The dialog box has the following options:

Type: Specifies the type of correlation rule. The available Source and Target options change depending upon the rule type. The following rule types are available:

- Client Set Cookie: This rule type automatically transforms web page cookie objects with dynamic data.
- Correlate Cookie Header: This rule type automatically transforms web page cookie header objects with dynamic data.
- Correlate Headers: This rule type automatically transforms web page header objects with dynamic data.
- Correlate Referer Header: This rule type automatically transforms web page referer header objects with dynamic data.

- DOM Correlation: This rule type automatically transforms web page Document Object Model (DOM) objects with dynamic data.
- Function/Text Substitution: This rule type lets you specify a user-defined function to replace a specific parameter or parameters.
- Java Session id: This rule type automatically transforms Java Session ID objects and replaces the ID with a variable value.
- Substitute Recorded Date: This rule type lets you specify a Regular Expression pattern to find and replace date parameters with a variable value.
- Title Verification: This rule type lets you specify the type of title verification test to add to each page.
- Variable Substitution: This rule type lets you specify a Regular Expression pattern to find and replace a specific parameter or parameters with a variable value.

Name: Specifies the name of the correlation rule. The name is required.

The following sections describe the rule types.

6.2.3.1 Client Set Cookie

When Client Set Cookie is selected, the **Source** and **Target** show the following options:

Source: Always cookies.

Target: Specifies which document object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.
- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the cookie parameter(s) to which to apply the correlation rule using a Regular Expression. (See Variable Substitution Rules below for examples.)

6.2.3.2 Correlate Cookie Header

When Correlate Cookie Header is selected, the **Source** and **Target** show the following options:

Source: Always cookie header.

Target: Specifies which document object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.
- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the cookie parameter(s) to which to apply the correlation rule using a Regular Expression. (See Variable Substitution Rules below for examples.)

6.2.3.3 Correlate Header

When Correlate Header is selected, the **Source** and **Target** show the following options:

Source: Always web page headers.

Target: Specifies which header object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.
- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the header parameter(s) to which to apply the correlation rule using a Regular Expression. (See Variable Substitution Rules below for examples.)

6.2.3.4 Correlate Referer Header

When Correlate Referer Header is selected, the **Source** and **Target** show the following options:

Source: Always web page referer headers.

Target: Specifies which referer header object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.
- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the referer header parameter(s) to which to apply the correlation rule using a Regular Expression. (See Variable Substitution Rules below for examples.)

6.2.3.5 DOM Correlation Rules

When DOM Correlation Rules is selected, the **Source** and **Target** show the following options:

Source: Specifies which document object(s) to substitute as dynamic data.

- **Links**: When selected, web page link objects with dynamic data will automatically transformed to variable values.
- **Action**: When selected, web page action objects with dynamic data will be automatically transformed to variable values.
- **Input**: When selected, web page input objects with dynamic data will be automatically transformed to variable values.
- **TextArea**: When selected, web page TextArea objects with dynamic data will be automatically transformed to variable values.
- **JavaScript**: When selected, web page JavaScript objects with dynamic data will be automatically transformed to variable values.
- **Frame**: When selected, web page Frame objects with dynamic data will be automatically transformed to variable values.
- **Option**: When selected, web page Option objects with dynamic data will be automatically transformed to variable values.
- **XML Text**: When selected, XML pages with dynamic data will be automatically transformed to variable values.

Target: Specifies which referer header object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.

- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the web page object(s)/parameter(s) to which to apply the correlation rule using a Regular Expression. (See Variable Substitution Rules below for examples.)

6.2.3.6 Function/Text Substitution Rules

When Function/Text Substitution Rules is selected, the **Source** and **Target** show the following options:

Source: Specifies the function name or text to use as the substitute for dynamic data.

- **Function/Text**: Specifies the name of the function to use to search the source location. The following functions are available:.
  - `{{@decode({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@decode({{myVariable}})}}`. The value is the encoded text string specified by the Target Regular Expression.
  - `{{@decrypt({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@decrypt({{myVariable}})}}`. The value is the encrypted text string specified by the Target Regular Expression.
  - `{{@deobfuscate({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@deobfuscate({{myVariable}})}}`. The value is the obfuscated text string specified by the Target Regular Expression.
  - `{{@encode({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@encode({{myVariable}})}}`. The value is the text string specified by the Target Regular Expression.
  - `{{@encrypt({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@encrypt({{myVariable}})}}`. The value is the text string specified by the Target Regular Expression.
  - `{{@file({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable {`{@file({{myVariable}})}}`. The value is the text string contained in the file specified by myVariable or a hard coded path such as `{{@file(c:\\OpenScript_Sample.txt)}}`.
  - `{{@getAndIncrement({{myVariable}},delta)}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@getAndIncrement({{myVariable}},` *`delta`*`)}}`. The value is the numeric value specified by the Target Regular Expression. `delta` is the amount to increment the value by each time.
  - `{{@hostip}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@hostip}}`. The value is the host IP address.
  - `{{@hostname}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@hostname}}`. The value is the host name.

- `{{@iterationnum}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@iterationnum}}`. The value is the script playback iteration number.
- `{{@jstr({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@jstr({{myVariable}})}}`. The value is a serialized Java string that specifies a Hexadecimal length value followed by the string contained in *`myVariable`* in the format `\00\09var_value`.
- `{{@len({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@len({{myVariable}})}}`. The value is the length of the string contained in *`myVariable`*.

Note: For `@jstr`, `@file`, and `@len` functions, *`myVariable`* is an OpenScript script variable defined using the `http.solve` or `http.solveXpath` methods. The variable name must be enclosed in double `{{ }}` braces within the function parenthesis. For example:

`{{@len(`**`{{`**`myVariable`**`}}`**`)}}`

- `{{@obfuscate({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@obfuscate({{myVariable}})}}`. The value is the text string specified by the Target Regular Expression.
- `{{@random(max)}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@random(`*`max`*`)}}`. The value is the numeric value specified by the Target Regular Expression. The default minimum value is 0. *`max`* is the maximum value limit for the random number. The generated random number is a uniformly distributed pseudorandom integer value between 0 (inclusive) and max value (exclusive), drawn from this random number generator's sequence. For example, `{{@random(2)}}` will return a number between 0 (inclusive) and 2 (exclusive). The random value is an integer from the formula `randomValue = min + m_random.nextInt(max - min)`, where `m_random.nextInt` uses the `java.util.Random.nextInt` method.
- `{{@random(min,max)}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@random(`*`min`*`, `*`max`*`)}}`. The value is the numeric value specified by the Target Regular Expression. *`min`* is the minimum value limit for the random number. *`max`* is the maximum value limit for the random number. The generated random number is a uniformly distributed pseudorandom integer value between `min` value (inclusive) and `max` value (exclusive), drawn from this random number generator's sequence. For example, `{{@random(1,4)}}` will return a number between 1 (inclusive) and 4 (exclusive). The random value is an integer from the formula `randomValue = min + m_random.nextInt(max - min)`, where `m_random.nextInt` uses the `java.util.Random.nextInt` method.
- `{{@sessionname}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@sessionname}}`. The value is the script playback session number.

- `{{@timestamp}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@timestamp}}`. The timestamp value is the difference, measured in milliseconds, between the current time and Midnight, January 1, 1970 UTC.
- `{{@timestampsecs}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@timestampsecs}}`. The value is the current timestamp in seconds instead of milliseconds.
- `{{@today(MM/dd/yyyy)}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@today(MM/dd/yyyy)}}`. The value is the current date in month/day/year format.
- `{{@topLevelStepName}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@topLevelStepName}}`. The value is the top level step group name.
- `{{@urlEncode({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@urlEncode({{myVariable}})}}`. The value is the text string specified by the Target Regular Expression. For example, the string, 'the file "abc" is in \root\etc', would be encoded as: the+file+%22abc%22+is+in+%5Croot%5Cetc.
- `{{@vuid}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@vuid}}`. The value is the virtual user ID specified by the Target Regular Expression.
- `{{@xmlDecode({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@xmlDecode({{myVariable}})}}`. The value is the encoded text string specified by the Target Regular Expression.
- `{{@xmlEncode({{myVariable}})}}`: Searches the source using the specified Target Regular Expression and replaces the data with the script variable `{{@xmlEncode({{myVariable}})}}`. The value is the text string specified by the Target Regular Expression.

Target: Specifies which referer header object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.
- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the web page object(s)/parameter(s) to which to apply the correlation rule using a Regular Expression. (See Variable Substitution Rules below for examples.)

6.2.3.7 Java Session id

When Java Session id is selected, the **Source** and **Target** show the following options:

Source: Specifies the attributes to use as the substitute for Java Session ID.

- **Variable Name**: Specifies the name of the variable to use as the substitute for dynamic data.

- **Pattern**: Specifies the Regular Expression to use to locate the dynamic data to replace. This rule searches the raw HTML for the Java Sesion ID Regular Expression pattern `jsessionid=(.+?)(?:"|&)` and replaces it with the variable name `http.jsessionid`.
- **Result Index**: Specifies a 0-based index value defining the specific result to retrieve if the Pattern returns multiple results.
- **Error Message**: Specifies an error message to report if the source data is not found on playback.
- **Source**: Specifies where to search for the dynamic data to replace: HTML Display Contents, Raw HTML or Response Header.
- **Encoding**: Specifies if encoding should be used for the search and the type.

Target: Specifies which object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.
- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the web page object(s)/parameter(s) to which to apply the correlation rule using a Regular Expression. (See Variable Substitution Rules below for examples.)

6.2.3.8 Substitute Recorded Date

When Substitute Recorded Date is selected, the **Source** and **Target** show the following options:

Source: Specifies the attribute to use as the substitute for dynamic data.

- **Attribute Name**: for internal use only and should only be set to "value" (without quotations).
- **Date Pattern**: Specifies the date pattern in the form M/dd/yyyy. The Date Pattern follows standard Java Date format string conventions. When correlating scripts, the time that the navigation was recorded is converted to a date using the specified Date Pattern. If the current date is found in a request, it is replaced with: `{{@today(date_pattern)}}`.

Target: Specifies which which document object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.
- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the referer header parameter(s) to which to apply the correlation rule using a Regular Expression. (See Variable Substitution Rules below for examples.)

6.2.3.9 Title Verification

When Title Verification is selected, the **Source** and **Target** show the following options:

Source: Specifies the type of title verification test to add to each page. When **Verify only, never fail** is selected, the title verification test is a verify only, never fail title verification test reporting a warning if a test fails. When **Verify only, never fail** is

cleared, the title verification test is a title verification test reporting a failure if a test fails.

Target: Specifies which document object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.
- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the referer header parameter(s) to which to apply the correlation rule using a Regular Expression. (See Variable Substitution Rules below for examples.)

6.2.3.10 Variable Substitution Rules

When Variable Substitution Rules is selected, the **Source** and **Target** show the following options:

Source: Specifies the attribute to use as the substitute for dynamic data.

- **Variable Name**: Specifies the name of the variable to use as the substitute for dynamic data.
- **Pattern**: Specifies the Regular Expression to use to locate the dynamic data to replace.
- **Error Message**: Specifies an error message to report if the source data is not found on playback.
- **Source**: Specifies where to search for the dynamic data to replace: HTML Display Contents, Raw HTML or Response Header.
- **Encoding**: Specifies if encoding should be used for the search and the type.

Target: Specifies which referer header object(s) to use as the target location of the transform.

- **Replace all locations**: When selected, the correlation rule applies to any object matching the source criteria.
- **Replace specified location**: When selected, the correlation rule applies only to the object matching the **Location** criteria.
- **Location**: Specifies the web page object(s)/parameter(s) to which to apply the correlation rule using a Regular Expression. Specify a regular expression to narrow down which part of a target request may be replaced with the correlated variable. All or part of a url, query string, and/or postdata may be substituted. Use `((.+?))` to indicate where the variable should be substituted.
  - The expression may be used to substitute a variable into a specific name=value pair. For example, to substitute the session ID in this post data: `Post Data: sessionId=123456&color=blue.`

    Specify the following expression: `sessionId=((.+?)).`

    Using the above expression, if the correlation rule's variable is found on a page and its value matches "123456", then the post data will become: `sessionId={{correlationRuleVariableName,123456}}&color=blue`.
  - If the name=value pair appears URL-encoded in the post data or query string, do NOT URL-encode the expression. For example, to substitute the

"file/folder" parameter in this post data: Post Data: `file%2Ffolder=folderXYZ%2FfileABC&session=ABC%2FDEF`.

Specify the following expression: `file/folder=((.+?))`.

– If substituting a variable into a non-URL-encoded postdata or querystring, do not URL-encode the expression. For example, to substitute the "id" value of the following unencoded XML post data: `Post Data: <xml><session id="12345"/></xml>`.

  Specify the following expression: `<session id="((.+?))"/>`.

  Using the above expression, if the correlation rule's variable is found on a page and its value matches "12345", then the entire post data will become: `<xml><session id="{{correlationRuleVariableName,12345}}"/></xml>`.

– If substituting a variable into a range of URL-encoded Name=Value pairs, then URL-encode the expression. For example, to replace all but the first parameter of the following URL-encoded query string data with one value: Query String: `file=root%2Fdata.txt&sessionId=123%2Fxyz&color=blue`.

  Specify the following expression: `file=root%2Fdata.txt((.+?))`.

  Using the above expression, if the correlation rule's variable is found on a page and its value matches `"&sessionId=123%2Fxyz&color=blue"`, then the entire query string will become:

  ```
  file=root%2Fdata.txt{{correlationRuleVariableName,&session
  Id=123%2Fxyz&color=blue}}
  ```

6.3 Recording Scripts

The OpenScript HTTP Module records parameters defined by each page of the Web application to a script which can then be played back, with parameters in the Web page filled in with values from a Databank file.

The HTTP Module records HTTP protocol requests generated by a Web browser for playback automation. The HTTP Recorder records Web browser events for playback correlation which allows users to correlate dynamic HTTP requests based on knowledge of the GUI events which generated the navigation (i.e. dynamic GET request originated from click on link "x").The HTTP Module provides a Record toolbar button that allows users to initiate the HTTP proxy recorder and captures Web page navigations to the script view. The toolbar includes start and stop recording toolbar buttons.

6.3.1 Setting HTTP Record Preferences

Before recording HTTP scripts, first set the HTTP record preferences.

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node, then expand the **Record** category.
4. Select **HTTP**.
5. Click the tabs and set the preferences. See Section 2.5.2, "HTTP Preferences" for descriptions of the Record Preferences settings.
6. Click **OK** when finished.

6.3.2 Recording a New HTTP Script

To create a new HTTP script, you essentially record the script.

1. Start OpenScript.
2. Set the HTTP Recording preferences if you haven't already.
3. Select **New** from the **File** menu.
4. Expand the Load Testing (Protocol Automation) node and select **Web/HTTP**.
5. Click **Next**.
6. Select the Repository and Workspace.
7. Enter a script name.
8. Click **Finish**. A new Script tree is created in the Script View.
9. Select **Record** from the **Script** menu or click the arrow on the Record toolbar button and select **Proxy Recorder**.
10. If you set the **Always launch a new browser** option in the HTTP Record preferences, the browser automatically opens when you start recording with the HTTP Proxy recorder. If you did not set the **Always launch a new browser** option, you will have to open a browser manually.
11. Load the web page where you want to start recording into the browser.
12. Navigate the web site to record page navigations. The page navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
13. When finished navigating pages, stop the script by selecting **Stop** from the **Script** menu or clicking the **Stop** button on the OpenScript toolbar.
14. If you set the **Close browser** option in the HTTP Record preferences, the browser automatically closes when you stop recording. If you did not set the **Close browser** option, you will have to close the browser manually.
15. Expand the nodes of the script to view the page navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

    Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

6.3.3 Using Client-Side Digital Certificates

This section explains how to configure recorded HTTP scripts to use client-side digital certificates through OpenScript for use as Oracle Load Testing virtual users. In order to use a client-side digital certificate in the Oracle Load Testing Java Agent, it is imperative that the certificate contain a private key. When generating a client certificate file, make sure to set Key Options as "Mark keys as exportable".

6.3.3.1 Exporting Client Certificates from Internet Explorer

OpenScript requires certificate files in the .PFX (PKCS#12) format.

To export client certificates from Internet Explorer to the .PFX format:

1. Select **Internet Options** from the Internet Explorer **Tools** menu.

2. Click the **Content** tab and click **Certificates**.
3. Select the client certificate to export and click **Export**.
4. Click **Next**.
5. Select **Yes, export the private key** at the prompt "Do you want to export the private key with the certificate?". If this option is not available, then the client certificate being exported does not contain a private key and cannot work with the Oracle Load Testing Java Agent.
6. Click **Next** and UNCHECK "Enable strong protection (requires IE 5.0, NT 4.0 SP4 or above)"
7. Click **Next** and enter a password. OpenScript requires private-key password of client certificates to store them to JKS format.
8. Click **Next** and specify a filename to export the PKCS#12 certificate. For example: `mycertfile.pfx`.

6.3.3.2 Configuring OpenScript to use the Client Certificate

Client Certificate are configured in the OpenScript HTTP Record Preferences. See Section 2.5.2.4, "Certificates" for additional information.

To configure the Client Certificate in OpenScript:

1. Start OpenScript and open the load testing script to use with the certificate.
2. Select **OpenScript Preferences** from the **Tools** menu.
3. Expand the **Record** preferences and select **HTTP**.
4. Click the **Certificates** tab.
5. Click the **Browse** button in the **Set Customized Certificate File** section.
6. Select your certificate .PFX file and click **Open**.
7. Enter the password for the certificate.
8. Click **OK**.

6.4 Playing Back Scripts

Once HTTP scripts have been recorded, you can play them back to validate functionality. Playback runs interactively in the OpenScript user interface and is also supported in the Oracle Load Testing.

The HTTP Module provides playback and iterate toolbar buttons that allows users to start the HTTP script playback for either a single playback through the script, or run through multiple iterations using data from a databank file. Playback results for HTTP scripts can be viewed in the Results and Console views.

6.4.1 Setting HTTP Playback Preferences

Before playing back scripts, you should set the playback preferences.

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node, then select the **Playback** category.
4. Select **HTTP**.

5. Expand the groups and set the preferences. See Section 2.4.3, "HTTP Preferences" for descriptions of the Playback Preferences settings.
6. Click **OK** when finished.

6.4.2 Playing Back HTTP Scripts

Once an HTTP script has been recorded, you can play it back.

1. Start OpenScript.
2. Open the HTTP script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.
4. You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

   See"Viewing Script Playback Results" on page 6-14 for more information.

6.4.3 Playing Back HTTP Scripts With Iterations

OpenScript allows repetitive playback of navigations in a script. The iterations can be performed with or without databanks.

1. Start OpenScript.
2. Open the HTTP script to play back.
3. Configure the script to use a databank as described in Section 4.2.1, "Configuring Databanks".
4. Select **Iterate** from the **Script** menu or click the toolbar button.
5. Select **Use Databanks**.
6. Select which databank file to specify the settings for if more than one database is configured for the script.
7. Specify the settings for the databank file.
8. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
9. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

6.4.4 Viewing Script Playback Results

To view HTTP script play back results:

1. Start OpenScript.
2. Open the HTTP script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.
4. When playback is finished, click the Results view. If necessary, select **Results** from the **View** menu to open the Results View.
5. In the Name column, expand the results nodes to view the results.

6. Select script result nodes in the Results View to view specific navigation results in the Details View. If necessary, select **Details** from the **View** menu to open the Details View.

   The Details View includes tabs for viewing the browser rendered content, HTML source, request and response headers, and a comparison tab for comparing playback details for content, request and response headers, cookies, and resources.

7. Click the top level script result in the Results View to view the Results Report in the Details view. You can view the report and export an HTML file of the report.

6.4.5 Resetting Encoding

The Reset Encoding menu option changes the character set used when displaying the recorded HTML. Use this option to reset the encoding for non-English web sites where the correct character encoding may not be set automatically.

To reset encoding:

1. Start OpenScript.
2. Open the HTTP script.
3. Expand the HTTP script and select a navigation node.
4. Right-click on the navigation node and select **Reset Encoding** from the shortcut menu.
5. Enter the encoding value for the recorded HTML page and click **OK**.

6.4.6 Comparing Recorded/Playback Results

To compare HTTP script play back results with the recorded navigations:

1. Start OpenScript.
2. Switch to the Tester Perspective and make sure the Details View is open. If not, select **Details** from the **View** menu to open the Details View.
3. Open the HTTP script to play back.
4. Select **Playback** from the **Script** menu or click the toolbar button.
5. When playback is finished, view the results. If necessary, select **Results** from the **View** menu to open the Results View.
6. In the Name column, expand the results node to view the results.
7. In the Name column, click a navigation node for a page.
8. In the Detail View, click the **Comparison** tab.
9. In the Comparison tab, select Content, Request Headers or Response Headers to view the Recorded and Playback text to compare in the lower panes.

6.4.7 Playing Back HTTP Scripts In Oracle Load Testing

Once recorded, you can play back HTTP scripts in Oracle Load Testing.

If OpenScript and Oracle Load Testing are on the same machine:

1. Start Oracle Load Testing.
2. Select the Repository and Workspace where the OpenScript scripts are located.
3. Select the script to play back.

4. Set the User Mode to Java Client.
5. Configure the scenario parameters as required for the test.
6. Run the scenario in the Autopilot mode.

If OpenScript and Oracle Load Testing are on the different machines:

1. Select **Export Script** from the **File** menu.
2. Select the additional files to export to a Zip file and click **OK**.
3. Copy the exported Zip file to the Oracle Load Testing machine.
4. Start Oracle Load Testing.
5. Select **Upload File** from the **Tools** menu.
6. Select OpenScript Zip as the file type.
7. Enter the name and location of the exported OpenScript Zip file.
8. Select the Repository and Workspace where the OpenScript scripts is to be uploaded.
9. Click **Upload** to upload the file.
10. In the **Build Scenarios** tab, select the script to play back.
11. Set the User Mode to Java Client.
12. Configure the scenario parameters as required for the test.
13. Run the scenario in the Autopilot.

Note: Any external files, such as databanks files, must available to Oracle Load Testing (i.e. located in the path in specified in script file).

6.4.8 Posting Binary or XML File Data

To post Binary or XML File Data:

1. Record an HTTP script as described in "Recording a New HTTP Script" on page 6-12.
2. Open the Java Code view.
3. Use the following code to specify a binary data variable and the `http.navigate()` method to post the data:

```
byte[] data =
oracle.oats.utilities.FileUtil.readBytesFromFile("c:\\image.jpg");
http.navigate(0, "http://www.mysite.com/", null, data, null, true);
```

   The same solution works for an XML file:

```
byte[] data = oracle.oats.utilities.FileUtil.readBytesFromFile("c:\\file.xml");
http.navigate(0, "http://www.mysite.com/", null, data, null, true);
```

If you want to store your binary or XML file inside the script itself:

1. Switch to the Developer perspective.
2. In the Navigator View, create a folder in the script project called "resources".
3. Add your jpg or XML file into the new "resources" folder.
4. Change the first line of the above code sample to this:

```
byte[] data =
getScriptPackage().getResourceFile("resources/yourfile.jpg").getData();
```

By storing the file locally with the script, the file will always be available to the agent, even if it is run on a remote agent machine through Oracle Load Testing.

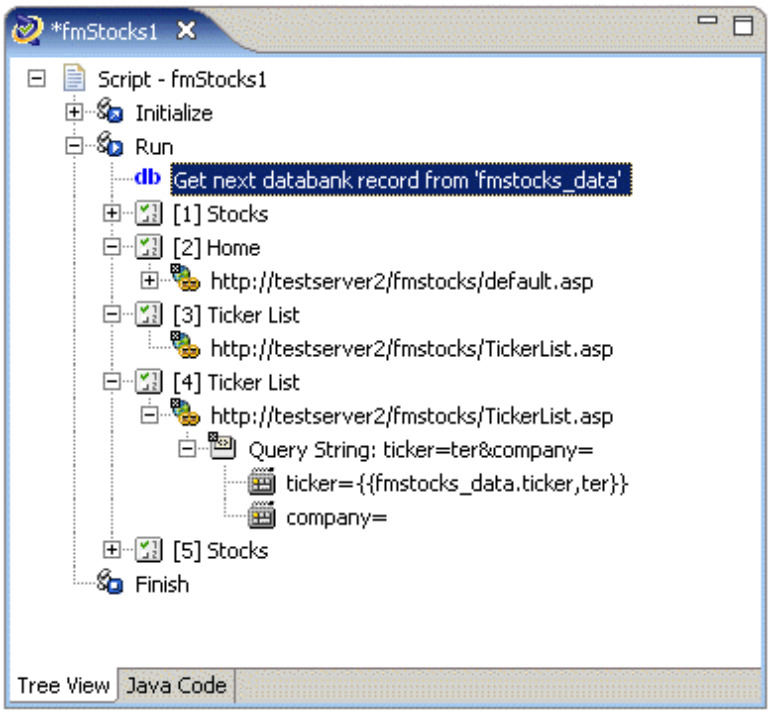
6.5 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

6.5.1 Understanding the HTTP Module Script View

The default display for an HTTP proxy recorded script is the Tree View GUI in the Script View. The HTTP Recorder generates the Tree View and code based upon the Step Group preferences set in OpenScript. The default Step Group settings will generate Step Groups based upon page navigations in the HTTP web application.

Figure 6–1 Script Tree View



Below each navigation will be child nodes for additional information about a page navigation, such as query strings and form action inputs. The right-click shortcut menu includes additional script modification options specific to the script generated using the HTTP Module.The Tree View is a graphical representation of the underlying code in the Code View. For example, "Page [4] Ticker List" in the above Tree View appears as Step Group and `http.get method` code in the Code View, as follows (line breaks and spacing added for clarity):

```
beginStep("[4] Ticker List", 3422);
{
   http.get(6, "http://testserver2/fmstocks/{{LINK_1_3,TickerList.asp}}",
        http.querystring(http.param("ticker", "ter"),
             http.param("company", "")),
```

```
            null, true, "ASCII", "ASCII");
}
endStep();
```

Changes made in the Tree View are automatically updated in the Code View. Changes made in the Code View are automatically updated in the Tree View. The HTTP Module includes an API with HTTP protocol-specific methods. The commands include (but are not limited to) methods to generate GET requests, POST requests, correlation substitutions, validation, etc.

6.5.2 Using Script Variables

OpenScript scripts use variables to pass dynamic data between navigations. The navigation can be in step groups or another code sequence. You can use the Tree view and **Script** menu options to add custom variables to a script or code them manually using the Code view. The HTTP Proxy recorder also defines variables when recording. The following example shows how the HTTP Proxy recorder's default settings record Step Groups, navigations and variables for form inputs and links.

The script is a three page script. the first page is a login page. The second page is the page returned after login which includes links to other pages. The page in the script is the page returned by a click on a link.

Step Group 1 (`beginStep("[1] Stocks", 0);`) gets the page navigation and form input parameters. The `http.solveXpath` method assigns the input values to script variables using XPaths (for example, `".//INPUT[@name='login']/@value"`) to find the input value in the page source.

```
beginStep("[1] Stocks", 0);
{
  http.get(2, "http://testserver2/fmstocks/", null, null, true, "ASCII", "ASCII");
  {
  http.solveXpath("FORMACTION_0", ".//FORM[@name='loginform']/@action",
      "default.asp", 0);
  http.solveXpath("INPUT_0", ".//INPUT[@name='login']/@value", "ta496", 0);
  http.solveXpath("INPUT_1", ".//INPUT[@name='password']/@value", "ta", 0);
  http.solveXpath("INPUT_2", ".//INPUT[@name='LoginButton']/@value", "Login", 0);
  }
}
endStep();
beginStep("[2] Home", 3246);
{
   http.post(4, "http://testserver2/fmstocks/{{FORMACTION_0,default.asp}}",
      null,       http.postdata(http.param("login", "{{INPUT_0,ta496}}"),
      http.param("password", "{{INPUT_1,ta}}"),
      http.param("LoginButton", "{{INPUT_2,Login}}")),
      null, true, "ASCII", "ASCII");
  {
  http.solveXpath("LINK_1_3", ".//A[text()='research a company']/@href",
     "TickerList.asp", 0);
  }
}
endStep();
beginStep("[3] Ticker List", 1703);
{
   http.get(13, "http://testserver2/fmstocks/{{LINK_1_3,TickerList.asp}}", null,
      null, true, "ASCII", "ASCII");
}
endStep();
```

In step Group 2 (`beginStep("[2] Home", 3264);`), the page navigation uses the variables defined in Step Group 1 to pass the data values as parameters to the `http.post` method. The he `http.solveXpath` method assigns links HREF values to variables using XPaths.

```
beginStep("[1] Stocks", 0);
{
   http.get(2, "http://testserver2/fmstocks/", null, null, true, "ASCII",
       "ASCII");
   {
      http.solveXpath("FORMACTION_0", ".//FORM[@name='loginform']/@action",
          "default.asp", 0);
      http.solveXpath("INPUT_0", ".//INPUT[@name='login']/@value", "ta496", 0);
      http.solveXpath("INPUT_1", ".//INPUT[@name='password']/@value", "ta", 0);
      http.solveXpath("INPUT_2", ".//INPUT[@name='LoginButton']/@value",
        "Login", 0);
   }
}
endStep();
beginStep("[2] Home", 3246);
{
  http.post(4, "http://testserver2/fmstocks/{{FORMACTION_0,default.asp}}",
     null,      http.postdata(http.param("login", "{{INPUT_0,ta496}}"),
          http.param("password", "{{INPUT_1,ta}}"),
          http.param("LoginButton", "{{INPUT_2,Login}}")),
          null,  true, "ASCII", "ASCII");
  {
   http.solveXpath("LINK_1_3", ".//A[text()='research a company']/@href",
   "TickerList.asp", 0);
  }
}
endStep();
beginStep("[3] Ticker List", 1703);
{
   http.get(13, "http://testserver2/fmstocks/{{LINK_1_3,TickerList.asp}}",
      null, null, true, "ASCII", "ASCII");
}
endStep();
```

In Step Group 3 (`beginStep("[3] Ticker List", 1703);`), the page navigation uses a variable defined in Step Group 2 to pass the data values as parameters to the `http.get` method.

```
beginStep("[1] Stocks", 0);
{
   http.get(2, "http://testserver2/fmstocks/", null, null, true, "ASCII",
     "ASCII");
   {
     http.solveXpath("FORMACTION_0", ".//FORM[@name='loginform']/@action",
       "default.asp", 0);
     http.solveXpath("INPUT_0", ".//INPUT[@name='login']/@value", "ta496", 0);
     http.solveXpath("INPUT_1", ".//INPUT[@name='password']/@value", "ta", 0);
     http.solveXpath("INPUT_2", ".//INPUT[@name='LoginButton']/@value",
         "Login", 0);
   }
}
endStep();
beginStep("[2] Home", 3246);
{
   http.post(4, "http://testserver2/fmstocks/{{FORMACTION_0,default.asp}}",
      null,      http.postdata(http.param("login", "{{INPUT_0,ta496}}"),
```

```
     http.param("password", "{{INPUT_1,ta}}"),
     http.param("LoginButton", "{{INPUT_2,Login}}")),
        null, true, "ASCII", "ASCII");
  {
   http.solveXpath("LINK_1_3", ".//A[text()='research a company']/@href",
      "TickerList.asp", 0);
  }
}
endStep();
beginStep("[3] Ticker List", 1703);
{
   http.get(13, "http://testserver2/fmstocks/{{LINK_1_3,TickerList.asp}}",
      null, null, true, "ASCII", "ASCII");
}
endStep();
```

6.5.3 Adding a Variable to a Script

The following describes how to add a variable to a script. A regular expression is used to search a specified location for a value to set for the variable.

1. Open or create a script project.
2. Select the script node in which you want to add the variable.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Variable node and select Set Variable.
5. Enter the variable name.
6. Enter the variable value.
7. Click **OK**. The Set: *variableName* = *value* node is added to the script tree.
8. In the Java Code view, the `getVariables().set("varName", "value");` method will be added to the script code.

6.5.4 Adding a Solve XPath to a Script

To add a SolveXpath to a script:

1. Open or create a script project.
2. Select the script node where you want to add the XPath value.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Variable node and select Solve XPath.
5. Enter the variable name.
6. Enter the XPath to use to search for the value for the variable.
7. Enter the Result index value to use for the variable method.
8. Click **OK**. The SolveXpath: *name* node is added to the script tree.
9. In the Java Code view, the `http.solveXpath();` method will be added to the script code:

```
http.solveXpath("var_MyDomVar", ".//INPUT[@name='login']/@value", "ta610", 0);

Example:
http.solveXpath("FORMACTION_0", ".//FORM[@name='loginform']/@action",
"default.asp", 0);
```

```
getLogger().info("Form Name: {{FORMACTION_0}}");
```

6.5.5 Finding a Variable in a Script

To find a variable in a script:

1. Open or create a script.
2. Right-click on a post data or query string parameter containing `{{ }}` syntax and select **Find Variable Source**.

The variable referenced inside the `{{ }}` will be selected in the script tree node.

If more than one variable exists inside the given parameter, OpenScript will display a dialog box from which to pick which variable to find.

6.5.6 Deleting Variables from a Script

Deleting a variable from the tree view causes references to the variable using `{{ }}` notation in any string in the script to be reverted to their recorded values. Variable references in the Java code are also reverted.

To delete a variable from a script:

1. Open or create a script.
2. Right-click on a post data or query string parameter containing `{{ }}` syntax and select **Delete**.

The variable referenced inside the `{{ }}` syntax will be reverted to its recorded value.

To delete similar variables from a script:

1. Open or create a script.
2. Right-click on a post data or query string parameter containing `{{ }}` syntax and select **Delete all *type*_ Variables**. This menu option appears for any variables whose name is prefixed with a word followed by an underscore (_) character.

All variables of *type*_ referenced inside the `{{ }}` syntax will be reverted to their recorded values.

6.5.7 Adding Authentication to a Script

Scripts can support Basic, NTLM, and Digest authentication during script recording. For web sites that require Basic Authentication, the browser opens a login dialog and waits for user input of the username and password. Once the username and password are entered, the browser sends the last request again with the Authentication information in the http request header. OpenScript records an Authentication step automatically with username and encrypted password as the step before the navigate step.

For web sites that require NTLM Authentication, the browser opens a login dialog and waits for user input of the username and password. Once the username and password are entered, the browser sends the last request again with the Authentication information in http request header. However, OpenScript uses a proxy to record the http request and response and cannot get the NTLM username and password information that the user inputs into the browser because the password is not included in the content transfer on the network. For NTLM Authentication, OpenScript opens a second login dialog for entering the username and password again during recording.

OpenScript records an Authentication step automatically with username and encrypted password as the step before the navigate step.

To add authentication to an HTTP script manually:

1. Record an HTTP script as described in "Recording a New HTTP Script" on page 6-12.
2. Select the **Run** node.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Select the **Authentication** node and click **OK**.
5. Enter the URL to access for authentication.
6. Enter a username.
7. Enter the password for the user. Passwords are encrypted using the Base-64 Crypt algorithm.
8. Click **OK** to add the Authentication node to the script tree.
9. In the Java Code view, the Authentication consists of the code executed in the `http.addAuthentication` procedure:

```
http.addAuthentication("http://testserver2", "username",
decrypt("KRT|J|xJPDP"));
```

6.5.8 Adding Text Matching Tests to a Script

You can use Text Matching Tests to report an error and/or abort the script if a single HTTP request does not match the Text Matching Test criteria.

To add a Text Matching Test to an HTTP script:

1. Record an HTTP script as described in "Recording a New HTTP Script" on page 6-12.
2. Expand the **Run** node.
3. Select the HTTP navigation node where you want to add the Text Matching test.
4. Select the **Script** menu and then select **Text Matching Test** from the **Add** menu.
5. Enter a name for the test.
6. Enter the text string or Regular Expression to match or click the Substitute Variable icon to select a databank or script variable to find in the source.

> **Note:** The pound (#) character and double brace ({{ and }}) character sequences need to be escaped with a preceding pound (#) character if used in the Text Matching Test as a literal string (not a string specifying an OpenScript databank or script variable). For example, the pound character should be doubled (##) and double braces should be preceded by a pound character (#{{ and #}}).

7. Select the source location that will be searched for the matching text:
   - **HTML Display Contents:** Search the browser rendered text of the page.
   - **Raw HTML:** Search HTML source of the page.
   - **Response Header:** Search the page Response Header.

8. Select the **Pass when** setting.
   - **Selected text is present:** The test case passes if the **Text to Match** string is found in the selected source.
   - **Selected text is absent:** The test case passes if the **Text to Match** string is not found in the selected source.
9. Select the **Match** type.
   - **Exact**: Matches the **Text to Match** string exactly.
   - **Regular Expression**: Matches using the Regular Expression specified in **Text to Match**.
   - **Wildcard**: Matches using the wildcard characters specified in **Text to Match**.
10. Set the **Verify only, never fail** option.
11. Click **OK**. The Text Matching Test node is added to the script tree.

    In the Java Code view, the `http.assertText` method will be added to the script code if the **Verify only, never fail** option is not selected:

    ```
    http.assertText("MyTextMatchTest", "Home", TextPresence.PassIfPresent,
    MatchOption.Exact)
    ```

    In the Java Code view, the `http.verifyText`method will be added to the script code if the **Verify only, never fail** option is selected:

    ```
    http.verifyText("MyTextMatchTest", "Home", TextPresence.PassIfPresent,
    MatchOption.Exact)
    ```

6.5.9 Adding Server Response Tests to a Script

You can use Server Response Tests to report an error and/or abort the script if a single HTTP request does not return back to the client within a specified time range.

To add a Server Response Test to an HTTP script:

1. Record an HTTP script as described in "Recording a New HTTP Script" on page 6-12.
2. Expand the **Run** node.
3. Select the HTTP navigation node where you want to add the Server Response test.
4. Select the **Script** menu and then select **Other** from the **Add** sub menu.
5. Select **Server Response Test** from the Validation group.
6. Enter a name for the test.
7. Enter the minimum and maximum time values.
8. Enter any error message text to log if the test fails.
9. Set the **Stop Iteration on Failure** option.
10. Click **OK** to add the Server Response node to the script tree.

6.5.10 Substituting Databank Variables

During playback, the parameters in the Web page are filled with values from the Databank file. However, you can substitute a databank variable for a query string parameter in an HTTP script.

1. Record an HTTP script as described in "Recording a New HTTP Script" on page 6-12 that has query string parameters.
2. Expand the **Run** node.
3. Expand the node containing query string parameters.
4. Right-click a query string parameter node and select **Substitute Variable** from the shortcut menu.
5. If you have not already configured the script with a databank, select **Add new databank** and click **Next**.
6. Select the database type and click **Next**.

   For CSV File databanks:

   a. Select the repository and folder then select the databank file to use.
   b. Click **Next**.
   c. Select the column (field name) to substitute for the text parameter and click **Finish**.

   For database databanks:

   a. Select the Database Driver to use and specify the database connection information. The **URL** will be populated with the connection information.
   b. Specify the **Username** and**Password** authentication for the database.
   c. Specify the database **Query** to use to retrieve the databank columns and data from the database.
   d. Specify the alias name to use for the databank.
   e. Click **Test** to verify the connection to the database.
   f. Click **Next**.
   g. Select the column (field name) to substitute for the text parameter and click **Finish**.

In the Tree View, the databank variable appears in place of the recorded value as {{*databankFileName.field,recordedValue*}}.

In the Java Code view, the databank variable appears as {{*databankFileName.field,recordedValue*}} in the `http.querystring(http.param()` parameter of the `http.get` method for scripts recorded using the HTTP record mode:

```
http.get(6, "http://testserver2/fmstocks/TickerList.asp",
     http.querystring(http.param("ticker", "{{db.fmstocks_data.ticker,orcl}}"),
     http.param("company", "")),
     null, true, "ASCII", "ASCII");
```

Or in the `http.postdata(http.param()` parameter of the `http.form().submit()` method for scripts recorded using the Web record mode:

```
http.form(27,"window[@index='0']//form[@action='{{web.link.researchacompany
     ,http://testserver2/fmstocks/TickerList.asp}}']")
     .submit(http.querystring(http.param("ticker",
     "{{db.fmstocks_data.ticker,orcl}}")), null, null, true, null, null, null,
     null, null);
```

6.5.11 Substituting Post Data Variables

To substitute a variable for a Post Data parameter in an HTTP script:

1. Record an HTTP script that has Post Data parameters.
2. Expand the **Run** node.
3. Expand the node containing Post Data parameters.
4. Right-click a Post Data parameter node and select **Substitute Variable** from the shortcut menu.
   - If you have already defined custom variables in the script, select the variable name to substitute for the Post Data parameter and click **Finish**.
   - If you have not already defined custom variables in the script, select **Create new script variable** and click **Next**.

     The Search for Value panel lists the navigation(s) that contain the post data value. If there are more than one navigation that contain the post data value, select the navigation were you want to substitute a variable. When you select a navigation, the data for that navigation appears below with the specific source highlighted along with a suggested Regular Expression.
     - Specify the Regular Expression to use for the substitute variable for the Post Data parameter and click **Next**.
     - Click **Test** to verify the Regular Expression locates the correct data value to substitute.
     - If the Regular Expression locates the correct data value to substitute, click **Next** to continue. If the Regular Expression does not return the correct data value, modify and test the Regular Expression until the desired data value is located and click **Next** to continue.
     - Enter a name for the substitute variable.
     - If you want to add the variable as a variable rule in a correlation library, click **Add to library** and specify the rule information. If not, click **Finish** to insert the substitute script variable into the script

In the Tree View, the script variable appears in place of the recorded value as {{*variableName*,*recordedValue*}}.

In the Java Code view, the script variable appears as *`variableName`*,*`Regular Expression`* parameters in the `http.solve` method:

```
http.solve("MY_VAR", "<INPUT id=login name=login value=\"(.+?)\">", null,
false, Source.Html, 0);
```

6.5.12 Adding a Cookie to a Script

To add a cookie to a script:

1. Open or create a script project.
2. Select the script node where you want to add the cookie.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Cookie node and select **Add Cookie**.

   This dialog box lets you add a cookie to a script.
5. Enter a valid cookie string.

- Cookie String: Specifies the cookie string to add
- Cookie String Charset: Specifies the character set to use for the cookie string

6. Click **OK**. The Cookie node is added to the script tree.
7. In the Java Code view, the `http.addCookie` method will be added to the script code:

```
http.addCookie("cookieString", "charset");

Example:

http.addCookie("username=testCookie", "ASCII");
java.util.List <Cookie> cookies =
http.getBrowser().getCookieJar().getAllCookies();
for (Cookie cookie : cookies) {
   info(cookie.getUrl());
   info(cookie.getCookieString());
   }
```

6.5.13 Removing a Cookie From Script

To remove a cookie from a script:

1. Open or create a script project.
2. Select the script node where you want to place the remove cookie node.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Cookie node and select Remove Cookie.
5. Enter the cookie string to remove.
6. Click **OK**. The Remove Cookie node is added to the script tree.
7. In the Java Code view, the `http.removeCookie` method will be added to the script code:

```
http.removeCookie("cookieString");
```

6.5.14 Adding a User Agent to a Script

To add a user agent to a script:

1. Open or create a script project.
2. Select the script node where you want to add the user agent.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the HTTP node and select **Set User Agent**.
5. Enter the user agent details.
6. Click **OK**. The user agent node is added to the script tree.
7. In the Java Code view, the *`http.setUserAgent("agent");`* method will be added to the script code:

```
http.setUserAgent("Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET
CLR 1.1.4322; InfoPath.1; .NET CLR 2.0.50727)");
```

6.6 Adding Navigation

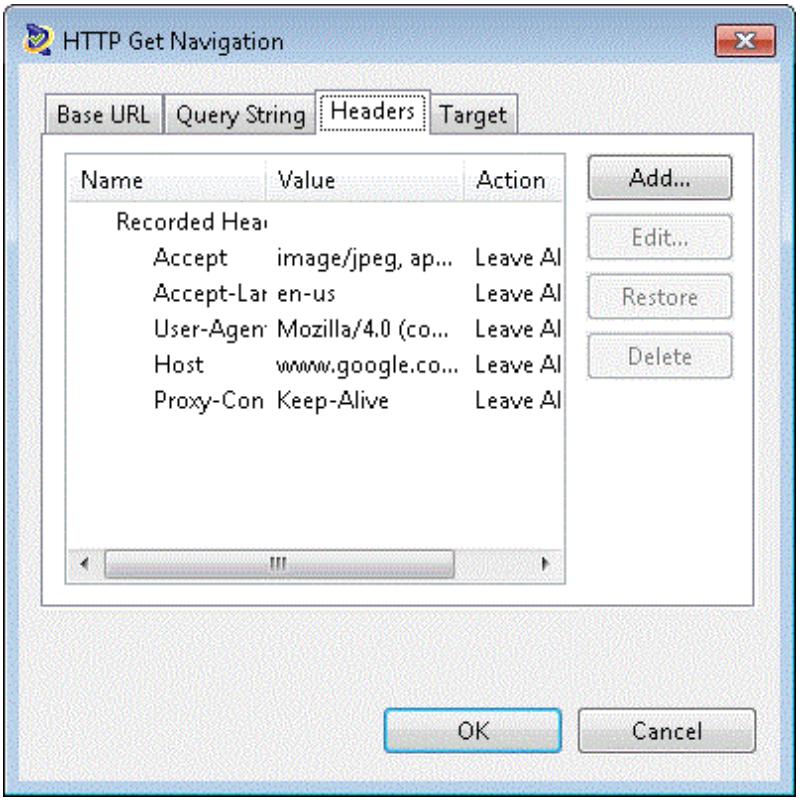
The HTTP Module allows you to view and edit all recorded navigations and related parameters (headers, post data, etc.) in either the script GUI Tree view or the Code View. It also enables you to view and edit any default correlation/parameterization of dynamic navigations and apply your own correlation to handle dynamic navigations.

6.6.1 Understanding Navigation Editing (Correlation)

You can use the Navigation Editing GUI View to configure the navigations you want to parameterize and the correlation rules you want to apply. The GUI allows viewing and editing properties for different types of navigations (for both Web/HTTP applications and non-Web/HTTP applications) and data inputs.

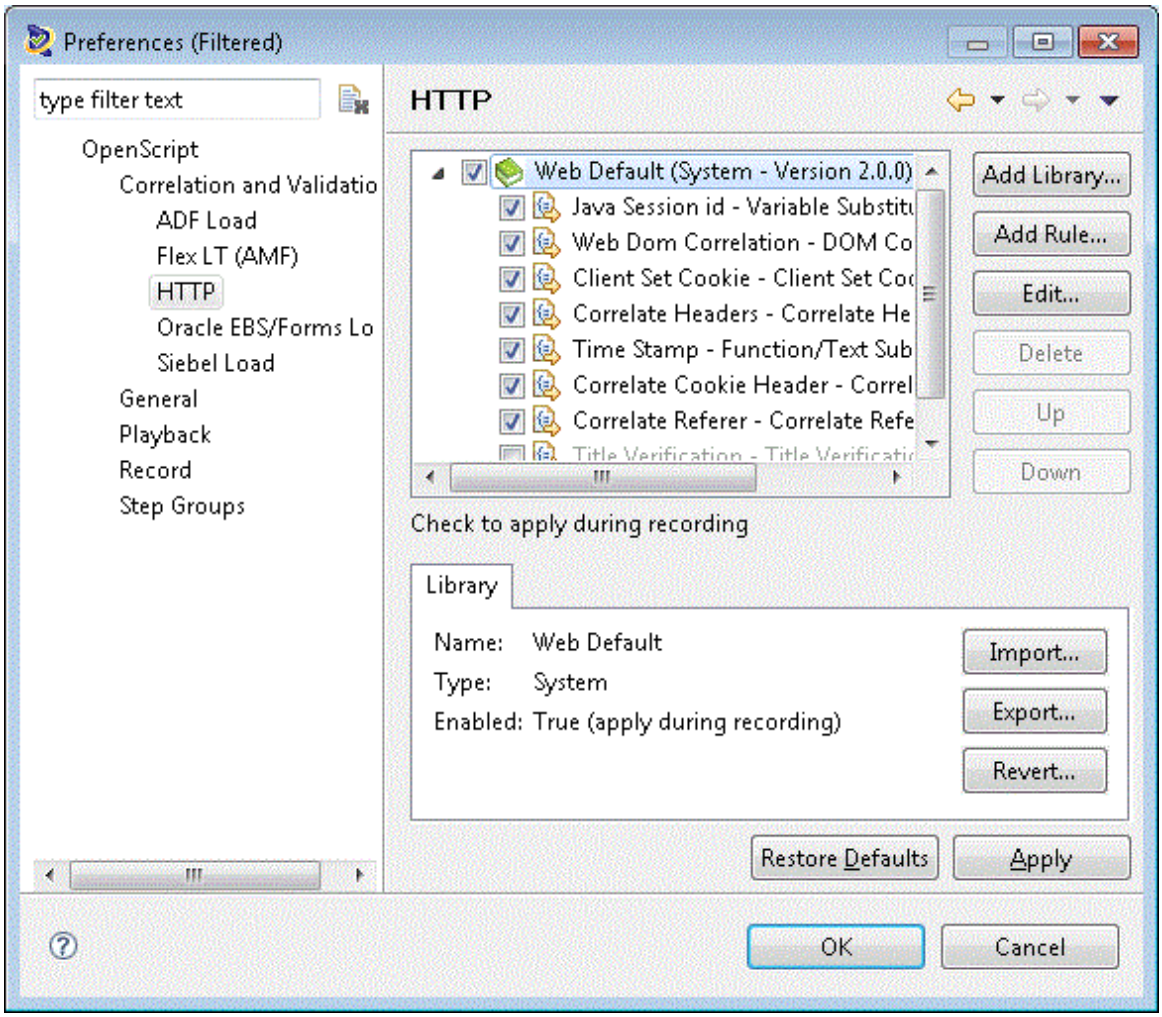
The Display & Editing for Recorded Navigations view includes recorded navigations and any navigation parameters like headers, etc. It also provides a mechanism for users to edit/add/delete navigations including dynamic parameter sources/targets.

Figure 6–2 Display & Editing for Recorded Navigations view



The Display & Editing for Correlation Rules Library & Editing view includes a list of all default correlation rules included in the module(s), listed by application type. It also provides a mechanism for adding, editing or deleting correlation rules.

Figure 6–3 The Display & Editing for Correlation Rules Library & Editing view



You can specify the navigations you want to parameterize and the data source you want to drive the inputs from through navigation editing commands in the code view. These commands map to the navigation editing GUI view.

```
beginStep("[2] Home", 3266);
{
   http.post(4, "http://testserver2/fmstocks/{{FORMACTION_0,default.asp}}",
        null,         http.postdata(http.param("login", "{{INPUT_0,ta616}}"),
             http.param("password", "{{INPUT_1,ta}}"),
             http.param("LoginButton", "{{INPUT_2,Login}}")),
             null, true, "ASCII", "ASCII");
             {
             http.solveXpath("LINK_1_3", ".//A[text()='research a
                  company']/@href", "TickerList.asp", 0);
             http.solveXpath("LINK_1_2", ".//A[text()='Logout']/@href",
                  "logout.asp", 0);
             }
}
endStep();
```

6.6.2 Adding HTTP Get Navigation

To add an HTTP Get Navigation to an HTTP script:

1. Record an HTTP script as described in "Recording a New HTTP Script" on page 6-12.
2. Select the **Run** node.
3. Select the **Script** menu, then select **HTTP Get Navigation** from the **Add** sub menu.
4. On the **Base URL** tab, specify the following:

   Path: The base URL path to use for the navigation.

   Request charset: The character set to use for the request.

   Response charset: The character set to use for the response.

   Encode strings: When selected, control and special characters in string are encoded to the Character entity references. When cleared, control and special characters in string are not encoded.
5. On the **Query String** tab, use the **Add** button to add the requested name/value pairs to the Base URL. Note that you can use the Up and Down buttons to move the selected query string parameter up or down one place in the search order.
6. On the **Headers** tab, use the **Add** button to add name/value pairs and actions to the Base URL.
7. Click **OK** to add the HTTP Get Navigation node to the script tree.
8. In the Java Code view, the HTTP Get Navigation consists of the code executed in the `http.Get` method (line breaks and spacing added for clarity):

```
http.Get(1, "http://testserver2",
       http.querystring(http.param("QueryString1", "QueryValue1"),
            http.param("QueryString2", "QueryValue2"),
                   http.param("QueryString3", "QueryValue3")),
       http.headers(http.header("HeaderString1", "HeaderValue1NoActions",
                    Header.HeaderAction.Add),
       http.header("HeaderString2", "HeaderValue2SetifNotSet",
                    Header.HeaderAction.SetIfNotSet),
       http.header("HeaderString3", "HeaderValue3ApplytoAll",
                    Header.HeaderAction.GlobalAdd),
       http.header("HeaderString4", "HeaderValue4BothActions",
                    Header.HeaderAction.GlobalSetIfNotSet)),
       false, "ASCII", "ASCII");
```

6.6.3 Adding HTTP Post Navigation

To add an HTTP Post Navigation to an HTTP script:

1. Record an HTTP script as described in "Recording a New HTTP Script" on page 6-12.
2. Select the **Run** node.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Select the HTTP Post Navigation node and click **OK**.
5. On the **Base URL** tab, specify the following:

   Path: The base URL path to use for the navigation.

   Request charset: The character set to use for the request.

Encode strings: When selected, control and special characters in string are encoded to the Character entity references. When cleared, control and special characters in string are not encoded.

6. On the **Query String** tab, use the **Add** button to add the requested name/value pairs to the Base URL. Note that you can use the Up and Down buttons to move the selected query string parameter up or down one place in the search order.
7. On the **Post Data** tab, use the **Add** button to add name/value pairs to the Base URL.
8. On the **Headers** tab, use the **Add** button to add name/value pairs and actions to the Base URL.
9. Click **OK** to add the HTTP Post Navigation node to the script tree.
10. In the Java Code view, the HTTP Post Navigation consists of the code executed in the `http.Post` method (line breaks and spacing added for clarity):

```
http.Post(12, "http://testserver2",
     http.querystring(http.param("QueryString1", "QueryValue1"),
                 http.param("QueryString2", "QueryValue2"),
                 http.param("QueryString3", "QueryValue3")),
     http.postdata(param("PostString1", "PostValue1"),
                 http.param("PostString2", "PostValue2"),
                 http.param("PostString3", "PostValue3")),
     http.headers(http.header("HeaderString1", "HeaderValue1NoActions",
                        Header.HeaderAction.Add),
                 http.header("HeaderString2", "HeaderValue2SetifNotSet",
                        Header.HeaderAction.SetIfNotSet),
                 http.header("HeaderString3", "HeaderValue3ApplytoAll",
                        Header.HeaderAction.GlobalAdd),
                 http.header("HeaderString4", "HeaderValue4BothActions",
                        Header.HeaderAction.GlobalSetIfNotSet)),
     true, "ASCII", "ASCII");
```

6.6.4 Adding an HTTP Multipart Post Navigation

To add an HTTP Multipart Post Navigation to an HTTP script:

1. Record an HTTP script as described in "Recording a New HTTP Script" on page 6-12.
2. Select the **Run** node.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the HTTP Multipart Navigation node and select the Multipart Post Navigation node and click **OK**.
5. On the **Base URL** tab, enter the following:

   Path: The base URL path to use for the navigation.

   Boundary: Specify the boundary to use to identify parts of a multipart form input navigation (file navigations). The boundary is used within the Content-Type: multipart/form-data; boundary="" response sent by the user agent. The specified boundary should not occur in any of the file data.

   Request charset: The character set to use for the request.

   Response charset: The character set to use for the response.

Encode strings: Select this option so that control and special characters in string are encoded to the Character entity references. When cleared, control and special characters in string are not encoded

6. On the **Query String** tab, use the **Add** button to add name/value pairs to the Base URL.
7. On the **Post Data** tab, use the **Add** button to add postdata to the Base URL. You can specify standard Postdata name/value pairs or postdata files. If you select **File**, specify the path, filename, and content type for each postdata file parameter.
8. On the **Headers** tab, use the **Add** button to add name/value pairs and actions to the Base URL.
9. Click **OK** to add the HTTP Multipart Post Navigation node to the script tree.
10. In the Java Code view, the HTTP Multipart Post Navigation consists of the code executed in the `http.multipartPost` method (line breaks and spacing added for clarity):

```
http.multipartPost(13, "http://testserver2",
        http.querystring(http.param("QueryString1", "QueryValue1"),
                    http.param("QueryString2", "QueryValue2"),
                    http.param("QueryString3", "QueryValue3")),
        http.postdata(http.param("PostDataString1", "PostDataValue1Standard"),
                    http.param("PostDataString2", "PostDataValue2FilePath",
                          "PostDataValue2FileName", "ASCII"),
                    http.param("PostDataString3",
                      "C:\\Oracle\\OFT\\DataBank\\fmstocks_data.csv",
                      "C:\\Oracle\\OFT\\DataBank\\fmstocks_data.csv", "CSV")),
        http.headers(http.header("HeaderString1", "HeaderValue1NoActions",
                          Header.HeaderAction.Add),
                    http.header("HeaderString2", "HeaderValue2IfNotSet",
                          Header.HeaderAction.SetIfNotSet),
                    http.header("HeaderString3", "HeaderValue3ApplytoAll",
                          Header.HeaderAction.GlobalAdd),
                    http.header("HeaderString4", "HeaderValue4Both",
                          Header.HeaderAction.GlobalSetIfNotSet)),
        "boundary", false, "ASCII", "ASCII");
```

6.6.5 Adding an HTTP XML Post Navigation

To add an HTTP XML Post Navigation to an HTTP script:

1. Record an HTTP script as described in "Recording a New HTTP Script" on page 6-12.
2. Select the **Run** node.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the XML Post Navigation node and select the XML Post Navigation node and click **OK**.
5. On the **Base URL** tab, specify the following:

   Path: The base URL path to use for the navigation.

   Request charset: The character set to use for the request.

   Response charset: The character set to use for the response.

Encode strings: When selected, control and special characters in string are encoded to the Character entity references. When cleared, control and special characters in string are not encoded.

6. On the **Query String** tab, use the **Add** button to add the requested name/value pairs to the Base URL. Note that you can use the Up and Down buttons to move the selected query string parameter up or down one place in the search order.
7. On the **Post Data** tab, enter the XML post data to add to the Base URL.
8. On the **Headers** tab, use the **Add** button to add name/value pairs and actions to the Base URL.
9. Click **OK** to add the HTTP XML Post Navigation node to the script tree.
10. In the Java Code view, the HTTP XML Post Navigation consists of the code executed in the `http.xmlPost` method (line breaks and spacing added for clarity):

```
http.xmlPost(0, "http://xmltest2",
        http.querystring(http.param("xmlQueryString1", "xmlQueryValue1"),
                  http.param("xmlQueryString2", "xmlQueryValue2"),
                  http.param("xmlQueryString3", "xmlQueryValue3")),
                        "xmlPostDataString",
        http.headers(http.header("xmlHeaderString1", "xmlHeaderValue1NoAction",
                           Header.HeaderAction.Add),
                  http.header("xmlHeaderString2", "xmlHeaderValue2IfNotSet",
                           Header.HeaderAction.SetIfNotSet),
                  http.header("xmlHeaderString3",
                    "xmlHeaderValue3ApplytoAll", Header.HeaderAction.GlobalAdd),
                  http.header("xmlHeaderString4", "xmlHeaderValue4Both",
                           Header.HeaderAction.GlobalSetIfNotSet)),
        false, "ASCII", "ASCII");
```

6.6.6 Using the HTTP Module API

The Web Functional Module includes a script Application Programming Interface (API) specific to Web HTTP protocol testing. The HTTP Module recorder creates the Java code that corresponds to the Tree View and displays the HTTP commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the HTTP API to enhance recorded scripts with additional testing functionality. Commands that are specific to the HTTP Module are part of the "http" class. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the HTTP Module API include:

- Adding authentication
- Adding and removing cookies
- Adding HTTP navigation (Get and Post)
- Adding HTTP Multipart Post navigation
- Adding HTTP Multipart name/value pairs
- Adding name/value pairs
- Adding XML Post navigation

- Setting the user agent

Many API methods can be added using the HTTP Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures and code examples. See the API Reference in the OpenScript help for additional programming information.

7

Using the Oracle EBS/Forms Functional Test Module

This chapter provides instructions on configuring and using the OpenScript Oracle EBS/Forms Functional Test Module, which provides support for functional testing of Oracle EBS/Forms web applications.

7.1 About the Oracle EBS/Forms Functional Test Module

The Oracle EBS/Forms Functional Test Module provides support for functional testing of Oracle EBS/Forms web applications. The Oracle EBS/Forms Functional Test Module is an extension to the Web Functional Test Module.

The Oracle EBS/Forms Functional Test Module is an extension module to the OpenScript Web Functional Test Module that extends the Web testing with Oracle EBS/Forms Functional Test recording and playback capabilities. The Oracle EBS/Forms Functional Test Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager.

The Oracle EBS/Forms Functional Test recorder displays commands in the Tree View in easy-to-understand commands. By default, script commands are grouped into Steps Groups by the Web page on which they were performed. Each Step Group contains one or more script commands corresponding to recorded actions that were performed on the page. The default name for the Step Group is the Web page Title (as specified in the "Title" tag).

OpenScript shows the results of Oracle EBS/Forms Functional Test script playback in the Results view. The Results view shows results for each script command (including duration and summary for failures). The Results Report compiles the same information into an HTML Results Report. Results can be exported from the OpenScript GUI in standard format (CSV / HTML). Results are also generated for unattended playback through the command line.

The Oracle EBS/Forms Functional Test Module API includes a "forms" class that provides additional programming functionality.

7.1.1 Key Features of the Oracle EBS/Forms Functional Test Module

- Supports Oracle E-Business Suite Release 12 (Forms 10g) running on Sun JRE and E-Business Suite Release 11i (Forms 6i) running on Jinitiator.
- Records Forms actions in the applet.

- Plays back recorded Forms actions/commands which consist of an event plus object identified by its attributes (for example: `forms.textField(28, "//forms:textField[(@name='DIST_LIST_NAME_0')]").input("LOREM IPSUM")`).
- Provides full script code view integration to support script generation for the Oracle EBS/Forms Functional Test Module. The Oracle EBS/Forms Functional Test Module includes an additional API to support Oracle EBS/Forms Functional Test protocol code scripting.
- Allows users to parameterize user inputs to Oracle EBS/Forms Functional Test scripts and drive those inputs from an external data file (Databank).
- Allows users to insert Tests to validate Oracle EBS/Forms content on playback.
- Provides additional automation of all Oracle EBS/Forms GUI components using options/settings that are specific to Oracle EBS/Forms Functional Test scripts within the Oracle EBS/Forms Functional Test categories in the preferences interface.
- Reports playback results for Oracle EBS/Forms Functional Test scripts in the Results and Console views.
- The Oracle EBS/Forms Functional Test Script Module API. The Oracle EBS/Forms Functional Test Application Programming Interface include Java code methods specific to functional testing of Oracle EBS/Forms applications.

The New Project wizard (Select **New** from the **File** menu) includes an "Oracle EBS/Forms " option in the Functional Test group to use when creating Oracle EBS/Forms functional testing projects in OpenScript. The Oracle EBS/Forms Functional Test Script Module records Oracle EBS/Forms applications using Object Identification. OpenScript captures user actions and records them to the OpenScript script nodes in a highly readable sequence of navigations and actions.

7.1.2 Prerequisites

The Oracle EBS/Forms Functional Test Module recorder has the following prerequisites:

- Before recording any script in Forms Functional Test Module, you must run the Forms/EBS application at least once before attempting to record a script with OpenScript on that machine. This ensures that required JRE/Jinitiator has been installed and also verifies that forms applications can run successfully on that machine inside of Internet Explorer.

7.2 Recording Oracle EBS/Forms Functional Tests

The Oracle EBS/Forms Functional Test Module records standard Oracle EBS/Forms components for Oracle E-Business Suite Release 12 (Forms 10g) running on Sun JRE and E-Business Suite Release 11i (Forms 6i). The Recorder creates functional and regression test scripts for automating testing of Oracle EBS/Forms applications.

Oracle EBS/Forms are applet based controls and the Oracle EBS/Forms Functional Test Module provides the object/attribute information for OpenScript to record interactions with those controls. Actions will be captured in the test script as OpenScript "forms" commands. For event-driven actions on Forms windows that do not trigger Forms messages, AWT event actions will be captured in the test script as OpenScript "applet" commands. Other components are standard Web controls which are captured as standard OpenScript "web" commands using Web Functional Test

object attributes. Object Identification attributes can later be modified by users through the Preferences global settings for new scripts or for already recorded commands in the tree view or code view. Recording can be configured through Internet Explorer only as the Oracle EBS/Forms Functional Test Module does not support Firefox.

The Oracle EBS/Forms Functional Test Module provides a record toolbar button that lets you initiate the Oracle EBS/Forms recorder and capture Web/Oracle EBS/Forms page actions to the script view. The record toolbar includes start and stop recording toolbar buttons. OpenScript recorders also open a floating toolbar that can be used while recording without having to switch between the browser and OpenScript.

7.2.1 Setting Oracle EBS/Forms Functional Test Record Preferences

To set Oracle EBS/Forms Functional Test record preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Record category.
4. Select **Oracle EBS/Forms Functional**.
5. Click the tabs and set the preferences. See Section 2.5.4, "Oracle EBS/Forms Functional Test Preferences" for descriptions of the Record Preferences settings.
6. Click **OK**.

7.2.2 Adding/Editing Object Identifiers

The Oracle EBS/Forms Functional Test Module uses object identification to specify attributes used to identify Oracle EBS/Forms objects. The Oracle EBS/Forms Functional Test Module uses the same predefined path attributes for common Web objects as the Web Functional Test Module; however, Oracle EBS/Forms Test Automation provides additional attributes to identify forms controls. Object paths are specified in XPath format. For example the object identification path appears as follows in Java code commands:

```
//forms:textField[(@name='DIST_LIST_NAME_0')]
```

You can set the default Web object attributes in the Oracle EBS/Forms Functional Test Module Record Preferences. You can also edit object attributes in recorded scripts in the tree view or the code view.

In addition to the predefined object identification, you can add an Object Library to the script to record paths into a library file. Object Library files may be shared and reused across other scripts. The Object Library files provide a more convenient "short name" for objects to provide for more convenient programming.

The Oracle EBS/Forms Functional Test Module includes object identifiers that specify how the recorder identifies Browser objects. You can add object identifiers or edit the existing object identifiers in the Record preferences.

To add or edit an object identifier:

1. Select the **OpenScript Preferences** from the **View** menu.
2. Expand the Record node and select Forms Functional Test.
3. Click the **Object Identification** tab. This tab lets you specify the Oracle EBS/Forms object identification attributes, as follows:

Active Profile: Specifies which object identification profile to use as the active profile. Profiles define a specific set of object identifiers to use when recording EBS/Forms functional tests. Use the **Add Profile** option to create a new custom profile.

Name: Shows the name(s) of the defined Oracle EBS/Forms object identifiers.

Attributes: Shows the pattern(s) specified for the defined Oracle EBS/Forms object identifiers.

Add Profile: Opens a dialog box for specifying a new Oracle EBS/Forms object identifier profile.

Add Object: Opens a dialog box for specifying a new Oracle EBS/Forms object identifier.

Edit: Opens a dialog box for editing the selected Oracle EBS/Forms object identifier.

Delete: Deletes the selected Oracle Forms object identifier.

Export: Opens a dialog box for exporting the currently selected Forms object identifier profile to an XML file.

Import: Opens a dialog box for importing a saved object identifier profile XML file.

Revert: Reverts the default EBS/Forms object identification profile to the default profile. Any change to the default profile are removed.

For each object element, you specify a name (typically an Oracle Forms object attribute), an operator, a value and a value type. As you add object elements, OpenScript builds the object identifier using logical OR between each object identifier element. Click **Edit** to change between logical OR and AND.

4. Click **Add** or select an existing object identifier and click **Edit**.
5. If adding a new object identifier, enter a name for the object identifier.
6. Add or edit object elements for the object identifier.

   See the Web Functional Test Module for additional information about adding and editing Object Identifiers.
7. Click **OK**. The object identifier is added to the record preferences.

7.2.3 Recording Oracle EBS/Forms Functional Test Scripts

To record Oracle EBS/Forms Functional Test scripts:

1. Start OpenScript.
2. Set the Oracle EBS/Forms Functional Test Recording preferences.
3. Select **New** from the **File** menu.
4. Expand the Functional Testing group.
5. Select **Oracle EBS/Forms** (The Oracle EBS/Forms script combines both Web and Oracle EBS/Forms technologies as part of the same script).
6. Click **Next**.
7. Select the Repository and Workspace.
8. Enter a script name.

9. Click **Finish**. A new Script tree is created in the Script View.
10. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
11. Load the web page where you want to start recording into the browser.
12. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
13. When finished navigating pages, close the browser.
14. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
15. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

> **Note:** Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

7.2.4 Event-Driven Recording

For event-driven actions on Forms windows that do not trigger Forms messages, AWT event actions will be captured in the test script as OpenScript "applet" commands. The following components and actions are supported on JRE 1.5 or later version:

Component	Actions	XPath Identifier
DTree	selectNode, collapseNode, expandNode, activateNode	TJavaDTree
ExpansionTrackingTree	selectNode, extendToNode, toggleNode, collapseNode, expandNode, activateNode, selectPopupMenu	TJavaExpansionTree
Grid	selectCell, focusCell, editCell	TJavaGrid
InfiniteScrollbar	scroll	TJavaInfiniteScrollBar
LWCheckBox	setCheck, isChecked	TJavaCheckbox
LWChoice	selectItem, selectItemByIndex	TJavaCombobox
LWList	select	TJavaList
LWMenu	menuClick	TJavaAppWindow
LWTextArea	replaceSelection, setSelectedRange	TJavaTextView
LWTextField	setText, setSelectedRange	TJavaTextfield
Ocgle Dcm layout edit	selectItem, deselectAll, move, resize, doubleClickItem, rangeSelect, createObject, deleteSelected	
PushButton	buttonClick	TJavaButton

Component	Actions	XPath Identifier
SchedulerWBTimeBrowser	resize, move, selectPopupMenu	TJavaTimeBrowser
SchedulerWBTreeBrowser	collapseNode, expandNode, selectNode, doubleClickNode, selectPopupMenu	TJavaTreeBrowser
TabBar	select	TJavaTabBar
Timeline	selectPopupMenu	TJavaTimeline
ToolBar	clickItemByIndex, clickItemByLabel	TJavaToolBar

Script commands for event-driven actions appear in the script in the following format:

```
applet.object(objectId).action(parameters);
```

The *objectId* includes the recorded ID and the XPath identifier. The recorded ID is a sequence number generated during recording. The XPath identifier for most components a 2-level path that identifies the window and component with each level starting with `/applet:`, as follows:

```
"/applet:TJavaWindow[@identifier]/applet:XPathIdentifier[@identifier]")
```

The *@identifier* is the component text or position index of the specific window or control. The position index is 0-based by component type starting with first of the type to appear in source. The *XPathIdentifier* is specific to the component type. For example, the *XPathIdentifier* for a Pushbutton is `TJavaButton`, as follows:

```
"/applet:TJavaWindow[@text='Editor' or @posIndex='3']
   /applet:TJavaButton[@text='Search' or @posIndex='2']")
```

A full command with method and action appears as follows:

```
applet.button(54,"/applet:TJavaWindow[@text='Editor'and @posIndex='3']
   /applet:TJavaButton[@text='Search' and @posIndex='2']")
   .buttonClick();
```

The action specifies any required parameters, as follows:

```
applet.comboBox(54,"/applet:TJavaWindow[@text='Editor'and @posIndex='3']
   /applet:TJavaCombobox[@posIndex='2']")
   .selectItem("item");
```

The XPath for menu actions is a single-level path that identifies the window, as follows:

```
"/applet:TJavaAppWindow[@text='Oracle Applications - Vision' or @posIndex='0']")
```

The menuClick action specifies the menu item to click as a parameter, as follows:

```
applet.appWindow(103,"/applet:TJavaAppWindow[@text='Oracle Applications - Vision'
      or @posIndex='0']")
   .menuClick("Window|1 Navigator - CRL 11i Projects");
```

Use Ctrl-space in the Java Code view to open an Intellisense window listing action parameters.

7.3 Playing Back Scripts

OpenScript plays back recorded Oracle EBS/Forms actions/commands which consist of an event plus an object identified by its attributes (for example: `forms.textField(28, "//forms:textField[(@name='DIST_LIST_NAME_0')]").input("LOREM IPSUM")`). The actions used for playback will either be those that are recorded or specified manually in the Java Code view. Playback can be configured through IE only as the Oracle EBS/Forms Functional Test Module does not support Firefox. Unattended playback is supported through Oracle Test Manager or third-party tools using OpenScript's command line interface. Oracle EBS/Forms Functional Test scripts do not play in Oracle Load Testing.

The Oracle EBS/Forms Functional Test Module provides playback and iterate toolbar buttons that allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for Oracle EBS/Forms Functional scripts can be viewed in the Results and Console views.

7.3.1 Setting Oracle EBS/Forms Functional Test Playback Preferences

To set Oracle EBS/Forms Functional Test Playback preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Playback category.
4. Select **Oracle EBS/Forms Functional Test**.
5. Expand the groups and set the preference. See Section 2.4.4, "Oracle EBS/Forms Functional Test Preferences" for descriptions of the Playback Preferences settings.
6. Click **OK**.

7.3.2 Playing Back Oracle EBS/Forms Functional Scripts

To play back Oracle EBS/Forms Functional scripts:

1. Start OpenScript.
2. Open the Oracle EBS/Forms Functional script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

7.3.3 Playing Back Oracle EBS/Forms Functional Scripts with Iterations

To play back Oracle EBS/Forms Functional scripts with iterations:

1. Start OpenScript.
2. Open the Oracle EBS/Forms Functional script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.
5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.

7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

7.4 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

7.4.1 Adding Forms Actions

The Oracle EBS/Forms Module includes actions for Oracle EBS/Forms objects that can be added to a script.

To add Forms actions to a script:

1. Record a EBS/Forms Functional Test script.
2. Select the script node where you want to add the action.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Forms Action node.
5. Expand an action node and select the action.
6. Click **OK**.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path.
8. Enter any required values to use for the object action.
9. Click **OK**. The action node is added to the script tree.

   In the Java Code view, a `forms.`*`object`*`(`*`objectId`*`).`*`action`*`(`*`parameters`*`)` method will be added to the script code:

   ```
   forms.textField(27, "//forms:textField[(@name='DIST_LIST_APPLICATION_
   0')]").openDialog();
   ```

   The Forms Action node includes actions for objects such as BlockScroller, Button, Calendar, CheckBox, ChoiceBox, EditBox, EditorDialog, FlexWindow, HelpDialog, InfoBox, List, ListOfValues, Notification, Calculator, RadioGroup, ResponseBox, SchedulingDataClient, SpreadTable, StatusBar, TabbedRegion, TextField, Tree, TreeList. Other object actions have corresponding Java code methods.

7.4.2 Using the Oracle EBS/Forms Functional Test Module API

The Oracle EBS/Forms Functional Test Module includes a script Application Programming Interface (API) specific to Oracle EBS/Forms functional testing. The Oracle EBS/Forms Functional Test Module recorder creates the Java code that corresponds to the Tree View and displays the Oracle EBS/Forms Functional Test commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the Oracle EBS/Forms Functional Test API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Oracle EBS/Forms Functional Testing Module are part of the "forms" class. For event-driven actions on Forms windows that do not trigger Forms messages, AWT event actions will be captured in the test script as OpenScript "applet" commands. Additional functional test methods are available in the "web" and "ft" classes. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the Oracle EBS/Forms Testing Module API include:

- Forms Action
- Button
- Calendar
- Checkbox
- Choice Box
- Edit Box
- Editor Dialog
- Flex Window
- Form Window
- HelpDialog
- Info Box
- List
- ListOfValues
- Notification
- RadioGroup
- ResponseBox
- SchedulingDataClient
- SpreadTable
- StatusBar
- TabbedRegion
- TextField
- Tree
- TreeList
- OtsHGridTable

Many API methods can be added using the Oracle EBS/Forms Functional Test Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures. See the API Reference in the OpenScript help for additional programming information.

8

Using the Oracle EBS/Forms Load Test Module

This chapter provides instructions on configuring and using the OpenScript Oracle EBS/Forms Load Test Module, which provides support for load testing of Oracle EBS/Forms web applications.

8.1 About the Oracle EBS/Forms Load Test Module

The Oracle EBS/Forms Load Test Module is an extension module to the OpenScript HTTP Module that extends the Web testing with Oracle EBS/Forms Load Test recording and playback capabilities. The Oracle EBS/Forms Load Test Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager.

The Oracle EBS/Forms Load Test recorder displays commands in the Tree View in easy-to-understand commands. By default, script commands are grouped into Steps Groups by the Web page on which they were performed. Each Step Group contains one or more script commands corresponding to recorded actions that were performed on the page. The default name for the Step Group is the Web page Title (as specified in the "Title" tag).

OpenScript shows the results of Oracle EBS/Forms Load Test script playback in the Results view. The Results view shows results for each script command (including duration and summary for failures). The Results Report compiles the same information into an HTML Results Report. Results can be exported from the OpenScript GUI in standard format (CSV / HTML). Results are also generated for unattended playback through the command line.

The Oracle EBS/Forms Load Test Module API includes a "nca" class that provides additional programming functionality.

8.1.1 Key Features of the Oracle EBS/Forms Load Test Module

- The Oracle EBS/Forms Load Test Script Module. The New Project wizard (**New** from the **File** menu) includes an "Oracle EBS/Forms" option in the Load Test Group to use when creating Oracle EBS/Forms load testing projects in OpenScript. The Oracle EBS/Forms Load Test Script Module records Oracle EBS/Forms applications at the protocol level. OpenScript captures user actions and records them to the OpenScript script based upon HTTP requests and post data or query strings.

- Correlation Library. The Oracle EBS/Forms Load Test Module includes an Oracle EBS/Forms-specific library of correlation rules for parameterizing scripts.
- Test Cases (Validation). The Oracle EBS/Forms Load Test Module includes a Status Bar test for validating validate Oracle EBS/Forms application content on playback.
- Oracle EBS/Forms-Specific Application Programming Interface (API). The Oracle EBS/Forms Load Test Module includes a Oracle EBS/Forms Load Test Module API Specification that can be used to customize Oracle EBS/Forms-specific scripts.

8.1.2 Prerequisites

The Oracle EBS/Forms Load Test Module recorder has the following prerequisites:

- Before recording any script in Forms Functional Test Module, you must run the Forms/EBS application at least once before attempting to record a script with OpenScript on that machine. This ensures that required JRE/Jinitiator has been installed and also verifies that forms applications can run successfully on that machine inside of Internet Explorer.
- You must set **Use Browser Settings** (selected) in the control panel proxy for *all* installed JVMs. Depending upon the Java version, the **Use Browser Settings** is either here:
  - Control Panel - Select JInitiator/Sun Java - Proxies tab

    or here:
  - Control Panel - Select Java - General tab - Network Settings
- If you want to report Forms End-User Performance Monitoring (EUM) metrics in Oracle Load Testing graphs, you must enable End User Monitoring metrics on the EBS R12 system being monitored. Go to the OAM configuration page for EBS and add the following two parameters to the desired configuration section in the EBS appsweb.cfg file:
  - `EndUserMonitoringEnabled`: Set to "true" to enable Chronos logging. The default value is NULL.
  - `EndUserMonitoringURL`: Specify the default value of the URL for the Web Listener associated with the Oracle Applications Web Server. This can be amended to an alternative URL which points to a WebCache or HTTP Server instance which may be located on another machine.

    Sample URL format:

    `http://<hostname>:<portnumber>/oracle_smp_chronos/oracle_smp_chronos_sdk.gif.`

  See the Installation and Configuration section of the *Oracle Real User Experience Insight Accelerator for Oracle E-Business Suite Guide* for additional information.

  Notes:

  Values are only reported when EUM metrics are enabled on the Forms server.

  Values are only reported in Oracle Load Testing if you set **Generate Timers for All Resources** to TRUE in your Oracle Load Testing script scenario preferences.

  Values are only reported to Oracle Load Testing graphs. They do not appear in the Oracle Load Testing session report.

8.2 Recording Oracle EBS/Forms Load Tests

The Oracle EBS/Forms Load Test Module records standard Oracle EBS/Forms components for Oracle E-Business Suite Release 12 (Forms 10g) running on Sun JRE and E-Business Suite Release 11i (Forms 6i). The Recorder creates load test scripts for automating testing of Oracle EBS/Forms applications.

Oracle EBS/Forms are applet based controls and the Oracle EBS/Forms Load Test Module provides the object/attribute information for OpenScript to record interactions with those controls. Actions will be captured in the test script as OpenScript "nca" commands. Other components are standard Web controls which are captured as standard OpenScript "http" navigation commands. Correlation rules can be modified by users through the Preferences settings for new scripts.

The Oracle EBS/Forms Load Test Module provides a record toolbar button that lets you initiate the Oracle EBS/Forms recorder and capture Web/Oracle EBS/Forms page actions to the script view. The record toolbar includes start and stop recording toolbar buttons. OpenScript recorders also open a floating toolbars that can be used while recording without having to switch between the browser and OpenScript.

Oracle EBS/Forms scripts supporting recording and playing back Forms applications that communicate over HTTP, HTTPS, and Forms Socket mode.

8.2.1 Setting Oracle EBS/Forms Load Test Record Preferences

To set Oracle EBS/Forms Load Test record preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Record category.
4. Select **Oracle EBS/ Forms Load Test**.
5. Set the parameters. See Section 2.5.5, "Oracle EBS/Forms Load Test Preferences" for descriptions of the Record Preferences settings.
6. Click **OK**.

8.2.2 Recording Oracle EBS/Forms Load Test Scripts

To record Oracle EBS/Forms Load Test scripts:

1. Start OpenScript.
2. Set the Oracle EBS/Forms Load Test Correlation preferences.
3. Set the Oracle EBS/Forms Load Test Recording preferences.
4. Select **New** from the **File** menu.
5. Expand the Load Testing group.
6. Select **Oracle EBS/Forms** (The Oracle EBS/Forms script combines both HTTP and Oracle EBS/Forms technologies as part of the same script).
7. Click **Next**.
8. Select the Repository and Workspace.
9. Enter a script name.
10. Click **Finish**. A new Script tree is created in the Script View.

11. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
12. Load the web page where you want to start recording into the browser.
13. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
14. When finished navigating pages, close the browser.
15. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
16. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

8.3 Playing Back Scripts

OpenScript plays back recorded Oracle EBS/Forms actions/commands which consist of an object identified by its attributes (for example: `nca.treeList("handlerName").selectByIndex(0);`). The actions used for playback will either be those that are recorded or specified manually in the Java Code view. Playback can be configured through IE only as the Oracle EBS/Forms Load Test Module does not support Firefox. Unattended playback is supported through Oracle Test Manager or third-party tools using OpenScript's command line interface.

The Oracle EBS/Forms Load Test Module provides playback and iterate toolbar buttons that allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for Oracle EBS/Forms Load scripts can be viewed in the Results and Console views.

8.3.1 Setting Oracle EBS/Forms Load Test Playback Preferences

To set Oracle EBS/Forms Load Test playback preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Playback category.
4. Select **Oracle EBS/Forms Load Test**.
5. Select or clear the message options. See Section 2.4.5, "Oracle EBS/Forms Load Test Preferences" for descriptions of the Playback Preferences settings.
6. Click **OK**.

8.3.2 Playing Back Oracle EBS/Forms Load Scripts

To play back Oracle EBS/Forms Load scripts:

1. Start OpenScript.
2. Open the Oracle EBS/Forms Load script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

8.3.3 Playing Back Oracle EBS/Forms Load Scripts with Iterations

To play back Oracle EBS/Forms Load scripts with iterations:

1. Start OpenScript.
2. Open the Oracle EBS/Forms Load script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.
5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.
7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

8.3.4 Playing Back Oracle EBS/Forms Load Scripts on Multiple "Same" Environments

In cases where you have multiple servers running the same EBS application for different purposes and want to do load testing for these sites using the same EBS/Forms load testing scripts, the Scripts may require some modifications. This section lists some areas to check and modify before using scripts recorded for one site for load testing on another site of the same EBS/Forms application.

- Parameterize URLs

  Use the **Parameterize URL** option on the **Tools** menu to create a variable name for the base URLs of the script.

- Correlate the server name and port number when in socket mode

  Locate the http navigation just before the "Connect to Forms server" Tree view or the `nca.connect` Java code. Correlate the forms server name and server port number to verify the host value and port number match the server on which you want to play back the script.

- Correlate the EBS/Forms Config Value

  EBS load script recorded against one site does not playback on another site because the "config" attribute value is different between sites. For example Site A may have `config='GDEV'` and Site B may have `config='OSIT'`. The Site A recorded script has `"config=sl8st213"` in the XML message sent to forms server while Site B requires `"config=mz8st213"` in the same message. Replace the value of the "config" attribute in the XML messages with a variable.

The **formsload.config** correlation rule (`config='(.+?)'`) is used for EBS/Forms application versions 11.x. The **formsload.config_12** correlation rule (`config\s*=\s*(?:'|')(.+?)(?:'|')`) is used for EBS/Forms application versions 12. The correlation rules replaces the value of the "config" attribute in the XML messages with a variable.

- Check Variables from JavaScript and Web page

  An `http.javaScriptPath` command may not play back correctly due to a slight difference between the JavaScript code of two sites. For example, Site A uses JavaScript such as:

  ```
  http.javaScriptPath("web.jscript.httprws60224remsuso_414", 1, 5, 86, 0)
  ```

  However, Site B uses JavaScript such as:

  ```
  http.javaScriptPath("web.jscript.httprws60224remsuso_414", 1, 5, 87, 0)
  ```

  Where a JavaScript variable does not play back correctly on a different site, in most cases the variables were for "static" content and can be safely removed. Right-click on the JavaScript variable that fails and choose "Delete" to delete the variable. If you have many static JavaScript variables that are failing in the script, right-click on any single Javascript variable and choose the option to remove all variables starting with `http.jscript`.

8.4 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

8.4.1 Adding Forms Actions

The Oracle EBS/Forms Module includes actions for Oracle EBS/Forms objects that can be added to a script.

To add Forms actions to a script:

1. Record a EBS/Forms Functional Test script.
2. Select the script node where you want to add the action.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Forms Load Action node.
5. Expand an action node and select the action.
6. Click **OK**.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path.
8. Enter any required values to use for the object action.
9. Click **OK**. The action node is added to the script tree.

   In the Java Code view, a `nca.`*`object`*`(`*`handlerName`*`).`*`action`*`()` method will be added to the script code:

   ```
   nca.treeList("handlerName").selectByIndex(0);
   ```

   The Forms Load Action node includes actions for objects such as Application, Button, CheckBox, ChoiceBox, List Item, List of Values, Pop List Item, Radio

Button, Tab, Text Field, Window, Generic Client, and Alert Dialog. Other object actions have corresponding Java code methods.

8.4.2 Converting Forms Actions to XML Messages

In some cases, you may want to convert EBS/Forms actions recorded to the script into the raw XML client messages to troubleshoot playback issues. For example, the script recorder creates actions in the script tree such as `Activate window ("WindowName")`. The Java code for the action would be similar to `nca.window(id, "WindowName").activate()`.

However, the actual XML message traffic between the application server and the Web client is more verbose. For example, the message traffic for a window activate action could be similar to the following:

```
<Messages>
  <ClientMessage Object="HEADER">
    <Message mActionString="MSG_UPDATE" mActionCode="2" mHandlerClassId="0"
          mHandlerId="170">
      <Property actionString="WINDOW_ACTIVATED" action="247"
          type="java.lang.Boolean" value="true"/>
    </Message>
  </ClientMessage>
  <ClientMessage Object="HEADER_ORIGINATING_BAL_SEG_VALUE_0">
    <Message mActionString="MSG_UPDATE" mActionCode="2" mHandlerClassId="0"
          mHandlerId="566">
      <Property actionString="CURSOR_POSITION" action="193"
          type ="java.lang.Integer" value="0"/>
    </Message>
  </ClientMessage>
</Messages>
```

If you need to troubleshoot specific script actions, you can convert the script action to the raw XML messages and try playing back the script again to see if the XML messages resolve the issue.

To convert EBS/Forms actions to XML client messages:

1. Right-click the script node you want to convert and select **Properties**. The Form Window opens with the following options:

   Automatically generate the Forms client messages: When selected, the script uses the recorded Forms action on playback.

   - **Path**: Shows the recorded Forms action path.

   Send recorded Forms client messages: When selected, the script uses the recorded raw XML client messages on playback.

   - **Description**: Shows a description of the client messages.

   Recorded Messages: Shows the raw XML client and server messages generated between the application server and the web client during recording.

2. Select the **Send recorded Forms client messages** option. If necessary, you can edit the raw XML messages.

3. Click **OK**.

> **Note:** Converting the script action to XML messages cannot be undone unless you close the script without saving.

The script node changes to a `Send Message` *`action`* action. The Java Code for the action will change to a method in the following form:

```
nca.sendMessages(id,"description","<Messages>XML string</Messages>");
```

The raw XML message string can be very long and can increase script size.

8.4.3 Using the Oracle EBS/Forms Load Test Module API

The Oracle EBS/Forms Load Test Module includes a script Application Programming Interface (API) specific to Oracle EBS/Forms load testing. The Oracle EBS/Forms Load Test Module recorder creates the Java code that corresponds to the Tree View and displays the Oracle EBS/Forms Load Test commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the Oracle EBS/Forms Load Test API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Oracle EBS/Forms Functional Testing Module are part of the "nca" class. Additional functional test methods are available in the "http" class. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the Oracle EBS/Forms Testing Module API include:

- Application
- Alert Dialog
- Button
- CheckBox
- ChoiceBox
- Generic Client
- List Item
- List of Values
- Pop List Item
- Radio Button
- Tab
- Text Field
- Window

Many API methods can be added using the Oracle EBS/Forms Load Test Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures. See the API Reference in the OpenScript help for additional programming information.

8.5 Setting Oracle EBS/Forms Load Test Correlation Preferences

To set Setting Oracle EBS/Forms Load Test Correlation preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Correlation category.
4. Expand the Oracle EBS/Forms Load Test library.

5. Select or clear the check boxes to enable or disable specific rules.
6. Click the **Add** or **Edit** buttons to modify rules in the library.
7. Click **OK**.

8.6 Oracle EBS/Forms Load Test Correlation Library

The Oracle EBS/Forms correlation library defines the correlation rules for Oracle EBS/Forms-based applications. The correlation rules specify the variable names and regular expressions to use to replace dynamic data in Oracle EBS/Forms applications and navigations. The default Oracle EBS/Forms correlation library provided with the OpenScript Oracle EBS/Forms Load Test Module includes the following correlation rules:

- **formsload.location - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `document.location='(.+?)'` and replaces it with the variable name `formsload.location` in all locations.
- **EBS Link and Form Action - DOM Correlation** - this rule implements the Web Document Object Model correlation rules for Links and Form actions for Oracle EBS/Forms applications.
- **ICX Ticket 11i - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `icx_ticket\s*=\s*(?:'|')(.+?)(?:'|')` and replaces it with the variable name `formsload.icx_ticket_11i` in all locations.
- **ICX Ticket R12 - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `icx_ticket&gv15=(.+?)&` and replaces it with the variable name `formsload.icx_ticket_r12` in all locations.
- **ICX Ticket R12s - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `gv15\s*=\s*"(.+?)"` and replaces it with the variable name `formsload.icx_ticket_r12s` in all locations.
- **JSLaunchForm - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `href="javascript:launchForm\('(.+?)'\)"` and replaces it with the variable name `formsload.launchform` in all locations.
- **formsload.module - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<PARAM (?:name|NAME)=(?:|'|")serverArgs(?:|'|")\s+(?:value|VALUE)="module=(.+?)` and replaces it with the variable name `formsload.module` in all locations.
- **formsload.config - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `config='(.+?)'` and replaces it with the variable name `formsload.config` in all locations. This rule is used for EBS/Forms application versions 11.x.
- **formsload.config_12 - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `config\s*=\s*(?:'|')(.+?)(?:'|')` and replaces it with the variable name `formsload.config` in all locations. This rule is used for EBS/Forms application versions 12.
- **SSO FormsID - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `sso_formsid=(.+?)` and replaces it with the variable name `formsload.ssoformsId` in all locations.

- **FormsLT Global Substitution - Oracle/EBS Forms Load Variable Substitution** - this rule adds the appropriate Connect Statement for Forms implementations running in HTTP mode. Substitute Forms URL transforms and ICX Ticket transform.
- **formsload.oas - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `oas(?:=|%3d|%3D)(.+?\.\.)` and replaces it with the variable name `formsload.oas` in all locations.
- **formsload.ti - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `&_ti(?:=|%3d|%3D)(.+?)&` and replaces it with the variable name `formsload.ti` in all locations.
- **EBS DOM Correlation - DOM Correlation** - this rule implements the default EBS Document Object Model correlation rules for Oracle EBS/Forms applications.
- **formsload.backslashJsVar - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `\\'\w+\\':\\'(.+?)\\'`, and replaces it with the variable name `formsload.backslashJsVar` in all locations.
- **formsload.formsvar.LOCATE_FG_BAR - Oracle/EBS Forms Load Variable Substitution Forms Only** - this rule locates text in the HTML matching the Regular Expression pattern `\[13\]LOCATE_FG_BAR\[15\](.+?)\"/>` and replaces it with the variable name `formsload.formsvar.LOCATE_FG_BAR` in all Requests and Responses locations.
- **formsload.formsvar.WINSYS_HYPERLINK_params - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern `<Property actionString=\"WINSYS_HYPERLINK\" action=\"283\" type=\"\[Ljava\.lang\.String;\" length=\"2\" value0=\"(.+?)params=(.+?)&oas` and replaces it with the variable name `formsload.formsvar.WINSYS_HYPERLINK_params` in all Requests and Responses locations.
- **formsload.formsvar.WINSYS_HYPERLINK_oas - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern `<Property actionString=\"WINSYS_HYPERLINK\" action=\"283\" type=\"\[Ljava\.lang\.String;\" length=\"2\" value0=\"(.+?)&oas=(.+?)\"` and replaces it with the variable name `formsload.formsvar.WINSYS_HYPERLINK_oas` in all Requests and Responses locations.
- **formsload.formsvar.CONFIGURATOR_session_key - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern `<param name="configurator_session_key">(.+?)</param>` and replaces it with the variable name `formsload.formsvar.CONFIGURATOR_session_key` in all Requests and Responses locations.
- **formsload.formsvar.CONFIGURATOR_icx_session_ticket - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern `<param name="icx_session_ticket">(.+?)</param>` and replaces it with the variable name `formsload.formsvar.CONFIGURATOR_icx_session_ticket` in all Requests and Responses locations.
- **formsload.formsvar.CONFIGURATOR_creation_date - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern `<param name="config_creation_`

date">(.+?)</param> and replaces it with the variable name formsload.formsvar.CONFIGURATOR_creation_date in all Requests and Responses locations.

- **formsload.formsvar.CONFIGURATOR_creation_date - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern <param name="config_creation_date">(.+?)</param> and replaces it with the variable name formsload.formsvar.CONFIGURATOR_creation_date in all Requests and Responses locations.
- **formsload.formsvar.CONFIGURATOR_model_lookup_date - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern <param name="config_model_lookup_date">(.+?)</param> and replaces it with the variable name formsload.formsvar.CONFIGURATOR_model_lookup_date in all Requests and Responses locations.
- **formsload.formsvar.CONFIGURATOR_requested_date - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern <param name="requested_date">(.+?)</param> and replaces it with the variable name formsload.formsvar.CONFIGURATOR_requested_date in all Requests and Responses locations.
- **formsload.formsvar.CONFIGURATOR_client_header - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern <param name="client_header">(.+?)</param> and replaces it with the variable name formsload.formsvar.CONFIGURATOR_client_header in all Requests and Responses locations.
- **formsload.formsvar.CONFIGURATOR_client_line - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern <param name="client_line">(.+?)</param> and replaces it with the variable name formsload.formsvar.CONFIGURATOR_client_line in all Requests and Responses locations.
- **formsload.formsvar.CONFIGURATOR_warehouse_id - Oracle/EBS Forms Load Variable Substitution Forms-Http** - this rule locates text in the HTML matching the Regular Expression pattern <param name="warehouse_id">(.+?)</param> and replaces it with the variable name formsload.formsvar.CONFIGURATOR_warehouse_id in all Requests and Responses locations.
- **Client Set Cookie - Client Set Cookie** - this rule set the client cookie.
- **formsload.formsstep.THINKTIME - Oracle/EBS Forms Load Step Generation** - this rule is used to generate think time in script step groups.

8.7 Troubleshooting Oracle EBS/Forms Load Test Scripts

The following are key concepts, tools, and functions you can use when debugging Oracle EBS/Forms Load Test scripts recorded during recording as well as playback.

8.7.1 Debugging Using the Message Log

This section explains how to use the Message Log for debugging Oracle EBS/Forms Load Test Scripts.

8.7.1.1 During Recording

When recording EBS/Forms Load scripts, the Oracle EBS/Forms Load Test module generates a complete log of messages that were exchanged between the forms server and the client browser. The log is generated for all types of communication (See Recording Oracle EBS/Forms Load Tests):

- Socket Connection - numerically identified by a 0
- HTTP Connection - numerically identified by a 1
- HTTPS Connection - numerically identified by a 2

This recorded log is located in `<installdir>/OFT/<ScriptName>/recordedData/FormsLT_Recorded_FormsMessage_Logs/RecordedMessages.log`.

8.7.1.2 Format of the Recorded Log

The format in the log is as follows:

- MESSAGE FROM CLIENT: indicates a Message sent from client to server, followed by an XML string representation of the message object.
- TERMINAL MESSAGE FROM CLIENT: indicates a Terminal Message sent from client to server, followed by an XML string representation of the terminal message object.
- MESSAGE FROM SERVER: indicates a response message sent from the server to client, followed by an XML string representation of the message object.
- TERMINAL MESSAGE FROM SERVER: indicates a Terminal Message indicating the end of a response sent from the server to the client, followed by an XML string representation of the terminal message object.

8.7.1.3 During Playback

If a script fails during playback, you can enable the message log for debug purposes.

To enable the message log:

1. Open the Oracle EBS/Forms load test script.
2. Enable the **Capture Playback Log** option in the EBS/Forms Playback Preferences.

After enabling message logging, click the play button to run the script. During script play back, all forms messages exchanged between server and the client are printed to the console window in OpenScript.

This is a rather large log of messages and using the recorded log as a reference, compare the messages being sent to the server by OpenScript. Check for messages that appear in the recorded log but not in the playback log. It is normal to have messages such as OUTERSIZE, LOCATION missing. The reason for this is because the EBS/Forms load script recorder was designed keeping in mind that the user should see actions being recorded in the script instead of messages.

8.7.1.4 After Playback

Message playback logs can be used to compare the recorded vs. played back forms messages to troubleshoot script problems. Select a result in the Results view and compare the recorded vs. playback messages in the Comparison tab of the Details view.

The Details view for EBS/Forms load scripts can have the following tabs:

- **Object Details Tree**: Shows a text-based representation of the property names and values of an EBS/Forms object in a tree hierarchy. You can right click a property to add a Text Matching test.
- **Screenshot**: Shows an image of the EBS/Forms window for visual identification. The Screenshot is taken on Window Activate actions.
- **Messages**: Shows the raw XML EBS/Forms messages recorded for the selected script node.
- **Message Tree**: Shows the XML EBS/Forms message parameters and values for the selected script node in a tree hierarchy.
- **Comparison**: Shows a comparison of the recorded vs. playback XML EBS/Forms messages or object details for the selected item in the Results view. Select the comparison type from the **Compare** list.

8.7.2 Analyzing Message Logs

Messages to check for when analyzing recorded and playback logs are as follows:

- FOCUS messages: Make sure the correct component has focus. The script cannot execute an action on a component without correctly setting a focus on it. For example, a text field component contains a button which displays a list of values dialog when pressed. This button is not enabled until the focus is correctly set on the text field. In this case, if OpenScript tries to execute a press button action and the focus in not correct, you will encounter an error such as "Component not available".
- WINDOW_ACTIVATED messages: Make sure the correct window is the currently active windows for the script action on a component. If the correct window is not currently active, the script will not be able to execute any actions on the components within that window.

8.7.3 Resolving "Component does not exist" Errors

"Component {0} does not exist" error messages (error code COMPONENT_DOES_NOT_EXIST) may occur when initially creating an Oracle Forms Load Testing script. This error is reported Possible causes:

1. If the error appears on the first action of the script, it could be indicative of "Issue 1. Connection Failures and ICX_Ticket Correlation" above.
2. The object specified in the script truly does not exist in the application at the time the action was performed on it.

   For example, when recording the script, if the tester searches for an order and deletes it, then during playback the script will fail because the order number no longer exists to delete. In some situations it may be helpful to insert status bar text validations, in cases where a status bar text message is expected.

   In other cases, the Forms application may display an unexpected dialog box error message indicating some validation failure or other business flow error in the

application itself. If such a dialog did not appear during recording, then any subsequent actions will fail. Check the error log in the script to see if any error dialog messages appeared. Check that all data inputs in the script are valid, and carefully review the actions specified in the script.

3. In other cases, it may be difficult to diagnose the root cause of the problem without technical support. For example, it could be that the application being tested is using a custom forms object that sends a custom message to the server that the Forms playback engine does not recognize.

8.7.4 Troubleshooting Forms ifError Messages

When playing the script in HTTP mode, the server can respond with error messages in plain/text which would require special handling by the VU. The following are the most common types of `ifError` messages returned by the server.

IfError Messages

These messages are returned as a response of any forms request against the server when playing the script in HTTP mode. The response content type for these messages is 'plain/text' and the response code is '200 OK'

ifError:3

The client does not handle cookies. Enable cookie handling for that virtual user.

ifError:4

The server was unable to create an Oracle EBS/Forms process. Problem exists on the server-side.

ifError:5

The server was unable to start the Oracle EBS/Forms process. Problem exists on the server-side.

ifError:6

The Oracle EBS/Forms session was aborted and the VU is unable to communicate with the server. In most cases this would occur when the server is experiencing a heavy load or if there is a problem with the script.

ifError:7

The server is currently busy, re-try the request later. The VU will re-try this request <n> number of times where <n> is a value specified in Forms Load Test playback preferences before it throws an error/stops playback.

ifError:11/<n>

The server is busy, re-try the request in <n> milliseconds. This type of message will not be displayed to the user, the VU will automatically re-try this request after <n> milliseconds.

9

Using the Oracle Fusion/ADF Functional Test Module

This chapter provides instructions on configuring and using the OpenScript Oracle Fusion/ADF Functional Test Module, which provides support for functional testing of Oracle Application Development Framework (ADF)-based applications.

9.1 About the Oracle Fusion/ADF Functional Test Module

The Oracle Fusion/ADF Functional Test Module provides support for functional testing of Oracle Application Development Framework (ADF) applications. The Oracle Fusion/ADF Functional Test Module is an extension to the Web Functional Module. The Oracle Fusion/ADF Functional Test Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager.

The Oracle Fusion/ADF Functional Test recorder displays commands in the Tree View in easy-to-understand commands. By default, script commands are grouped into Steps Groups by the Web page on which they were performed. Each Step Group contains one or more script commands corresponding to recorded actions that were performed on the page. The default name for the Step Group is the ADF component name.

OpenScript shows the results of Oracle Fusion/ADF Functional Test script playback in the Results view. The Results view shows results for each script command (including duration and summary for failures). The Results Report compiles the same information into an HTML Results Report. Results can be exported from the OpenScript GUI in standard format (CSV / HTML). Results are also generated for unattended playback through the command line.

The Oracle Fusion/ADF Functional Test Module API includes a "adf" class that provides additional programming functionality.

9.1.1 Prerequisites

The Oracle Fusion/ADF Functional Test Module recorder has the following prerequisite:

- Before recording any script in Oracle Fusion/ADF Functional Test Module, you must configure the ADF Server so that the ADF application uses uncompressed class names.

9.1.2 Key Features of the Oracle Fusion/ADF Functional Test Module

- The Oracle Fusion/ADF Functional Test Script Module. The New Project wizard (Select **New** from the **File** menu) includes an "Oracle Fusion/ADF" option in the Functional Test Group to use when creating Oracle Fusion/ADF functional testing projects in OpenScript. The Oracle Fusion/ADF Functional Test Script Module records functional scripts against ADF Faces applications (Oracle Application Development Framework 11*g* Release 1 (11.1.1)).
- ADF Functional-Specific Application Programming Interface (API). The Oracle Fusion/ADF Functional Test Module includes an ADF Functional Test Module API Specification that can be used to customize ADF Functional test-specific scripts.

9.2 Configuring the ADF Server

The Oracle Fusion/ADF Fusion Test Module recorder requires that the ADF application use uncompressed class names. You must configure the application server to specify that uncompressed class names are used for testing purposes.

There are three settings that must be configured in the following files on the ADF server:

- WEB-INF/web.xml
- trinidad-config.xml

9.2.1 Configuring the WEB-INF/web.xml File

This section explains the settings that must be specified in the ADF server web.xml file.

In the ADF server's WEB-INF/web.xml file, set the `oracle.adf.view.rich.automation.ENABLED` parameter to `true`. This is required in order for the Oracle Fusion/ADF Functional Test Module to find objects using scope IDs (also known as test IDs or Sub IDs).

The following XML shows how the `oracle.adf.view.rich.automation.ENABLED` parameter must be specified in the web.xml file before recording Oracle Fusion/ADF Functional Test scripts:

```
<context-param>
<param-name>oracle.adf.view.rich.automation.ENABLED</param-name>
<param-value>true</param-value>
</context-param>
```

In the ADF server's WEB-INF/web.xml file, set the `org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION` parameter to `true`. This is required in order for the Oracle Fusion/ADF Functional Test Module to identify ADF component class names.

The following XML shows how the `org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION` parameter must be specified in the web.xml file before recording Oracle Fusion/ADF Functional Test scripts:

```
<context-param>
<param-name>org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION</param-name>
<param-value>true</param-value>
</context-param>
```

9.2.2 Configuring the trinidad-config.xml File

This section explains the settings that must be specified in the ADF server trinidad-config.xml file.

In the ADF server's trinidad-config.xml, set the `animation-enabled` element to `false`.

The following XML shows how the animation-enabled element must be specified in the trinidad-config.xml file before recording Oracle Fusion/ADF Functional Test scripts:

```
<animation-enabled>false</animation-enabled>
```

9.2.3 Verifying the Compression Settings

This section explains how to verify that the ADF application is using uncompressed class names.

To verify the ADF application is using uncompressed class names:

1. Navigate to the ADF site.
2. Press F12 or select **Developer Tools** from the **Tools** menu.
3. Click the HTML tab and check the elements of the DOM Tree. If the majority of the class names are in the form `class="af_`*`name`*`"`, the ADF application is using uncompressed class names. If the majority of the class names are in the form `class="x??"`, the ADF application is using compressed class names.

9.3 Recording Oracle Fusion/ADF Functional Tests

The Oracle Fusion/ADF Functional Test Module records standard ADF components for Oracle Application Development Framework 11*g* Release 1 (11.1.1). The Recorder creates functional test scripts for automating testing of Oracle Fusion/ADF applications.

ADF components are applet based controls and the Oracle Fusion/ADF Functional Test Module provides the object/attribute information for OpenScript to record interactions with those controls. Actions will be captured in the test script as OpenScript "adf" commands. Other components are standard Web controls which are captured as standard OpenScript "web" and "ft" navigation commands. Correlation rules can be modified by users through the Preferences settings for new scripts.

OpenScript plays back recorded ADF actions/commands which consist of an event plus an object identified by its attributes (for example: `adf.inputText(11,"/web:window[@index='0' or @title='inputText Demo']/web:document[@index='0']/web:ADFInputText[@id='dmoTpl:idInputText' and @label='String value']").setValue("String")`. The actions used for playback will either be those that are recorded or specified manually in the Java Code view. Playback can be configured through IE only as the Oracle Fusion/ADF Functional Test Module does not support Firefox. Unattended playback is supported through Oracle Test Manager or third-party tools using OpenScript's command line interface. Oracle Fusion/ADF Functional Test scripts do not play in Oracle Load Testing.

The Oracle Fusion/ADF Functional Test Module provides a record toolbar button that lets you initiate the ADF recorder and capture Web/ADF page actions to the script view. The record toolbar includes start and stop recording toolbar buttons. OpenScript

recorders also open a floating toolbar that can be used while recording without having to switch between the browser and OpenScript.

9.3.1 Setting Oracle ADF Functional Test Record Preferences

To set Oracle ADF Functional Test record preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Record category.
4. Select **Oracle ADF Functional**.
5. Click the tab and set the preferences. See Section 2.5.3, "Oracle ADF Functional Test Preferences" for descriptions of the Record Preferences settings.
6. Click **OK**.

9.3.2 Adding/Editing Object Identifiers

The Oracle ADF Functional Test Module uses object identification to specify attributes used to identify Oracle ADF objects. The Oracle ADF Functional Test Module uses the same predefined path attributes for common Web objects as the Web Functional Test Module; however, Oracle ADF Test Automation provides additional attributes to identify ADF controls. Object paths are specified in XPath format. For example the object identification path appears as follows in Java code commands:

```
"/web:ADFCommandButton[@id='dmoTpl:usewindowButton' and @text='Click here']")
```

The full path to the control appears as follows:

```
adf.commandButton("/web:window[@index='0' or @title='commandButton Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFCommandButton[@id='dmoTpl:usewindowButton' and @text='Click here']")
 .click();
```

You can set the default Web object attributes in the Oracle ADF Functional Test Module Record Preferences. You can also edit object attributes in recorded scripts in the tree view or the code view.

In addition to the predefined object identification, you can add an Object Library to the script to record paths into a library file. Object Library files may be shared and reused across other scripts. The Object Library files provide a more convenient "short name" for objects to provide for more convenient programming.

The Oracle ADF Functional Test Module includes object identifiers that specify how the recorder identifies Browser objects. You can add object identifiers or edit the existing object identifiers in the Record preferences.

To add or edit an object identifier:

1. Select the **OpenScript Preferences** from the **View** menu.
2. Expand the Record node and select ADF Functional Test.
3. Click the **Object Identification** tab. This tab lets you specify the Oracle ADF object identification attributes, as follows:

   Active Profile: Specifies which object identification profile to use as the active profile when recording scripts. Profiles define a specific set of object identifiers to use when recording ADF functional tests. Use the **Add Profile** option to create a

new custom profile. Once you have created a profile, select the profile name in the **Name** column and use **Add Object** to define custom objects and attributes in the custom profile.

Name: Shows the name(s) of the defined Oracle ADF object identifiers.

Attributes: Shows the pattern(s) specified for the defined Oracle ADF object identifiers.

Add Profile: Opens a dialog box for specifying a new Oracle ADF object identifier profile.

Add Object: Opens a dialog box for specifying a new Oracle ADF object identifier.

Edit: Opens a dialog box for editing the selected Oracle ADF object identifier.

Delete: Deletes the selected Oracle ADF object identifier or custom profile. The default profile cannot be deleted.

Export: Opens a dialog box for exporting the currently selected ADF object identifier profile to an XML file.

Import: Opens a dialog box for importing a saved object identifier profile XML file.

Revert: Reverts the default ADF object identification profile to the default profile. Any changes to the default profile are removed. Select the default profile name in the **Name** column to activate the revert option.

For each object element, you specify a name (typically an Oracle ADF object attribute), an operator, a value and a value type. As you add object elements, OpenScript builds the object identifier using logical OR between each object identifier element. Click **Edit** to change between logical OR and AND.

4. Click **Add** or select an existing object identifier and click **Edit**.
5. If adding a new object identifier, enter a name for the object identifier.
6. Add or edit object elements for the object identifier.

   See the Web Functional Test Module for additional information about adding and editing Object Identifiers.
7. Click **OK**. The object identifier is added to the record preferences.

9.3.3 Recording Oracle Fusion/ADF Functional Test Scripts

To record ADF Functional Test scripts:

1. Start OpenScript.
2. Set the Oracle Fusion/ADF Functional Test Correlation preferences.
3. Select **New** from the **File** menu.
4. Expand the Functional Testing group.
5. Select **ADF** (The Oracle Fusion/ADF script combines both Web and ADF technologies as part of the same script).
6. Click **Next**.
7. Select the Repository and Workspace.
8. Enter a script name.
9. Click **Finish**. A new Script tree is created in the Script View.

10. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
11. Load the web page where you want to start recording into the browser.
12. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
13. When finished navigating pages, close the browser.
14. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
15. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

9.4 Playing Back Scripts

OpenScript plays back recorded ADF actions/commands which consist of an object identified by its attributes. The actions used for playback will either be those that are recorded or specified manually in the Java Code view. Unattended playback is supported through Oracle Test Manager or third-party tools using OpenScript's command line interface.

The Oracle Fusion/ADF Functional Test Module provides playback and iterate toolbar buttons that allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for Oracle Fusion/ADF Functional scripts can be viewed in the Results and Console views.

9.4.1 Playing Back Oracle Fusion/ADF Functional Scripts

To play back Oracle Fusion/ADF Functional scripts:

1. Start OpenScript.
2. Open the Oracle Fusion/ADF Functional script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

9.4.2 Playing Back Oracle Fusion/ADF Functional Scripts with Iterations

To play back Oracle Fusion/ADF Functional scripts with iterations:

1. Start OpenScript.
2. Open the Oracle Fusion/ADF Functional script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.

5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.
7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

9.5 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

9.5.1 Adding Fusion/ADF Actions

The Oracle Fusion/ADF Module includes actions for Oracle Fusion/ADF objects that can be added to a script.

To add Fusion/ADF actions to a script:

1. Record a Fusion/ADF Functional Test script.
2. Select the script node where you want to add the action.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the ADF Action node.
5. Expand an action node and select the action.
6. Click **OK**.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path.
8. Enter any required values to use for the object action.
9. Click **OK**. The action node is added to the script tree.

   In the Java Code view, an `adf.`*`object`*`(`*`objectId`*`).`*`action`*`()` method will be added to the script code:

```
adf.inputText("/web:window[@index='0' or @title='inputText Demo']
    /web:document[@index='0' or @name='w0']
    /web:ADFInputText[@id='dmoTpl:idInputText'
        and @label='']").setValue("My Text Input");
```

   The ADF Action node includes actions for objects such as Calendar, Command Button, Command Image Link, Command Link, Command Menu Item, Command Toolbar, Dialog, Guage, Go Menu Item, Graph, Input Color, Input Combobox List of Values, Input dateSelect ManyChoice, Input List of Values, Input Number Slider, Input Number Spinbox, Input Range Slider, Input Text, Menu, Navigation Page, Page, Panel Accordion, Panel Box, Panel, Splitter, Panel Tabbed, Panel window, Query, Quick Query, Reset Button, Rich Text Editor, Select Boolean Checkbox, Select Boolean Radio, Select Many Checkbox, Select Many Choice, Select Many Listbox, Select Many Shuttle, Select One Choice, Select One Listbox, Select One Radio, Select Order Shuttle, Show Detail, Show Detail Header, Table,

Toolbar, Train, Train Button Bar, Tree, Tree Table. Other object actions have corresponding Java code methods.

9.5.2 Oracle Fusion/ADF Functional Test Module API

The Oracle Fusion/ADF Functional Test Module includes a script Application Programming Interface (API) specific to ADF functional testing. The Oracle Fusion/ADF Functional Test Module recorder creates the Java code that corresponds to the Tree View and displays the Oracle Fusion/ADF Functional Test commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the Oracle Fusion/ADF Functional Test API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Oracle Fusion/ADF Functional Testing Module are part of the "adf" class. Additional test methods are available in the "web" or "ft" classes. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the Oracle Fusion/ADF Testing Module API include:

- Calendar
- Command Button
- Command Image Link
- Command Link
- Command Menu Item
- Command Toolbar
- Dialog
- Gauge
- Go Menu Item
- Graph
- Input Color
- Input Combobox List of Values
- Input dateSelect ManyChoice
- Input List of Values
- Input Number Slider
- Input Number Spinbox
- Input Range Slider
- Input Text
- Menu
- Navigation Page
- Page
- Panel Accordion
- Panel Box
- Panel Splitter

- Panel Tabbed
- Panel window
- Query
- Quick Query
- Reset Button
- Rich Text Editor
- Select Boolean Checkbox
- Select Boolean Radio
- Select Many Checkbox
- Select Many Choice
- Select Many Listbox
- Select Many Shuttle
- Select One Choice
- Select One Listbox
- Select One Radio
- Select Order Shuttle
- Show Detail
- Show Detail Header
- Table
- Toolbar
- Train
- Train Button Bar
- Tree
- Tree Table

Many API methods can be added using the Oracle Fusion/ADF Testing Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures. See the API Reference in the OpenScript help for additional programming information.

10

Using the Oracle Fusion/ADF Load Test Module

This chapter provides instructions on configuring and using the OpenScript Oracle Fusion/ADF Load Test Module, which provides support for load testing of Oracle Application Development Framework (ADF)-based applications.

10.1 About the Oracle Fusion/ADF Load Test Module

The Oracle Fusion/ADF Load Test Module provides support for load testing of Oracle Application Development Framework (ADF) applications. The Oracle Fusion/ADF Load Test Module is an extension to the HTTP Module. The Oracle Fusion/ADF Load Test Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager.

The Oracle Fusion/ADF Load Test recorder displays commands in the Tree View in easy-to-understand commands. By default, script commands are grouped into Steps Groups by the Web page on which they were performed. Each Step Group contains one or more script commands corresponding to recorded actions that were performed on the page. The default name for the Step Group is the ADF component name.

OpenScript shows the results of Oracle Fusion/ADF Load Test script playback in the Results view. The Results view shows results for each script command (including duration and summary for failures). The Results Report compiles the same information into an HTML Results Report. Results can be exported from the OpenScript GUI in standard format (CSV / HTML). Results are also generated for unattended playback through the command line.

The Oracle Fusion/ADF Load Test Module API includes a "adfload" class that provides additional programming functionality.

10.1.1 Key Features of the Oracle Fusion/ADF Load Test Module

- The Oracle Fusion/ADF Load Test Script Module. The New Project wizard (**New** from the **File** menu) includes an "Oracle Fusion/ADF" option in the Load Test Group to use when creating Oracle Fusion/ADF load testing projects in OpenScript. The Oracle Fusion/ADF Load Test Script Module records load scripts against ADF Faces applications (Oracle Application Development Framework 11*g* Release 1 (11.1.1)). OpenScript captures user actions and records them to the OpenScript script based upon HTTP requests and post data or query strings.

- Correlation Library. The Oracle Fusion/ADF Load Test Module includes an Oracle ADF-specific library of correlation rules for parameterizing scripts that allows playback of standard ADF Faces applications without manual correlation.
- ADF Load-Specific Application Programming Interface (API). The Oracle Fusion/ADF Load Test Module includes an ADF Load Test Module API Specification that can be used to customize ADF Load test-specific scripts.

10.1.2 Prerequisites

ADF application developers should ensure that all ADF components in the JSP pages have hard-coded and unique component IDs.

10.2 Recording Oracle Fusion/ADF Load Tests

The Oracle Fusion/ADF Load Test Module records standard ADF components for Oracle Application Development Framework 11*g* Release 1 (11.1.1). The Recorder creates load test scripts for automating testing of Oracle Fusion/ADF applications.

ADF components are applet based controls and the Oracle Fusion/ADF Load Test Module provides the object/attribute information for OpenScript to record interactions with those controls. Actions will be captured in the test script as OpenScript "adfload" commands. Other components are standard Web controls which are captured as standard OpenScript "http" navigation commands. Correlation rules can be modified by users through the Preferences settings for new scripts.

The Oracle Fusion/ADF Load Test Module provides a record toolbar button that lets you initiate the ADF recorder and capture Web/ADF page actions to the script view. The record toolbar includes start and stop recording toolbar buttons. OpenScript recorders also open a floating toolbar that can be used while recording without having to switch between the browser and OpenScript.

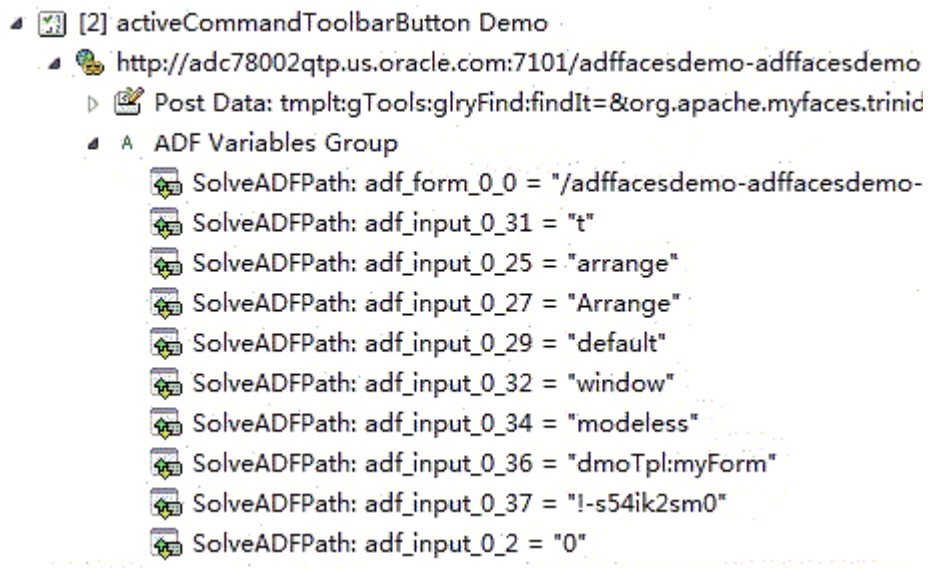
10.2.1 Recording Oracle Fusion/ADF Load Test Scripts

To record ADF Load Test scripts:

1. Start OpenScript.
2. Set the Oracle Fusion/ADF Load Test Correlation preferences.
3. Select **New** from the **File** menu.
4. Expand the Load Testing group.
5. Select **ADF** (The Oracle Fusion/ADF script combines both HTTP and ADF technologies as part of the same script).
6. Click **Next**.
7. Select the Repository and Workspace.
8. Enter a script name.
9. Click **Finish**. A new Script tree is created in the Script View.
10. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
11. Load the web page where you want to start recording into the browser.

12. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
13. When finished navigating pages, close the browser.
14. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
15. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree. ADF scripts include ADF Variables Group nodes that show the attributes and values of ADF variable elements.

Figure 10–1 ADF Variables Group in Script Tree View



Selecting an ADF variable path element in the tree view highlights the value of the selected element in the ADF Rich Response Details View.

Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

You can customize the script using the menu options or the Code View for specific testing requirements. The Code View shows the Variable group and variable elements in the `solveGroupAdf()` and `getAdfVariable()` methods.

```
adfload.solveGroupAdf(
   adfload.getAdfVariable(
      "adf_input_0_5",
      "fragment",
      ".//INPUT[@id='emT:stsch:0:smc1:_0']/@value",
      "0", 0, 0, VariableType.INPUT,
      EncodeOptions.None),

   adfload.getAdfVariable(
      "adf_input_0_3_urlEncode",
      "fragment",
      ".//INPUT[@id='emT:stsch:0:startDate::tzDispId']/@value",
```

```
    "America%2FLos_Angeles", 0, 0,
    VariableType.INPUT,
    EncodeOptions.URLEncode),

adfload.getAdfVariable(
    "adf_input_0_6_",
    "fragment",
    ".//INPUT[@id='emT:stsch:0:smc1:_1']/@value",
    "1", 0, 0, VariableType.INPUT,
    EncodeOptions.None),

adfload.getAdfVariable(
    "adf_comp_emTfinishbtn",
    "component",
    ".//AdfRichCommandButton[@disabled='true']/@fullId",
    "emT:finishbtn", -1, -1,
    VariableType.RICHCOMPONENT,
    EncodeOptions.None));
```

10.2.1.1 Editing ADF Variables Groups

You can use the ADF Group Variables dialog to edit, delete, and move ADF Variables located on the current page.

Double-clicking an "ADF Variables Group" node in the tree view opens the ADF Group Variables dialog. The ADF Group Variables dialog shows the following information:

Name - shows the name of the ADF variable path element.

XPath - show the XPath to the element.

Value - shows the value of the element.

Segment Index - shows the ADF Rich Response Content segment index value.

Variable Index - shows the variable index value.

Variable Type - show the type of variable element which is the index of appearance that matches the XPath in a segment.

Encode Option - shows which encoding option is used for the element.

Edit - opens a dialog for editing the selected variable element.

Delete - removes the selected variable element from the group.

Up - moves the selected variable element up one row in the group.

Down - moves the selected variable element down one row in the group.

10.2.1.2 Editing ADF Variables XPaths

You can edit an ADF Variable element's XPath. Expand an "ADF Variables Group" in the tree view to view the "SolveADFPath" nodes.

Double-clicking a "SolveADFPath" node in the tree view opens the ADF XPath dialog. The ADF XPath dialog shows the following information:

Variable Type - specifies the type of variable.

XPath - specifies the XPath to the element. Use the toolbar to toggle between the tree view and text view to edit the element's XPath.

Name - specifies the name of the ADF variable path element.

Segment Index - specifies the ADF Rich Response Content segment index value.

Variable Index - specifies the type of variable element which is the index of appearance that matches the XPath in a segment.

Encode Option - specifies which encoding option is used for the element.

10.2.1.3 ADF Component Tree View

The Details View for any ADF page contains a "ADF Component Tree View" tab that displays all the ADF components on the page. OpenScript builds the component hierarchy by analyzing the ADF-Rich-Xml-Content. The tree resembles the parent/child relationship between components displayed on a page. The ADF Component Tree View tab provides a view of the components that exist inside an ADF Rich XML Content page.

The ADF Component Tree View tab of the Details view shows the following options and panes:

Find - select the [Match Type], enter the component attribute text to locate in the tree hierarchy, then click **Next** or **Previous** to highlight the attribute in the tree hierarchy.

[Match Type] - specifies the method to use to match the **Find** text.

- **Full Match** - finds attributes that match the exact text specified in the **Find** field.
- **Partial Match** - finds attributes that match any portion of the text specified in the **Find** field.
- **Match XPath** - finds attributes that match the XPath specified in the **Find** field.

Next - locates the next match in the tree hierarchy.

Previous - locates the previous match in the tree hierarchy.

The tree view pane shows the component attributes in a tree hierarchy. Expand the nodes to show individual attributes.

The Properties pane shows the name and value of the component selected in the tree view.

- **Attribute** - shows the component attribute name.
- **Value** - shows the component attribute value.

The XML Contents pane show the raw XML for the component. If the raw XML contents exceeds 30KB, a message appears indicating the contents have been truncated. Click **Load Remaining Content** to load the full XML content.

10.3 Playing Back Scripts

OpenScript plays back recorded ADF actions/commands which consist of an object identified by its attributes. The actions used for playback will either be those that are recorded or specified manually in the Java Code view. Unattended playback is supported through Oracle Test Manager or third-party tools using OpenScript's command line interface.

The Oracle Fusion/ADF Load Test Module provides playback and iterate toolbar buttons that allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for Oracle Fusion/ADF Load scripts can be viewed in the Results and Console views.

10.3.1 Playing Back Oracle Fusion/ADF Load Scripts

To play back Oracle Fusion/ADF Load scripts:

1. Start OpenScript.
2. Open the Oracle Fusion/ADF Load script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

10.3.1.1 ADF Comparison View

After playing back an ADF script, the recorded and played back ADF Rich XML Content can be compared using a textual view of the ADF Rich Component hierarchy in the Details View. Open the Details View and select the Comparison tab. Select the ADF Component Hierarchy Compare option.

10.3.2 Playing Back Oracle Fusion/ADF Load Scripts with Iterations

To play back Oracle Fusion/ADF Load scripts with iterations:

1. Start OpenScript.
2. Open the Oracle Fusion/ADF Load script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.
5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.
7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

10.4 Setting Oracle Fusion/ADF Load Test Correlation Preferences

To set Setting Oracle Fusion/ADF Load Test Correlation preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Correlation category.
4. Expand the ADF Load Test library.
5. Select or clear the check boxes to enable or disable specific rules.
6. Click the **Add** or **Edit** buttons to modify rules in the library.
7. Click **OK**.

10.5 Oracle Fusion/ADF Load Test Correlation Library

The Oracle Fusion/ADF correlation library defines the correlation rules for ADF-based applications. The correlation rules specify the variable names and regular expressions to use to replace dynamic data in ADF applications and navigations. The default Oracle Fusion/ADF correlation library provided with the OpenScript Oracle Fusion/ADF Load Test Module includes the following correlation rules:

- **Correlate ADF Headers - Correlate Headers** - this rule implements the default Correlate Headers correlation rules for ADF applications that use dynamic headers.
- **_afrLoop - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_afrLoop=(\d{10,})` and replaces it with the variable name `_afrLoop` in all locations.
- **_adf.ctrl-state - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_adf\.ctrl-state=(.+?)["&']` and replaces it with the variable name `_adf.ctrl-state` in all locations.
- **ADF Rich Response Correlation - ADF Rich Response Correlation** - this rule implements the ADF Rich component Object Model correlation rules for ADF applications.
- **ADFContentAction - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<content action="(.+?)">` and replaces it with the variable name `ADFContentAction` in all locations.
- **RedirectURL - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<redirect>(.+?)<` and replaces it with the variable name `RedirectURL` in all locations.
- **ContextID - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<contextId>(.+?)</contextId>` and replaces it with the variable name `ContextId` in all locations.
- **ADF Web Correlation - DOM Correlation** - this rule implements the default Web Document Object Model correlation rules for ADF applications.

10.6 Oracle Fusion/ADF Load Test Module API

The Oracle Fusion/ADF Load Test Module includes a script Application Programming Interface (API) specific to ADF load testing. The Oracle Fusion/ADF Load Test Module recorder creates the Java code that corresponds to the Tree View and displays the Oracle Fusion/ADF Load Test commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the Oracle Fusion/ADF Load Test API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Oracle Fusion/ADF Load Testing Module are part of the "adfload" class. Additional test methods are available in the "http" class. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Many API methods can be added using the Oracle Fusion/ADF Testing Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures. See the API Reference in the OpenScript help for additional programming information.

11

Using the Adobe Flex Functional Test Module

This chapter provides instructions on configuring and using the OpenScript Adobe Flex Functional Test Module, which provides support for functional testing of Adobe Flex-based web applications.

11.1 About the Adobe Flex Functional Test Module

The OpenScript Adobe Flex Functional Test Module provides support for functional testing of web applications that use the Adobe Flex Automation Framework. The OpenScript Adobe Flex Functional Test Module provides the ability to record, playback and validate transactions inside Adobe Flex applications embedded in web pages. The OpenScript Adobe Flex Functional Test Module is an extension to the Web Functional Test Module.

The Adobe Flex Functional Test Module is an extension module to the OpenScript Web Functional Test Module that extends the Web testing with Adobe Flex Functional Test recording and playback capabilities. The Adobe Flex Functional Test Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager.

The Adobe Flex Functional Test recorder displays commands in the Tree View in easy-to-understand commands. Script commands correspond to Adobe Flex events generated by the Adobe Flex Automation Framework as OpenScript actions. By default, script commands are recorded sequentially. Step groups can be added manually but are not created by default.

OpenScript shows the results of Adobe Flex Functional Test script playback in the Results view. The Results view shows results for each script command (including duration and summary for failures). The Results Report compiles the same information into an HTML Results Report. Results can be exported from the OpenScript GUI in standard format (CSV / HTML). Results are also generated for unattended playback through the command line.

The Adobe Flex Functional Test Module API includes a "flexFT" class that provides additional programming functionality.

11.1.1 Key Features of the Adobe Flex Functional Test Module

- The OpenScript Adobe Flex Functional Test Module is based on the Adobe Flex Automation Framework. The Flex Automation Framework (the framework) provides the following features:

  - A wide range of reliable record and playback functionality across the broad suite of built-in Flex controls.
  - Documented and supported means of extending the record and playback functionality to custom controls.
  - Standardized object identification.
  - Product behavior consistent with Adobe's documentation about how to test Flex applications.

- Records Adobe Flex actions for automation and validation of Flex applications of any version (2, 3, 4) using any browser (IE or FF). Supports Flash application created with Flex framework.
- Plays back recorded Adobe Flex actions/commands which consist of a Flex control and event plus the object identified by its attributes (for example: `flexFT.textarea(138, "Path").input("LOREM IPSUM");`).
- Provides full script code view integration to support script generation for the OpenScript Adobe Flex Functional Test Module. The OpenScript Adobe Flex Functional Test Module includes an additional API to support Flex Functional Test protocol code scripting.
- Allows users to parameterize user inputs to OpenScript Adobe Flex Functional Test scripts and drive those inputs from an external data file (Databank).
- Allows users to insert Tests to validate Adobe Flex content on playback.
- Reports playback results for OpenScript Adobe Flex Functional Test scripts in the Results and Console views.
- The OpenScript Adobe Flex Functional Test Script Module API. The OpenScript Adobe Flex Functional Test Application Programming Interface include Java code methods specific to functional testing of Adobe Flex applications. The API provides support for the MX component classes available in Flex 3.

The New Project wizard (Select **New** from the **File** menu) includes an "Adobe Flex " option to use when creating Flex functional testing projects in OpenScript. OpenScript captures user actions and records them to the OpenScript script nodes in a highly readable sequence of navigations and actions.

11.1.2 Prerequisites

The Adobe Flex Functional Test Module recorder has the following prerequisites:

> **Note:** The automation libraries/swc files are required for Flex Functional Testing only. This does not apply for Adobe Flex (AMF) load testing which records at the protocol level.

- The Flex application must include the Adobe Flex automation libraries either at compiles time or at run time. You need at least `automation.swc` and `automation_agent.swc` from the Adobe <flex builder>\sdks\3.5.0\frameworks\libs folder (3.5.0 is an Adobe sdk version). Also, `automation_dmv.swc` is required for charts, advanceddatagrid and olapdatagrid support. See the Creating Applications for Testing section of the Adobe Flex Data Visualization Developer's Guide for additional information about the tasks required to include the Flex automation libraries.

  Creating Applications for Testing:

http://livedocs.adobe.com/flex/3/html/help.html?content=functest_components2_15.html#178953

Adobe Flex Data Visualization Developer's Guide:

http://livedocs.adobe.com/flex/3/html/help.html?content=functest_components2_15.html#178953

- The Oracle OpenScript `openscript_agent.swc` file must be included when re-compiling Flex applications. The Flex application must be linked with the OpenScript Flex agent located in <*installdir*>\OpenScript\plugins\oracle.oats.scripting.modules.flexFT_*version*\flexagent\openscript_agent.swc or equivalent.

11.2 Recording Adobe Flex Functional Tests

The Adobe Flex Functional Test Module records web applications that use the Adobe Flex Automation Framework. The Recorder creates functional and regression test scripts for automating testing of Adobe Flex applications.

Events are recorded as they are generated by the Adobe Flex Automation Framework and played back through the framework.

The Adobe Flex Functional Test Module provides the object/attribute information for OpenScript to record interactions with those controls. Actions will be captured in the test script as OpenScript "flexFT" commands. Other components are standard Web controls which are captured as standard OpenScript "web" commands using Web Functional Test object attributes. Object Identification attributes can later be modified by users through the Preferences global settings for new scripts or for already recorded commands in the tree view or code view.

The Adobe Flex Functional Test Module provides a record toolbar button that lets you initiate the Adobe Flex recorder and capture Web/Adobe Flex events to the script view. The record toolbar includes start and stop recording toolbar buttons. OpenScript recorders also open a floating toolbar that can be used while recording without having to switch between the browser and OpenScript.

11.2.1 Recording Adobe Flex Functional Test Scripts

To record Adobe Flex Functional Test scripts:

1. Start OpenScript.
2. Set the Adobe Flex Functional Test Recording preferences.
3. Select **New** from the **File** menu.
4. Expand the Functional Testing group.
5. Select **Adobe Flex** (The Adobe Flex script combines both Web and Adobe Flex technologies as part of the same script).
6. Click **Next**.
7. Select the Repository and Workspace.
8. Enter a script name.
9. Click **Finish**. A new Script tree is created in the Script View.
10. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.

11. Load the web page where you want to start recording into the browser.
12. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
13. When finished navigating pages, close the browser.
14. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
15. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

11.3 Playing Back Scripts

OpenScript plays back recorded Adobe Flex actions/commands which consist of an event plus an object identified by its attributes (for example: `flexFT.combobox(`*`objectId Path`*`).select("Visa", TriggerEvent.Mouse, KeyModifier.None);`). The *`objectId Path`* are hierarchical sets of object properties used to identify a specific UI component in the flex application.

The actions used for playback will either be those that are recorded or specified manually in the Java Code view. Unattended playback is supported through Oracle Test Manager or third-party tools using OpenScript's command line interface. Adobe Flex Functional Test scripts do not play in Oracle Load Testing.

The Adobe Flex Functional Test Module provides playback and iterate toolbar buttons that allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for Adobe Flex Functional scripts can be viewed in the Results and Console views.

11.3.1 Adobe Flex Object Identification

The Adobe Flex Functional Test Module uses object identification to specify attributes used to identify Adobe Flex objects. The Adobe Flex Functional Test Module uses the same predefined path attributes for common Web objects as the Web Functional Test Module; however, Adobe Flex Test Automation provides additional attributes to identify Flex controls. Object paths are specified in XPath format. For example the object identification path appears as follows in Java code commands:

```
flexFT.tree(8,"/web:window[@index='0' or @title='Adobe Flex 3 Component Explorer']
 /web:document[@index='0']
   /flex:application[@automationIndex='index:-1' and
     @automationName='explorer' and
     @automationClassName='FlexApplication' and
     @className='explorer' and
     @label='' and
     @id='explorer']
   /flex:dividedBox[@className='mx.containers.HDividedBox' and
     @id='null' and
```

```
  @automationIndex='index:0' and
  @automationName='index:0' and
  @automationClassName='FlexDividedBox' and
  @label='']
/flex:panel[@automationIndex='index:0' and
  @automationName='Adobe%20Flex%203%20Component%20Explorer' and
  @automationClassName='FlexPanel' and
  @className='mx.containers.Panel' and
  @label='' and
  @id='null']
/flex:tree[@automationIndex='index:0' and
  @automationName='compLibTree' and
  @automationClassName='FlexTree' and
  @className='mx.controls.Tree' and
  @id='compLibTree']")
.select("Visual Components>Button Controls>CheckBox",
  TriggerEvent.Mouse, KeyModifier.None);
```

You can set the default Web object attributes in the Web Functional Test Module Record Preferences. You can also edit object attributes in recorded scripts in the tree view or the code view. The Adobe Flex object identification is provided by the Adobe Flex Automation Framework when recording a component and are not set in OpenScript Preferences.<

In addition to the predefined object identification, you can add an Object Library to the script to record paths into a library file. Object Library files may be shared and reused across other scripts. The Object Library files provide a more convenient "short name" for objects to provide for more convenient programming.

11.3.2 Playing Back Adobe Flex Functional Scripts

To play back Adobe Flex Functional scripts:

1. Start OpenScript.
2. Open the Adobe Flex Functional script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

11.3.3 Playing Back Adobe Flex Functional Scripts with Iterations

To play back Adobe Flex Functional scripts with iterations:

1. Start OpenScript.
2. Open the Adobe Flex Functional script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Set the iteration count.
5. Select which databank file to use, if necessary.
6. Set the starting record for the selected databank in the Databank Settings section. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
7. Click **OK**.

You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

11.4 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

11.4.1 Adding Flex Actions

The Adobe Flex Module includes actions for Adobe Flex objects that can be added to a script.

To add Forms actions to a script:

1. Record a Adobe Flex Functional Test script.
2. Select the script node where you want to add the action.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Flex Action node.
5. Expand an action node and select the action.
6. Click **OK**.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path.
8. Enter any required values to use for the object action.
9. Click **OK**. The action node is added to the script tree.

   In the Java Code view, a `flexFT.`*`object`*`(`*`objectId`*`).`*`action`*`()` method will be added to the script code:

```
flexFT.tree(8,"/web:window[@index='0' or @title='Adobe Flex 3 Explorer']
 /web:document[@index='0']
   /flex:application[@automationIndex='index:-1' and
     @automationName='explorer' and
     @automationClassName='FlexApplication' and
     @className='explorer' and
     @label='' and
     @id='explorer']
   /flex:dividedBox[@className='mx.containers.HDividedBox' and
     @id='null' and
     @automationIndex='index:0' and
     @automationName='index:0' and
     @automationClassName='FlexDividedBox' and
     @label='']
   /flex:panel[@automationIndex='index:0' and
     @automationName='Adobe%20Flex%203%20Explorer' and
     @automationClassName='FlexPanel' and
     @className='mx.containers.Panel' and
     @label='' and
     @id='null']
   /flex:tree[@automationIndex='index:0' and
     @automationName='compLibTree' and
     @automationClassName='FlexTree' and
     @className='mx.controls.Tree' and
     @id='compLibTree']")
   .select("Visual Components>Button Controls>CheckBox",
```

```
TriggerEvent.Mouse, KeyModifier.None);
```

Note: When adding Table, Object, or other OpenScript tests on Flex controls, you should add some Think time to the script between the OpenScript test node and the Flex control being tested. The Think time should be enough time to allow the Flex control to load completely before the OpenScript test is executed. If the OpenScript test is executed before the Flex control loads, an error will occur.

The Adobe Flex Functional Test Module can also perform actions on Adobe Flex-specific objects such as Accordion, AdvancedDataGrid, Alert, Application, AreaChart, AreaSeries, AxisRenderer, BarChart, BarSeries, Box, BubbleSeries, Button, ButtonBar, Canvas, CartesianChart, Chart, ChartLegend, ChartSeries, Checkbox, ColorPicker, ColumnChart, ColumnSeries, ComboBase, ComboBox, Container, DataGrid, DateChoser, DateField, DisplayObject, DivideBox, Form, FormItem, HLOChart, Image, Label, LineChart, LineSeries, LinkBar, List, ListBase, ListLabel, Loader, Menu, MenuBar, NavigationBar, NumericStepper, OLAPDataGrid, Panel, PieChart, PlotSeries, PopUpButton, ProgressBar, RadioButton, Repeater, Rule, ScrollBar, ScrollBase, Slider, TabNavigator, TextArea, TitleWindow, ToggleButtonBar, Tree, UIMovieClip, VideoDisplay, ViewStack.

11.4.2 Adobe Flex Action Dialog Box

The Adobe Flex Action Dialog Box lets you specify actions to perform on Flex-specific objects. The available options and settings depend upon the type of Flex object and the specific action to perform on the object. The resulting dialog box can display the following sections and fields:

Action: Shows the type of action to perform on the Flex object.

Object: Shows the type of Flex object on which to perform on the action.

Path: Specifies the object identification path used to identify the object in the application.

Value/Values: Specifies the values to use to perform an action on a Flex object. The values vary depending upon the object type and action. All index values are zero-based.

- **Action**: A String that specifies the action for drag and drop operations on controls.
- **Begin Index**: An Integer that specifies the beginning index of the SelectText action for ColorPicker, ComboBase, ComboBox, DateField, NumericStepper, and TextArea controls.
- **Click Target**: Specifies the target of a click action. Enter "thumb" or "track" to specify either the slider track or slider thumb. This setting is used with the Change action for Slider controls.
- **Color**: A String that specifies the Hexadecimal RGB (i.e. #003366) color value for change actions on ColorPicker controls.
- **Column Index**: An Integer that specifies the index of the column being clicked, edited, or stretched in Grid-type controls.
- **Data Field**: A String that specifies the Header data field for click actions on Grid-type controls.
- **Delt**a: An Integer value that specifies the change in value for mouse scroll actions.

- **Dragged Item**: A String that specifies the item dragged in DragStart and DragDrop actions on controls.
- **End Index**: An Integer that specifies the ending index of the SelectText action for ColorPicker, ComboBase, ComboBox, DateField, NumericStepper, and TextArea controls.
- **Header Part**: A String that specifies the headerTextPart or headerIconPart for HeaderClick actions on AdvancedDataGrid and OLAPDataGrid controls.
- **Hit Set**: A Double that represents chart series data point for ClickSeries, DoubleClick, or ItemRollOver actions on AreaSeries, BarSeries, BubbleSeries, ChartSeries, ColumnSeries, LineSeries, PieSeries, and PlotSeries controls.
- **Item Renderer**: A String that specifies the item within Combo, Grid, Tree, and List-type controls.
- **Key Code**: A String that specifies the key code to use for Type actions on controls. The Key Codes correspond to the Adobe ActionScript key codes.
- **Key Modifier**: Specifies the modifier to use for keyboard trigger events. Keyboard actions can include modifiers such as Ctrl, Alt, Shift, or combinations of modifiers such as Ctrl-Alt, or Shift-Alt.
  - **None**: When selected, no keyboard modifier is used for keyboard trigger events.
  - **Ctrl**: When selected, the Ctrl key is used as the keyboard modifier for keyboard trigger events. For example, Ctrl-*key.*
  - **CtrlShift**: When selected, the Ctrl-Shift key combination is used as the keyboard modifier for keyboard trigger events. For example, Ctrl-Shift-*key*.
  - **Alt**: When selected, the Alt key is used as the keyboard modifier for keyboard trigger events. For example, Alt-*key*.
  - **CtrlAlt**: When selected, the Ctrl-Alt key combination is used as the keyboard modifier for keyboard trigger events. For example, Ctrl-Alt-*key*.
  - **ShiftAlt**: When selected, the Shift-Alt key combination is used as the keyboard modifier for keyboard trigger events. For example, Shift-Alt-*key*.
  - **CtrlShiftAlt**: When selected, the Ctrl-Shift-Alt key combination is used as the keyboard modifier for keyboard trigger events. For example, Ctrl-Shift-Alt-*key*.
- **Local X**: Specifies the mouse coordinate X position within the control for ColumnStretch and MouseMove actions on controls.
- **Local Y**: Specifies the mouse coordinate X position within the control for MouseMove actions on controls.
- **Moving Column Index**: An Integer that specifies the index of the ColumGroupedADGHeader being shifted in AdvancedDataGrid and OLAPDataGrid controls.
- **New Column Index**: An Integer that specifies the new index of the ColumGroupedADGHeader being shifted in AdvancedDataGrid and OLAPDataGrid controls.
- **New Index**: An Integer that specifies the new index of the header being shifted in AdvancedDataGrid and OLAPDataGrid controls.
- **Old Column Index**: An Integer that specifies the old index of the olumGroupedADGHeader being shifted in AdvancedDataGrid and OLAPDataGrid controls.

- **Old Index**: An Integer that specifies the old index of the header being shifted in AdvancedDataGrid and OLAPDataGrid controls.
- **Position**: An Integer that specifies the position number within scroll-type controls.
- **Scroll Direction**: Specifies the direction to scroll an object.
  - **vertical:** When selected, the scroll action is vertical.
  - **horizontal**: When selected, the scroll action is horizontal.
- **Scroll Detail**: Specifies the scroll bar detail within scroll-type controls.
  - **atLeft**: When selected, the scroll bar is put to the left of its scrolling range.
  - **atTop**: When selected, the scroll bar is put to the top of its scrolling range.
  - **atRight**: When selected, the scroll bar is put to the right of its scrolling range.
  - **atBottom**: When selected, the scroll bar is put to the bottom of its scrolling range.
  - **lineLeft**: When selected, the scroll bar is moved one line to the left in the scrolling range.
  - **lineUp**: When selected, the scroll bar is moved one line up in the scrolling range.
  - **lineRight**: When selected, the scroll bar is moved one line to the right in the scrolling range.
  - **lineDown**: When selected, the scroll bar is moved one line down in the scrolling range.
  - **pageLeft**: When selected, the scroll bar is moved one page to the left in the scrolling range.
  - **pageUp**: When selected, the scroll bar is moved one page up in the scrolling range.
  - **pageRight:** When selected, the scroll bar is moved one page to the right in the scrolling range.
  - **pageDown**: When selected, the scroll bar is moved one page down in the scrolling range.
  - **thumbTrack**: When selected, the scroll action is moving.
  - **thumbPosition**: When selected, the scroll action is stopped.
- **Related Object**: Specifies a related object for container-type controls such as Accordion, ButtonBar, LinkBar, NavigationBar, NumericStepper, TabNavigator, and ToggleButtonBar.
- **Row Index**: An Integer that specifies the index of the row being edited in AdvancedDataGrid and OLAPDataGrid controls.
- **Shift Key**: Specifies if the Shift Key should be used with the Key code. Valid values are True and False. This setting is used with the ChangeFocus action in all control types.
- **Text**: A String that specifies the text or password text use for Input or Password Input actions on TextArea controls.
- **Thumb Index**: Specifies the index of the thumb on which to perform Change actions on Slider controls.
- **Trigger Event**: Specifies the action to use to trigger the event on the object.

  - **Mouse**: When selected, a mouse action triggers the event on the object.
  - **Keyboard**: When selected, a keyboard action triggers the event on the object. Specify the Key Modifier to use with the keyboard event trigger.
- **Value**: A Double that specifies the value to use for Change actions on Slider controls.

11.4.3 Using the Adobe Flex Functional Test Module API

The Adobe Flex Functional Test Module includes a script Application Programming Interface (API) specific to Adobe Flex functional testing. The Adobe Flex Functional Test Module recorder creates the Java code that corresponds to the Tree View and displays the Adobe Flex Functional Test commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the Adobe Flex Functional Test API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Adobe Flex Functional Testing Module are part of the "flexFT" class. Additional functional test methods are available in the "web" and "ft" classes. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the Adobe Flex Testing Module API include:

- Accordion
- AdvancedDataGrid
- Alert
- Application
- AreaChart
- AreaSeries
- AxisRenderer
- BarChart
- BarSeries Box
- BubbleSeries
- Button
- ButtonBar
- Canvas
- CartesianChart
- Chart
- ChartLegend
- ChartSeries
- Checkbox
- ColorPicker
- ColumnChart
- ColumnSeries

- ComboBase
- ComboBox
- Container
- DataGrid
- DateChoser
- DateField
- DisplayObject
- DivideBox
- Form
- FormItem
- HLOChart
- Image
- Label
- LineChart
- LineSeries
- LinkBar
- List
- ListBase
- ListLabel
- Loader
- Menu
- MenuBar
- NavigationBar
- NumericStepper
- OLAPDataGrid
- Panel
- PieChart
- PlotSeries
- PopUpButton
- ProgressBar
- RadioButton
- Repeater
- Rule
- ScrollBar
- ScrollBase
- Slider
- TabNavigator

- TextArea
- TitleWindow
- ToggleButtonBar
- Tree
- UIMovieClip
- VideoDisplay
- ViewStack

Many API methods can be added using the Adobe Flex Functional Test Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures. See the API Reference in the OpenScript help for additional programming information.

12

Using the Adobe Flex (AMF) Load Test Module

This chapter provides instructions on configuring and using the OpenScript Adobe Flex (AMF) Load Test Module, which provides support for load testing of Adobe Flex (AMF) web applications.

12.1 About the Adobe Flex (AMF) Load Test Module

The Adobe Flex (AMF) Load Test Module is an extension module to the OpenScript HTTP Module that extends the Web testing with Adobe Flex Action Message Format Load Test recording and playback capabilities. The Adobe Flex (AMF) Load Test Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager. The Adobe Flex (AMF) Load Test Module provides the following features:

The Adobe Flex (AMF) Load Test recorder displays commands in the Tree View in easy-to-understand commands. By default, script commands are grouped into Steps Groups by the Web page on which they were performed. Each Step Group contains one or more script commands corresponding to recorded actions that were performed on the page. The default name for the Step Group is the Web page Title (as specified in the "Title" tag) or the Flex application operation name.

OpenScript shows the results of Adobe Flex (AMF) Load Test script playback in the Results view. The Results view shows results for each script command (including duration and summary for failures). The Results Report compiles the same information into an HTML Results Report. Results can be exported from the OpenScript GUI in standard format (CSV / HTML). Results are also generated for unattended playback through the command line.

The Adobe Flex (AMF) Load Test Module API includes an "amf" class that provides additional programming functionality.

12.1.1 Key Features of the Adobe Flex (AMF) Load Test Module

- The Adobe Flex (AMF) Load Test Script Module. The New Project wizard (**New** from the **File** menu) includes an "Adobe Flex (AMF)" option in the Load Test Group to use when creating Adobe Flex (AMF) load testing projects in OpenScript. The Adobe Flex (AMF) Load Test Script Module records Adobe Flex (AMF) applications by converting binary AMF streams to XML/text format during recording to allow easy editing of parameters. XML/text format is converted back to binary AMF streams during playback.

- Correlation Library. The Adobe Flex (AMF) Load Test Module includes an Adobe Flex (AMF)-specific library of correlation rules for parameterizing scripts.
- Test Cases (Validation). The Adobe Flex (AMF) Load Test Module can perform Adobe Flex AMF requests, Text Matching tests, and solve variables by regular expressions or XPath.
- Adobe Flex (AMF)-Specific Application Programming Interface (API). The Adobe Flex (AMF) Load Test Module includes a Adobe Flex (AMF) Load Test Module API Specification that can be used to customize Adobe Flex (AMF)-specific scripts.

12.2 Recording Adobe Flex (AMF) Load Tests

The Adobe Flex (AMF) Load Test Module records Adobe Flex-based web applications that use the Action Message Format (AMF). The Recorder creates load test scripts for automating testing of Adobe Flex (AMF) applications.

Adobe Flex is a collection of technologies released by Adobe Systems for the development and deployment of cross platform, rich Internet applications based on the proprietary Adobe Flash platform. The OpenScript Adobe Flex (AMF) Load Test Module captures the binary AMF and converts the stream to human-readable XML to record interactions with those controls. Actions will be captured in the test script as OpenScript "amf" commands. Other components are standard Web controls which are captured as standard OpenScript "http" navigation commands. Correlation rules can be modified by users through the Preferences settings for new scripts.

The Adobe Flex (AMF) Load Test Module provides a record toolbar button that lets you initiate the Adobe Flex (AMF) recorder and capture Web/Adobe Flex (AMF) page actions to the script view. The record toolbar includes start and stop recording toolbar buttons. OpenScript recorders also open a floating toolbar that can be used while recording without having to switch between the browser and OpenScript.

12.2.1 Recording Adobe Flex (AMF) Load Test Scripts

To record Adobe Flex (AMF) Load Test scripts:

1. Start OpenScript.
2. Set the Adobe Flex (AMF) Load Test Correlation preferences.
3. Select **New** from the **File** menu.
4. Expand the Load Testing group.
5. Select **Adobe Flex (AMF)** (The Adobe Flex (AMF) script combines both HTTP and Adobe Flex (AMF) technologies as part of the same script).
6. Click **Next**.
7. Select the Repository and Workspace.
8. Enter a script name.
9. Click **Finish**. A new Script tree is created in the Script View.
10. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
11. Load the web page where you want to start recording into the browser.
12. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).

13. When finished navigating pages, close the browser.
14. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
15. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

12.3 Playing Back Scripts

OpenScript plays back recorded Adobe Flex (AMF) requests by converting the XML post message (for example: `amf.post("description","urlPath", "postMessageXML");`) back to a binary stream before sending it to the server. The actions used for playback will either be those that are recorded or specified manually in the Java Code view. Unattended playback is supported through Oracle Test Manager or third-party tools using OpenScript's command line interface.

The Adobe Flex (AMF) Load Test Module provides playback and iterate toolbar buttons that allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for Adobe Flex (AMF) Load scripts can be viewed in the Results and Console views.

12.3.1 Playing Back Adobe Flex (AMF) Load Scripts

To play back Adobe Flex (AMF) Load scripts:

1. Start OpenScript.
2. Open the Adobe Flex (AMF) Load script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

12.3.2 Playing Back Adobe Flex (AMF) Load Scripts with Iterations

To play back Adobe Flex (AMF) Load scripts with iterations:

1. Start OpenScript.
2. Open the Adobe Flex (AMF) Load script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.
5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.

7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

12.4 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

12.4.1 Adding Adobe Flex (AMF) Load Actions

The Adobe Flex (AMF) Load Module includes actions for Adobe Flex (AMF) requests that can be added to a script.

To add Adobe Flex (AMF) Load actions to a script:

1. Record an Adobe Flex (AMF) Load Test script.
2. Select the script node where you want to add the action.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Flex LT (AMF) node.
5. Expand an action node and select the action.
6. Click **OK**.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path.
8. Enter any required values to use for the object action.
9. Click **OK**.
10. For AMF Requests, enter the description, URL and XML post message for the request.
11. Click **OK**. The action node is added to the script tree.

    In the Java Code view, a `amf.post("`*`description`*`","`*`urlPath`*`", "`*`postMessageXML`*`")` method will be added to the script code (line breaks and spacing added for clarity):

```
amf.post("Click the button", "http://oracle-xyz:8080/openamf/gateway",
 "<?xml version=\"1.0\" encoding=\"utf-8\"?>\r\n
 <amf version=\"0\">
  <headers><header name=\"amf_server_debug\" required=\"true\">
   <asobject type=\"NetDebugConfig\">
    <entry key=\"m_debug\" type=\"boolean\">true</entry>
    <entry key=\"coldfusion\" type=\"boolean\">true</entry>
    <entry key=\"error\" type=\"boolean\">true</entry>
    <entry key=\"amf\" type=\"boolean\">false</entry>
    <entry key=\"trace\" type=\"boolean\">true</entry>
    <entry key=\"httpheaders\" type=\"boolean\">false</entry>
    <entry key=\"recordset\" type=\"boolean\">true</entry>
    <entry key=\"amfheaders\" type=\"boolean\">false</entry>
   </asobject>
  </header></headers>
```

```
 <bodies>
  <body operation=\"org.openamf.examples.Test3.getNumber\" response=\"/1\">
   <list><item type=\"double\">123.456</item></list>
  </body>
 </bodies>
</amf>");
```

12.4.2 Using the Adobe Flex (AMF) Load Test Module API

The Adobe Flex (AMF) Load Test Module includes a script Application Programming Interface (API) specific to Adobe Flex (AMF) load testing. The Adobe Flex (AMF) Load Test Module recorder creates the Java code that corresponds to the Tree View and displays the Adobe Flex (AMF) Load Test commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.<

You can use the Adobe Flex (AMF) Load Test API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Adobe Flex (AMF) Load Testing Module are part of the "amf" class. Additional test methods are available in the "http" class. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the Adobe Flex (AMF) Testing Module API include:

- AMF Request (post)
- Text Matching Test
- Solve Regular Expression Variable
- Solve XPath Variables

Many API methods can be added using the Adobe Flex (AMF) Load Test Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures. See the API Reference in the OpenScript help for additional programming information.

12.5 Setting Adobe Flex (AMF) Load Test Correlation Preferences

To set Setting Adobe Flex (AMF) Load Test Correlation preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Correlation category.
4. Select Adobe Flex LT (AMF).
5. Expand the Adobe Flex (AMF) Load Test library.
6. Select or clear the check boxes to enable or disable specific rules.
7. Click the **Add** or **Edit** buttons to modify rules in the library.
8. Click **OK**.

12.6 Adobe Flex (AMF) Load Test Correlation Library

The Adobe Flex (AMF) Load Test Module includes a correlation library that automatically recognizes and substitutes Adobe Flex (AMF) parameters with OpenScript variables and functions during script recording. The script nodes show the Adobe Flex (AMF) entities included in the page navigation. The Adobe Flex (AMF)

Load Testing recorder automatically recognizes and parameterizes Adobe Flex (AMF) entities for Load testing.

The tree view nodes show the recorded description for Adobe Flex (AMF) operations. The Java code view shows the post parameters for the AMF request. For example:

```
amf.solve("flex.dsid","<entry key=\"DSId\" type=\"string\">(.+?)</entry>",
  "DSId is a required field", false, AmfSource.Content, 0, null);

amf.post(10,"Operation('todoService.getList')",
 "http://oracle-xyz:8080/todolist-web/messagebroker/amf",
 "<?xml version=\"1.0\" encoding=\"utf-8\"?>\r\n
  <amf version=\"3\">
  <headers />
   <bodies>
    <body operation=\"null\" response=\"/2\">
     <list>
      <item type=\"object\">
       <flex.messaging.messages.RemotingMessage>
        <body type=\"object\"><list /></body>
        <clientId type=\"object\" isNull=\"true\" />
        <correlationId type=\"object\" isNull=\"true\" />
        <destination type=\"string\">todoService</destination>
        <headers type=\"object\">
         <map>
          <entry key=\"DSId\" type=\"string\">
            {{flex.dsid,FC6E956E-1309-78B0-B487-BB01891A57DD}}</entry>
          <entry key=\"DSEndpoint\" type=\"string\">channel-amf</entry>
         </map>
        </headers>
        <messageId type=\"string\">{{@guid()}}</messageId>
        <operation type=\"string\">getList</operation>
        <source type=\"string\" />
        <timeToLive type=\"int\">0.0</timeToLive>
        <timestamp type=\"int\">0.0</timestamp>
       </flex.messaging.messages.RemotingMessage>
      </item>
     </list>
    </body>
   </bodies>
  </amf>");
```

Additional libraries and rules can be added using the OpenScript Correlation library Preferences for Adobe Flex (AMF) Load Test.

The Adobe Flex (AMF) correlation library defines the correlation rules for Adobe Flex (AMF)-based applications. The correlation rules specify the variable names and regular expressions to use to replace dynamic data in Adobe Flex (AMF) applications and navigations. The default Adobe Flex (AMF) correlation library provided with the OpenScript Adobe Flex (AMF) Load Test Module includes the following correlation rules:

- **Java Session id - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `jsessionid=(.+?)(?:"|&;)` and replaces it with the variable name `http.jsessionid` in all locations.
- **Web Dom Correlation - DOM Correlation** - this rule implements the default Web Document Object Model correlation rules for Adobe Flex (AMF) applications.
- **Client Set Cookie - Client Set Cookie** - this rule type automatically transforms web page cookie objects with dynamic data.

- **Correlate Header - Correlate Headers** - this rule type automatically transforms web page header objects with dynamic data.
- **Time Stamp - Function/Text Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `timestamp=((\d{13,}))` and replaces it with the `{{@timestamp}}` function in the specified locations.
- **Correlate Cookie Header - Correlate Cookie Header** - this rule type automatically transforms web page cookie header objects with dynamic data.
- **Correlate Destination Id - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<entry key="DSId" type="string">(.+?)</entry>` and replaces it with the variable name `flex.dsid` in all locations.
- **Message Id - Function/Text Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<messageId type="string">((.+?))</messageId>` and replaces it with the `{{@guid}}` function in the specified locations.

13
Using the Hyperion Load Test Module

This chapter provides instructions on configuring and using the OpenScript Hyperion Load Test Module, which provides support for load testing of Hyperion applications.

13.1 About the Hyperion Load Test Module

The Hyperion Load Test Module is an extension module to the OpenScript HTTP Module that extends the Web testing with Hyperion-specific correlation rules.

13.1.1 Key Features of the Hyperion Load Test Module

- The Hyperion Load Test Script Module. The New Project wizard (**New** from the **File** menu) includes an "Oracle Hyperion" option in the Load Test Group to use when creating Hyperion load testing projects in OpenScript.
- Correlation Library. The Hyperion Load Test Module includes a Hyperion-specific library of correlation rules for parameterizing HTTP scripts.

13.2 Recording Hyperion Load Tests

The Hyperion Load Test Module records Hyperion-specific correlation rules into HTTP load testing scripts for automating testing of Hyperion EnterpriseOne applications.

13.2.1 Recording Hyperion Load Test Scripts

To record Hyperion Load Test scripts:

1. Start OpenScript.
2. Set the Hyperion Load Test Correlation preferences.
3. Select **New** from the **File** menu.
4. Expand the Load Testing group.
5. Select **Oracle Hyperion** (The Hyperion script combines both HTTP and Hyperion technologies as part of the same script).
6. Click **Next**.
7. Select the Repository and Workspace.
8. Enter a script name.
9. Click **Finish**. A new Script tree is created in the Script View.

10. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
11. Load the web page where you want to start recording into the browser.
12. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
13. When finished navigating pages, close the browser.
14. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
15. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

> **Note:** Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

13.3 Playing Back Scripts

OpenScript plays back recorded Hyperion actions. The actions used for playback will either be those that are recorded or specified manually in the Java Code view.

The Hyperion Load Test Module uses the OpenScript playback and iterate toolbar buttons which allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for Hyperion Load scripts can be viewed in the Results and Console views.

13.3.1 Playing Back Hyperion Load Scripts

To play back Hyperion Load scripts:

1. Start OpenScript.
2. Open the Hyperion Load script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

13.3.2 Playing Back Hyperion Load Scripts with Iterations

To play back Hyperion Load scripts with iterations:

1. Start OpenScript.
2. Open the Hyperion Load script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.
5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.

7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

13.4 Setting Hyperion Load Test Correlation Preferences

To set Setting Hyperion Load Test Correlation preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Correlation category.
4. Select Hyperion Load Testing Module.
5. Expand the Hyperion Load Test library.
6. Select or clear the check boxes to enable or disable specific rules.
7. Click the **Add** or **Edit** buttons to modify rules in the library.
8. Click **OK**.

13.5 Hyperion Load Test Correlation Library

The Hyperion Load Test Module includes a correlation library that automatically recognizes and substitutes Hyperion parameters with OpenScript variables and functions during script recording. The script nodes show the Hyperion entities included in the page navigation. The HTTP recorder automatically recognizes and parameterizes Hyperion entities for Load testing.

Additional libraries and rules can be added using the OpenScript Correlation library Preferences for Hyperion Load Test.

The Hyperion correlation library defines the correlation rules for Hyperion-based applications. The correlation rules specify the variable names and regular expressions to use to replace dynamic data in Hyperion applications and navigations. The default Hyperion correlation library provided with the OpenScript Hyperion Load Test Module includes the following correlation rules:

- **ssoToken- Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_multiline_<token><!\[CDATA\[(.+?)]]> </token>` and replaces it with the variable name `ssoToken` in all locations.
- **repositoryToken- Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_multiline_'sessionId':'(.+?)'` and replaces it with the variable name `repositoryToken` in all locations.
- **ssnkey - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `name="ssnkeystate" value="(.+?)" />` and replaces it with the variable name `ssnkey` in all locations.
- **viewState - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `ViewState" value="(.+?)">` and replaces it with the variable name `viewState` in all locations.

- **afrLoop - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_afrLoop=(.+?)"` and replaces it with the variable name `afrLoop` in all locations.
- **ctrlState - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_adf.ctrl-state=(.+?)">` and replaces it with the variable name `ctrlState` in all locations.
- **asserterToken - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_multiline_<assertertoken><!\[CDATA\[(.+?)]]></assertertoken>` and replaces it with the variable name `asserterToken` in all locations.
- **extraToken - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `ExtraToken = "(.+?)"` and replaces it with the variable name `extraToken` in all locations.
- **headers - Correlate Headers** - this rule implements the default Correlate Headers correlation rules for Hyperion applications that use dynamic headers.
- **bpmui - Variable Substitution** - this rule locates text in the Response Header matching the Regular Expression pattern `X-ORACLE-BPMUI-CSRF:(.*)(?:\\r|$)` and replaces it with the variable name `bpmui` in all locations.
- **reportId - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `REPORTID="(.+?)"` and replaces it with the variable name `reportId` in all locations.
- **SmartView sID - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<sID>(.+?)</sID>` and replaces it with the variable name `sID` in all locations.
- **SmartView sso - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<sso>(.+?)</sso>` and replaces it with the variable name `sso` in all locations.

14

Using the JD Edwards Functional Test Module

This chapter provides instructions on configuring and using the OpenScript JD Edwards EnterpriseOne Functional Test Module, which provides support for testing of JD Edwards EnterpriseOne applications with Grid controls.

14.1 About the JD Ewards Functional Test Module

The JD Edwards Functional Test Module is an extension module to the OpenScript Web Functional Test Module that extends the Web testing with support for functional test record/playback of JD Edwards Enterprise One Grid controls.

14.1.1 Key Features of the JD Edwards EnterpriseOne Functional Test Module

- The JD Edwards EnterpriseOne Functional Test Script Module. The New Project wizard (**New** from the **File** menu) includes an "Oracle JD Edwards EnterpriseOne" option in the Functional Test Group to use when creating JD Edwards EnterpriseOne functional testing projects in OpenScript.
- JD Edwards EnterpriseOne Functional-Specific Application Programming Interface (API). The JD Edwards EnterpriseOne Functional Test Module includes an JD Edwards Functional Test Module API Specification that can be used to customize JD Edwards EnterpriseOne Functional test-specific scripts.

14.2 Recording JD Edwards EnterpriseOne Functional Tests

The Oracle JD Edwards EnterpriseOne Functional Test Module records standard JD Edwards EnterpriseOne-specific components for JD Edwards EnterpriseOne 9.0 and 9.1. **Recording JD Edwards EnterpriseOne 9.0 is supported with Internet Explorer but is not supported with Firefox**. Recording JD Edwards EnterpriseOne 9.1 is supported with Internet Explorer and Firefox. The Recorder creates functional test scripts for automating testing of JD Edwards EnterpriseOne web applications.

Oracle JD Edwards EnterpriseOne Grid components are object based controls and the Oracle JD Edwards EnterpriseOne Functional Test Module provides the object/attribute information for OpenScript to record interactions with the Grid control. Actions will be captured in the test script as OpenScript "eone" commands. Other components are standard Web controls which are captured as standard OpenScript "web" and "ft" navigation commands.

OpenScript plays back recorded JD Edwards EnterpriseOne actions/commands which consist of an event plus an object identified by its attributes (for example:

```
eone.grid("/web:window[@index='0' or @title='Work With Leave
```

```
Verification Rules']/web:document[@index='7' or
@name='e1menuAppIframe']/web:EOneGrid[@gridId='0_1' or
@gridName='Work With Leave Verification Rules']").selectRow(3);.
```

The actions used for playback will either be those that are recorded or specified manually in the Java Code view. Unattended playback is supported through Oracle Test Manager or third-party tools using OpenScript's command line interface. Oracle JD Edwards EnterpriseOne Functional Test scripts do not play in Oracle Load Testing.

The Oracle JD Edwards EnterpriseOne Functional Test Module provides a record toolbar button that lets you initiate the JD Edwards EnterpriseOne recorder and capture Web/JD Edwards EnterpriseOne page actions to the script view. The record toolbar includes start and stop recording toolbar buttons. OpenScript recorders also open a floating toolbar that can be used while recording without having to switch between the browser and OpenScript.

14.2.1 Setting JD Edwards EnterpriseOne Functional Test Preferences

To set JD Edwards EnterpriseOne Functional Test preferences:

1. Start OpenScript.
2. Open or record an OracleJD Edwards EnterpriseOne script.
3. Select **OpenScript Preferences** from the **View** menu.
4. Expand the OpenScript node and the Record category.
5. Select **JDE EnterpriseOne Functional**.
6. Click the tab and set the preferences. See Section 2.5.6, "Oracle JDE EnterpriseOne Functional Test Preferences" for descriptions of the Record Preferences settings.
7. Click **OK**.

14.2.2 Adding/Editing Object Identifiers

The Oracle JD Edwards EnterpriseOne Functional Test Module uses object identification to specify attributes used to identify JD Edwards EnterpriseOne objects. The Oracle JD Edwards EnterpriseOne Functional Test Module uses the same predefined path attributes for common Web objects as the Web Functional Test Module; however, JD Edwards EnterpriseOne Functional Test Automation provides additional attributes to identify JD Edwards EnterpriseOne Functional controls. Object paths are specified in XPath format. For example the object identification path appears as follows in Java code commands:

```
"/web:EOneGrid[@gridId='0_1' or @gridName='Work With Leave Verification Rules']")
```

The full path to the control appears as follows:

```
eone.grid("/web:window[@index='0' " +
   "or @title='Work With Leave Verification Rules']" +
 "/web:document[@index='7' or @name='e1menuAppIframe']" +
 "/web:EOneGrid[@gridId='0_1' or @gridName='Work With Leave Verification Rules']")
 .selectRow(3);
```

You can set the default Web object attributes in the Oracle JD Edwards EnterpriseOne Functional Test Module Record Preferences. You can also edit object attributes in recorded scripts in the tree view or the code view.

In addition to the predefined object identification, you can add an Object Library to the script to record paths into a library file. Object Library files may be shared and reused

across other scripts. The Object Library files provide a more convenient "short name" for objects to provide for more convenient programming.

The Oracle JD Edwards EnterpriseOne Functional Test Module includes object identifiers that specify how the recorder identifies Browser objects. You can add object identifiers or edit the existing object identifiers in the Record preferences.

To add or edit an object identifier:

1. Select the **OpenScript Preferences** from the **View** menu.
2. Expand the Record node and select JDE EnterpriseOne Functional.
3. Click the **Object Identification** tab. This tab lets you specify the JDE EnterpriseOne object identification attributes, as follows:

   Active Profile: Specifies which object identification profile to use as the active profile when recording scripts. Profiles define a specific set of object identifiers to use when recording JD Edwards EnterpriseOne functional tests. Use the **Add Profile** option to create a new custom profile. Once you have created a profile, select the profile name in the **Name** column and use **Add Object** to define custom objects and attributes in the custom profile.

   Name: Shows the name(s) of the defined JD Edwards EnterpriseOne object identifiers.

   Attributes: Shows the pattern(s) specified for the defined JD Edwards EnterpriseOne object identifiers.

   Add Profile: Opens a dialog box for specifying a new JD Edwards EnterpriseOne object identifier profile.

   Add Object: Opens a dialog box for specifying a new JD Edwards EnterpriseOne object identifier.

   Edit: Opens a dialog box for editing the selected JD Edwards EnterpriseOne object identifier.

   Delete: Deletes the selected JD Edwards EnterpriseOne object identifier or custom profile. The default profile cannot be deleted.

   Export: Opens a dialog box for exporting the currently selected JD Edwards EnterpriseOne object identifier profile to an XML file.

   Import: Opens a dialog box for importing a saved object identifier profile XML file.

   Revert: Reverts the default JD Edwards EnterpriseOne object identification profile to the default profile. Any changes to the default profile are removed. Select the default profile name in the **Name** column to activate the revert option.

   For each object element, you specify a name (typically an JD Edwards EnterpriseOne object attribute), an operator, a value and a value type. As you add object elements, OpenScript builds the object identifier using logical OR between each object identifier element. Click **Edit** to change between logical OR and AND.
4. Click **Add** or select an existing object identifier and click **Edit**.
5. If adding a new object identifier, enter a name for the object identifier.
6. Add or edit object elements for the object identifier.

   See the Web Functional Test Module for additional information about adding and editing Object Identifiers.
7. Click **OK**. The object identifier is added to the record preferences.

14.2.3 Recording JD Edwards EnterpriseOne Functional Test Scripts

To record JD Edwards EnterpriseOne Functional Test scripts:

1. Start OpenScript.
2. Select **New** from the **File** menu.
3. Expand the Functional Testing group.
4. Select **Oracle JD Edwards EnterpriseOne** (The JD Edwards EnterpriseOne script combines both Web and JD Edwards EnterpriseOne technologies as part of the same script).
5. Click **Next**.
6. Select the Repository and Workspace.
7. Enter a script name.
8. Click **Finish**. A new Script tree is created in the Script View.
9. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
10. Load the web page where you want to start recording into the browser.
11. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
12. When finished navigating pages, close the browser.
13. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
14. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

14.3 Playing Back Scripts

OpenScript plays back recordedJD Edwards EnterpriseOne actions/commands which consist of an object identified by its attributes. The actions used for playback will either be those that are recorded or specified manually in the Java Code view.

The JD Edwards EnterpriseOne Functional Test Module uses the OpenScript playback and iterate toolbar buttons which allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for JD Edwards Enterprise One functional test scripts can be viewed in the Results and Console views.

14.3.1 Playing Back JD Edwards EnterpriseOne Functional Scripts

To play back JD Edwards EnterpriseOne Functional scripts:

1. Start OpenScript.

2. Open the JD Edwards EnterpriseOne functional test script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

14.3.2 Playing Back JD Edwards EnterpriseOne Functional Scripts with Iterations

To play back JD Edwards EnterpriseOne functional test scripts with iterations:

1. Start OpenScript.
2. Open the JD Edwards EnterpriseOne functional test script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.
5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.
7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

14.4 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

14.4.1 Capturing JD E EnterpriseOne Grid Control Attributes

The JD Edwards EnterpriseOne Module includes an option for capturing JD Edwards EnterpriseOne Grid contol objects that can be added to a script.

To capture a JD Edwards EnterpriseOne Grid control:

1. Record a JD EdwardsEnterpriseOne Functional Test script.
2. Click Inpect Path.
3. When the browser starts, load the JD Edwards EnterpriseOne application and log in.
4. Navigate to the grid control you want to capture.
5. Mouse over the grid and Press F10 to capture the control's object idenfication attributes.

14.4.2 Adding JD Edwards EnterpriseOne Grid Control Actions

The JD EdwardsEnterpriseOne Module includes actions for JD Edwards EnterpriseOne Grid contol objects that can be added to a script.

To add JD Edwards EnterpriseOne actions to a script:

1. Record a JD Edwards EnterpriseOne Functional Test script.
2. Select the script node where you want to add the action.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the EnterpriseOne Actions node.
5. Expand the EOneGrid node and select the action.
6. Click **OK**.
7. Enter the object identification path for the object. You can use the **Capture** or **Select** menu options to capture or select an object path.
8. Enter any required values to use for the object action.
9. Click **OK**. The action node is added to the script tree.

   In the Java Code view, an `eone.`*`object`*`(`*`objectId`*`).`*`action`*`()` method will be added to the script code:

```
eone.grid("/web:window[@index='0' " +
   "or @title='Work With Leave Verification Rules']" +
 "/web:document[@index='7' or @name='e1menuAppIframe']" +
 "/web:EOneGrid[@gridId='0_1'" +
   or @gridName='Work With Leave Verification Rules']")
 .selectRow(3);
```

   The EOneGrid Action node includes actions for objects such as clickAttachmentCell, clickCell, clickCellIcon, clickCellLink, collapseTreeNode, deselectAllRows, deselectRow, expandTreeNode, goNextPage, goNextRecordSection, goPreviousPage, goPreviousRecordSection, goToLastPage, launchCellVisualAssist, launchExportAssistant, launchImportAssistant, launchQBEVisualAssist, selectAllRows, selectComboboxCellByIndex, selectComboboxCellByText, selectRecordSection, selectRow, setCellTextFieldValue, setCellCheckBox, setFocusOnCell, setQBEValue, dragTo, waitFor.

14.4.3 Oracle JD Ewards EnterpriseOne Functional Test Module API

The Oracle JD Edwards EnterpriseOne Functional Test Module includes a script Application Programming Interface (API) specific to ADF functional testing. The Oracle JJD Edwards EnterpriseOne Functional Test Module recorder creates the Java code that corresponds to the Tree View and displays the Oracle JD Edwards EnterpriseOne Functional Test commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the Oracle JD Edwards EnterpriseOne Functional Test API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Oracle JD Edwards EnterpriseOne Functional Testing Module are part of the "eone" class. Additional test methods are available in the "web" or "ft" classes. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the Oracle JD Edwards EnterpriseOne Testing Module API include:

- clickAttachmentCell
- clickCell
- clickCellIcon
- clickCellLink

- collapseTreeNode
- deselectAllRows
- deselectRow
- expandTreeNode
- goNextPage
- goNextRecordSection
- goPreviousPage
- goPreviousRecordSection
- goToLastPage
- launchCellVisualAssist
- launchExportAssistant
- launchImportAssistant
- launchQBEVisualAssist
- selectAllRows
- selectComboboxCellByIndex
- selectComboboxCellByText
- selectRecordSection
- selectRow
- setCellTextFieldValue
- setCellCheckBox
- setFocusOnCell
- setQBEValue
- dragTo
- waitFor

Many API methods can be added using the Oracle JD Edwards EnterpriseOne Testing Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures. See the API Reference in the OpenScript help for additional programming information.

15
Using the JD Edwards Load Test Module

This chapter provides instructions on configuring and using the OpenScript JD Edwards Load Test Module, which provides support for load testing of JD Edwards EnterpriseOne applications.

15.1 About the JD Edwards Load Test Module

The JD Edwards Load Test Module is an extension module to the OpenScript HTTP Module that extends the Web testing with JD Edwards-specific correlation rules.

15.1.1 Key Features of the JD Edwards Load Test Module

- The JD Edwards Load Test Script Module. The New Project wizard (**New** from the **File** menu) includes an "Oracle JD Edwards EnterpriseOne" option in the Load Test Group to use when creating JD Edwards load testing projects in OpenScript.
- Correlation Library. The JD Edwards Load Test Module includes a JD Edwards-specific library of correlation rules for parameterizing HTTP scripts.

15.2 Recording JD Edwards Load Tests

The JD Edwards Load Test Module records JD Edwards-specific correlation rules into HTTP load testing scripts for automating testing of JD Edwards EnterpriseOne applications.

15.2.1 Recording JD Edwards Load Test Scripts

To record JD Edwards Load Test scripts:

1. Start OpenScript.
2. Set the JD Edwards Load Test Correlation preferences.
3. Select **New** from the **File** menu.
4. Expand the Load Testing group.
5. Select **Oracle JD Edwards EnterpriseOne** (The JD Edwards script combines both HTTP and JD Edwards technologies as part of the same script).
6. Click **Next**.
7. Select the Repository and Workspace.
8. Enter a script name.
9. Click **Finish**. A new Script tree is created in the Script View.

10. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
11. Load the web page where you want to start recording into the browser.
12. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
13. When finished navigating pages, close the browser.
14. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
15. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

15.3 Playing Back Scripts

OpenScript plays back recorded JD Edwards actions. The actions used for playback will either be those that are recorded or specified manually in the Java Code view.

The JD Edwards Load Test Module uses the OpenScript playback and iterate toolbar buttons which allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for JD Edwards Load scripts can be viewed in the Results and Console views.

15.3.1 Playing Back JD Edwards Load Scripts

To play back JD Edwards Load scripts:

1. Start OpenScript.
2. Open the JD Edwards Load script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

15.3.2 Playing Back JD Edwards Load Scripts with Iterations

To play back JD Edwards Load scripts with iterations:

1. Start OpenScript.
2. Open the JD Edwards Load script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.
5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.

7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

15.4 Setting JD Edwards Load Test Correlation Preferences

To set Setting JD Edwards Load Test Correlation preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Correlation category.
4. Select JD Edwards Load Testing Module.
5. Expand the JD Edwards Load Test library.
6. Select or clear the check boxes to enable or disable specific rules.
7. Click the **Add** or **Edit** buttons to modify rules in the library.
8. Click **OK**.

15.5 JD Edwards Load Test Correlation Library

The JD Edwards Load Test Module includes a correlation library that automatically recognizes and substitutes JD Edwards parameters with OpenScript variables and functions during script recording. The script nodes show the JD Edwards entities included in the page navigation. The HTTP recorder automatically recognizes and parameterizes JD Edwards entities for Load testing.

Additional libraries and rules can be added using the OpenScript Correlation library Preferences for JD Edwards Load Test.

The JD Edwards correlation library defines the correlation rules for JD Edwards-based applications. The correlation rules specify the variable names and regular expressions to use to replace dynamic data in JD Edwards applications and navigations. The default JD Edwards correlation library provided with the OpenScript JD Edwards Load Test Module includes the following correlation rules:

- **Time Stamp - Function/Text Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `timestamp=((\d{13,}))` and replaces it with the function name `{{@timestamp}}` in the location specified by the Regular Expression `timestamp=((\d{13,}))`.
- **GLOBAL_RI - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<input type=hidden name="globalRID" value="(.+?)">` and replaces it with the variable name `GLOBAL_RID` in the location matching the Regular Expression `RID=((.+?))(&|$)`.
- **JdeMafJasLinkTarget - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `&jdemafjasLinkTarget=E1MENUMAIN_(.+?)&RENDER_MAFLET` and replaces it with the variable name `JdeMafJasLinkTarget` in the location matching the Regular Expression `jdemafjasLinkTarget=E1MENUMAIN_((.+?))(&|$`.

- **JdeMafJasCacheUid - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<input type=hidden name="jdemafjascacheUID" value="(.+?)">` and replaces it with the variable name `JdeMafJasCacheUid` in the location matching the Regular Expression `jdemafjascacheUID=((.+?))(&|$)`.
- **JdeMafJasUID - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<input type=hidden name="jdemafjasUID" value="(.+?)">` and replaces it with the variable name `JdeMafJasUID` in the location matching the Regular Expression `jdemafjasUID=((.+?))(&|$)`.

16
Using the PeopleSoft Load Test Module

This chapter provides instructions on configuring and using the OpenScript PeopleSoft Load Test Module, which provides support for load testing of PeopleSoft applications.

16.1 About the PeopleSoft Load Test Module

The PeopleSoft Load Test Module is an extension module to the OpenScript HTTP Module that extends the Web testing with PeopleSoft-specific correlation rules.

16.1.1 Key Features of the PeopleSoft Load Test Module

- The PeopleSoft Load Test Script Module. The New Project wizard (**New** from the **File** menu) includes an "Oracle PeopleSoft" option in the Load Test Group to use when creating PeopleSoft load testing projects in OpenScript.
- Correlation Library. The PeopleSoft Load Test Module includes a PeopleSoft-specific library of correlation rules for parameterizing HTTP scripts.

16.2 Recording PeopleSoft Load Tests

The PeopleSoft Load Test Module records PeopleSoft-specific correlation rules into HTTP load testing scripts for automating testing of PeopleSoft EnterpriseOne applications.

16.2.1 Recording PeopleSoft Load Test Scripts

To record PeopleSoft Load Test scripts:

1. Start OpenScript.
2. Set the PeopleSoft Load Test Correlation preferences.
3. Select **New** from the **File** menu.
4. Expand the Load Testing group.
5. Select **Oracle PeopleSoft** (The PeopleSoft script combines both HTTP and PeopleSoft technologies as part of the same script).
6. Click **Next**.
7. Select the Repository and Workspace.
8. Enter a script name.
9. Click **Finish**. A new Script tree is created in the Script View.

10. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
11. Load the web page where you want to start recording into the browser.
12. Navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
13. When finished navigating pages, close the browser.
14. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
15. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

16.3 Playing Back Scripts

OpenScript plays back recorded PeopleSoft actions. The actions used for playback will either be those that are recorded or specified manually in the Java Code view.

The PeopleSoft Load Test Module uses the OpenScript playback and iterate toolbar buttons which allows users to start the script playback for either a single playback through the script or multiple iterations using data from a databank file. Playback results for PeopleSoft Load scripts can be viewed in the Results and Console views.

16.3.1 Playing Back PeopleSoft Load Scripts

To play back PeopleSoft Load scripts:

1. Start OpenScript.
2. Open the PeopleSoft Load script to play back.
3. Select **Playback** from the **Script** menu or click the toolbar button.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

16.3.2 Playing Back PeopleSoft Load Scripts with Iterations

To play back PeopleSoft Load scripts with iterations:

1. Start OpenScript.
2. Open the PeopleSoft Load script to play back.
3. Select **Iterate** from the **Script** menu or click the toolbar button.
4. Select **Use Databanks**.
5. Select which databank file to specify the settings for if more than one database is configured for the script.
6. Specify the settings for the databank file.

7. Select the **Run no more than [ ] iterations** option and set the iteration count to the desired number of playback iterations. See Section 4.2.4, "Playing Back Scripts With Iterations" for additional information about iteration settings.
8. Click **OK**.

   You can view the progress of the script playback in the Console View. You can review the results of script playback in the Results View.

16.4 Setting PeopleSoft Load Test Correlation Preferences

To set Setting PeopleSoft Load Test Correlation preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Correlation category.
4. Select PeopleSoft Load Testing Module.
5. Expand the PeopleSoft Load Test library.
6. Select or clear the check boxes to enable or disable specific rules.
7. Click the **Add** or **Edit** buttons to modify rules in the library.
8. Click **OK**.

16.5 PeopleSoft Load Test Correlation Library

The PeopleSoft Load Test Module includes a correlation library that automatically recognizes and substitutes PeopleSoft parameters with OpenScript variables and functions during script recording. The script nodes show the PeopleSoft entities included in the page navigation. The HTTP recorder automatically recognizes and parameterizes PeopleSoft entities for Load testing.

Additional libraries and rules can be added using the OpenScript Correlation library Preferences for PeopleSoft Load Test.

The PeopleSoft correlation library defines the correlation rules for PeopleSoft-based applications. The correlation rules specify the variable names and regular expressions to use to replace dynamic data in PeopleSoft applications and navigations. The default PeopleSoft correlation library provided with the OpenScript PeopleSoft Load Test Module includes the following correlation rules:

- **ICStateNum_v1 - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<input type='hidden' name='ICStateNum' id='ICStateNum' value='(.+?)' />` and replaces it with the variable name `ICStateNum_v1` in the location matching the Regular Expression `&ICStateNum=((.+?))&ICAction`.
- **ICSID_v1 - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<input type='hidden' name='ICSID' value='(.+?)' />` and replaces it with the variable name `ICSID_v1` in all locations.
- **ICStateNum_v2 - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `ICStateNum\.value=(.+?); initVars_win` and replaces it with the variable name `ICStateNum_v2` in the location matching the Regular Expression `&ICStateNum=((.+?))&`.

- **ICSID_v2 - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `<input type='hidden' name='ICSID' id='ICSID' value='(.+?)' />` and replaces it with the variable name `ICSID_v2` in all locations.
- **ViewAttach - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `\?cmd=viewattach&userfile=(.+?)&` and replaces it with the variable name `ViewAttach` in all locations.
- **SDirName - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `&sdirname=(.+?)'` and replaces it with the variable name `SDirName` in all locations.
- **PSFileProc - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `psfileproc/(.+?)\.pdf` and replaces it with the variable name `PSFileProc` in all locations.

17
Using the Web Services Module

This chapter provides instructions on using the OpenScript Web Services Module, which supports testing of Web Services.

17.1 About the Web Services Module

The Web Services Module is an application module that supports testing of Web Services. The Web Services Module is an extension to the HTTP Module. The OpenScript Web Services module includes the following features:

- The Web Services Module. The New Project wizard (select **New** from the **File** menu) includes a "Web Services Script" option to use when creating Web Services scripts in OpenScript.
- Support for SOAP Protocols. The Web Service Module supports the SOAP 1.1 and 1.2 protocols.
- Support for multiple parsers. The Web Service Module supports the multiple WSDL parsers. In addition to the default OpenScript and Oracle parsers, OpenScript can also be configure to use apache AXIS and .Net parsers.
- WSDL Manager. The OpenScript WSDL Manager lets you import and store Web Services Definition files for creating Web Services scripts. Using the WSDL Manager, you add Web Services methods to the OpenScript tree.
- XML Editor. The OpenScript XML Editor lets you edit Web Services requests to include either static values or variables.
- Web Services-specific Application Programming Interface (API). The Web Services Module includes a Web Services Module API Specification that can be used to customize Web Services scripts.

17.1.1 Key Features of the Web Services Module

The Web Services Module is an extension module to HTTP Module that extends the platform with Web Services testing capabilities. The Web Services Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager.

The Web Services methods are added to the Script tree using the WSDL Manager. Web Services method postdata parameters can be edited using the XML Editor features of the Details View.

OpenScript shows the results of Web Services script playback in the Results view. The Results view shows results for each script command (including duration and

summary for failures). The Results Report compiles the same information into an HTML Results Report. Results can be exported from the OpenScript GUI in standard format (CSV / HTML). Results are also generated for unattended playback through the command line.

The Web Services Module API includes a "ws" class that provides additional programming functionality.

17.2 Creating Web Services Scripts Using WSDL Manager

Creating Web Services scripts using WSDL Manager involves the following major steps:

1. Create a Web Services script tree.
2. Add WSDL files to the WSDL Manager view.
3. Add methods from the WSDL Manager to the script tree.
4. Edit method parameters in the Details view.

The following sections explain each of the major steps.

17.2.1 Creating the Web Services Script Tree

To create a Web Services script tree:

1. Start OpenScript.
2. Select **New** from the **File** menu.
3. Expand the General group and select **Web Services**.
4. Click **Next**.
5. Select the Repository and Workspace.
6. Enter a script name.
7. Click **Finish**. A new Script tree is created in the Script View and the WSDL Manager view appears.

17.2.2 Adding WSDL Files to the WSDL Manager View

To add files to the WSDL Manager view:

1. Click the Add icon in the WSDL Manager view.
2. Enter the URL to a the WSDL file or click **Browse** to select a local file.
3. Select the parser to use and set the **Roll over** option.
4. Click **Next**. The parsed methods appear.
5. Click **Finish** to add the parsed methods to the WSDL Manager view.

17.2.3 Adding Methods to the Script Tree

To add WSDL file methods from the WSDL Manager view to the script tree:

1. Expand the WSDL file tree in the WSDL Manager view.
2. Right-click the method to add and select **Add to script** from the shortcut menu. The method will be added to the **Run** node of the script tree.

In the Java Code view, a `ws.method(method)/ws.endMethod()` group with a `ws.post()` method will be added to the script code, as follows:

```
ws.method("findApprovedPatientsByLastName");
{
  ws.post(2, "http://server:7011/medrec-jaxws-services/PatientFacadeService",
    "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>
    <soapenv:Envelope
      xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
      xmlns:med=\"http://www.oracle.com/medrec\">\r\n
        <soapenv:Header/>\r\n
          <soapenv:Body>\r\n
            <med:findApprovedPatientsByLastName>\r\n
             <!--Optional:-->\r\n
             <arg0>String</arg0>\r\n
            </med:findApprovedPatientsByLastName>\r\n
          </soapenv:Body>\r\n
    </soapenv:Envelope>", null, true, null, null);
}
ws.endMethod();
```

17.2.4 Editing Method Parameters in the Details View

To edit WSDL file methods in the Details View:

1. Expand the **Run** node in the Script tree view.
2. Expand the WSDL method node in the Script tree view.
3. Select an XML post data node in the Script tree view.
4. Open the Details view and select the XML Tree tab.
5. Click a value in the right column of the XML Tree tab to edit the value.

   or

   Right-click a parameter in the left column of the XML Tree tab and select **Substitute Variable** to select a variable name or Databank value to substitute for the parameter value. If you parameterize a value with a Databank, the databank variable appears as `{{db.databankFileName.field,recordedValue}}` in the SOAP parameters. For example, the optional argument `<arg0>` in the above postdata example would appear as `<arg0>{{db.customer.LastName,String}}</arg0>\r\n`.

   The XML source in the XML tab is not decoded by default. Right-click the XML source in the XML tab and select **Format** to format the contents. When you format the xml contents, the inner text of an element will be decoded. The following is an example of the inner text of an element before formatting:

   ```
   <soapenv name="google&amp;quot;">google&amp;quot;</soapenv>
   ```

   The following is an example of the inner text of an element after formatting:

   ```
   <soapenv name="google&amp;quot;">google"</soapenv>
   ```

17.3 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

17.3.1 Adding a Web Services Post Navigation

To add a Web Services Post Navigation to a script:

1. Create a Web Services script.
2. Select the **Run** node.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the HTTP node.
5. Select the Web Services Post Navigation node from the Web Services group and click **OK**.
6. On the **Base URL** tab, enter the URL, request and response charsets, and set the **Encode strings** option.
7. On the **Post Data** tab, enter the SOAP protocol postdata XML.
8. On the **Headers** tab, use the **Add** button to add name/value pairs and actions to the Base URL.
9. Click **OK** to add the Web Services Post Navigation node to the script tree.

   In the Java Code view, the Web Services Post Navigation consists of the code executed in the `ws.Post` method (line breaks and spacing added for clarity):

```
ws.post(2, "http://testserver2/EmployeeLookup/EmployeeLookup.asmx",
  "<?xml version=\"1.0\" encoding=\"utf-8\"?>\r\n\r\n
    <soapenv:Envelope
       xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
       xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
       xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"
       xmlns:web=\"http://oracle.com/webservices\"> \r\n
     <soapenv:Header/> \r\n
       <soapenv:Body> \r\n
         <web:findEmployee soapenv:
           encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\"> \r\n
           <criteria xmlns:enc=\"http://oracle.com/webservices/encodedTypes\"
             xsi:type=\"enc:SearchCriteria\"> \r\n
             <FirstName xsi:type=\"xsd:string\">string</FirstName> \r\n
             <LastName xsi:type=\"xsd:string\">string</LastName> \r\n
             <EmployeeID xsi:type=\"xsd:int\">3</EmployeeID> \r\n
           </criteria> \r\n
         </web:findEmployee> \r\n
       </soapenv:Body> \r\n
    </soapenv:Envelope>",
  http.headers(http.header("Content-Type", "text/xml",
    Header.HeaderAction.Modify),
    http.header("SOAPAction","\"http://oracle.com/webservices/findEmployee\"",
      Header.HeaderAction.Add)),
  false, "UTF-8", "UTF-8");
```

17.3.2 Adding a Text Matching Test

You can use Text Matching Tests to report an error and/or abort the script if a request does not match the Text Matching Test criteria.

To add a Text Matching Test to a Web Services script:

1. Create a Web Services script.
2. Expand the **Run** node.

3. Select the Web Services postdata node where you want to add the Text Matching test.
4. Select the **Script** menu and then select **Other** from the **Add** menu.
5. Select **Text Matching Test** from the Validation group.
6. Enter a name for the test.
7. Enter the Text to Match.
8. Enter any error message text to log if the test fails.
9. Select the source location to look for the text to match: HTML or Response Header.
10. Select the **Pass when** setting.

    Selected text is present: the test case passes if the **Text to Match** string is found in the selected source.

    Selected text is absent: the test case passes if the **Text to Match** string is not found in the selected source.
11. Select the **Regular Expression** option if the Text to Match is a Regular Expression. Clear the **Regular Expression** option if the Text to Match is plain text.
12. Click **OK** to add the Text Matching Test node to the script tree.

    In the Java Code view, the Text Matching Test consists of the code executed in the `http.match` method (line breaks and spacing added for clarity):

```
http.match("Test name", "Text to Match", "Error Message",
      Source location = Source.Html | Source.ResponseHeader,
      pass when present = false | pass when absent = true,
      is not RegExp = false | is RegExp = true);
```

    Example:

```
http.match("MyTXTMatch", "Login", "Could not find login", Source.Html, false,
false);
```

Set the default Error recovery setting for the HTTP Text Matching test in the OpenScript playback preferences.

17.3.3 Adding Security Extensions

You can add security extensions to Web Services scripts.

To add security extensions to a Web Services script:

1. Create a Web Services script.
2. Expand the **Run** node.
3. Select the Web Services method node where you want to add the security and attachments.
4. Select the **Script** menu and then select **Other** from the **Add** menu.
5. Expand the HTTP node.
6. Select **Web Services Security Attachments** from the Web Services group and click **OK**.
7. If necessary click the **WS-Security** tab.

8. Enter a URL. If you selected a Web Services navigation node in the script tree, the URL will be automatically entered.
9. Select **User Username Token**
10. Enter the user name and password.

    Username: specifies the user name to use for the Username Token in the XML request.

    Password: specifies the password to use for the Username Token in the XML request.

    Confirm Password: confirms the password.
11. Select the password type: Password Text or Password Digest.

    Password Text: when selected, the password in the XML request is included as plain text. The URI attribute for the <wsse:Password> element is set to #PasswordText.

    Password Digest: when selected, the password is encrypted. The URI attribute for the <wsse:Password> element is set to #PasswordDigest.
12. Select or clear the **Add Created Header**, **Add Nonce** and **Add Timestamp** options.

    Add Created Header: when selected, a creation timestamp is included in the Username Token of the XML request for use in setting the server cache limit of used nonces.

    Add Nonce: when selected, a cryptographically random nonce value is included in the Username Token of the XML request to provide a countermeasure for replay attacks.

    Add Timestamp: when selected, a timestamp value is included in the Web Services security element of the XML request. The timestamp includes both Created and Expires elements. Specify the **Valid For** number of seconds.
13. Click **OK** to add the Security Attachment node to the script tree.

    In the Java Code view, the Security Attachment consists of the code executed in the `ws.addSecurityAttachments` method (line breaks and spacing added for clarity), as follows:

```
ws.addSecurityAttachments("url",
  ws.security("userName", deobfuscate("password"),addCreatedHeader,
    addNonce, addTimestamp, validFor), null);
```

    If you add security and file attachments together, the `ws.addSecurityAttachments` method includes both the `ws.security` and `ws.attachments` methods (line breaks and spacing added for clarity), as follows:

```
ws.addSecurityAttachments("url",
   ws.security("userName", deobfuscate("password"),true, true, true, 10),
   ws.attachments(AttachmentMechanism.transferType,
     ws.attachment("filename","attachmentPart")));
```

17.3.4 Adding Attachments

You can add file attachments to Web Services scripts.

To add file attachments to a Web Services script:

1. Create a Web Services script.

2. Expand the **Run** node.
3. Select the Web Services method node where you want to add the security and attachments.
4. Select the **Script** menu and then select **Other** from the **Add** menu.
5. Select **Web Services Security Attachments** from the Web Services group.
6. If necessary click the **WS-Security** tab.
7. Enter a URL. If you selected a Web Services navigation node in the script tree, the URL will be automatically entered.
8. Click the **Attachments** tab.
9. Select the **Transfer Type**.
   - DEFAULT - uses the default transfer type specified by the Content-Type header.
   - SWA - Security SOAP Messages with Attachments
   - MTOM - SOAP Message Transmission Optimization Mechanism
   - DIME - Direct Internet Message Encapsulation
10. Click **Add**.
11. Enter the path and file name or click **Browse** to select a file.
12. If the Web Services method includes any Attachment Part object identifiers, select an Attachment Part from the list. If the Web Services method does not include any Attachment Part object identifiers, the list will be empty.
13. Click **OK** to add the Security Attachment node to the script tree.

    In the Java Code view, the Security Attachment consists of the code executed in the `ws.addSecurityAttachments` method (line breaks and spacing added for clarity) as follows:

```
ws.addSecurityAttachments("url", null,
   ws.attachments(AttachmentMechanism.transferType,
     ws.attachment("filename","attachmentPart")));
```

    If you add security and file attachments together, the `ws.addSecurityAttachments` method includes both the `ws.security` and `ws.attachments` methods (line breaks and spacing added for clarity), as follows:

```
ws.addSecurityAttachments("url",
   ws.security("userName", deobfuscate("password"),true, true, true, 10),
   ws.attachments(AttachmentMechanism.transferType,
     ws.attachment("filename","attachmentPart")));
```

    The following example Web Services script method shows the `ws.addSecurityAttachments` method with a `ws.post` postdata method used to upload a file. The `ws.post` method specifies the SOAP Envelope postdata, Content-Type, and SOAP Action.

```
ws.method("upload");
{
  ws.addSecurityAttachments("http://myurl.com:8080/services/MTOMService",
    ws.security("username",deobfuscate("5blNah5kX/XuZnepYwInFw=="),
      true, true, true, 20),
    ws.attachments(AttachmentMechanism.MTOM,
      ws.attachment("C:\\OracleATS\\OFT\\test.txt", "<upload>776598931581")));
```

```
  ws.post(15, "http://myurl.com:8080/services/MTOMService",
    "<soapenv:Envelope
        xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\"
        xmlns:ser=\"http://service.interop.mtom.sample\">\r\n
      <soapenv:Header/>\r\n
      <soapenv:Body>\r\n
        <ser:upload>\r\n
          <!--Optional:-->\r\n
          <ser:fileName>string</ser:fileName>\r\n
          <!--Optional:-->\r\n
          <ser:contents>cid:776598931581</ser:contents>\r\n
        </ser:upload>\r\n
      </soapenv:Body>\r\n
    </soapenv:Envelope>",
    http.headers(http.header("Content-Type", "text/xml;charset=UTF-8",
        Header.HeaderAction.Modify),
      http.header("SOAPAction", "\"urn:upload\"",
        Header.HeaderAction.Modify)),
    true, null, null);
}
ws.endMethod();
```

17.3.5 Web Services Module API

The Web Services Module includes a script Application Programming Interface (API) specific to Web Services testing. The Web Services script creates the Java code that corresponds to the Tree View and displays the Web Services commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the Web Services API to enhance scripts with additional testing functionality. Commands that are specific to the Web Services Module are part of the "ws" class. Additional functional test methods are available in the "http" class. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Many API methods can be added using the Web Services Test Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures. See the API Reference in the OpenScript help for additional programming information.

17.4 Recording Web Services Scripts

If you have a Web Services client application written already that communicates over HTTP and which communicates through a proxy, you can record the traffic using thee OpenScript HTTP recorder.

17.4.1 Setting Web Services Record Preferences

To set Web Srvices record preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Record category.
4. Select **Web Services**.

5. Click the tabs and set the preferences.
6. Click **OK**.

17.4.2 Recording Web Services Scripts

To record Web Services script:

1. Start OpenScript.
2. Set the Web Services Recording preferences.
3. Select **New** from the **File** menu.
4. Expand the General group and select **Web Services**.
5. Click **Next**.
6. Select the Repository and Workspace.
7. Enter a script name.
8. Click **Finish**. A new Script tree is created in the Script View and the WSDL Manager view appears.
9. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
10. Load the web page where you want to start recording into the browser.
11. Navigate the web site to record navigations. The navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
12. When finished navigating pages, close the browser.
13. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
14. Expand the **Run** node of the script to view the page navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

    Note: Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

18

Using the Siebel Functional Test Module

This chapter provides instructions on configuring and using the OpenScript Siebel Functional Test Module, which tests Siebel applications by accessing objects through the Document Object Model (DOM) of the Web browser and the Siebel test automation framework.

18.1 About the Siebel Functional Test Module

The Siebel Functional Test Module provides support for functional testing of Siebel web applications. The Siebel Functional Test Module is an extension module to the OpenScript Web Functional Test Module that extends the Web testing with Siebel Functional Test recording and playback capabilities. The Siebel Functional Test Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager.

18.1.1 Key Features of the Siebel Functional Test Module

The Siebel Functional Test Module includes the following features:

- Records Standard Siebel High Interactivity (HI) and Standard Interactivity (SI) components for Siebel versions 7.7, 7.8, 8.0 and 8.1 through integration with Siebel Test Automation CAS Library.
- Plays back recorded Siebel actions/commands which consist of an event plus object identified by its attributes (for example: `GotoScreenlink pageTabs("SiebePageTabs") Accounts Screen`).
- Provides full script code view integration to support script generation for the Siebel Functional Test Module. The Siebel Functional Test Module includes an additional API to support Siebel Functional Test protocol code scripting.
- Allows users to parameterize user inputs to Siebel Functional Test scripts and drive those inputs from an external data file (Databank).
- Allows users to insert Tests to validate Siebel HI and SI content on playback.
- Provides additional options/settings that are specific to Siebel Functional Test scripts within the Siebel Functional Test categories in the preferences interface.
- Reports playback results for Siebel Functional Test scripts in the Results and Console views.
- The Siebel Functional Test Script Module API. The Siebel Functional Test Application Programming Interface include Java code methods specific to functional testing of Siebel applications.

The New Project wizard (select **New** from the **File** menu) includes a "Siebel Functional Test Script" option to use when creating Siebel functional testing projects in OpenScript. The Siebel Functional Test Script Module records Siebel applications using the Siebel test automation framework. OpenScript captures user actions and records them to the OpenScript script nodes in a highly readable sequence of navigations and actions.

18.2 Functional Testing Siebel Applications

The following is an outline of the procedures and best practices used to perform functional testing of Siebel applications with the OpenScript application.

18.2.1 Prerequisites

The instructions in this document assume the following prerequisites:

- Testing hardware/environment is available.
- The Siebel applications are installed on a Siebel Server.
- The Oracle Application Testing Suite applications have been installed on a testing machine.
- The test machine can access the Siebel applications.
- Some steps may require system administrator level privileges for the Siebel Server.
- License for Siebel Test Automation required.

18.2.2 Setting up the Siebel Test Environment

The functional test environment should approximate as closely as possible a working Siebel deployment environment. However, hardware cost constraints may be a limiting factor. This section provides recommendations about the basic test system configuration. Additional test system configurations can be used based upon hardware and network availability.

The basic *n*-tier configuration should consist of the following systems:

- Web Server
- Siebel Server
- Database Server
- Database Storage

See the *Siebel Installation Guide* in the in the Siebel document bookshelf for information about hardware and Siebel installation requirements.

Notes:

- Siebel applications can contain High-Interactivity (HI) components, which use ActiveX controls, and Standard-Interactivity (SI) applications, which use standard HTML. Applications may also use a combination of HI and SI components. Testing methods vary depending upon the type of components being tested. See the "Automating Functional Tests" chapter in the *Testing Siebel eBusiness Applications* document in the Siebel document bookshelf for a description of the component types.

- In general, Siebel applications are more memory intensive than CPU intensive. If trade-offs need to be made in test hardware decisions, memory should be given higher consideration than CPU speed.
- Siebel web applications use Cookies to maintain the state information.

18.2.3 Enabling Siebel Test Automation

The Siebel test automation framework must be activated on the Siebel server for OpenScript to access it. Changing the Siebel configuration file may require system administrator level privileges for the Siebel server. A license is also required to use the Siebel Test Automation framework. Contact your account representative for additional information about licence requirements.

18.2.3.1 Siebel 7.x

To enable the test automation framework in Siebel 7.x:

1. Open the .CFG file for the Siebel application on the Siebel server.
2. Set the `EnableAutomation` and `AllowAnonUsers` switches in the `[SWE]` section as follows:

   ```
   [SWE]
   EnableAutomation = TRUE
   AllowAnonUsers = TRUE
   ...
   ```

   See the *Siebel Testing Siebel eBusiness Applications* documentation if you need to set up secure access to the Siebel Test Automation framework.
3. Restart the Siebel Server.

18.2.3.2 Siebel 8.x

To enable the test automation framework in Siebel 8.x:

1. Log into Siebel as Administrator.
2. Go to "Site Map".
3. Go to "Administration - Server Configuration".
4. Select "Call Center Object Manager" (provided you want to enable automation for Call Center).
5. Under list of Components, click the Parameters tab.
6. Find EnableAutomation and AllowAnonUsers parameters and set both to TRUE.
7. Restart the Siebel Server.

18.2.4 Script Creation Techniques

The following are tips and techniques to use when creating Siebel load test scripts using the OpenScript application:

- Disable browser caching to make sure the pages are returned from the server rather than the browser cache.
- Record actions from login through logout to make sure parameters are passed properly between page navigations.

- Record actions slowly in the Siebel environment to make sure the recorder records all actions to the OpenScript script. If possible, watch as the OpenScript script pages are added to the script tree.
- Siebel Popup windows may initially appear incorrect. Resize the window slightly to refresh the page in the popup window.
- Do not insert Siebel Tests in Siebel popup windows unless the test is necessary.
- Save the script in OpenScript using Save As on the File menu. When you save a Siebel proxy-recorded script, OpenScript automatically creates a Java Agent versions of the script in the workspace. Depending upon the size of the script, the file save operation may take some time.

18.2.5 Setting Browser Options

When recording and playing back scripts to test an application, you want to make sure the pages returned are from the server and not the browser cache. To verify or change the browser settings:

1. Open Internet Options from the Control Panel.
2. Click **Settings** in the **Temporary Internet files** section.
3. Select **Every visit to the page**.
4. Click **View Objects** in the **Temporary Internet files folder** section.
5. Verify that the downloaded Program files directory does not contain multiple versions of the Siebel High Interactivity Framework and Siebel Test Automation programs installed.
6. If necessary, remove the duplicate or older versions.
7. Close the Downloaded Program Files window.
8. Click **OK** to close the Temporary Settings.
9. Click **OK** to close the Internet Properties.

18.2.6 Starting the Siebel Application

When you start the Siebel application in the browser, the URL must include the Siebel Web Engine (SWE) command to generate the test automation framework information. The `AutoOn` Siebel Web Engine command (SWECmd) is added to the URL as follows (SWECmd is case-sensitive):

`http://`*`hostname`*`/`*`application`*`/start.swe`**`?SWECmd=AutoOn`**

where *`hostname`* is the machine name or IP address of the Siebel server and *`application`* is the name of the Siebel application. For example, *`application`* could be `callcenter` or `callcenter_enu` depending upon the Siebel version.

Enter the start URL and command into the browser address. for example:

```
http://siebelServer/callcenter_enu/start.swe?SWECmd=AutoOn
```

As you navigate pages, OpenScript records page navigation to the OpenScript script tree.

18.2.7 Determining a Siebel Component Type

Siebel applications can include High-Interactivity (HI) object and Standard-Interactivity (SI) object types. You can use the Inspect Path feature of OpenScript to determine the type of an object in an application.

1. Click the Inspect Path toolbar button or the Capture button to open the Select Object dialog box.
2. Move the mouse cursor over the page in the Siebel application to view the component type. The current component is highlighted in the OpenScript browser and the path appears in the Select Element dialog box. Siebel HI component types are indicated by `/siebelft:cas[ClassName=` in the **Path/Object** fields:
3. The path to HI component types is also referred to as the Object Descriptor String (ODS) and is used in OpenScript to recognize applets used with the Siebel application.

   SI component types are indicated by `/web:window[index='0']...` in the **Path** field.
4. Press F10 to capture the object path. You can copy the path from the dialog box using Ctrl-C and paste using Ctrl-V.

The following are examples of complete object paths (line breaks added for clarity):

High-Interactivity (HI) Path.

```
/siebelft:cas[ClassName='SiebApplication' and
        RepositoryName='Siebel Universal Agent']
   /siebelft:cas[ClassName='SiebScreen' and
           RepositoryName='Web Call Center Home Screen']
   /siebelft:cas[ClassName='SiebView' and
           RepositoryName='Home Page View (WCC)']
   /siebelft:cas[ClassName='SiebApplet' and
           RepositoryName='Sales Message Alert List Applet Tiny']
   /siebelft:cas[ClassName='SiebList' and
           RepositoryName='SiebList']
```

Standard-Interactivity (SI) Path:

```
/web:window[index='0']
   /web:document[index='10']
   /web:span[text='Search' or index='1']
```

or

```
/web:window[index='0']/web:document[index='10']
   /web:form[index='0' or name='SWEForm1_0']
   /web:input_text[id='s_1_1_16_0' or name='s_1_1_16_0' or index='1']
```

18.3 Recording Siebel Functional Test Scripts

The Siebel Functional Test Module records standard Siebel High Interactivity (HI) and Standard Interactivity (SI) components for Siebel versions 7.7, 7.8, 8.0 and 8.1 through integration with Siebel Test Automation CAS Library. Siebel Test Automation must be enabled on the Siebel server side in order to successfully record these events. The Recorder creates functional and regression test scripts for automating testing of Siebel applications.

Siebel HI components are Active-X based controls and Siebel Test Automation provides the object/attribute information for OpenScript to record interactions with

those controls. Actions on HI controls will be captured in the test script as OpenScript "siebelFT" commands. Siebel SI components are standard Web controls which are captured as standard OpenScript "web" commands using Web Functional Test object attributes; however, Siebel Test Automation may provide additional attributes to identify SI controls which take precedence over standard Web object/attributes. Object Identification attributes can later be modified by users through the Preferences global settings for new scripts or for already recorded commands in the tree view or code view. Recording can be configured through Internet Explorer only as Siebel does not support Firefox.

The Siebel Functional Test Module provides a record toolbar button that lets you initiate the Siebel recorder and capture Web/Siebel page actions to the script view. The record toolbar includes start and stop recording toolbar buttons. OpenScript recorders also open a floating toolbar that can be used while recording without having to switch between the browser and OpenScript.

Before recording Siebel Functional test scripts, make sure the Siebel test automation framework is activated on the Siebel server. See Functional Testing Siebel Applications for details about Prerequisites and the Siebel Test Environment.

18.3.1 Setting Siebel Functional Test Record Preferences

To set Siebel Functional Test record preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Record category.
4. Select **Siebel Functional Test**.
5. Set the General preferences. See Section 2.5.7, "Siebel Functional Test Preferences" for descriptions of the Record Preferences settings.
6. Click **OK**.

18.3.2 Adding/Editing SI Element and Site Map Link Paths

The Siebel Functional Test Module can use special object identifier paths when recording Standard Interactivity (SI) Web objects and Sitemap links. The object identifier paths specify the element attributes to use to identify Siebel SI controls or Sitemap links. The Siebel attributes are RN (repository name), OT (object type) and UN (unique name).

To add or edit an SI element or Sitemap link path:

1. Select the **OpenScript Preferences** from the **View** menu.
2. Expand the Record node and select Siebel Functional Test.
3. Click the **General** tab.
4. Click **Edit** for SI Element path or Sitemap path.
5. Click **Add** or select an existing attribute and click **Edit**.
6. If adding a new attribute, enter a name for the attribute.
7. Add or edit attributes for the path.

   For each attribute, you specify a name (typically a Siebel object attribute), an operator, a value and a value type. As you add attributes, OpenScript builds the

object identifier path using logical AND between each attribute. Click **Edit** to change between logical OR and AND.

8. Click **OK**. The object identifier path is updated in the record preferences.

18.3.3 Recording Siebel Functional Test Scripts

To record Siebel Functional Test Scripts:

1. Start OpenScript.
2. Select **New** from the **File** menu.
3. Select **Siebel Functional Test Script**.
4. Click **Next**.
5. Select the Repository and Workspace.
6. Enter a script name.
7. Click **Finish**. A new Script tree is created in the Script View.
8. Select **Record** from the **Script** menu. The browser automatically opens when you start recording.
9. Load the Siebel application using the AutoOn Siebel Web Engine command (`?SWECmd=AutoOn`) in the URL into the browser.
10. Log in and navigate the web site to record page objects, actions, and navigations. The page objects, actions, and navigations will be added to the node of the script tree specified by the **Set Record Section** setting (the **Run** node is the default).
11. When finished navigating pages, log out and close the browser.
12. Select **Stop** from the **Script** menu or click the Stop button on the OpenScript toolbar.
13. Expand the **Run** node of the script to view the page objects, actions, and navigation nodes in the script tree.

    You can customize the script using the menu options or the Code View for specific testing requirements.

> **Note:** Do not close the script editor view or script project while recording or playing back scripts. Doing so could result in unpredictable behavior in the OpenScript application.

18.4 Modifying Scripts

Once a script has been created/recorded, you can make modifications to customize the script for your specific testing needs.

18.4.1 Adding Siebel Actions

The Siebel Functional Test Module includes actions for Siebel objects that can be added to a script.

To add Siebel actions to a script:

1. Record a Siebel Functional Test script.
2. Select the script node where you want to add the action.

3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Siebel Functional Test node.
5. Expand an action node and select the action.
6. Click **OK**.

   The Siebel action dialog boxes let you define the action to perform during playback of a Siebel Functional Test script. This dialog box is used for most Siebel actions including Application, Button, Calculator, Calendar, Checkbox, Communications Toolbar, Currency, List, Menu, Page Tabs, PDQ, Rich Text, Screen Views, Task Assistant, Text, Text Area, Threadbar, Toolbar, Tree, and View Applets. Specific values may be required for specific actions.

   - **Action**: Shows the action to perform. Additional values for variables or attributes may be required depending upon the action to perform.
   - **Path**: Specify the object ID of the Siebel or Web object on which to perform the action. You can use the **Capture** or **Select** menu options to capture or select an object path.
   - **Value(s)**: Specify the variables or attributes as required for the specific action on an object.
7. Enter the object identification path for the object.
8. Enter any required values to use for the object action.
9. Click **OK**. The action node is added to the script tree.

   In the Java Code view, a `siebelFT.`*`object`*`(`*`objectId`*`).`*`action`*`()` method will be added to the script code:

```
siebelFT.menu(100,"/siebelft:cas[@ClassName='SiebApplication' and
   @RepositoryName='Siebel Universal Agent']
 /siebelft:cas[@ClassName='SiebMenu' and
   @RepositoryName='SiebMenu']").select("File\\\\File - Logout");
```

   The Siebel Functional Test node includes actions for objects such as Application, Calculator, Calendar, Communications Toolbar, Page Tabs, Task Assistant, Threadbar, and View Applets, etc. Other object actions have corresponding Java code methods.

18.4.2 Handling Non-Standard Siebel Dialog Boxes

Most dialog boxes invoked in Siebel applications will be dismissed or closed properly during script playback. However, in certain circumstances with customized Siebel applications, some non-standard Alert/Confirmation dialog boxes may not be dismissed or closed during script playback.

This may happen with custom dialog boxes written while customizing default Siebel applications if the Siebel developer implements the custom dialog box in a way that blocks callbacks from the server. Typically, the custom dialog boxes have the caption “Internet Explorer” or something other than "Siebel" (the default caption for default Siebel dialog boxes). The default Siebel dialog boxes are implemented in a way that does not block callbacks.

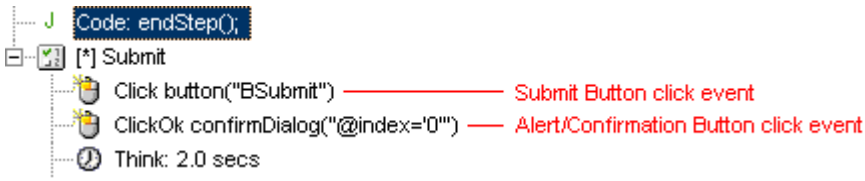
For example, when recording a Siebel transaction, at a certain point you click a button (or do something else) and an Alert/Confirmation dialog box appears. You dismiss the Alert/Confirmation dialog box by clicking OK and proceed further. When you play back the script, the script may halt because it is unable to dismiss or close the custom Alert/Confirmation dialog box.

When OpenScript clicks the Button, it waits for a callback from the Siebel server before considering the click event completed. If OpenScript does not receive a callback from the server, OpenScript considers click event to have failed and throws an exception and terminates script playback.

You can customize the script to handle non-standard Siebel dialog boxes during script playback. This involves adding custom Java code to the script to dismiss the dialog without relying on a callback call from the Siebel server.

In the Tree view, the click events for the dialog box actions will appear similar to the following click events:

Figure 18–1 Script Tree View for Click Events



In the Java Code view, the click events for the dialog box actions will appear similar to the following click events code:

```
siebelFT.button(311,"/siebelft:cas[@ClassName='SiebApplication' and
       @RepositoryName='Siebel Power Communications']
    /siebelft:cas[@ClassName='SiebScreen' and @RepositoryName='Orders']
    /siebelft:cas[@ClassName='SiebView' and
       @RepositoryName='Order Entry - Line Items View (Sales)']
    /siebelft:cas[@ClassName='SiebApplet' and
       @RepositoryName='Jawwal Order Entry - Order Form Applet Dashboard (Sales)']
    /siebelft:cas[@ClassName='SiebButton' and
       @RepositoryName='BSubmit']").click();

web.confirmDialog(312,"/web:dialog_confirm[@index='0' and
    @text='You are about to submit the order #: 1-3832871\n\n Do you want to
    continue?']").clickOk();
```

In the Java Code view, add the following code *before* the Submit button click event:

```
new Thread(new Runnable() {
     public void run() {
    try {
        Thread.sleep(4000);
        java.awt.Robot robot = new java.awt.Robot();
        robot.keyPress(java.awt.event.KeyEvent.VK_ENTER);
        robot.keyRelease(java.awt.event.KeyEvent.VK_ENTER);
        } catch(Exception x) {
             }
           }
         }
     ).start();
```

Next, comment or delete the code for the recorded click event for dismissing the Alert/Confirmation dialog box. The resulting code should appear similar to the following code:

```
//initiate a new thread which will dismiss the Alert/Confirmation dialog box
new Thread(new Runnable() {
     public void run() {
    try {
```

```
        Thread.sleep(4000);
        java.awt.Robot robot = new java.awt.Robot();
        robot.keyPress(java.awt.event.KeyEvent.VK_ENTER);
        robot.keyRelease(java.awt.event.KeyEvent.VK_ENTER);
        } catch(Exception x) {
             }
           }
         }
     ).start();

//invoke the dialog box that generates the Alert/Confirmation dialog box
siebelFT.button(311,"/siebelft:cas[@ClassName='SiebApplication' and
       @RepositoryName='Siebel Power Communications']
    /siebelft:cas[@ClassName='SiebScreen' and @RepositoryName='Orders']
    /siebelft:cas[@ClassName='SiebView' and
       @RepositoryName='Order Entry - Line Items View (Sales)']
    /siebelft:cas[@ClassName='SiebApplet' and
       @RepositoryName='Jawwal Order Entry - Order Form Applet Dashboard (Sales)']
    /siebelft:cas[@ClassName='SiebButton' and
       @RepositoryName='BSubmit']").click();
```

The above custom code does the following: before clicking the Siebel button that generates the Alert/Confirmation dialog box, a new thread is started. The thread sleeps for 4 seconds (4000 ms, which can be changed to your own delay). While the thread is asleep, the Siebel button gets clicked (it happens in the major thread, so no thread locks each other) and the Alert/Confirmation dialog box appears. The thread wakes up after the specified delay and invokes an ENTER key event (the same as a manual Enter key press). Since the Alert/Confirmation dialog box's OK button always has focus, it gets dismissed by the thread's ENTER key call.

18.4.3 Siebel Functional Test Module API

The Siebel Functional Test Module includes a script Application Programming Interface (API) specific to Siebel functional testing. The Siebel Functional Test Module recorder creates the Java code that corresponds to the Tree View and displays the Siebel Functional Test commands in the Java Code view using easy-to-understand function names. The Java Code view commands correspond to the Tree View and you can edit your scripts in either view.

You can use the Siebel Functional Test API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Siebel Functional Testing Module are part of the "siebelFT" class. Additional functional test methods are available in the "web" and "ft" classes. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the Siebel Testing Module API include:

- Applet
- Application
- Attribute
- Button
- Calculator
- Calendar
- Cells
- Checkbox

- Communications Toolbar
- Currency
- Page Tabs
- PDQ
- Pick List
- Rich Text
- Screen Views
- Task Assistant
- Text
- Text Area
- Threadbar
- Toolbar
- Tree
- View Applets

Many API methods can be added using the Siebel Functional Test Module Tree View. Additional methods can be added using the Java Code view. Use Ctrl-space in the Java Code view to open an Intellisense window listing available procedures. See the API Reference in the OpenScript help for additional programming information.

19
Using the Siebel Load Test Module

This chapter provides instructions on configuring and using the OpenScript Siebel Load Test Module, which tests Siebel-based applications by automating the underlying HTTP protocol traffic.

19.1 About the Siebel Load Test Module

The Siebel Load Test Module provides support for load testing of Siebel web applications. The Siebel module is an extension to the HTTP Module.

19.1.1 Key Features of the Siebel Load Test Module

The Siebel Load Test Module is an extension module to the OpenScript HTTP Module that extends Web load testing with Siebel Load Test recording and playback capabilities. The Siebel Load Test Module is fully integrated with the OpenScript platform including the Results view, Details view, Properties view, Console/Problems views, Preferences, Step Groups, Script Manager, and Workspace Manager. The OpenScript Siebel Load Test module includes the following features:

- The Siebel Load Test Script Module. The New Project wizard (Select **New** from the **File** menu.) includes a "Siebel Load Test Script" option to use when creating Siebel load testing projects in OpenScript. The Siebel Load Test Script Module records Siebel applications at the protocol level. OpenScript captures user actions and records them to the OpenScript script based upon HTTP requests and post data or query strings.
- Siebel-Specific Correlation Library. The Siebel module includes a transform library for automatically finding dynamic values inside recorded Siebel pages and substituting them into the appropriate Siebel HTTP requests.
- Siebel-Specific Correlation Rules. These rules define various transform rules for automatically finding/substituting Siebel parameters into a navigation. The Siebel rules are specified in a Siebel-specific correlation library that is added to the OpenScript correlation preferences.
- Siebel-Specific Application Programming Interface (API). The Siebel module includes a Siebel Module API Specification that can be used to customize Siebel-specific scripts.

The Siebel Load Test recorder displays commands in the Tree View in easy-to-understand commands. By default, script commands are grouped into Steps Groups by the Web page on which they were performed. Each Step Group contains one or more script commands corresponding to recorded actions that were performed on the page. The default name for the Step Group is the Web page Title (as specified in the "Title" tag).

OpenScript shows the results of Siebel Load Test script playback in the Results view. The Results view shows results for each script command (including duration and summary for failures). The Results Report compiles the same information into an HTML Results Report. Results can be exported from the OpenScript GUI in standard format (CSV / HTML). Results are also generated for unattended playback through the command line.

The Siebel Load Test Module API includes a "siebel" class that provides additional programming functionality.

19.1.2 Prerequisites

The instructions in this document assume the following prerequisites:

- Testing hardware/environment is available.
- The Siebel applications are installed on a Siebel Server.
- The Oracle Application Testing Suite have been installed on a testing machine.
- The test machine can access the Siebel applications.
- Some steps may require system administrator level privileges for the Siebel Server.

19.2 Load Testing Siebel Applications

The following is an outline of the procedures and best practices used to load test Siebel applications with the OpenScript application.

19.2.1 Setting Up Siebel Load Test Environments

The load test environment should approximate as closely as possible a working Siebel deployment environment. However, hardware cost constraints may be a limiting factor. The following sections provide recommendations about basic test system configurations. Additional test system configurations can be used based upon hardware and network availability.

19.2.1.1 Basic Configuration

The basic *n*-tier configuration should consist of the following systems:

- Web Server
- Siebel Server
- Database Server
- Database Storage

See the *Siebel Installation Guide* in the in the Siebel document bookshelf for information hardware and Siebel installation and any licensing requirements.

Note: In general, Siebel applications are more memory intensive than CPU intensive. If trade-offs need to be made in test hardware decisions, memory should be given higher consideration than CPU speed.

19.2.1.2 Floating Load Balancing Test Server

In addition to the basic tier configuration, various load balancing tests should include another system that can be configured on a single system as a movable server between tiers. The floating load balancing server can be used to test fail-over of clustered servers and recovery of servers if one server (on any one of the tiers) in a multiple server configuration goes down.

A floating Server could be configured as Web Server, Siebel Server, and Database Server on the same machine.

19.2.1.3 Clustered Web Server Configuration

The clustered Web server configuration tests two or more Web servers accessing a single Siebel server. This configuration is used to test how Siebel and the database server handles load balancing from multiple users accessing from multiple Web browsers and systems.

19.2.1.4 Clustered Siebel Servers Configuration

The clustered Siebel server configuration tests two or more Siebel servers handing Web traffic and accessing a single database server. This configuration is used to test Web traffic load balancing on clustered Siebel servers and how the database server handles load balancing from multiple Siebel servers accessing from multiple systems.

19.2.1.5 Clustered Database Server Configuration

The clustered database server configuration tests two or more database servers handing Siebel data and accessing the database storage. This configuration is used to test Siebel data load balancing on clustered database servers and how the database storage handles load balancing from multiple database servers accessing from multiple systems.

Notes:

- Siebel web applications use Cookies to maintain the state information.
- Each User must be logged into the same Siebel Application Server as first logged into.
- Do not use round robin load-balancing for clustered Siebel servers

19.2.2 Siebel Correlation Library

OpenScript includes a Siebel Test Automation library that Oracle Application Testing Suite applications can communicate with when creating Scripts. The Siebel Correlation Library is installed automatically as part of he OpenScript installation.

19.2.3 Script Creation Techniques

The following are tips and techniques to use when creating Siebel load test scripts using the Oracle Functional Testing application:

- Disable browser caching to make sure the pages are returned from the server rather than the browser cache.
- Record actions from login through logout to make sure parameters are passed properly between page navigations.

- Record actions slowly in the Siebel environment to make sure the recorder records all actions to the OpenScript script. If possible, watch as the script pages are added to the script tree.
- Close the external browser window after recording.
- Save the script. When you save a Siebel proxy-recorded script, OpenScript automatically creates a Java Agent version of the script in the workspace. Depending upon the size of the script, the file save operation may take some time.

19.2.4 Recording Scripts for Load Tests

Siebel load testing scripts are recording in an external browser window using the OpenScript proxy recorder. When you record a Siebel Load Test Script, OpenScript automatically starts the proxy recorder and opens an external browser window when you click the Record button on the toolbar. Once the external browser opens, you can load your Siebel application and start recording page navigation.

OpenScript does not support record and playback of the CTI Toolbar. URLS with the SWECmd=WaitForCmd are filtered out by default.

19.2.5 Starting the Siebel Application

Start the Siebel application in the browser using the start URL:

`http://`*`hostname`*`/`*`application`*`/start.swe`

where *`hostname`* is the machine name or IP address of the Siebel server and *`application`* is the Siebel application to start. For example:

```
http://siebelserver/callcenter_enu/start.swe
```

Enter the start URL and command into the browser address. As you navigate pages, the OpenScript proxy recorder records page navigation to the Script tree. You can view the nodes in the script tree and in the Java code.

19.2.6 Playing Back Scripts

OpenScript playback provides a convenient way to test and verify the page navigation recorded to the script.

1. Open a Siebel load test script in OpenScript.
2. Select **Playback** from the **Script** menu or click the toolbar button to verify the script plays back correctly.
3. Select items in the Results view and review the tabs in the Details view to check for any errors. Click the Headers tab to view request and response header data.
4. Verify that the response headers do not contain content or data value errors. One type of content error to check for is a "204 No Content" error. For example:

```
HTTP/1.1 100 Continue
Server: Microsoft-IIS/5.0
Date: Fri, 20 Mar 2009 15:51:47 GMT
X-Powered-By: ASP.NET
HTTP/1.1 200 OK:
Server: Microsoft-IIS/5.0
Date: Fri, 20 Mar 2009 15:51:47 GMT
X-Powered-By: ASP.NET
content-language: en
```

```
cache-control: no-cache
content-type: text/html;charset=UTF-8
content-length: 3762
```

See the *Troubleshooting Load Testing Issues* section in the *Testing Siebel eBusiness Applications* documentation in the Siebel document bookshelf for additional examples of common issues to resolve for load test scripts.

19.2.7 Resolving Script Issues

Each navigation node in the script tree shows the URL, Post Data, Recorded Headers, and Custom Dynamic values recorded to the script.

Expand nodes in the script tree to view the navigation sequence.

The `PostData` node shows the Siebel entities included in the page navigation. OpenScript Siebel Load Testing recorder automatically recognizes and parameterizes Siebel entities for Load testing. Playing back a script verifies that the recorder parameterized the Siebel entities properly.

The tree view nodes show the automatically created dynamic value names and Siebel path for parameters required for the *next* page. The name and Siebel path are shown as variable nodes under the post data node. It also shows automatically parameterized Siebel functions in curly braces, for example:

```
SWEC={{@SWECount}}
```

If a script does not playback correctly, or has errors, you may need to add custom dynamic values for Siebel parameters.

Check the PostData name/value pairs for the page navigation to verify the Siebel entities have been properly parameterized during recording. Any Siebel parameters that pass dynamic data from one page to the next should have custom dynamic values.

The PostData on the next page shows the destination of the dynamic values passed from the previous page.

19.2.7.1 Siebel Entities to Parameterize

The following table shows some common Siebel commands that may appear in the PostData of the page navigation:

Siebel Command	Name
SWEACn	Application Count
SWEBMC	Bookmark
SWEBRS	Browser Retry Sequence
SWEBID	Browser ID
_sn	Cookie
SWEFI	Form ID
SWEVLC	View Layout Cache
SWETS	Timestamp
SWEC	SWE Count
SWERowId, SWERowIds	Row IDs
s_#_#_#_#	Record Data

See the *SWE API* section of the *Siebel Portal Framework Guide* in the Siebel document bookshelf for additional information about Siebel Web Engine (SWE) commands, methods, and arguments.

19.2.8 Using Databanks with Siebel

Data values in Siebel post data strings can be parameterized in a script and connected to a Databank file that provides input data for data-driven tests. The OpenScript script editing options let you specify additional Siebel method names that use parameterized data.

1. Record or open a Siebel script in OpenScript.
2. Select **Find/Replace** from the **Edit** menu.
3. Type `SWEMethod` and click **Find**.
4. Continue clicking **Find** until you locate the SWEMethod that requires databanked values.

   For example, the PostData in a script page may contain the following record data name=value pair:

   ```
   s_1_2_49_0=doctest
   ```

   The SWEMethod that posts the data is `SWEMethod=Mirror Add GotoView`.

To use Databank parameters for the data values:

1. Select the **Assets** tab in the script view.
2. Select **Databanks**.
3. Click **Add**.
4. Select the databank type.
5. Select the Repository from the **My Repositories** tree.
6. Select the Databank file from the repository or file folder.
7. Enter an alias name to use for the Databank or leave the default alias name. The default alias name is the name of the .CSV Databank file.
8. Click **OK**.
9. Click **OK** to add the Databank file.
10. Right click the parameter node in the script tree that you want to substitute with a databank variable and select **Substitute Variable**.
11. If necessary expand the Databanks node and select the databank field you want to use as the input parameter data.
12. Click **Finish**.
13. The script node name/value pair changes to show the Databank alias name, field name, and recorded value as a variable value. For example:

    ```
    SWEUserName={{siebel_data.login,sadmin}}
    ```

14. Click the Playback toolbar button to playback the script once to verify the it plays back correctly.
15. Click the Iterate toolbar button to playback the script with a Databank.
16. Set the Iteration count, starting record, and data usage and click **OK**.

17. Verify the script plays back correctly.
18. Save the script.
19. In the Oracle Load Testing application, add the script to the Scenario.
20. Double-click the script name in the Scenario to define the Scenario details for the script.
21. Make sure Java Client is selected as the User mode.
22. Set the **Use Databanks** setting is set to `True` (if the **Use Databanks** setting is not shown, open the Scenario Details and set the option in the **Main** section).
23. Click **Run Test** and run the load test.

See the following sections of this document for details about defining ServerStats metrics and running tests in the Oracle Load Testing console. See the *Oracle Load Testing User's Guide* for additional information about using the features and options in the Oracle Load Testing application.

19.2.9 Preparing the Siebel Server Manager Commands

The Oracle Load Testing ServerStats uses the Siebel Server Manager program to retrieve statistics from the Siebel Server while running Virtual Users in a load test. The ServerStats Metrics need to be configured to run the Siebel Server Manager with input commands from a batch file and a file containing the input commands. The batch file and command input file must be created and placed on the Siebel Server where the ServerStats Metrics can access and run the batch file. This section explains the basic requirements of the batch file and command input file.

> **Note:** Starting the Siebel command-line server-monitoring program may require system administrator level privileges for the Siebel server. The Oracle Load Testing system testing the Siebel server needs the required user permissions to access the Siebel server and run the Siebel Server Manager program from the local host.

19.2.9.1 Creating the Batch File

Use any ASCII editor to create a batch file and a commands file to run the Siebel Server Manager program. The batch file name will be referenced in the Oracle Load Testing ServerStats metrics. You can use any name for the batch file (for example `srvrmgr_cmds.txt`).

Use the following syntax to specify the command in the batch file to start the Siebel Server Manager program:

```
\\machine IP\path to Siebel server bin\srvrmgr -g gateway -e enterprise -u
username  p password -i input_File
```

Parameter	Description
machine IP	The machine name or IP address or the Siebel Server.
path to Siebel server bin	The drive and directory path to the Siebel Server Manager program on the Siebel server.
gateway	The Network address of the Siebel Gateway Name Server machine.

Parameter	Description
enterprise	Siebel Enterprise Server name.
username	Siebel Server administrator username.
password	Siebel Server administrator password.
input_File	The name of a file containing commands to run in the Server Manager program.

For additional information about using the Siebel command-line server-monitoring program (`srvrmgr`) and command line flags, see the *Siebel System Administration Guide Version 7.7* (or newer) documentation in the Siebel document bookshelf.

The following is an example of a command in a batch file to start the Siebel Server Manager:

```
\\10.16.111.00\c$\sea77\siebsrvr\bin\srvrmgr -g gateway -e siebel -u sadmin -p
sadmin -i srvrmgr_cmds.txt
```

19.2.9.2 Creating the Command Input File

The command input file contains the commands to run in the Siebel Server Manager program. The command input file will be automatically run by the batch file at each Oracle Load Testing ServerStats Collection Interval. The example in the previous section uses the file name `srvrmgr_cmds.txt` as the input file in the srvrmgr command. You can use any file name as long as the batch command matches the file name of the input file.

`srvrmgr_cmds.txt` is a text file that contains the sequence of commands to run in the Siebel Server Manager program. The following is an example of an input file with Server Manager commands:

```
configure list statistics show STAT_ALIAS, CURR_VAL
list statistics
quit
```

The `configure list statistics show STAT_ALIAS, CURR_VAL` command specifies which Siebel Statistics to return from the srvrmgr program. `STAT_ALIAS, CURR_VAL` are the column names of the data values to return. `STAT_ALIAS` is the alias for the Statistic name. `CURR_VAL` is the current value for the statistic. `quit` closes the Siebel Server Manager session.

The `list statistics` command returns the statistics to Siebel Server terminal. The Oracle Load Testing Data Collector uses the Regular Expressions defined in the ServerStats metrics to extract specific data from the statistics returned from the `srvrmgr` program.

You can configure the srvrmgr commands to provide any of the available statistics data that can be returned by the srvrmgr program. See the *Siebel System Administration Guide Version 7.7* (or newer) documentation in the Siebel document bookshelf for additional information about Siebel Server Manager commands.

19.2.9.3 Siebel Statistics

The `srvrmgr` program returns the following statistics:

Name	Alias	Description
Average Connect Time	AvgConnTime	Average connect time for Object Manager sessions
Average Reply Size	AvgRepSize	Average size of reply messages (in bytes)
Average Request Size	AvgReqSize	Average size of request messages (in bytes)
Average Requests Per Session	AvgReqs	Average number of requests per Object Manager session
Average Response Time	AvgRespTime	Average Object Manager response time
Average Think Time	AvgThinkTime	Average end-user think time between requests
Avg SQL Execute Time	AvgSQLExecTime	Average time for SQL execute operations (in seconds)
Avg SQL Fetch Time	AvgSQLFetchTime	Average time for SQL fetch operations (in seconds)
Avg SQL Parse Time	AvgSQLParseTime	Average time for SQL parse operations (in seconds)
CPU Time	CPUTime	Total CPU time for component tasks (in seconds)
Elapsed Time	ElapsedTime	Total elapsed (running) time for component tasks (in seconds)
Maximum Peak Memory Usage	MaxPeakMemory	Peak Mem used by task. Rolls up differently from MinPeakMemory
Minimum Peak Memory Usage	MinPeakMemory	Peak Mem used by task. Rolls up differently than MaxPeakMemory
Num of DBConn Retries	NumDBConnRtrs	Number of Retries due to DB Connection Loss
Num of DLRbk Retries	NumDLRbkRtrs	Number of Retries due to Deadlock Rollbacks
Num of Exhausted Retries	NumExhstRtrs	Number of Times All Retries are Exhausted
Number of SQL Executes	SQLExecs	Total number of SQL execute operations
Number of SQL Fetches	SQLFetches	Total number of SQL fetch operations
Number of SQL Parses	SQLParses	Total number of SQL parse operations
Number of Sleeps	Sleeps	Total number of sleeps for component tasks
Object Manager Errors	Errors	Number of errors encountered during Object Manager session
Reply Messages	RepMsgs	Number of reply messages sent by the server

Name	Alias	Description
Request Messages	ReqMsgs	Number of request message received by the server
SQL Execute Time	SQLExecTime	Total elapsed time for SQL execute operations (in seconds)
SQL Fetch Time	SQLFetchTime	Total elapsed time for SQL fetch operations (in seconds)
SQL Parse Time	SQLParseTime	Total elapsed time for SQL parse operations (in seconds)
Sleep Time	SleepTime	Total amount of sleep time for component tasks (in seconds)
Tasks Exceeding Configured Cap	TskXcdCfgCpt	Number of tasks stated that exceeded configured capacity
Tests Attempted	TestsAttempted	Number of tests that were started
Tests Failed	TestsFailed	Number of tests that failed
Tests Successful	TestsSuccessful	Number of tests that were successful
Total Database Response Time	DBRespTime	Total Database Response/Processing Time (milliseconds)
Total Reply Size	RepSize	Total size (in bytes) of reply messages
Total Request Size	ReqSize	Total size (in bytes) of request messages
Total Response Time	RespTime	Total Object Manager response time (in seconds)
Total Tasks	TotalTasks	Total number of tasks completed for server components
Total Think Time	ThinkTime	Total end-user think time (in seconds)

For additional information about monitoring Siebel servers, see the *System Monitoring and Diagnostics Guide for Siebel eBusiness Applications Version 7.7* (or newer) documentation in the Siebel document bookshelf.

19.2.9.4 Batch File Location

Once you create the batch file and command input file, copy the files to the Oracle Load Testing local host in the `C:\Oracle\DataCollector` directory.

19.2.10 Defining ServerStats Metrics

Oracle Load Testing ServerStats metrics are used to collect the data from the Siebel Server Manager program. This section explains how to set up Virtual Agents in ServerStats (Oracle Load Testing) to run the Siebel Server Manager program (`srvrmgr`) from the command-line interface.

1. Start the Oracle Load Testing application.

2. Select **ServerStats** from the **Tools** menu.
3. Select the Metrics node to view the metric categories.
4. Click **New**.
5. Enter a name for the metric.
6. Enter a description for the metric.
7. Select **Virtual Agent** as the Metric type.
8. Click **Next**.
9. Enter the name of the batch file you created to run the srvrmgr program in the **Command Line** field.
10. Enter a Regular expression to parse the data returned from the `srvrmgr` program in the **Matching Regexp** field. For the Server Manager `srvrmgr` program commands:

    ```
    configure list statistics show STAT_ALIAS, CURR_VAL
    list statistics
    quit
    ```

    use the following format for the Regular Expression:

    ```
    /aliasName\s+([0-9]+)/
    ```

    For example, for the server statistic `Average Connect Time`, the Regular Expression would be as follows:

    ```
    /AvgConnTime\s+([0-9]+)/
    ```

11. Enter the Key of value to use to parse the Regular Expression. The key of value specifies which set of parenthesis in the Regular Expression is the value to return. For Siebel statistics using the above Regular Expression, set the value to 1.
12. Enter the Sample Multiplier value. The following window shows a metric configured to retrieve the Average Connect Time.
13. Click the **Test** button to get to the Test Setup window:
14. Click **OK** to start the test.
15. Verify the results returned the correct data value for the statistic from the Siebel Server Manager program and did not return any errors.

    Note: Manually run the `srvrmgr` program and list statistics on the Siebel Server to verify the Regular Expression returns the correct data value/format.
16. Click **Close**.
17. Click **Finish**. The New metric appears in the Metrics tree under the User Defined node.
18. Repeat steps 4-17 to configure additional Siebel metrics in ServerStats.

For additional information about monitoring Siebel servers, see the *System Monitoring and Diagnostics Guide for Siebel eBusiness Applications Version 7.7* (or newer) documentation in the Siebel document bookshelf.

19.2.11 Defining a ServerStats Configuration

Oracle Load Testing ServerStats configurations are used to specify which metrics to include when collecting the data from the Siebel Server Manager program and update

Oracle Load Testing graphs and reports. You can also create a metric profile for Siebel metric and use the profile as part of the configuration. This section explains how to define a ServerStats configuration and add metrics to the configuration.

1. If necessary, start Oracle Load Testing and select **ServerStats** from the **Tools** menu.
2. Click the Configurations node to view existing configurations.
3. Click **New**.
4. Enter a name for the configuration.
5. Enter a description for the configuration.
6. Click **Save**. The configuration window adds new options for adding and updating monitors:
7. Click **Add a new monitor**.
8. Expand the User defined node and select a Siebel metric.
9. Click **Next**.
10. Set the monitored system, data collector, and collection interval.
11. Click **Next**. The metric is added to the list of monitors in the configuration.
12. Click **Finish**.
13. Repeat steps 7-12 to add additional metrics to the configuration.
14. Click **Test**.
15. Verify the results returned the correct data values for the statistics from the Siebel Server Manager program and did not return any errors.
16. Click **Close**.
17. Click **Update**.

19.2.12 Importing Pre-Configured Metrics and Profiles to Oracle Load Testing

If you have pre-configured files for Siebel metrics and metric profiles, you can import the files into Oracle Load Testing rather than manually configure the metrics and profiles.

1. If necessary, start the Oracle Load Testing application.
2. Select **Import File** from the **Tools** menu.
3. Select the File Type. The ServerStats file types are as follows:

Type	Extension
ServerStats Metric	.metric
ServerStats Metric Profile	.hwm
ServerStats Configuration	.config

4. Click **Browse** to select the file location.
5. Select the drive and directory location.
6. Select the file to import.
7. Click **Open**.

8. Click **Upload**.
9. Click **OK**.
10. Repeat steps 4-9 for each file to upload.

19.2.13 Running Load Tests in the Oracle Load Testing Console

Select the script or a user-defined profile from the **Select scripts & user-defined profiles** list.

1. Select a script.
2. Click **Add to scenario**.
3. Set the **# VUs**.
4. Set the **System** to use to test.
5. Set the **User Mode** to Java Client.
6. Set the **Iteration Delay** to 1.
7. Set the **VU Pacing (Think Time)** to Recorded.
8. Click **Add to Autopilot**.
9. Set the **Start** and **Stop** test options.
10. Set the **Virtual User Rampup**.
11. Select the ServerStats configuration you defined earlier.
12. Click the **Run Test** button.
13. Specify the Session to Save.
14. Click **OK**.

19.2.13.1 Viewing VU Grid

The Virtual User grid lets you view the progress of the script playback for each virtual user. If necessary, click the **Watch VU Grid** tab to switch to the grid.

19.2.13.2 Viewing ServerStats

The ServerStats display lets you view the Siebel Server statistics in real time using the ServerStats display window. Select **ServerStats Display** from the **Tools** menu to open the ServerStats display.

19.2.14 Generating Graphs and Reports Using Oracle Load Testing

You can generate graphs from Virtual User and ServerStats data during run time and for post testing analysis.

19.2.14.1 Creating Custom Runtime Graphs

The **View Run Graphs** tab lets you generate custom graphs during test runtime.

1. While the load test is running, click the View Run Graphs tab.
2. Click **New Graph**. A new blank graph tab is added to the Reports and Graphs section.
3. Scroll down to the **Filters** section.
4. Enter a graph name.

5. Expand ServerStats Monitors in the Available Data Series tree.
6. Select the data series to add to the graph.
7. Click **Add Data Series**.
8. Repeat to add other monitors to the data series.
9. Specify the **Plot Data Series** and **Y-Axis Scaling** options.
10. Click **Generate Graphs**. The custom graph appears as a new tab in the Reports and Graphs section.

19.2.14.2 Creating Custom Reports

The **Create Reports** tab lets you generate custom reports and graphs after the test for post-testing analysis.

1. Click **Create Reports** tab.
2. Click **New Graph**.
3. Select the ServerStats session as the **Available Data Series**.
4. Expand **Available Data Series** tree.
5. Select the data series to add to the graph.
6. Click **Add Data Series**.
7. Click **Generate Graph**. The custom graph appears as a new tab in the Reports and Graphs section.

   You can export the graph to Adobe PDF, Microsoft Excel, or Comma Separated Value formats.

The **Create Reports** tab also lets you retrieve session performance reports after the test for post-testing analysis.

1. If necessary, click **Create Report**s tab.
2. Click the **Sessions** tab in the Reports and Graphs section.
3. Select the Session. The report appears in the Reports and Graphs section.

   You can export or print the session report.

19.3 Setting Siebel Correlation Preferences

To set Setting Siebel Correlation preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Correlation category.
4. Expand the Siebel Load Library.
5. Select or clear the check boxes to enable or disable specific rules.
6. Click the **Add** or **Edit** buttons to modify rules in the library. See Section 19.4, "Siebel Correlation Library" for a list of correlation rules.
7. Click **OK**.

19.3.1 Enabling the Java Correlation Mode

By default, Siebel load scripts play back using a DLL-based correlation library if running on Windows platforms or a .SO-based correlation library if running on Linux. The OpenScript Siebel load testing module also includes a platform independent, JAVA-based correlation library that can be enabled using an additional playback setting.

To enable the JAVA-based correlation library:

1. Select **OpenScript Preferences** from the **View** menu.
2. Expand the OpenScript node and select the Playback category.
3. Enter the following setting into the Additional Arguments field:

   ```
   -siebel.correlationMode java
   ```

4. Click **OK**.

19.4 Siebel Correlation Library

The Siebel correlation library defines the correlation rules for Siebel (Siebel versions 7.7, 7.8, 8.0, 8.1). The correlation rules specify the variable names and regular expressions to use to replace dynamic data in Siebel applications and navigations.

The default Siebel correlation library provided with the OpenScript Siebel Module includes the following correlation rules:

- **Siebel SWEACn - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `SWEACn=(\d+)` and replaces it with the variable name `SWEACn` in all locations.
- **Siebel SN - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_sn=((.+?))&` and replaces it with the variable name `siebelsn` in the specified location. The variable name `siebelsn` uses the Regular Expression pattern `name="_sn" value="(.+?)"`.
- **Siebel SN - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_sn=((.+?))&` and replaces it with the variable name `siebelsn` in the specified location. The variable name `siebelsn` uses the Regular Expression pattern `_sn=(.+?)"`.
- **Siebel SN - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `_sn=((.+?))&` and replaces it with the variable name `siebelsn` in the specified location. The variable name `siebelsn` uses the Regular Expression pattern `_sn=(.+?)&`.
- **Siebel SWEBID - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `SWEBID(=|%3d|%3D)((\d+))` and replaces it with the variable name `SWEBID` in the specified location. The variable name `SWEBID` uses the Regular Expression pattern `navigator.id = ([0-9]+?);`.
- **Siebel SWEBID - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern `SWEBID(=|%3d|%3D)((\d+))` and replaces it with the variable name `SWEBID` in the specified location. The variable name `SWEBID` uses the Regular Expression pattern `navigator.id = "([0-9]+?)"`.
- **Siebel SWEBMC - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern

SWEBMC=(((|%3d|%3D)\d+([&%]|$|\s))) and replaces it with the variable name SWEBMC in the specified location. The variable name SWEBMC uses the Regular Expression pattern SWEBMC(?:=|%3d|%3D)(\d+)[?&?%].

- **Siebel SWEBRS - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern SWEBRS=(((|%3d|%3D)\d+([&%]|$|\s))) and replaces it with the variable name SWEBRS in the specified location. The variable name SWEBRS uses the Regular Expression pattern <input type = "hidden" name="SWEBRS"\s+?value="(\d+?)">.
- **Siebel SWEFI - Variable Substitution** - this rule locates text in the HTML matching the Regular Expression pattern SWEFI=(((|%3d|%3D)\d+([&%]|$|\s))) and replaces it with the variable name SWEFI in the specified location. The variable name SWEFI uses the Regular Expression pattern (.)SWEFI\1(\d+)\1.
- **SWETS - Function/Text Substitution** - this rule locates text in the HTML matching the Regular Expression pattern SWETS=((\d{13,})) and replaces it with the function {{@timestamp}} in the specified location.
- **SWSECancelID - Function/Text Substitution** - this rule locates text in the HTML matching the Regular Expression pattern SWSECancelID=(=|%3d|%3D)((\d{10,})) and replaces it with the function {{@timestampsecs}} in the specified location.
- **RefID - Function/Text Substitution** - this rule locates text in the HTML matching the Regular Expression pattern refID(=|%3d|%3D)((\d+)) and replaces it with the function {{@CounterRefID}} in the specified location.
- **Siebel Correlation Rule** - this rule locates all Siebel SWEC and RowIDs.
- **alarmDate - Substitute Recorded Date** - this rule locates the alarmDate in the HTML matching ther Regular Expression pattern alarmDate(((\d{1,2})*(\d{1,2})/(\d{1,2})/(\d{4}))) and replaces it with the function {{@today(n,M/d/yyyy)}} in the specified location.
- **currentDate - Substitute Recorded Date** - this rule locates the date the script was recorded and replaces it with the date pattern M/dd/yyyy.

19.5 Siebel Script Functions

Dynamic values returned from a Siebel server can be replaced by Siebel-specific OpenScript functions. The following script functions are available specifically for Siebel scripts when you substitute a variable value:

- {{@CounterRefID,x}}: this function is used to replace the refID parameter in QueryString or Postdata strings. x is the recorded value.
- {{@siebeltimestampsecs}}: this function is used to replace the Siebel time stamp with the script variable {{@siebeltimestampsecs}}. The value is the current timestamp in seconds instead of milliseconds.
- {{@SWECount}}, {{@SWECInc(x)}}, {{@SWECSet()}}: these functions are used to replace SWEC parameters in QueryString or Postdata strings.

20
Using the Utilities Module

This chapter provides instructions on using the OpenScript Utilities Module, which provides commonly used testing functions.

20.1 About the Utilities Module

The Utilities Module is an extension to the Basic Module. The OpenScript Utilities module includes the following features:

20.1.1 Key Features of the Utilities Module

- Text File Processing. Read values from text files including CSV and XML files as well as copy and move files in the file system.
- Databases. Read values from various databases such as Oracle as well as other JDBC-ODBC Compliant databases
- XML XPath Expressions. Generate XPath expressions from valid XML files.

  You can use the Utilities Module API to enhance recorded scripts with additional testing functionality. Commands that are specific to the Utilities Module are part of the "utilities" class.

20.2 Using Text File Processing

You can use the `utilities` API to read values from text files including CSV and XML. The following sections explain how to use the utilities API.

20.2.1 Working with Text Files

The Utilities API includes a `getFileService()` object with methods for working with text files such as reading lines of text from a file or appending to a file. The following examples show some ways to use `getFileService`.

To add code that reads text from a file:

1. Create a script project.
2. Open the Java Code view.
3. Add the `readLines()` method to specify the file to read. The following example shows how to parse the lines of text in a file and print to the OpenScript console view:

```
import java.io.File;
//[...]
```

```
String[] lines = utilities.getFileService().readLines("C:/Sample.txt");
for (String line : lines) {
info(line);
}
```

To add code that appends text to a file:

1. Create a script project.
2. Open the Java Code view.
3. Add the `appendStringToFile()` method to specify the file to which to append text strings. The following example shows how to create a new file and append lines of text to the file:

```
import java.io.File;
//[...]
utilities.getFileService().createDestinationFile("myFile.txt", false);
String line1 = "This is a new line 1";
String line2 = "This is a another new line 2";
String contents = "\n" + line1 + "\n" + line2;
utilities.getFileService().appendStringToFile("myFile.txt", contents);
```

20.2.2 Working with CSV Files

The Utilities API includes a `loadCSV()` object for working with data from a Comma Separated Value text file.

To add code that loads and prints data from a .CSV file:

1. Create a script project.
2. Open the Java Code view.
3. Add the `loadCSV` method to specify the file to read. For this example the file, "C:\customer.csv" contains this data:

```
FirstName,LastName,MiddleInitial
John,James,R
Mary,Simpson,J
```

The following example shows one way to parse a table of text in a .CSV file and print values to the OpenScript console view:

```
import java.io.File;
import java.util.List;
//[...]
String filePath = "c:\\";
String csvFile = filePath + "fmstocks_data.csv";
File file = new File(csvFile);

Table table = utilities.loadCSV(csvFile);

//Print the CSV file
String columns = "";
int columnNumber = table.getColumns().getColumnCount();
String [] columnNames = table.getColumns().getColumnNames();
for (int index=0; index<columnNumber; index++)
   columns += columnNames[index] + "  ";
info(columns);

List <Row> rows = table.getRows();
for (int index=0; index<rows.size(); index++) {
```

```
    String [] rowValue = rows.get(index).getAll();
    String rowContent = "";
    for (int columnIndex=0; columnIndex<rowValue.length; columnIndex++)
        rowContent += rowValue[columnIndex] + "  ";
    info(rowContent);
}
```

20.2.3 Working with XML Files

The Utilities API includes a `loadXML()` object for reading text from a XML formatted text file.

To add code that reads text from a .XML file:

1. Create a script project.
2. Open the Java Code view.
3. Add the `loadXML` method to specify the file to read. For this example the file, "C:\grocery.xml" contains this data:

```
<?xml version="1.0" encoding="utf-8"?>
<Oceans>
  <ocean name="Arctic"/>
  <ocean name="Atlantic"/>
  <ocean name="Indian"/>
  <ocean name="Pacific"/>
  <ocean name="Southern"/>
</Oceans>
```

The following example shows how to parse a table of text in a .XML file and print values to the OpenScript console view:

```
XML xml = utilities.loadXML("C:/oceans.xml");
XML root = xml.getChildren()[0];
info(root.getTagName());
XML[] oceans = root.getChildren();

for (XML ocean : oceans){
info(ocean.getAttribute("name"));
}
```

20.3 Getting Values from a Database

Getting values from a database requires a database definition, a database SQL query or SQL execute and a disconnect from the database. This section explains how to manually add database actions to a script. See Section 3.2.7, "Importing Database Capture Files" for additional information about importing a DBReplay capture file or SQL statements from a plain SQL and PL/SQL statements .SQL script file to generate an OpenScript load testing script.

To get values from a database:

1. Create a database script project.
2. Select the node where you want to add the database definition.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Database node and select Database Definition.
5. Click **OK**.

6. Specify the database definition information.

   Database Driver - specify the database driver to use.

   - **Oracle Thin (oracle.jdbc.driver.OracleDriver)** - when selected, the database connection uses the Oracle Thin database driver. Specify the following connection information:

     Hostname - specify the name of the host machine on which the database is located.

     Port - specify the port number to use.

     – **SID** - when selected, specify the System ID to identify the particular database on the system.

     – **Service Name** - when selected, specify the Service Name defined as the alias for the database instance.

   - **ODBC (sun.jdbc.odbc.JdbcOdbcDriver)** - when selected, the database connection uses the ODBC database driver. Specify the following connection information:

     Data source - specify the name of the ODBC data source to which to connect.

   URL - specify the URL to use to access the database.

   Username - specify a user name to log into the database.

   Password - specify a password to log into the database.

   Alias - specify an alias name to use to identify the database definition. The alias appears in the script tree for the definition and is selected when adding database actions to the script.

   Test - test the connection to the database based upon the specified driver and database information.

7. Click **Test** to verify a successful connection.
8. Click **OK**.
9. Select the node where you want to add the database connection. The OpenScript database connect method is optional. The database connect is invoked automatically when calling execute or query methods
10. Select the **Script** menu and then select **Other** from the **Add** sub menu.
11. Expand the Database node and select Connect.
12. Select the database alias and click **OK**.
13. Select the node where you want to add the database query or execute statement.
14. Select the **Script** menu and then select **Other** from the **Add** sub menu.
15. Expand the Database node and select SQL Query or SQL Execute.
16. Specify the SQL statement to query or execute and click **Add**.
17. Specify a data type and define a name for the parameter.
18. Click **OK**.
19. Click **OK**.
20. Select the node where you want to add the database disconnect.
21. Select the **Script** menu and then select **Other** from the **Add** sub menu.

22. Expand the Database node and select Disconnect.
23. Select the database alias and click **OK**.

    In the Java Code view, the `utilities.getSQLService()` methods will be added to the script code for each database script action (additional code and comments added):

```
//define database
utilities.getSQLService().define("oracledb",
   "oracle.jdbc.driver.OracleDriver", "00.000.000.000", "myuserID",
   decrypt("ZgEQLMIUx8EVDAhfAenvyg=="));

//connect to database
utilities.getSQLService().connect("oracledb");

//execute SQL statement
String query = "Create table Employee (ID number(4) not null unique, " +
"FirstName varchar2(40) not null, LastName varchar2(40) not null, " +
"Country varchar2(40), HireDate date)";
info("Query: " + query);
utilities.getSQLService().execute("oracledb", query);

//execute update SQL statement
query = "Insert into Employee (ID, FirstName, LastName, Country, HireDate) " +
   "Values (101, 'Tom', 'Smith', 'USA', '01-JAN-95')";
utilities.getSQLService().executeUpdate("oracledb", query);

//query SQL statement
query = "Select * from Employee";
Table table = utilities.getSQLService().query("oracledb", query);

//print table
for (int i=0; i<table.getRowCount(); i++) {
   Row row = table.getRow(i);
   String [] rowValue = row.getAll();
   String rowContent = "";
   for (int col=0; col<rowValue.length; col++)
      rowContent += rowValue[col] + "  ";
   info(rowContent);
}

//disconnect from database
utilities.getSQLService().disconnect("oracledb");
```

20.3.1 Adding a SQL Query Test

A SQL Query test can be used to test data values retrieved from a database using a SQL query against expected values.

To add a SQL Query test:

1. Create a database script project.
2. Select the node where you want to add the SQL Query test.
3. Select the **Script** menu and then select **Other** from the **Add** sub menu.
4. Expand the Database node and select SQL Query.
5. Click **OK**.

6. If you have already created a database connection, select the database alias. If you have not already created a database connection, click **New** and specify a new database definition. See Section 20.3, "Getting Values from a Database" for additional information about creating a database definition.
7. Enter a SQL Statement to use to define the data to retrieve from the database.
8. Click **Test** to verify the data is retrieved from the database. If the **Test Results** dialog does not appear listing the data from the SQL Statement, verify the SQL Statement is correctly structured.
9. Click **Close** to close the **Test Results** dialog box.
10. Click **Create SQL Query Test**.
11. Enter a name for the test.
12. Set the **Verify only, never fail** option.
13. Select the **Test Data** option:
    - **Entire Table** - when selected, the entire table of data retrieved from the SQL Statement is included in the test.
    - **Filter table by query**- when selected, only the data that matches the filter query is included in the test. Enter a SQL query and click **Apply** to apply the filter to the test data.
    - **Apply** - applies the query filter to the test data.
14. Enable testing on specific data values by selecting or clearing the check boxes in each cell. Click **Enable All** to enable testing on all data values in the grid. Click **Disable All** to disable testing on all data values in the grid, then select individual cells manually.
    - [Row] - shows the row number of the test data.
    - [Table column name(s)] - shows the name(s) of the database table field(s) retrieved from the database by the SQL Statement. Each column in the SQL Query Test will show table field names as the column headers. Select the check box to enable testing or clear the check box to disable testing on specific data values.
15. For each selected table cell, specify the SQL Query Test Details:

    SQL Query Test Details - shows the test details for the selected table cell.
    - **Value from DB** - shows the actual value of the data retrieved from the database for the selected table cell.

      Enable - when selected, testing for the selected cell is enabled.
    - **Cell** - shows the row and column information for the selected table cell.

      Row - the row number of the selected table cell.

      Column Name - the column name of the selected table cell. This is the field name retrieved from the database.
    - **Value Type** - specifies the data type for the value in the selected table cell.
    - **Operator** - specifies the test operator used to compare the data value in the selected table cell against the expected value.
    - **Expected Value** - specifies the expected value to compare against the value retrieved from the database.

Substitute Variable - opens a dialog box for selecting a script variable to use for the Expected Value.

16. Click **OK** when finished. New Query and Query Test nodes will be added to the script tree.

    In the Java Code view, the `utilities.getSQLService()` methods will be added to the script code for the database query and SQL test:

```
utilities.getSQLService().query(9, "mydb", "Select FName from Employee", null);
{
utilities.getSQLService().assertQuery(
   null, "mySQLtest", null,
   utilities.getSQLService().cell(1, "FNAME",
      "Aaron", SQLTestOperator.StringExact),
   utilities.getSQLService().cell(2, "FNAME",
      "Adrian",SQLTestOperator.StringExact)));
}
```

    When you play back the script, the test results appear in the Results view.

20.3.2 Calling a Database Procedure Statement

You can use the `utilities` API to execute a SQL call database procedure statement and return a list object for an out type parameter value list.

You can use the `utilities.getSQLService().`**`callProcedure`**`("`*`recid`*`", "`*`alias`*`", "`*`sql`*`", "`*`params`*`")` method to call the procedure and return a list object for out type, where:

- *`recid`* is an optional Integer specifying the recorded ID.
- *`alias`* is a String specifying the user-defined database alias specified for the database containing the procedure.
- *`sql`* is a String specifying the SQL statement to be sent to the database, typically a static SQL to call database procedure statement.
- *`params`* is an optional `List<Object>` object containing all the `SQLService.parameter` or `SQLService.SQLParameterType`-wrapped parameter values by index. The index starts with 1.

The following example shows the code used to define and connect to a database, call a database procedure, and disconnect from the database:

```
utilities.getSQLService().define("local_XE_DB",
   "oracle.jdbc.driver.OracleDriver",
   "jdbc:oracle:thin:@localhost:1521/XE", "system",
   ;deobfuscate("6GaD7eW3kGVe5TKHmuI/+w=="));

utilities.getSQLService().connect("local_XE_DB");

utilities.getSQLService().callProcedure(44, "local_XE_DB",
   "Begin\n  insertInfo2(014,'anna14',21,'F','ecnu14',
     'History','1288',to_date(?,'yyyy-mm-dd'));\nEnd;",
   utilities.parameters(
     SQLService.parameter("1989-02-18",
     SQLService.SQLParameterType.In)));

utilities.getSQLService().disconnect("local_XE_DB");
```

20.4 Using the XPath Generator

The Utilities Module includes an XPath generator utility that you can use to generate an XPath Expression to a selected element from a valid XML file.

To use the XPath Generator:

1. Create an XML file that contains the tags and values to use to generate the XPath expression. The following is an example of a simple XML file that can be used with the XPath Generator:

```
<?xml version="1.0" encoding="utf-8"?>
<Oceans>
   <ocean name="Artic"/>
   <ocean name="Atlantic"/>
   <ocean name="Indian"/>
   <ocean name="Pacific"/>
   <ocean name="Southern"/>
</Oceans>
```

2. Create and record a test script. The **Tools** menu appears on the OpenScript menu bar for functional and load test scripts.
3. Select **Generate XPaths** from the **Tools** menu.
4. Click **Browse** and select the XML file to load.
5. Expand the XML tree under the Tags section of the XML file.
6. Select the XML tag to use to generate the XPath. The generated XPath appears in the XPath Expression field in a form similar to `/Oceans/ocean[1]/@name`.
7. Use the Ctrl+C and Ctrl+V keyboard combinations to copy and paste the generated XPath to a method in the Java Code tab of the script view.

   The XPath Expression can be used in the utilities `findByXPath` API method, as follows:

```
utilities.loadXML("filePath").findByXPath(xpath, xml)
```

21
Using the Shared Data Module

This chapter provides instructions on using the OpenScript Shared Data Module, which allows data to be passed between scripts using a shared data queue.

21.1 About the Shared Data Module

The Shared Data Module is an extension module to the OpenScript Basic Module that extends the other testing modules with message queue and hash map capabilities.

Shared Data is typically used to pass data between load testing scripts running as Virtual Users in Oracle Load Testing. The Shared Data can also be used to pass data between functional testing scripts running from the command line.

Virtual users can put a message object in a queue, and take a message object from the same queue. Virtual users can also create a hash map or get an existing map and put key-values into the hash map. The virtual users can put or get message objects to the same queue or hash map in different agents/machines.

21.1.1 Key Features of the Shared Data Module

The Shared Data module provides the following features:

- Preferences - shared data connection preferences can be set under the OpenScript Playback preferences.
- Message Queue Manipulation - create, peek, and poll message queues.
- Hash Map Manipulation - create, put, and get key-value data in hash maps.
- Shared Data Module API - The Shared Data Module API includes a "sharedData" class that provides additional programming functionality.

21.2 Setting Shared Data Preferences

To set Shared Data preferences:

1. Start OpenScript.
2. Select **OpenScript Preferences** from the **View** menu.
3. Expand the OpenScript node and the Playback category.
4. Select **Shared Data**.
5. Set the Shared Data Preferences as follows:

   OATS Credentials: Specifies the authentication credentials to use to establish the communication between the shared queue and the Virtual User.

- **Enable global shared data access credentials**: When selected, when selected, the shard data access credentials are enabled. Specify the **Address**, **User Name**, and **Password**.
- **Address**: Specifies the address of the Oracle Load Testing for Web Application server to use for the shared data service.
- **User name**: Specifies the user name to use for authentication. The default name is `oats` unless changed in the Oracle Application Testing Suite configuration.
- **Password**: Specifies the password to use for authentication. This should be the same password specified in the Encryption setting of the General preferences if the **Encrypt script data** setting is selected.

Actions on Shared Data: Specifies actions on shared data.

- **Timeout**: Specifies the maximum number of seconds to wait for actions on shared data to occur before timing out.

6. Click **OK**.

21.3 Using the Shared Data Service

This section describes how to enable and use the Shared Data Service.

21.3.1 Basic Scenarios

The following are the basic scenarios for using the Shared Data Service:

- **Queue Mode**: Items are stored sequentially in queues. Scripts can get the first or last data item in the queue. Other items cannot be accessed randomly.

  Script A is run by 100 Virtual Users, which act as message producers putting message objects to the shared data queues.

  Script B is run by another 100 Virtual Users, which act as consumers getting message objects from the shared data queues.

  Consumer Virtual Users can get an object from the beginning or end and the information from a queue. If the queue is empty, the consumer Virtual User is blocked until the timeout is reached. Once the object can be retrieved, the consumer Virtual User is resumed.

- **Hash Map mode**: Any item can be accessed using a key. The hash map may already contain a mapping for a key. The hash map needs to be checked before a new item is put into a hash map.

  Script A is run by 100 Virtual Users, which put key-value objects to a shared data hash map.

  Script B is run by another 100 Virtual Users, which get values with keys from the shared data hash map.

  If a Virtual User cannot get the value to which the specified key is mapped, it will be blocked until the timeout is reached or the key-value is added to the map.

21.3.2 Enabling the Shared Data Service

To enable the Shared Data Service:

1. Start OpenScript.

2. Open an existing script or create and record a new script.
3. Select **Script Properties** from the **Script** menu.
4. Select the Modules category.
5. Select the **Shared Data** module.
6. Click **OK**. The Shared Data Service will be added to the script class in the Java Code as follows.

   ```
   @ScriptService oracle.oats.scripting.modules.sharedData.api.SharedDataService
   sharedData;
   ```

   Once you have enabled the Shared Data service, you can set the password encryption and the connection parameters and then use the Shared Data API to manipulate message queues or hash maps. You use the `sharedData` class in the Java code view to create manipulate message queues and hash maps.

21.3.3 Setting the Password Encryption

The Password encryption is set in the General Preferences. To set the password encryption:

1. Select **OpenScript Preferences** from the **View** menu.
2. Expand the OpenScript node and the General category.
3. Select **Encryption**.
4. Select **Obfuscate script data** or **Encrypt script data** to make sure the connection to the Shared Data Service uses an obfuscated or encrypted password.
5. If you select **Encrypt script data**, enter a password.
6. Click **OK**.

21.3.4 Setting the Connection Parameters

The connection parameters specify the Oracle Load Testing server to use for the Shared Data Service and the authentication settings. During a load test, the Shared Data Service is limited to running only on the Oracle Load Testing controller running the test. To set the connection parameters:

1. Make sure the Shared Data Service is enabled and the password encryption is specified as previously described.
2. Select the script node where you want to set the connection parameters.
3. Select **Add** from the **Script** menu and then select **Other**.
4. Expand the Shared Data folder.
5. Select **Set Connection Parameters** and click **OK**.
6. Set the connection parameters as follows:

   Address: Specify the address of the machine to use for the Shared Data Service. For example: t3://localhost:8088 or t3://*machinename*.com:8088.

   User Name: Specify the user name to use for authentication. The default name is `oats` unless changed in the Oracle Application Testing Suite configuration.

   Password: Specify the password to use for authentication.
7. Click **OK**. A Connection Parameters node will be added to the script tree.

8. In the Java Code view, the Connection Parameters consist of the code executed in the `sharedData.setConnectionParameters` procedure:

```
sharedData.setConnectionParameters("t3://localhost:8088", "oats",
  decrypt("L4I57b+KpnI2BQSRKPG88w=="));
```

After setting the connection parameters, you can user the Shared Data API in the Java Code view to manipulate data in message queues and hash maps.

21.3.5 Creating a Shared Data Queue

To create a shared data queue:

1. Create an script project.
2. Make sure the Shared Data Service is enabled, the password encryption, and connection parameters are specified as previously described.
3. Open the Java Code view and insert the `sharedData.createQueue` code with a life time value into the script where you want to create the queue, as follows:

```
info("Create queueA with life time of 10 minutes");
sharedData.createQueue("queueA", 10);
```

The maximum number of queues is 1000. The maximum capacity of a queue is 65535. If the maximum is exceeded, an exception occurs. Once the life time expires, the queue is destroyed.

21.3.6 Inserting Data into a Shared Data Queue

The types of the "values" that can be put into a queue are as follows:

- String
- boolean
- integer
- long
- double
- float
- a List of any of the above data types
- User-defined serializable java objects.

To insert data into an existing queue:

1. Set up the shared data service and create a queue as previously described.
2. Open the Java Code view and insert the `sharedData.offerFirst` or `sharedData.offerLast` code with a value into the script where you want to insert data into the queue, as follows:

```
int iterationNum = getIteration().getTotalIterationsCompleted() + 1;
info("Insert data at the front of an existing queueA");
sharedData.offerFirst("queueA", "first" + iterationNum);
```

or

```
int iterationNum = getIteration().getTotalIterationsCompleted() + 1;
info("Insert data at the end of an existing queueA");
sharedData.offerLast("queueA", "last" + iterationNum);
```

```
info("parameter type - String");
sharedData.offerFirst("queueA", "value");

info("parameter type - list of Strings");
ArrayList<String> listOfStr = new ArrayList<String>();
listOfStr.add(0, "val1");
listOfStr.add(1, "val2");
sharedData.offerFirst("queueA", listOfStr);
ArrayList<String> queueValue = (ArrayList<String>)
   sharedData.pollFirst("queueA");

info("parameter type - boolean");
sharedData.offerFirst("queueA", true);

info("parameter type - int");
sharedData.offerFirst("queueA", 10);

info("parameter type - double");
sharedData.offerFirst("queueA", 10.5);

info("parameter type - long");
sharedData.offerFirst("queueA", 100);
```

21.3.7 Getting Data from a Shared Data Queue

To get data from a queue:

1. Set up the shared data service, create a queue, and insert data to the queue as previously described.
2. Open the Java Code view and insert the Shared Data method(s) to use to get data into the script where you want to get data from the queue. The Shared Data Service includes methods for getting the length and peeking (gets the data) and polling (gets and removes the data) data, as follows:

```
info("Get the length of queueA");
int actualLength = sharedData.getLengthOfQueue("queueA");

info("Get the most current item of queueA");
String queueValue1 = (String) sharedData.peekFirst("queueA");

info("Get the most current item of queueA - timeout after 5 seconds");
String queueValue2 = (String) sharedData.peekFirst("queueA", 5000);

info("Get the oldest item of queueA");
String queueValue1 = (String) sharedData.peekLast("queueA");

info("Get the oldest item of queueA - timeout after 5 seconds");
String queueValue2 = (String) sharedData.peekLast("queueA", 5000);

info("Get and remove the most current item from queueA");
String pollValue1 = (String) sharedData.pollFirst("queueA");

info("Remove the most current item from queueA - Timeout after 5 seconds");
String pollValue2 = (String) sharedData.pollFirst("queueA", 5000);

info("Remove the oldest item from queueA");
String pollValue1 = (String) sharedData.pollLast("queueA");

info("Remove the oldest item from queueA - Timeout after 5 seconds");
```

```
String pollValue2 = (String) sharedData.pollLast("queueA", 5000);

info("Waiting for an existing value 100 in queueA");
boolean isFound1 = sharedData.waitFor("queueA", 100);

info("Waiting for an existing value 100 in queueA - timeout afer 5 seconds");
boolean isFound2 = sharedData.waitFor("queueA", 100, 5000);
```

3. Add other custom code to the script to use the data from the queue.

21.3.8 Clearing a Shared Data Queue

Clear queues when the script is finished using the data.

To clear a shared data queue:

1. Open the Java Code view and insert the `sharedData.clearQueue` code with a the name of the queue to clear into the script where you want to clear the queue, as follows:

```
info("Clear queueA");
sharedData.clearQueue("queueA");
```

21.3.9 Destroying a Shared Queue

Destroy queues when the script is finished using the data. Destroying queues releases the queues' data and its listeners and release the memory that is allocated to a queue.

To destroy a shared data queue:

1. Open the Java Code view and insert the `sharedData.destroyQueue` code with a the name of the queue to destroy into the script where you want to destroy the queue, as follows:

```
info("Destroy queueA");
sharedData.destroyQueue("queueA");
```

21.3.10 Creating a Shared Data Hash Map

To create a shared data hash map:

1. Create an script project.
2. Make sure the Shared Data Service is enabled, the password encryption, and connection parameters are specified as previously described.
3. Open the Java Code view and insert the `sharedData.createMap` code with a life time value into the script where you want to create the hash map, as follows:

```
info("Create mapA with life time of 10 minutes");
sharedData.createMap("mapA", 10);
```

The maximum number of hash maps is 1000. The maximum capacity of a hash map is 65535. If the maximum is exceeded, an exception occurs. Once the life time expires, the hash map is destroyed.

21.3.11 Inserting Data into a Shared Data Hash Map

The types of the "values" that can be put into a hash map are the same as for queues. See "Inserting Data into a Shared Data Queue" on page 21-4 for a list of the data types.

To insert data into an existing hash map:

1. Set up the shared data service and create a hash map as previously described.
2. Open the Java Code view and insert the `sharedData.putToMap` code with a key and value into the script where you want to insert data into the hash map, as follows:

```
int iterationNum = getIteration().getTotalIterationsCompleted() + 1;
info("put key/value pair to an existing mapA");
sharedData.putToMap("mapA", "key" + iterationNum, "value" + iterationNum);

info("parameter type - String");
sharedData.putToMap("mapA", "key", "value");
String mapValue = (String) sharedData.getFromMap("mapA", "key");

info("parameter type - list of Strings");
ArrayList<String> listOfStr = new ArrayList<String>();
listOfStr.add(0, "val1");
listOfStr.add(1, "val2");
sharedData.putToMap("mapA", "key", listOfStr);
ArrayList<String> mapVal = (ArrayList<String>) sharedData.getFromMap("mapA",
   "key");

info("parameter type - boolean");
sharedData.putToMap("mapA", "key", true);

info("parameter type - int");
sharedData.putToMap("mapA", "key", 10);

info("parameter type - double");
sharedData.putToMap("mapA", "key", 10.5);

info("parameter type - long");
sharedData.putToMap("mapA", "key", 100);
```

21.3.12 Getting Data from a Shared Data Hash Map

To get data from a hash map:

1. Set up the shared data service, create a queue, and insert data to the hash map as previously described.
2. Open the Java Code view and insert the Shared Data method(s) to use to get data into the script where you want to get data from the hash map. The Shared Data Service includes methods for getting the keys of the hash map and getting data and removing data from the hash map, as follows:

```
info("Get all keys of mapA");
String [] lsKey = sharedData.getKeysOfMap("mapA");

info("Get key/value pair from mapA");
String actualValue = (String) sharedData.getFromMap("mapA", "key");

info("Get key/value pair from mapA - timeout after 5 seconds");
String actualValue1 = (String) sharedData.getFromMap("mapA", "key", 5000);

info("Remove key/value pair from mapA");
String removeValue = (String) sharedData.removeFromMap("mapA", "key");

info("Remove key/value pair from mapA - timeout after 5 seconds");
String removeValue1 = (String) sharedData.removeFromMap("mapA", "key", 5000);
```

3. Add other custom code to the script to use the data from the hash map.

21.3.13 Clearing a Shared Data Hash Map

Clear hash maps when the script is finished using the data.

To clear a shared data hash map:

1. Open the Java Code view and insert the `sharedData.clearMap` code with a the name of the map to clear into the script where you want to clear the hash map, as follows:

```
info("Clear mapA");
sharedData.clearMap("mapA");
```

21.3.14 Destroying a Shared Data Hash Map

Destroy hash maps when the script is finished using the data. Destroying hash maps releases the map's data and its listeners and release the memory that is allocated to a map.

To destroy a shared data hash map:

1. Open the Java Code view and insert the `sharedData.destroyMap` code with a the name of the map to destroy into the script where you want to destroy the hash map, as follows:

```
info("Destroy mapA");
sharedData.destroyMap("mapA");
```

21.4 Using The Shared Data API

The Shared Data Module includes a script Application Programming Interface (API) for Shared Data actions. You can use the Shared Data API to pass data between load testing scripts running as Virtual Users in Oracle Load Testing. The Shared Data can also be used to pass data between functional testing scripts running from the command line. Commands that are specific to the Shared Data Module are part of the "sharedData" class. The Shared Data service must be enabled separately for use with other types of scripts. You can also leverage other commands from other enabled classes (services) or general Java commands in your scripts.

Some examples of the Shared Data Module API include:

- clearMap
- clearQueue
- createMap
- createQueue
- destroyMap
- destroyQueue
- getFromMap
- getKeysOfMap
- getLengthOfQueue
- offerFirst
- offerLast

- peekFirst
- peekLast
- pollFirst
- pollLast
- putToMap
- removeFromMap
- setConnectionParameters
- waitFor

The setConnectionParameters API method can be added using the script Tree View. Additional methods can be added using the Java Code view.

22
Using the Block Scenarios Module

This chapter provides instructions on using the OpenScript Block Scenarios Module, which provides support for generating complex Virtual User scenarios.

22.1 About the Block Scenario Module

The Block Scenarios Module is an auxiliary module to generate complex Virtual User scenarios. These scenarios can have multiple execution paths. When creating Block Scenarios, you specify the distribution of the execution paths by selecting the percentage of blocks and scripts that make up a path.

Block scenarios can contain both virtual user scripts and script blocks. A script block can contain one or more scripts as well as child blocks and child scripts. Specific functions within a script can also be called. Each block and script can be set to run always or be assigned a percentage that specifies the run distribution of the blocks and scripts within the block scenario. The following example shows a block scenario script in XML format (a tree view is also available in OpenScript):

```
<block name="Main" iterations=3">
 <script name="Login"/>
 <block name="SearchOrAct">
   <script name="Search" percent="90"/>
   <block name="Act" percent="10">
      <script name="Buy" percent="50"/>
      <script name="Sell" percent="50"/>
   </block>
 </block>
 <script name="Logout"/>
</block>
```

The "Main" block is the parent block of the block scenario which contains two scripts, "Login" and "Logout", and a script block, "SearchOrAct". The "SearchOrAct" script block contains one script, "Search", and a child block containing two scripts, "Buy" and "Sell".

The "Main" block is set to run three iterations. The "Main" block will always run the "Login" script, "SearchOrAct" block, and "Logout" scripts during each iteration. When the "SearchOrAct" block is run, there is a 90 percent chance the "Search" script will be run and 10 percent chance the "Act" block will be run. If the "Act" block is run, there is a 50 percent chance the "Buy" script will be run and a 50 percent chance the "Sell" script will be run.

The scenario runs scripts using a random distribution within the script blocks. Each iteration, the outcome of what the Virtual User does could be different. The percentage distribution is random, so it is not guaranteed to match the exact percentages

specified. However, when run enough times to generate a large enough sample size, the results should be close to the specified percentages. The percentages set must equal 100 percent for scripts within a block, child blocks within a block, and blocks within a scenario. The percentage value displays in red in the block XML tree view of the **Details** view if the percentage total does not equal 100 percent.

When running with multiple virtual users, random probability will decide which path each Virtual User goes through the scenario.

Scripts must be added to the Block Scenario project as script assets before the building the block scenario in the UI.

22.2 Creating Block Scenario Projects

Block Scenario project are created similar to other script projects. However, instead of recording the scenario, scripts are added manually as script assets. The scripts and blocks are added to the block scenario tree based upon how you want to build the scenario and execution paths.

To creating a block scenario project:

1. Start OpenScript.
2. Select **New** from the **File** menu.
3. Expand the Load Testing group.
4. Select **Block Scenarios Script**.
5. Click **Next**.
6. Select the Repository and Workspace.
7. Enter a script name.
8. Click **Finish**. A new Script tree is created in the Script View.
9. Add scripts to the block scenario project as script assets. See Section 22.2.1, "Adding Script Assets to Block Scenario Projects".
10. Expand the **Run** node of the script and select the **Run Scenario** node. The **Details** view shows the Block XML and Block XML Tree tabs.
11. Right-click nodes in the Block XML Tree of the **Details** view to add blocks and scripts to the tree.
12. Save the block scenario script when finished.

22.2.1 Adding Script Assets to Block Scenario Projects

You must add scripts as script assets to block scenario projects before building the block scenario structure. The scripts must exist before they can be added.

To add assets to a block scenario project:

1. Open or create a block scenario project.
2. Select the **Assets** tab in the script view.
3. Select **Script** as the type of asset to add and click **Add**.
4. Set the **Relative to** option. The **Relative to current script** and **Relative to a repository** options specify how the current script will locate the specified script asset. The **Relative to a repository** option locates the script asset by a repository path such as, `[Repository: Default] Default!/WebTutor`, if the asset is

selected from a repository. The **Relative to current script** option locates the script asset by a relative path such as `../WebTutor`. Selecting the The **Relative to current script** option is not recommended as script-relative paths are more brittle than repository-relative paths if scripts are moved or shared.

The following are guidelines when using script assets in a team or distributed environment:

- Do not use Absolute Paths when referring to assets or saving assets. Oracle Load Testing does not support absolute paths.
- OpenScript, Oracle Test Manager, Oracle Load Testing, and all command-line agents should all use the same shared repository names and paths.
- Do not refer to an asset in another repository by a relative path.

5. Click **OK** to add the asset to the script properties.
6. Repeat the steps to add all of the scripts that will be included in the block scenario project.
7. Click **OK** when finished adding script assets to close the script properties.
8. Use the right-click menu options of the Block XML Tree tab of the **Details** view to add blocks and scripts to the scenario.
9. Save the block scenario script when finished.

22.3 Modifying Block Scenarios

This section explains the procedures for adding and editing blocks and scripts to the block scenario tree.

22.3.1 Adding Blocks

Blocks are used in a block scenario to create groupings of scripts and child blocks that create the structure and execution paths that make up the scenario.

To add a block to a block scenario project:

1. Open or create a Block Scenario project and add script assets.
2. Expand the **Run** node of the script tree and select the **Run Scenario** node. The **Details** view shows the Block XML and Block XML Tree tabs.
3. Right-click nodes in the Block XML Tree of the **Details** view and select **Add Block**. This dialog box lets you add or edit a script block in a Block Scenarios project.

   Name: Specifies the name of the script block.

   Iterations: Specifies the number of iterations to run the script block.

   Schedule: Specifies the scheduling of the script block.

   - **Always schedule to run**: When selected, the block is always run during each iteration of the block scenario.
   - **Only schedule to run with [] % of VUs**: When selected, the block (and script(s)/block(s) within the block) will be run the based upon the specified percentage. The scenario runs scripts using a random distribution within the script block. For example, if you define a block scenario as follows:

```
<block name="Main" iterations=3">
 <script name="Login"/>
 <block name="SearchOrAct">
```

```
    <script name="Search" percent="90"/>
    <block name="Act" percent="10">
       <script name="Buy" percent="50"/>
       <script name="Sell" percent="50"/>
    </block>
  </block>
  <script name="Logout"/>
</block>
```

During playback with 1 VU, the VU will iterate over the "Main" block for 3 iterations. During each iteration of the "Main" block, the VU will always run the "Login" script, run either the "Search" script or the "Act" block, then always run the "Logout" script. During each iteration of the "SearchOrAct" block, there is a 90 percent chance the VU will run the "Search" script and there is a 10 percent chance the VU will run the "Act" block. In the 10 percent chance that the VU will run the "Act" block, there will be a 50 percent chance the VU will run the Buy script, and a 50 percent chance the VU will run the "Sell" script.

The percentages set must equal 100 percent for scripts within a block, child blocks within a block, and blocks within a scenario. The percentage value displays in red in the block tree view of the **Details** view if the percentage total does not equal 100 percent.

4. Enter a name for the block.
5. Specify the number of iterations to run the block.
6. Specify the Schedule for how the block will run. If you want the block to run always during each iteration, select **Always schedule to run**. If you want the block to run a percentage of the time within the scenario, select **Only schedule to run with [] % of VUs** and specify the percentage.
7. Click **OK**.
8. Repeat the steps to add other blocks to the scenario.

22.3.2 Adding Scripts

Scripts perform the user actions that make up the scenario.

To add a script to a block scenario project:

1. Open or create a Block Scenario project and add script assets.
2. Expand the **Run** node of the script tree and select the **Run Scenario** node. The **Details** view shows the Block XML and Block XML Tree tabs.
3. Right-click nodes in the Block XML Tree of the **Details** view and select **Add Script**. This dialog box lets you specify the scripts to include in a script block of block scenario project.

   Script: Specifies the name of the script to include in the script block.

   Run: Specifies if the entire script or a specific function within the script will be run.

   - **Script**: When selected, the entire script is run.
   - **Specific Function**: When selected, only the selected script function is run. Select the function from the list.

   Iterations: Specifies the number of iterations to run the script block.

Schedule: Specifies the scheduling of the script block.

- **Always schedule to run**: When selected, the block is always run during each iteration of the block scenario.
- **Only schedule to run with [] % of VUs**: When selected, the block (and script(s)/block(s) within the block) will be run the based upon the specified percentage. The scenario runs scripts using a random distribution within the script block. For example, if you define a block scenario as follows:

```
<block name="Main" iterations=3">
 <script name="Login"/>
 <block name="SearchOrAct">
   <script name="Search" percent="90"/>
   <block name="Act" percent="10">
      <script name="Buy" percent="50"/>
      <script name="Sell" percent="50"/>
   </block>
 </block>
 <script name="Logout"/>
</block>
```

  During playback with 1 VU, the VU will iterate over the "Main" block for 3 iterations. During each iteration of the "Main" block, the VU will always run the "Login" script, run either the "Search" script or the "Act" block, then always run the "Logout" script. During each iteration of the "SearchOrAct" block, there is a 90 percent chance the VU will run the "Search" script and there is a 10 percent chance the VU will run the "Act" block. In the 10 percent chance that the VU will run the "Act" block, there will be a 50 percent chance the VU will run the Buy script, and a 50 percent chance the VU will run the "Sell" script.

  The percentages set must equal 100 percent for scripts within a block, child blocks within a block, and blocks within a scenario. The percentage value displays in red in the block tree view of the **Details** view if the percentage total does not equal 100 percent.

4. Select the script to run from the **Script** list or click **New** to add a new script.
5. Specify the number of iterations to run the script.
6. Specify the Schedule for how the script will run. If you want the script to run always during each iteration, select **Always schedule to run**. If you want the script to run a percentage of the time within the block or scenario, select **Only schedule to run with [] % of VUs** and specify the percentage.
7. Click **OK**.
8. Repeat the steps to add other scripts to the scenario.

22.3.3 Adding Child Blocks

Child blocks can be used to create groupings of scripts and other child blocks that create the structure and execution paths that make up the scenario.

To add a child block to a block scenario project:

1. Open or create a Block Scenario project and add script assets.
2. Expand the **Run** node of the script tree and select the **Run Scenario** node. The **Details** view shows the Block XML and Block XML Tree tabs.
3. Select the node in the Block XML Tree where you want to add the child block.

4. Right-click and select **Add Block as Child**.
5. Enter a name for the child block.
6. Specify the number of iterations to run the block.
7. Specify the Schedule for how the block will run. If you want the block to run always during each iteration, select **Always schedule to run**. If you want the block to run a percentage of the time within the scenario, select **Only schedule to run with [] % of VUs** and specify the percentage.
8. Click **OK**.
9. Repeat the steps to add other child blocks to the scenario.

22.3.4 Adding Child Scripts

To add a child script to a block scenario project:

1. Open or create a Block Scenario project and add script assets.
2. Expand the **Run** node of the script tree and select the **Run Scenario** node. The **Details** view shows the Block XML and Block XML Tree tabs.
3. Select the node in the Block XML tree where you want to add the child script.
4. Right-click and select **Add Script as Child**.
5. Select the script to run from the **Script** list or click **New** to add a new script.
6. Specify the number of iterations to run the script.
7. Specify the Schedule for how the script will run. If you want the script to run always during each iteration,, select **Always schedule to run**. If you want the script to run a percentage of the time within the scenario, select **Only schedule to run with [] % of VUs** and specify the percentage.
8. Click **OK**.
9. Repeat the steps to add other child scripts to the scenario.

22.3.5 Editing Block and Script Settings

To edit block or script settings in a block scenario project:

1. Open or create a Block Scenario project and add script assets.
2. Expand the **Run** node of the script tree and select the **Run Scenario** node. The **Details** view shows the Block XML and Block XML Tree tabs.
3. Select the node in the Block XML Tree that you want to edit.
4. Right-click and select **Edit**.
5. Edit the block or script settings.
6. Click **OK**.

22.3.6 Moving Blocks and Scripts within a Scenario

To move a block or script within a block scenario project:

1. Open or create a Block Scenario project and add script assets.
2. Expand the **Run** node of the script tree and select the **Run Scenario** node. The **Details** view shows the Block XML and Block XML Tree tabs.

3. Select the block or script you wish to move in the Block XML Tree tab of the **Details** view.
4. Right-click the node and select **Move Up** or **Move Down**.

22.3.7 Deleting Blocks and Scripts from a Scenario

To delete blocks or scripts from a block scenario project:

1. Open or create a Block Scenario project and add script assets.
2. Expand the **Run** node of the script tree and select the **Run Scenario** node. The **Details** view shows the Block XML and Block XML Tree tabs.
3. Select the node in the Block XML Tree that you want to delete.
4. Right-click the node and select **Delete**.

22.4 Playing Back Block Scenario Scripts

Block scenario scripts can be played back in OpenScript as a single Virtual User or in Oracle Load Testing in multi-user scenarios.

The scenario runs scripts using a random distribution within the script blocks. Each iteration, the outcome of what the Virtual User does could be different. The percentage distribution is random, so it is not guaranteed to match the exact percentages specified. However, when run enough times to generate a large enough sample size, the results should be close to the specified percentages.

When running with multiple virtual users, random probability will decide which path each Virtual User goes through the scenario.

A

Script Command Line Reference

This appendix lists the parameters for running OpenScript scripts from the command line. All OpenScript scripts may be run from the command line assuming that:

- All resources that the script depends on, including databank files, object library files, and other scripts it runs, must be accessible from the machine where the script is run.
- The installed version of OpenScript or agent is newer or the same as the version used to create the script.
- Only Firefox browser is supported on Linux.

To run a script from the Windows command line, type:

`[OpenScript Install Dir]/runScript.bat` *`Path/ScriptName`*`.jwg [`*`options`*`]`

If OpenScript is not installed, but the OATS agent is installed, type on Windows and Linux respectively:

`[OATS Install Dir]/agent/runScript.bat` *`Path/ScriptName`*`.jwg [`*`options`*`]`

`[OATS Install Dir]/agent/runScript.sh` *`Path/ScriptName`*`.jwg [`*`options`*`]`

`Path` is the full drive and directory path of the file location. When specifying file paths, use forward-slash (/) instead of back-slash (\) to ensure the back-slash notation is not misinterpreted in Java. If you must use back-slash notation, use double back-slashes (\\) in the file paths. For example, `[OATS Install Dir]\\agent\\runScript.bat`.

`[`*`options`*`]` may consist of any number of agent command line settings.

A.1 Specifying Command Line Settings

This section describes how to use the command line settings.

- Zero or more agent command line settings may be specified using the following format:

  `-settingId settingValue`

  Example:

  ```
  runScript
  "C:/OracleATS/OpenScript/DefaultRepository/Default!/script1/script1.jwg"
  -iterations 5 -iterationDelay 5
  ```

- If a *settingValue* contains spaces, the value may be enclosed inside double-quotation marks.
- If a setting specifies a boolean true/false value, and no *settingValue* is specified, then a true value is assumed.
- Except where specifically noted, all *settingId* values are NOT case-sensitive. For example, `-iterations 10` means the same thing as `-ITERATIONS 10`.
- It is possible to view all settings passed to a script by typing the following code inside an OpenScript script:

```
info(getSettings().toString());
```

- Custom settings may be used in a script and specified on the command line. For example, consider a script with a setting "todaysURLToTest":

```
public void run() throws Exception {
    info("Today we will test the URL " + getSettings().get("todaysURLToTest"));
  }
```

  The setting may be specified on the command line using:

```
-todaysURLToTest "http://www.oracle.com/"
```

  The following are reserved keywords that cannot be used as custom setting names:

```
-noUpdate
-boot
-classloaderProperties
-plugins
-firstUse
-newUpdates
-update
-password
-keyring
```

- If many settings are required, use the `-propertiesPath` setting to specify a file containing a larger number of properties. Select **Export Playback Settings** from the **Tools** menu to generate a .properties file for the script. See the `-propertiesPath` setting in the General Settings on page A-9 for additional information.

A.2 Supported Agent Command Line Settings

Certain settings only apply to scripts that have a particular module applied. For example, HTTP load test script settings do not apply to Web functional test scripts.

A.2.1 General Settings

The following table lists the General command line settings.

Setting	Description
`-ENCRYPTION_PROPERTIES_FILE` *path*`\encryption.properties`	Specify the full path and file name to use for encrypted password authentication. The file name is C:\Documents and Settings*username*\osworkspace\.metadata\.plugins\oracle.oats.scripting.utilities\encryption.properties.

Setting	Description
`-databank_preparation_timeout` *`timeout`*	Specifies how much time to spend preparing a databank for use before timing out. The value is in seconds. If this option is not specified, the default value of 30 seconds will be used. This setting includes the total time to do all of the following activities: If using a Database-backed databank: ■ Connect to the database ■ Query ■ Read records, write into the file ■ Create the index simultaneously ■ Disconnect If using a CSV-backed databank: ■ Time required to parse the CSV file and create the index If using Random Unique: ■ Time to shuffle the index Example usage: `-databank_preparation_timeout 60`

Setting	Description
`-dbopts "dboption, dboption, ..."`	Specify which databank records to use when playing back the script, where *dboption* exists for each databank being configured. One or more *dboption* may be specified. Each *dboption* has the form: *"settingName*=settingValue:*settingName=settingValue*:..." At least one setting name=value pair must be specified. Only the `alias` setting is required. All other settings are optional. The following setting names are valid for a *dboption*: `alias=`*alias* - (Required) Specify the alias of the databank being configured. `startIndex=`*startIndex* - Specify the starting databank record to use. First record is 1. Default value is 1. `rangeMin=`*rangeMin* - Specify the starting databank record range (inclusive). First record is 1. Default value is 1. `rangeMax=`*rangeMax* - Specify the ending databank record range (inclusive). First record is 1. Specify -1 to indicate that the range max is equal to the number of databank records. Default is -1. `advance=`*advance_mode* - Specify the **Advance to Next Record** setting. *advance_mode* value may be one of the following: ■ `advance=WHEN_SCRIPT_REQUESTS` - Corresponds to the option, "Advance to Next Record: **When Script Requests a Record**". This is the default value if no setting specified. ■ `advance=EACH_OCCURRENCE` - Corresponds to the option, "Advance to Next Record: **Each Occurrence**". Optionally use `advance-column` to specify a column name. If no column is specified, then any column will trigger record advancement. ■ `advance=EACH_ITERATION` - Corresponds to the option, "Advance to Next Record: **Each Iteration**". ■ `advance=KEEP_FIRST_RECORD` - Corresponds to the option, "Advance to Next Record: **Keep the First Record Assigned**". Example inputs: ■ `advance=WHEN_SCRIPT_REQUESTS` ■ `advance=EACH_OCCURRENCE` ■ `advance=EACH_OCCURRENCE:advance-column=firstName` ■ `advance=EACH_ITERATION` ■ `advance=KEEP_FIRST_RECORD` `advance-column=`*columnName* - Specify the name of a databank column. If no column is specified, then any column will trigger record advancement.

Setting	Description

`-dbopts` (cont'd)	`select=`*`select_mode`* - Specify the **Select Next Record** setting. *`select_mode`* value may be one of the following:
	■ `select=SEQUENTIAL` - Corresponds to the option, "Select Next Record: **Sequentially**". This is the default selection mode if no setting specified.
	■ `select=RANDOM` - Corresponds to the option, "Select Next Record: **Randomly**".
	■ `select=SHUFFLE` - Corresponds to the option, "Select Next Record: **By Shuffling**".
	`select-seed=`*`seed_number`* - Specify a seed to use when choosing a random record. This setting is optional, and is only used when `select=RANDOM` or when `select=SHUFFLE`. *`select_mode`* value is any non-negative long value. Specify 0 to let OpenScript generate a seed using the current system time. Default value is 0.
	Example inputs:
	■ `select=RANDOM:select-seed=437292634`
	■ `select=SHUFFLE:select-seed=372389237`
	■ `select=RANDOM:select-seed=0`
	■ `select=SHUFFLE:select-seed=0`
	`whenOut=`*`whenOut_mode`* - Specify the **When Out of Records** setting. *`whenOut_mode`* value may be one of the following:
	■ `whenOut=LOOP_FOREVER` - Corresponds to the option, "When Out of Records: **Loop Over Range**". Default value if none specified.
	■ `whenOut=STOP_USER` - Corresponds to the option, "When Out of Records: **Stop the User**".
	■ `whenOut=KEEP_SAME` - Corresponds to the option, "When Out of Records: **Keep the First Record Assigned**".
	Example inputs:
	■ `whenOut=LOOP_FOREVER`
	■ `whenOut=STOP_USER`
	■ `whenOut=KEEP_SAME`
	`independentCursors=true|false` - Specify the **Let Each User Iterate Over Records Independently** setting. This setting is not recommended for command-line playback, because it is only useful for scenarios with more than one virtual user. Value may be one of the following:
	■ `independentCursors=FALSE` - Default if not specified.
	■ `independentCursors=TRUE`
	Example usage of `-dbopts`:
	`-dbopts alias=fmstocks_ data:startIndex=2:rangeMin=2:rangeMax=5 :whenOut=STOP_USER`
	`-iterations 10 -dbopts alias=fmstocks_ data:select=SHUFFLE`

Setting	Description

-dbopts (cont'd)	-iterations=10 -dbopts alias=fmstocks_data:select=RANDOM:select-seed=4728292:advance=EACH_ITERATION
	
	-iterations=100 -dbopts alias=products:select=SEQUENTIAL:advance=EACH_ITERATION, alias=prices:select=SEQUENTIAL:advance=EACH_OCCURRENCE:advance-column=price
	
	Notes and Limitations
	
	Certain setting combinations are not allowed, or may cause exceptions when the script is run. The following are situations to be aware of when using -dbopts.
	
	1. The whenOut and independentCursors options are not available when select=RANDOM. When random is selected, an infinite supply of random records exists, and it requires that independentCursors=FALSE.
	2. Virtual users may still request an individual record using getRecord(*n*) after all records are used up, and whenOut=KEEP_SAME.
	3. The getRecord(*n*), getFirstRecord(), and getLastRecord() Java code methods do not advance the record cursor used by getNextDatabankRecord(). Therefore:
	
	getNextDatabankRecord();// returns 1
	getRecord(7);// returns 7
	getNextDatabankRecord();//returns 2, not 7
	
	4. The getRecord(*n*), getFirstRecord(), and getLastRecord() Java code methods throw an exception when they are invoked if select=SHUFFLE or select=RANDOM.
	5. The getRecord(*n*), getFirstRecord(), and getLastRecord() Java code methods throw an exception if they are invoked and the databank is not indexed.
	6. select-seed is only available when select=RANDOM or select=SHUFFLE.
	7. When individualCursors=true and select=SHUFFLE, all virtual users will get the same set of random records in the same order.
	8. A specific databank range and starting index may not be set if the databank cannot be indexed.
	9. select=RANDOM or select=SHUFFLE is only allowed when the databank can be indexed.
	10. select=SHUFFLE is only allowed when the databank can be indexed and when there are fewer than 200,000 records.
	11. individualCursors=true is only allowed when a databank can be indexed.

Setting	Description

`-dboptions` *`alias`*`:`*`index`*`:`*`mode`*`:`*`range`*`,`*`alias`*`:` *`index`*`:`*`mode`*`:`*`range`*`,...`	(Deprecated - used for version 9.10) Specify which databank records to use when playing back the script. Use
	`-dboptions` *`alias`*`:`*`index`*`:`*`mode`*`:`*`range`*`,`*`alias`*`:`*`index`*`:`*`mode`*`:`*`range`*`,...` where *`alias`* is a databank alias, *`index`* is the first databank record to retrieve (first record is 1), and *`mode`* is one of the following strings:
	■ `FIRST_RECORD_ONLY` - The Virtual User will only use the first record assigned to it.
	■ `USE_ALL_RECORDS` - Stop after all records are used.
	■ `STOP_AFTER_LAST_RECORD` - Stop after the last record in the databank file is used.
	■ `LOOP_FOREVER` - Loop over all records for as many iterations are specified in `-iterations`. This is the default mode if no mode is specified.
	`range` (optional) specifies which range of records to retrieve, in the format: *`rangeMin`*`:`*`rangeMax`*. For example, `5:10` means the range of records starting at record 5 and ending at record 10. If not specified, assumes all records in the databank.
	Regardless of the databank setting, the VU will never run for more iterations than specified in the `-iterations` setting.
	Example 1: To loop over records starting at the 5th row in the databank with alias "fmstocks_data", use:
	`-dboptions fmstocks_data:5:LOOP_FOREVER`
	Example 2: A databank with alias "fmstocks_data" contains 15 records. To iterate over records 5 through 10, starting at record 7 (i.e. 7,8,9,10,5,6,7,8,9,10,5,6...), for at most 100 iterations, use:
	`-iterations 100 -dboptions fmstocks_data:7:LOOP_FOREVER:5:10`

Setting	Description
`-delayPercentage` *mode*	Specify how long to delay between steps in a script. Depending on the *mode*, this setting may be used in conjunction with other settings. *mode* should be one of the following numbers: `-2` No Delay. `1` Use Recorded Delay. Optionally specify a minimum and maximum amount of delay using `-delayMin` *n* and `-delayMax` *m*, where *n* and *m* are given in seconds. If unspecified, the default minimum and maximum amount of delay to use are 0 and 5 seconds, respectively. `-1` Delay for a random number of seconds. Optionally specify a minimum and maximum amount of delay using `-delayMin` *n* and `-delayMax` *m*, where *n* and *m* are given in seconds. If unspecified, the default minimum and maximum amount of delay to use are 0 and 5 seconds, respectively. `0` Delay for a random number of seconds using a percentage threshold range around the recorded delay. Optionally specify a lower and upper percentage delay range using `-delayLower` *p* and `-delayUpper` *q*, where *p* and *q* are a decimal value between 0 and 1. If unspecified, 0 will be used for the percentage threshold, resulting in the actual recorded delay being used.
`-iterationDelay` *n*	Pause for *n* seconds between iterations.
`-iterations` *n*	Run *n* iterations of the script.
`-loglocalvudisplay true|false`	Create VUDisplay.txt and VUDisplay.csv result output files in the folder from which the agent was launched. The Virtual User Display output is not supported by functional test modules. The `loglocalvudisplay` setting is case-sensitive and must be exactly `loglocalvudisplay`.
`-noReport true|false`	Set to true to disable generation of the results report. The default is `false`.
`-portnumbermaximum` *portNumber*	Specifies the port number to use as the maximum port number for the record and playback port range to avoid port conflicts when multiple users run OpenScript from a single installation using multiple concurrent interactive desktop sessions (for example, Terminal Server or Remote Desktop sessions). The default is `7888`.
`-portnumberminimum` *portNumber*	Specifies the port number to use as the minimum port number for the record and playback port range to avoid port conflicts when multiple users run OpenScript from a single installation using multiple concurrent interactive desktop sessions (for example, Terminal Server or Remote Desktop sessions). The default is `7777`.
`-preserveVariables true|false`	Set to true to preserve variables between iterations, false to clear variables between iterations. Variables refer to variables set using the Variables service, i.e. `getVariables().set()`, not local Java code variables.

Setting	Description
`-propertiesPath` *`path`*	Specify the full path of a Java .properties file containing additional command line setting name=value pairs to add. Example usage: `-propertiesPath "C:/additionalSettings.properties"`

`-replaceURLs` *`originalURL1`*`=`*`replacementURL1`*`,`*`originalURL2`*`=`*`replacementURL2`*`,[...]`	Specifies the URL replacement string. During playback, anytime the agent makes a request to a URL starting with a segment, *`originalURL`*, the agent replaces the original URL segment with *`replacementURL`*. This feature is only supported for Load Test scripts.
	
	■ *`originalURL`* - Specify the starting segment of the *`URL`*`:`*`port`* that appears in the script that should be replaced. This value is case-sensitive.
	
	■ *`replacementURL`* - Specify the new starting segment *`URL`*`:`*`port`* that the agent requests instead of *`originalURL`*.
	
	For both parameters, if the protocol is omitted, HTTP protocol is assumed. If no port is specified after the host, port 80 is assumed for HTTP protocol, and port 443 is assumed for HTTPS protocol. URLs are replaced after all correlations are applied. One or more URL replacement pairs may be specified, separating each replacement pair with a comma. The following examples show the format of strings:
	
	`-replaceURLs "test_server:7789=production_server:7789"`
	
	`-replaceURLs "test_server:7789=production_server:7789,https://staging.oracle.com/main=https://production.oracle.com/welcome"`

`-reportName` *`name`*	Override the file name used when generating the result report HTML file. Specify the name of the report including an extension, but excluding any path information. For example: `-reportName results.html` If not specified, OpenScript will determine a meaningful name for the result report HTML file based on the type of script being run. For example, an HTTP load test script calls its report httpReport.htm. A functional test script such as a Web functional test script calls its report FTReport.htm.

Setting	Description
`-repository` *repositoryString*	Specify the physical locations of one or more named repositories. *repositoryString* takes the form `repositoryName=`*fullPathToRepository*`,repositoryName2=`*fullPathToRepository2*`,...` This setting is required. When you run Version 9.0 OpenScript scripts from the Version 9.10 Command Line Interface, you must use the `-repository` command line argument to pass to the Oracle Load Testing Virtual User Agent all repositories used by the script. This includes all repositories used for all Databanks, Object Libraries, and Child Scripts X levels deep. In Version 9.0, The Oracle Load Testing Virtual User Agent would use the default repository (`C:/<`*installdir*`>/OFT`) if the repository command line argument was not specified. In Version 9.10, the multi-user environment default repository changed to `<`*yourmachinename*`>.<`*yourusername*`>.Default` in the <*user.home*> location. This change requires that the *repositoryName=repositoryPath* string be specified as a command line argument for all repositories including the default repository. Always specify the *repositoryString* inside double-quotes. Example usage: `-repository "JohnComputer.johndoe.Default=D:/OracleATS/OFT,SharedRepository=\\testserver/OurScripts"`
`-resultReportFolder` *path*	Specify the output folder for all result report files. If it is not specified, the default output folder will be: `[`*script folder*`]/`*SessionId* Example usage: `-resultReportFolder "C:/result"` The output folder would be: `C:/result/`*SessionId*
`-stopVUserOnFailure true|false`	Specify whether or not the virtual user will continue running more iterations after a fatal error. Use the Error Recovery settings to control which errors should be fatal.

A.2.2 Browser Settings

The following table lists the Browser command line settings.

Setting	Description
`-browser.type` *`type`*	Specify the browser type to use for script playback where *`type`* is one of the following (use exact case and no spaces): ▪ `InternetExplorer` ▪ `Firefox` The default is `InternetExplorer` on Windows. The default is `Firefox` on Linux.
`-browser.startupTimeout` *`n`*	Specify *`n`* seconds to wait for the browser to start before timing out. The default is 15 seconds.
`-browser.overridePath` *`path`*	OpenScript automatically detects where Internet Explorer and Firefox browser processes physically exist in the file system. In case the path to one of these browsers is incorrect, specify an alternative path to use when launching the specified browser type. This setting is not intended to be used to specify the path to an unsupported browser. Example usage: `-browser.pathOverride "D:/Programs/Firefox/firefox.exe"`
`-browser.extraArgs` *`args`*	Specify any additional startup arguments that OpenSript should use when launching the browser process on playback. The default is no additional arguments other than what OpenScript may require internally.
`-browser.hide true|false`	For Internet Explorer browser only, specify `true` to hide the browser during playback. This setting has no effect on Firefox. If certain actions in the script require the browser to be visible, such as key presses and physical mouse clicks, the script will not playback correctly. The default is false; browser is visible.

A.2.3 HTTP Settings

The following tables list the HTTP command line settings.

A.2.3.1 Proxy

The following table lists the proxy command line settings.

Setting	Description
`-http.proxyHost` *`host`*	Set the proxy host to the specified *`host`*. If no proxy host is specified, then no proxy is used. Example usage: `-http.proxyHost "proxyserver.mycompany.com"`
`-http.proxyPort` *`port_number`*	Set the proxy port to the specified *`port_number`*. Example usage: `-http.proxyPort 1234`

Setting	Description
`-http.proxyUsername` *`username`*	If the proxy host requires authentication, provide the *`username`*. Ignored if `http.proxyHost` and `http.proxyPort` are not set.
`-http.proxyPassword` *`password`*	If the proxy host requires authentication, provide the *`password`*. Ignored if `http.proxyHost` and `http.proxyPort` are not set. The password is *not* encrypted when provided on the command line.
`-http.nonProxyHosts` *`hostsList`*	Specify a list of host names that the agent should *not* forward through the specified proxy defined by `http.proxyHost` and `http.proxyPort`. Separate multiple host names by a `|` delimeter. The host name must match the host being requested, as seen in the `Host` HTTP request header.

A.2.3.2 Compression

The following table lists the compression command line settings.

Setting	Description
`-http.useGzip true|false`	Enable GZIP compression. The client will add `gzip` to every `Accept-Encoding` HTTP request header. If the server responds with GZIP-compressed data, OpenScript will decompress it. The default value is `true`.
`-http.useDeflate true|false`	Enable Deflate compression. The client will add `deflate` to every `Accept-Encoding` HTTP request header. If the server responds with Deflate-compressed data, OpenScript will decompress it. The default value is `false`.

A.2.3.3 Headers

The following table lists the header command line settings.

Setting	Description
`-http.userAgentHeader`	Browser Emulation. Specify the `User-Agent` header string to use when making requests. It is not required and harmless to include the `User-Agent:` prefix in the value itself. Most OpenScript scripts override this setting with the `User-Agent` header value found during recording. Example usage: `-http.userAgentHeader "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)"`
`-http.globalHeaders` *`headersList`*	Specify any custom "Global Headers: string to use in the Request header for script playback. The format is in the form: *name1*:*value1*;*name2*:*value2*;*name3*:*value3*. For example: x-oracle-slm-message-id: bcn=<*beacon_name*>; svc=<*service_name*>;test=<*test_name*>;step={{@getTopLevelStepName())}}.
`-http.language` *`language`*	Set the value of the `Accept-Language` HTTP header. If no value specified, the agent tries to determine the language header by looking at the system language.

Setting	Description
`-http.version` *version*	Specify the HTTP version string to append to each request, i.e. `HTTP/1.1` or `HTTP/1.0`. If no value specified, agent will use `HTTP/1.1`
`-http.accept` *acceptString*	Specify the HTTP `Accept` header to send by default for all requests.

`-http.requestInOne true|false`	Specify if the HTTP request is sent as one TCP package or two TCP packages. An HTTP request may contain two parts: headers and postdata. If the setting is set to `true`, the headers together with the postdata is sent in one TCP package. If `false`, the headers and post data are sent in order in two TCP packages. The default is `false`.
	Example usage:
	`getSettings().set("http.requestInOne", true);`
	`http.post(...);`
	`getSettings().set("http.requestInOne", false);`

A.2.3.4 Connections

The following table lists the connection command line settings.

Setting	Description
`-http.useKeepAlive true|false`	Specify whether or not to keep HTTP connections alive after using them. Default is `true` to enable keep-alive connections.
`-http.maxKeepAliveRequests` *n*	Specify an integer *n* number of requests to make before closing a keep-alive connection. Must be used together with `http.useKeepAlive`.
`-http.maxConnections` *n*	Specify an integer *n* maximum number of connections a single virtual user may have open concurrently before blocking (waiting) for a new connection to become available. This setting only applies in OpenScript when concurrent sections are used in the script. The default value if not set is `2` connections per virtual user.
`-http.expireDuration` *n*	Specify the number of seconds *n* to use for the socket 'idle timeout' and use a new connection when reusing an idle-timeout socket. This is used to specify the timeout for a socket that gets closed by the server side after a long idle period.
`-http.connectionSpeed` *n*	Specify the number of bytes/sec *n* to limit the download speed. For example, to download at 56KBps, specify `56000`. If no value specified, or an empty string is specified, this method assumes true line speed should be used.

A.2.3.5 Other

The following table lists other HTTP command line settings.

Setting	Description
`-http.ignoredUrls` *`list_of_urls`*	Specify a comma-separated list of case-insensitive URLs not to request during playback. The URLs only need to be the ENDS of URLs, not the entire URLs. Example usage: `-http.ignoredUrls ".jpg,.gif,.css."`
`-http.cacheEmulation` *`mode`*	Specify if/how the browser should cache downloaded contents. Each virtual user maintains its own cache. The *`mode`* value is one of the following values: ■ `1` - Repeat User; use a cache and do not clear the cache after each iteration. ■ `2` - First Time User; use a cache and clear the cache after each iteration. ■ `3` - Disable "Cache download pages" (do not cache). ■ `4` - Enable "Cache download pages" and "Automatically". ■ `5` - Enable "Cache download pages" and "Every Visit to the page". The default value is `3`, do not cache.
`-http.clearCacheAfterIteration true|false`	Specify whether or not the browser's cache should be cleared after the script completes each iteration of its run() section. This setting only applies when the cache is enabled using `http.cacheEmulation`. The default value is `true`, clear cache.
`-http.maxCacheSize` *`size`*	Specify the maximum amount of in-memory storage to allocate for cached document contents. This setting applies to all virtual users in the process, even though each virtual user keeps its own cached documents. After the in-memory cache is exhausted, document contents will be cached to a temporary folder on disk in <*installDir*>\agent\cache. There is no upper bound on how much disk storage may be used to store cached documents. The disk cache is cleared every time the agent process starts. The default value for OpenScript is 16MB. The default value for Oracle Load Testing is 128MB.
`-http.useCookies true|false`	Specify whether or not to allow the browser to remember and use session or persistent cookies during playback. The default value is `true`, use cookies.
`-http.downloadLocalFiles true|false`	Specify whether or not the agent should download files that exist in the local file system. When local files are not downloaded, the contents are not displayed in the Oracle Load Testing Virtual User Display, and it is not possible to solve variables against the contents, for example. The default value is `false`, do not download local files.
`-http.preserveCookies true|false`	Set to `true` if cookies should be preserved between iterations. Set to `false` if cookies will be cleared between iterations. The default value is `false`, do not preserve cookies between iterations.

Setting	Description
`-http.preserveConnections true|false`	Set to `true` if the browser should attempt to reuse any open browser connections if possible between iterations. Each virtual user maintains its own set of connections that it never shares with other virtual users. The default value is `true`, preserve connections between iterations.
`-http.maxContentSize` *n*	Specify the maximum number of kilobytes (KB) to download from a server for a given request. Set to `-1` to mean unlimited download size. The default value is `-1`, unlimited.
`-http.socketTimeout` *n*	Specify the socket timeout in *n* seconds. The default value is `120` seconds.

A.2.3.6 Download Manager

The following table lists the Download Manager command line settings.

Setting	Description
`-http.useDownloadManager true|false`	Specify if the Download manager is enabled during playback. The default is `false`.
`-http.downloadManager.css true|false`	Specify if css resources in <Link> tags are downloaded during playback. The default is `false`.
`-http.downloadManager.Image true|false`	Specify if image resources in <Img> tags, in the "background" attribute of a tag, or in <style> tags with "background:url" patterns are downloaded during playback. The default is `false`.
`-http.downloadManager.embeded object true|false`	Specify if object resources in <Embed> tags or in <Object> tags are downloaded during playback. The default is `false`.
`-http.downloadManager.script true|false`	Specify if script resources in <Script> tags are downloaded during playback. The default is `false`.
`-http.downloadManager.applet true|false`	Specify if applet resources in <Applet> tags are downloaded during playback. The default is `false`.
`-http.ignoredUrlRegexps` *string*	Specify the Regular Expression(s) string to use to ignore specific resources. For example, the expression `Login_Banner(.+?)` would not download resources such as Login_Banner1.gif and Login_Banner2.gif. Multiple Regular Expressions can be separated using a comma (,).

A.2.4 Functional Test Settings

The following table lists the functional test command line settings.

Setting	Description
`-ft.smartMatch` *`format`*	Specifies which format to use to match attributes in an object path. The match format can be a wildcard-formatted or a regular-expression-formatted expression. *`format`* is one of the following settings: `wildcard` - (default) Attributes in the given path may contain wildcards for unknown characters. For example, title="Welcome, user *". An asterisk "*" matches any number of characters. A question mark "?" matches any single character. `wildcardThenRegex` - Attributes in the given path may contain a wildcard-formatted expression, or a regular-expression-formatted expression. During playback, an attempt is first made to find the object assuming a wildcard format, then an attempt is made to find the object assuming a regular-expression format. `regex` - Attributes in the given path may contain a regular expression.
`-ft.smartMatch true|false`	When true, the OpenScript Smart Match object identification ranking feature is enabled. The default value is `true`, enabled.

A.2.5 Oracle EBS/Forms Functional Test Settings

The following table lists the Oracle EBS/Forms functional test command line settings.

Setting	Description
`-formsft.startup_timeout` *`n`*	Specify *`n`* seconds to wait for the Forms applet to start up before failing the script due to a timeout. The default value is 30 seconds.
`-formsft.action_timeout` *`n`*	Specify *`n`* seconds to wait when trying to play an action before timing out because the object required could not be found. The default value is 30 seconds.
`-formsft.capture_screenshot true|false`	Set to `true` to capture screenshots of the Forms applet during playback. The default value is `true`.
`-formsft.SUPPRESS_JPSECURITY_DIALOG true|false`	Set to `true` to supress "Java Security Windows" and "Warning - Security" pop up windows during either script recording or play back when the client machine has Java Runtime Environement (JRE) that is 1.6.0_24 or newer. If the client machine has JRE that is 1.6.0_23 or older, no matter if this setting is enabled or disabled, "Java Security Windows" and "Warning - Security" pop up windows will not show during either script recording or play back. The default value is `true`.

A.2.6 Oracle EBS/Forms Load Test Settings

The following table lists the Oracle EBS/Forms load test command line settings.

Setting	Description
`-formslt.capture_message_details true|false`	Set to `true` to capture message details during playback. When true, OpenScript captures and stores Forms message requests, responses, and information about all loaded Forms components during playback. This information is useful to have when debugging the script. OpenScript displays captured details in the "Messages" and "Object Details" tabs of the Details view. Oracle Load Testing displays this information in the Virtual User Display based on the "Virtual User Display" settings. Capturing message details is a memory-intensive operation. During heavy load testing, it is recommended to clear this setting to reduce the amount of heap space required by the agent.
`-formslt.enable_message_log true|false`	Set to `true` to show forms message log details in the Console tab. When `false`, the message log details are not shown in the console. The default is `false`.
`-formslt.heartBeatInterval=`*`interval in seconds`*	Specifies how often to notify the forms server that the forms client is still alive when there is no user activitiy in the forms client. The default is `0`.

A.2.7 Siebel Load Test Settings

The following table lists the Siebel load test command line settings.

Setting	Description
`-siebel.correlationMode java`	Enables the JAVA-based correlation library. By default, Siebel load scripts play back using a DLL-based correlation library if running on Windows platforms or a .SO-based correlation library if running on Linux. The OpenScript Siebel load testing module also includes a platform independent, JAVA-based correlation library that can be enabled using an additional playback setting.

A.2.8 Shared Data Settings

The following table lists the shared data command line settings.

Setting	Description
`-sharedData.actionTimeout` *`timeout`*	Specify the maximum time in seconds to wait for the requested data to return.
`-sharedData.Address` *`address`*	Specify the address of the Oracle Load Testing for Web Application server to use for the shared data service. For example: t3//:localhost:8088 or t3://machinename.com:8088.
`-sharedData.Password` *`password`*	Specify the password to use for authentication. Note that this is a clear text password that is not secure.

Setting	Description

`-sharedData.encryptedPassword` *`encryptedPassword`*	Specify the encrypted password to use for authentication. To generate an *`encryptedPassword`* value for use in this command:
	1. In OpenScript, set the password encryption to **Encrypt script data** in the General Encryption Preferences.
	2. Select **Add** from the **Script** menu, expand the Shared Data folder and select **Set Connection Parameters**.
	3. Specify the connection parameters then copy the encrypted password from the Java Code view to your command line settings. For example, an encrypted password may look similar to the following example:
	`gLM7NB+n7Yba0FlspMDX8A`
	This setting must also be used in conjunction with the `-ENCRYPTION_PROPERTIES_FILE` command-line setting.
	Example usage:
	`--sharedData.encryptedPassword gLM7NB+n7Yba0FlspMDX8A -ENCRYPTION_PROPERTIES_FILE C:/Documents and Settings/`*`username`*`/osworkspace/.metadata/.plugins/oracle.oats.scripting.utilities/encryption.properties`

`-sharedData.obfuscatedPassword` *`obfuscatedPassword`*	Specify the obfuscated password to use for authentication. To generate an *`obfuscatedPassword`* value for use in this command:
	1. In OpenScript, set the password encryption to **Obfuscate script data** in the General Encryption Preferences.
	2. Select **Add** from the **Script** menu, expand the Shared Data folder and select **Set Connection Parameters**.
	3. Specify the connection parameters then copy the obfuscated password from the Java Code view to your command line. For example, an obfuscated password may look similar to the following example:
	`RG5CE6b1u8cG9VzbbnW6cQ==`
	Example usage:
	`-sharedData.obfuscatedPassword RG5CE6b1u8cG9VzbbnW6cQ==`

`-sharedData.UserName` *`name`*	Specify the user name to use for authentication. The default name is `oats` unless changed in the Oracle Application Testing Suite configuration.

A.2.9 Web Functional Test Settings

The following table lists the Web functional test command line settings.

Setting	Description
`-web.auto_dismiss_alert true|false`	Specify whether or not JavaScript alert dialog boxes are automatically dismissed if they appear during playback. The default value is `false`.
`-web.browser_id n`	Specify the browser index value *n*. The default value is `0`.
`-web.capture_browser_only true|false`	Specify whether or not to capture only the contents of the browser window as a screenshot. The default value is `true`.
`-web.capture_entire_screen true|false`	Specify whether or not the entire screen is captured as a screenshot. The default value is `false`.
`-web.capture_entire_screen_ on_fail true|false`	Specify whether or not a screen capture of the entire screen is saved to the results if a failure occurs during playback. The default value is `true`.
`-web.capture_frame true|false`	Specify whether or not to capture the HTML frames on the page during playback. The default value is `true`.
`-web.capture_html true|false`	Specify whether or not to capture the browser HTML during playback. The default value is `true`.
`-web.capture_screenshot true|false`	Specify whether or not to capture screenshots during playback. The default value is `true`.
`-web.capture_screenshot_ interval n`	Specify the number of milliseconds to wait after a new page is detected and the screenshot is captured. The default value is `500`.
`-web.capture_url true|false`	Specify whether or not to capture the page URL during playback. The default value is `true`.
`-web.clear_cache true|false`	Specify whether or not pages are cleared from cache between playback iterations. The default value is `false`.
`-web.clear_cache_before true|false`	Specify whether or not pages are cleared from cache before playback. The default value is `false`.
`-web.clear_persistent_cookies true|false`	Specify whether or not to clear browser persistent cookies between iterations. The default value is `false`.
`-web.clear_persistent_ cookies_before true|false`	Specify whether or not to clear browser persistent cookies before starting playback. The default value is `false`.
`-web.clear_persistent_ cookies_before true|false`	Specify whether or not to clear browser persistent cookies before starting playback. The default value is `false`.
`-web.clear_session_cookies true|false`	Specify whether or not to clear browser session cookies between iterations. The default value is `false`.
`-web.clear_session_cookies_ before true|false`	Specify whether or not to clear browser session cookies before starting playback. The default value is `false`.
`-web.clear_cache true|false`	Specify whether or not to clear the browser cache between iterations. The default value is `false`.
`-web.date_format date_format_ string`	Specify a date format string to override the default date format string used when performing date validation tests.

Setting	Description
`-web.event_time_out n`	Specify the Object Timeout *n* in seconds. If an object cannot be found in *n* seconds, the action will fail. The default value is `60`.
`-web.launch_new_browser true|false`	Specify whether or not to launch a new browser when playing back a different script. The default value is `true`.
`-web.log_jsevent true|false`	Specify whether or not script playback creates a log of the Javascript events (such as onmouseover, onmousedown, click, etc.) fired on HTML elements. The default value is `false`.
`-web.webdom_proxy_port n`	Specify the port used for communication between the web browser and the agent. The default value is `7666`.

A.2.10 Error Recovery Settings

All Error Recovery settings are specified using the form:

`-errorRecoverySettingId action`

`action` is one of the following constants (case-sensitive):

- `Ignore` - proceed with the script as if the error did not occur. The error will still be logged to the console/log file.
- `Warn` - report the error, but continue running the script.
- `Fail` - report the error and fail the current iteration of the script.
- `ReportErrorAndContinue`- Report the error to the results log and continue script execution.
- `Pause` - Pause playback and wait for user's decision to continue or abort script execution.

A.2.10.1 General

The following table lists the general error recovery settings.

Setting	Description
`err.basic.BINARY_DECODING_EXCEPTION`	Binary Decode Failed
`err.basic.CHILD_SCRIPT_EXCEPTION`	Child Script Failed
`err.basic.CREATE_VARIABLE_ERROR`	Create Variable Failed
`err.basic.ENCRYPTION_SERVICE_NOT_ INITIALIZED`	Encryption Service Not Initialized
`err.basic.FILE_NOT_FOUND`	File Not Found
`err.basic.FUNCTION_EXCEPTION`	Function Failed
`err.basic.GENERAL_SCRIPT_EXCEPTION`	Unexpected Script Error
`err.basic.SEGMENT_PARSER_ERROR`	Segment Parser Failed
`err.basic.VARIABLE_NOT_FOUND`	Variable Not Found.
`err.basic.ENCRYPTION_SERVICE_NOT_ INITIALIZED`	Encryption Service Not Initialized

A.2.10.2 Flex Load Testing (AMF)

The following table lists the Flex load testing (AMF) error recovery settings.

Setting	Description
err.amfLT.OPERATION_INVOCATION_ERROR	An operation on an object failed
err.amfLT.PLAYBACK_ERROR	Playback failed

A.2.10.3 Functional Testing

The following table lists the functional testing error recovery settings.

Setting	Description
err.functionalTest.FT_MATCH_ERROR	Text Matching Test failed
err.functionalTest.OBJECT_TEST_ERROR	Object Test failed
err.functionalTest.TABLE_TEST_ERROR	Table Test failed

A.2.10.4 HTTP

The following table lists the HTTP error recovery settings.

Setting	Description
err.http.HTML_PARSING_ERROR	HTML Parsing error
err.http.INTERNET_INVALID_URL	Invalid URL
err.http.INVALID_HTTP_RESPONSE_CODE	Invalid HTTP Response Code
err.http.KEYSTORE_LOAD_ERROR	Client Certificate Keystore error
err.http.MATCH_ERROR	Text Matching Test failed
err.http.NODE_NOT_FOUND_EXCEPTION	Element node not found with xpath exception
err.http.RESPONSE_TIME_ERROR	Response Time error
err.http.SOLVE_ERROR	Solve Variable failed
err.http.ZERO_LENGTH_DOWNLOAD	Zero Length Downloads

A.2.10.5 Oracle EBS/Forms Functional Testing

The following table lists the Oracle EBS/Forms Functional Testing error recovery settings.

Setting	Description
err.formsFT.FORMS_FT_ERROR	Oracle Forms Error
err.formsFT.STATUSBAR_TEST_ERROR	Status Bar Test Error

A.2.10.6 Oracle EBS/Forms Load Testing

The following table lists the Oracle EBS/Forms Load Testing error recovery settings.

Setting	Description
err.formsLT.CONNECT_ERROR	Forms Connect Error

Setting	Description
err.formsLT.IO_ERROR	Forms Input/Output Communication Error
err.formsLT.MATCH_ERROR	Forms Content Match Failed
err.formsLT.PLAYBACK_ERROR	Forms Playback Error
err.formsLT.COMPONENT_NOT_FOUND	Forms Component Not Found

A.2.10.7 Web Functional Testing

The following table lists the Web Functional Testing error recovery settings.

Setting	Description
err.webdom.FAIL_TO_PLAYBACK	Playback Failed
err.webdom.HTML_TEST_ERROR	HTML Test Failed
err.webdom.OBJECT_NOT_FOUND_ERROR	Object Not Found
err.webdom.RESPONSE_TIME_ERROR	Response Time Error
err.webdom.TITLE_TEST_ERROR	Title Test Failed
err.webdom.WAIT_FOR_PAGE_TIMEOUT_ERROR	Wait for Page Timeout
err.webdom.WEBDOM_SOLVE_ERROR	Solve Variable Failed

A.2.10.8 Utilities

The following table lists the Utilities error recovery settings.

Setting	Description
err.utilities.SQL_ERROR	SQL Execute Error
err.utilities.XML_PARSING_ERROR	XML Parsing Error
err.utilities.VALIDATION_ROWCOUNT_ERROR	SQL Validation Row Count Error
err.utilities.CSV_LOADING_ERROR	CSV Loading Error

B

Proxy Command Line Reference

This appendix lists the parameters for running OpenScript proxy from the command line. This is an HTTP/HTTPS proxy that may be used for inspecting HTTP traffic originating from clients such as web browsers and Java applets. It is not intended to be used for heavy traffic loads from more than a few browsers. After running the proxy, configure your clients proxy settings to redirect traffic to this proxy which, by default, is running on localhost:7777.

To run the proxy from the command line, type:

`[OpenScript Install Dir]/proxy.bat [options]`

`[options]` may consist of any number of proxy command line settings. All Arguments are optional and are case-insensitive.

B.1 Specifying Command Line Settings

This section describes how to use the proxy command line settings.

- Zero or more proxy command line settings may be specified using the following format:

  `-settingId settingValue`

  Example 1: Run the proxy listening on port 7777 and log a CSV file of all trafic it captures.

  ```
  proxy.bat -logCSV -logDir C:\Temp\ProxyLogOutput
  ```

  Example 2: Log all HTTP/S traffic except JavaScript and CSS files.

  ```
  proxy.bat -logCSV -logDir "C:\Temp\ProxyLogOutput" -excludeRegexp
  (.+?)js;(.+?)css
  ```

  Example 3: Run the proxy and route all requests to another proxy except for requests to the hosts called "test-server.com" and "production-server.com".

  ```
  proxy.bat -chainproxy corporateProxy.us.corp.com:7777 -chainProxyNonProxyHosts
  test-server.com;production-server.com
  ```

- If many settings are required, use the `-propertiesPath` setting to specify a file containing a larger number of properties. See the `-propertiesPath` setting in the General Settings on page B-2 for additional information.

B.2 Supported Proxy Command Line Settings

This section lists the command line settings that can be used with the proxy.

B.2.1 General Settings

The following table lists the General proxy command line settings.

Setting	Description
`-acceptEncodingHeader [gzip, deflate]`	Overrides the outgoing client HTTP request header value for "Accept Encoding". If no value is specified, then the "Accept Encoding" header is stripped from the outgoing request. Default value is "gzip, deflate". Example usage: `-acceptEncodingHeader gzip`
`-bindIp [ip]`	Specify a specific local IP address to bind. Use this when connecting through a VPN and the proxy auto-detects the wrong local address to bind. Example usage: `-bindIp 127.0.0.1`
`-debug`	Turn on debug log messages. Example usage: `-debug`
`-handleEncodedResponse true|false`	Instructs the proxy to attempt to decode/uncompress response contents in gzip or deflate format. The proxy de-compresses contents before logging them or returning them to the client browser. The default is `true`. Example usage: `-handleEncodedResponse false`
`-httpPort [Port Number]`	Specify the port on which the proxy will listen for traffic. The default proxy listening port if none specified is 7777. Example usage: `-httpPort 7898`
`-propertiesPath path`	Specify the file path to a properties file listing all of the applicable proxy start-up Command Line Arguments in name=value pair format. If used it is presumed all arguments will be inside the file rather than directly on the command line. Example usage: `-propertiesPath C:/proxyStartup.properties`
`-replace "serverUrl=localFilePath"`	When *serverUrl* is downloaded, replace its contents with the contents from the local File located at the *localFilePath*. This is used to inject a different response content into the browser than what the server would normally return. Example usage: `-replace "test.server.com\mocApp.htm = C:\Temp\Replacement\mocApp.htm,http://production.com = C:\Temp\Replacement\testFile.txt"`
`-SSLVersion [TLS | SSL]`	Specify the SSL version to be used by the proxy while running. The default is TLS. Example usage: `-sslversion SSL`

Setting	Description
`-timeout[`*`milliseconds`*`]`	Specify how long the proxy should wait on a connection before timing out waiting for a response. The default is 120000ms. Example usage: `-timeout 200000`

B.2.2 Chain Proxy Settings

The following table lists the Chain Proxy command line settings.

Setting	Description
`-chainproxy "`*`hostname`*`,`*`portNumber`*`"`	Specify a second proxy host name or IP address and port through which to tunnel captured traffic. Example usage: `-chainproxy "corporateproxy,80"`
`-chainProxyAuthenticate`	Flag argument, required if the proxy specified in the argument `-chainProxy` requires authentication. If specified, you will be prompted for username and password information directly before the proxy initiates. Example usage: `-chainproxy "authenticatedHost,7721" -chainproxyAuthenticate`
`-chainProxyNonProxyHosts [`*`hostname1`*`;`*`hostname2`*`]`	Specify one or more web server host names or IP addresses, delimited by a semi-colon (;), for which traffic should not be directed through to the proxy specified by `-chainproxy`. Requests and responses will bypass the chained proxy and go directly to the host. Example usage: `-chainProxyNonProxyHosts "test-server.com;production-server.com"`

B.2.3 Logging Settings

The following tables list the proxy logging command line settings.

Setting	Description
`-logDir [`*`filepath`*`]`	Specify a local directory in which to log traffic. Used in conjunction with other logging settings. Example usage: `-logDir "C:\temp\log1" -logCSV`
`-logCSV`	Log the traffic to a CSV file in the directory specified by the "-logDir" argument. Example usage: `-logDir "C:\temp\log1" -logCSV`

Setting	Description
`-logXML`	Log the traffic to an XML file in the directory specified by the "-logDir" argument. Example usage: `-logDir "C:\temp\log1" -logXML`
`-logHeadersOnly`	When logging content using the other logging arguments, only log headers. Response headers are not logged. Example usage: `-logDir "C:\temp\log1" -logCSV -logHeadersOnly`
`-logImages true|false`	Specify whether or not to log image file responses to the file specified by the `-logDir` argument. Used in conjunction with other `-log` arguments. Disabled by default, no images are logged. Example usage: `-logDir "C:\temp\log1" -logCSV -logImages true`
`-preExclude [`*`URL SubString`*`; `*`URL Substring`*`]`	Exclude URLs that contain a value matching any substring provided in the semi-colon (;) delimited list. Example usage: `-preexclude testserver;oracle.com`
`-exclude [`*`contenttype1`*`;`*`contenttype2`*`]`	Exclude HTTP responses with the specified Content-type headers when logging traffic. Example usage: `-exclude text/javascript;text/html`
`-excludeRegexp [`*`RegExp1`*`;`*`RegExp2`*`]`	Exclude HTTP requests whose URLs match the given regular expressions when logging traffic. Example usage: `-excludeRegexp (.+?)js;(.+?)css`

B.2.4 Security Settings

The following table lists the proxy security command line settings.

Setting	Description
`-allowedClients [`*`IP1`*`;`*`IP2`*`;`*`IP3`*`]`	Specify if the proxy will accept requests from clients/browsers running on different machines. Default value is local addresses only "127.0.0.1;localhost;0:0:0:0:0:0:0:1"; the proxy only accepts requests from clients browsers running on the local machine.
`-clientCertificate [`*`file`*`]`	Client certificate in pem format.

C

Command Line Tools Reference

This appendix lists the parameters for running the OpenScript Command-Line Tools Interface (CTLI). The Command-Line Tools Interface is an umbrella interface for running OpenScript command-line tools. The command-line tools include the OpenScript Command-Line Compiler and the OpenScript Command-Line Asset Updater.

To run the Command-Line Tools Interface, type:

`[OpenScript Install Dir]/osclti.bat [options]`

`[options]` may consist of options for the OpenScript Command-Line Compiler or the OpenScript Command-Line Asset Updater, or help information.

C.1 Using the Command-Line Tools Interface

This section describes how to use the command line tools settings.

- Prints out General help:

  ```
  osclti.bat help
  ```

- Runs OpenScript Command-Line Compiler with supplied options. If options are incorrect, it prints the Compiler help and exits:

  ```
  osclti.bat compile <options>
  ```

- Prints out OpenScript Command-Line Compiler help:

  ```
  osclti.bat compile -help
  ```

- Runs OpenScript Command-Line Asset Updater with supplied options. If options are incorrect, it prints Update Assets help and exits:

  ```
  osclti.bat updateAssets <options>
  ```

- Prints out OpenScript Command-Line Asset Updater help:

  ```
  osclti.bat updateAssets -help
  ```

- Prints out General help:

  ```
  osclti.bat
  ```

- Prints out General help:

  ```
  osclti.bat <any unknown argument>
  ```

C.2 Supported Command Line Tools Interface Options

This section lists the command line settings that can be used with the Command-Line Tools Interface.

C.2.1 Command-Line Compiler Options

The following table lists the OpenScript Command-Line Compiler options.

Setting	Description
`-script [path_to_script_folder]`	Specify the path to a single script's folder for single script compilation. One and only one option from `-rootFolder` or `-script` options must be specified. Use `-rootFolder` option for batch build, use `-script` option for the single script build. Example usage: `osclti compile -script "C:\OracleATS\OFT\myScript"`
`-rootFolder [path_to_folder]`	Specify a folder containing multiple scripts for batch compilation. When `-rootFolder` is specified, the tool builds all scripts in the subtree of folders underneath it, recursively. One and only one option from `-rootFolder` or `-script` options must be specified. Use `-rootFolder` option for batch build, use `-script` option for the single script build. Example usage: `osclti compile -rootFolder "C:\OracleATS\OFT\"`
`-logFile [path_to_log]`	Optional. Specify where the compile log output file should go. User must specify a writable location. If `-logFile` is not specified, then `build.log` will be created in the user's folder (i.e. C:\Documents and Settings\<username>) or in the OpenScript directory if the user's folder is not writable. Example usage: `osclti compile -rootFolder "C:\OracleATS\OFT\" -logfile "C:\logs"`
`-verbose`	Optional. Specify whether the compiler should use verbose output logging. If `-verbose` option is specified, then the compiler prints accessed/processed units into the output log file. Example usage: `osclti compile -rootFolder "C:\OracleATS\OFT\" -logfile "C:\logs" - verbose`
`-stopBatch`	Optional. Specify if the batch compiler should stop on any script failure, or continue compiling other scripts even if a failure happens. The default behavior is not to stop the batch compilation on failure. Example usage: `osclti compile -rootFolder "C:\OracleATS\OFT\" -stopBatch`

Setting	Description
`@optionsFile`	Optional. Specify a file containing one or more command-line options. Options can be specified on command-line or/and in the options file specified by the `@optionsFile` option. The options file can specify any option in the same format as they can be specified on the command-line. Additionally, an options file can use a new line as an option delimiter. If the same option is specified on the command-line and in the options file, then the last argument in order of appearance has a precedence. Example usage: `osclti compile @C:\OracleATS\OFT\options.txt`

C.2.2 Command-Line Asset Updater Options

The following table lists the OpenScript Command-Line Asset Updater options.

Setting	Description
`-add`	Specify to add a new asset. One and only one option from `-add` or `-update` options must be specified. Example usage: `osclti updateAssets -add -rootFolder "C:/OracleATS/OFT" -type script -asset alias=globalLib:Repo/GlobalLib;alias=Login:/Repo/Login`
`-change`	Specify to change an existing asset. One and only one option from `-add` or `-change` options must be specified. Example usage: `osclti updateAssets -change -rootFolder "C:/OracleATS/OFT" -type databank -asset alias=user:Repo/databanks/user.csv`
`-rootFolder [path_to_folder]`	Specify a folder containing multiple scripts to update. When `-rootFolder` is specified, the tool builds all scripts in the subtree of folders underneath it, recursively. Example usage: `osclti updateAssets -change -rootFolder "C:/OracleATS/OFT" -type databank -asset alias=user:Repo/databanks/user.csv`

Setting	Description
`-type [AssetType]`	Specify the type of asset being added or updated. AssetType should be one of the following: ■ jar ■ databank ■ objectLib ■ script The asset updater automatically detects if this asset is a dedicated function library and, if it is, then it makes appropriate changes in scripts to change. Example usage: `osclti updateAssets -change -rootFolder "C:/OracleATS/OFT" -type databank -asset alias=user:Repo/databanks/user.csv`
`-asset [AssetList]`	Specify the list of one or more asset descriptions to add or change. If changing an asset, then *AssetList* should contain a single asset description. Separate multiple asset descriptions with a semicolon. For example: assetDisc;assetDisc;... ;assetDisc. Each asset description should take the form: alias=chosenAlias:repoName/path/to/assetFolder/relative/toReposiotry/assetFileName This is the only possible form of specifying assets to add. In this case, each script will be assigned the asset of this type in the specified repository location. Assets will be known to each script by the specified alias. If an asset of the same type with the same alias exists for a script, then the location of the asset will be changed according to command-line arguments. When changing assets, each script with an asset of the same type with the same alias will change its location according to the command-line arguments. When changing assets, the asset description can be specified in another form: oldRepoName/old/path/toAssetFolder/OldAsset FileName=repoName/path/to/assetFolder/relative/toReposiotry/assetFileName In this case, if any script has an asset in the old location, then without changing the asset alias, the asset location will be changed according to the command-line arguments. Example usage: `osclti updateAssets -change -rootFolder "C:/OracleATS/OFT" -type databank -asset alias=user:Repo/databanks/user.csv`

Setting	Description
`-logFile [path_to_log]`	Optional. Specify where the compile log output file should go. The logFile should be located in a directory where the user has rights to read, write, change, create and delete this file. If `-logFile` is not specified, then `build.log` will be created in the OpenScript directory. Example usage: `osclti updateAssets -update -rootFolder "C:/OracleATS/OFT" -type databank -asset alias=user:Repo/databanks/user.csv -logfile "C:\logs"`

D

Error Message Reference

This appendix lists the error messages for the OpenScript Workbench Platform and HTTP and Siebel Modules.

D.1 Basic Module Error Messages

This section list the Error messages for the OpenScript Workbench Basic Module.

D.1.1 General Script Exceptions

Error Message	Description
File not found: {0}	The file was not found. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: ERR_FILE_NOT_FOUND
Failed to create variable {0}, path={1}	Failed to create the specified variable. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: ERR_CREATE_VARIABLE_ERRORCODE
An unexpected exception occurred in the script. Script section: {0}.	Represents an unexpected exception in the user's script code. Error Component ID: oracle.oats.scripting.modulesbasic.api Error Code ID: ERR_GENERIC_ERROR_CODE

D.1.2 Binary Decoding Exceptions

Error Message	Description
Failed to decode character at offset {1} in string "{0}"	This exception may be thrown while converting a string into binary using the {@link oracle.oats.scripting.modules.basic.api.utilities.BinaryUtil#hexString2Binary(String)} method. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: ERR_BINARY_DECODE

D.1.3 Script Creation Exceptions

Error Message	Description
Could not find script "{0}" in workspace "{1}" in repository "{2}"	Indicates that the specified script could not be found in the indicated workspace and repository. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: SCRIPT_NOT_FOUND
Script not found: {0}	Indicates that the specified script could not be found in the path. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: SCRIPT_PATH_NOT_FOUND
Exception occurred while reading script {0}.	An exception occurred while reading the specified script. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: EXCEPTION_READING_SCRIPT
Failed to load script class {0}. Class not found.	Indicates that a script class failed to load. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: FAILED_TO_LOAD_SCRIPT_CLASS_NOT_FOUND
Failed to load virtual user class {0}.	Indicates that a virtual user class failed to load. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: FAILED_TO_LOAD_VUSER_CLASS
Failed to create instance of virtual user class {0}.	Indicates that an instance of a virtual user class failed to be created. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: FAILED_TO_CREATE_VUSER_INSTANCE
Failed to load script class {0}. {1}	Indicates that a script class failed to load. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: FAILED_TO_LOAD_SCRIPT_CLASS
Failed to create instance of script "{0}".	Indicates that an instance of script class failed to be created. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: FAILED_TO_CREATE_SCRIPT_INSTANCE

D.1.4 Segment Parser Exceptions

Error Message	Description
Parameter missing: {0}	Indicates that a parameter was not found when parsing {{ }} syntax calling Transforms.transform(String, Variables) Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: PARAMETER_NOT_FOUND
Unknown segment type: {0}	Indicates that a parameter was not recognized when parsing {{ }} syntax calling Transforms.transform(String, Variables) Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: UNKNOWN_SEGMENT_TYPE
Unexpected end of string found while parsing string "{0}"	Indicates that an end of string was not recognized when parsing {{ }} syntax calling Transforms.transform(String, Variables) Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: UNEXPECTED_END_OF_STRING
Unexpected parent segment type {0} found while parsing string "{1}"	Indicates that a parent type was not recognized when parsing {{ }} syntax calling Transforms.transform(String, Variables) Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: UNEXPECTED_PARENT_SEGMENT
Unexpected end of function found near character offset {0} while parsing string "{1}"	Indicates that a function was not recognized when parsing {{ }} syntax calling Transforms.transform(String, Variables) Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: UNEXPECTED_END_OF_ FUNCTION
Function name not registered: {0}.	Indicates that a function was not registered. Use SegmentParser.addCustomTransformFunction(oracle .oats.scripting.main.CustomFunction *functionName*) to register custom transform functions. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: UNREGISTERED_FUNCTION
The following characters cannot be escaped: {0}	Indicates that a invalid characters were found when parsing {{ }} syntax. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: INVALID_CHARACTERS_IN_ SEGMENT

D.1.5 Script Service Exceptions

Error Message	Description
Failed to create script service {0}	Indicates that the specified script service failed. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: FAILED_TO_CREATE_SCRIPT_SERVICE

D.1.6 URL Encoding Exceptions

Error Message	Description
Failed to URL-encode string "{0}"	Indicates that the specified URL encoding failed. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: URL_ENCODING_EXCEPTION

D.1.7 Variable Exceptions

Error Message	Description
Variable not found: {0}	Indicates that a variable could not be found when evaluating a string containing curly brace {{ }} formatted data. This exceptions is typically thrown when {@link oracle.oats.scripting.modules.basic.api.Transforms#transform(String, oracle.oats.scripting.modules.basic.api.Variables)} is invoked using a string that references a non-existent variable. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: ERR_VARIABLE_NOT_FOUND
Variable "{0}" not found for string: {1}	Indicates that a variable could not be found when evaluating a string containing curly brace {{ }} formatted data. This exceptions is typically thrown when {@link oracle.oats.scripting.modules.basic.api.Transforms#transform(String, oracle.oats.scripting.modules.basic.api.Variables)} is invoked using a string that references a non-existent variable. Error Component ID: oracle.oats.scripting.modules.basic.api Error Code ID: VARIABLE_NOT_FOUND_FOR_STRING

D.2 Platform Error Messages

This section lists the error messages for the OpenScript Workbench platform.

D.2.1 Browser Exceptions

Error Message	Description
Exception parsing NTLM response: {0}	Indicates an error parsing the NTLM response authentication protocol. Error Component ID: oracle.oats.lbrowser Error Code ID: BROWSER_NTLM_EXCEPTION
Cache look-up error.	Could not find document with host {0}, file {1} in the local browser cache. Indicates an error looking for a document in the browser cache. Error Component ID: oracle.oats.lbrowser Error Code ID: BROWSER_CACHE_LOOKUP_ERROR
Too many redirects.	Browser will not redirect more than {0} times. Comparable WinInet error code: Error 12156: Redirect Failed. The redirection failed because either the scheme changed (for example HTTP to FTP) or all attempts to redirect failed (default is five attempts). Error Component ID: oracle.oats.lbrowser Error Code ID: BROWSER_TOO_MANY_REDIRECTS

D.2.2 SSL Exceptions

Error Message	Description
Error creating new SSL session.	Indicates an error creating a Secure Socket Layer session. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_CREATING_SSL_SESSION
Failed to prepare SSL socket factory.	Indicates an error preparing a Secure Socket Layer socket factory. Error Component ID: oracle.oats.lbrowser Error Code ID: FAILED_TO_PREPARE_SOCKET_FACTORY

D.2.3 TCP Exceptions

Error Message	Description
The server name {0} could not be resolved.	Comparable WinInet error code: Error 12007: Name Not resolved. The server name could not be resolved. This happens when Java throws an UnknownHostException. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_INTERNET_NAME_NOT_RESOLVED

Error Message	Description
The attempt to connect to the server {0} on port {1} failed.	Comparable WinInet error code: Error 12029: Cannot Connect. The attempt to connect to the server failed. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_INTERNET_CANNOT_ CONNECT
The request to the proxy was invalid.	Comparable to WinInet error code: Error 12033: Invalid Proxy Request. The request to the proxy was invalid. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_INTERNET_INVALID_ PROXY_REQUEST
Invalid response received from proxy server: {0}.	Comparable to WinInet error code: Error 12033: Invalid Proxy Request. The request to the proxy was invalid. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_INTERNET_INVALID_ PROXY_RESPONSE
The connection with the server has been terminated.	Comparable to WinInet error code: Error 12030: Connection aborted. The connection with the server has been terminated. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_INTERNET_ CONNECTION_ABORTED
Timeout occurred while sending request to server.	Comparable to WinInet error code: Error 12002: Timeout. The request has timed out. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_INTERNET_REQUEST_ TIMEOUT
Timeout occurred while waiting for server response.	Comparable to WinInet error code: Error 12002: Timeout. The request has timed out. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_INTERNET_RESPONSE_ TIMEOUT
SSL exception occurred while connecting to {0} on port {1}. {2}	Indicates a Secure Socket Layer exception while connecting to the indicated server and port. Error Component ID: oracle.oats.lbrowser Error Code ID: SSL_CONNECT_EXCEPTION
Failed to bind socket to local port {0}	Indicates a failure to bind to the socket on the indicated local port. Error Component ID: oracle.oats.lbrowser Error Code ID: FAILED_TO_BIND

D.2.4 HTTP Exceptions

Error Message	Description
The URL is invalid.	Comparable to WinInet error code: Error 12005: Invalid URL. The Uniform Resource Locator (URL) is invalid. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_INTERNET_INVALID_URL
Failed to write HTTP request header: {0}.	Comparable to WinInet error code: Error 12030: Connection Aborted. The connection with the server has been terminated. Error Component ID: oracle.oats.lbrowser Error Code ID: WRITE_REQUEST_EXCEPTION_HEADER
Failed to write request HTTP postdata: {1}.	Comparable to WinInet error code: Error 12030: Connection Aborted. The connection with the server has been terminated. Error Component ID: oracle.oats.lbrowser Error Code ID: WRITE_REQUEST_EXCEPTION_POSTDATA
Failed to read HTTP response header from server.	Comparable to WinInet error code: Error 12152: Invalid Server Response. The server response could not be parsed. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_READING_HTTP_RESPONSE_HEADER
Failed to read HTTP response contents from server.	Comparable to WinInet error code: Error 12152: Invalid Server Response. The server response could not be parsed. Error Component ID: oracle.oats.lbrowser Error Code ID: ERROR_READING_HTTP_RESPONSE_CONTENTS

D.3 HTTP Error Messages

This section lists the error messages for the HTTP Module.

D.3.1 HTTP Service Exceptions

Error Message	Description
Comparable to WinInet error code: Error 12005: Invalid URL. The Uniform Resource Locator (URL) is invalid.	Indicates a failure when attempting to validate the contents of a Siebel page. Error Component ID: oracle.oats.scripting.modules.http.api Error Code ID: ERROR_INTERNET_INVALID_URL

Error Message	Description
Invalid HTTP response code: {0}	The HTTP Service browser returns this error if an HTTP response is received from a web server containing a response code greater than or equal to 400. Error Component ID: oracle.oats.scripting.modules.http.api Error Code ID: ERROR_INVALID_HTTP_RESPONSE_CODE
Error parsing HTML. {0}	Indicates an unexpected parsing failure in the HTTP service while parsing HTML. Error Component ID: oracle.oats.scripting.modules.http.api Error Code ID: HTML_PARSING_ERROR
Failed to solve variable {0}	Indicates a failure during a match() operation. See oracle.oats.scripting.modules.http.api.ThinIteratingVUserScript#match(String, String, String, oracle.oats.scripting.modules.http.api.ThinIteratingVUserScript.Source, boolean, boolean). The actual error message is defined by the script. The error message shown is an example default error message. Error Component ID: oracle.oats.scripting.modules.http.api Error Code ID: MATCH_ERROR

D.4 Oracle EBS/Forms Functional Test Error Messages

This section lists the error messages for the Oracle EBS/Forms Functional Test Module.

D.4.1 Oracle EBS/Forms Functional Test

Error Message	Description
Install Forms FT Helper Failure: Cause: {0}	Indicates a failure when attempting to install the Oracle EBS/Forms browser helper object. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code ID: ERROR_COMMON_INSTALL_HELPER_FAILURE
Send Playback Command Timeout! command:{0}, timeout: {1}	Indicates a playback command cannot complete within the default timeout setting. Adjust the Forms timeout preferences based upon which command timed out and the timeout value. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code ID: ERROR_PLAYBACK_TIMEOUT_NO_RESPONSE

Error Message	Description
Failed to connect to the Oracle Forms applet after {0} seconds. Verify that the applet launching page opens, and that the Forms Startup Timeout setting is high enough for this site.	Indicates a connection to the Forms server did not complete within the default timeout setting. Adjust the Forms startup timeout preferences to increase the timeout value. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code ID: ERROR_PLAYBACK_TIMEOUT_STARTUP
Forms Object Not Found! XPath: {0}, Type: {1}, Cause: {2}	Indicates the Forms object was not found showing the XPath, object type, and cause. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code ID: ERROR_PLAYBACK_OBJECT_NOT_FOUND
Forms Object Not Found! XPath: {0}, Type: {1}, Cause: No Matches	Indicates the Forms object was not found showing the XPath and object type. The cause is there was no match. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code ID: ERROR_PLAYBACK_OBJECT_NOT_FOUND_NO_MATCHES
Forms Object Not Found! XPath: {0}, Type: {1}, Cause: Illegal Access	Indicates the Forms object was not found showing the XPath and object type. The cause is an illegal access. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code ID: ERROR_PLAYBACK_OBJECT_ILLEGAL_ACCESS
Replay Action: {0} failed. Cause: {1}	Indicates a failure of a playback action. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code: ERROR_PLAYBACK_ACTION_PLAYBACK_FAILURE
VU is NULL	Indicates the Virtual User is a null value. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code: ERROR_PLAYBACK_VU_IS_NULL
Unknown FormsFT Error: {0}	Indicates an unknown error occurred. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code: ERROR_UNKNOWN
Status Bar Test failed.	Indicates a Status Bar test failure. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code: STATUSBAR_TEST_FAILED

Error Message	Description
Status Bar Test {0} failed: actual text "{1}" failed to match the expected one "{2}"	Indicates a Status Bar test failure showing the actual text and the expected text. Error Component ID: oracle.oats.scripting.modules.formsFT.api Error Code: STATUSBAR_TEST_FAILED_WITH_DETAIL_INFO

D.5 Oracle Forms Load Test Error Messages

This section lists the error messages for the Oracle Forms Load Test Module.

D.5.1 Connect Errors

Error Message	Description
Connection Error (Generic): "{0}"	Indicates a failure when attempting connect to the Oracle Forms server. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_GENERIC
Connection Error: Failed to connect to forms server at URL "{0}	Indicates a failure when attempting connect to the Oracle Forms server at the specified URL. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_HTTP_CONNECT_EXCEPTION
Connection Error: Failed to connect to forms server over socket at "{0}:{1}".	Indicates a failure when attempting connect to the Oracle Forms server over the specified socket. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_SOCKET_CONNECT_EXCEPTION
Connection Error: No servlet session id found.	Indicates the session ID for a servlet was not found. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_NOSERVLETSESSIONID
Connection Error: Invalid Session Cookie found.	Indicates the session cookie was invalid. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_NEED_COOKIE
Connection Error: Unable to initialize runtime process.	Indicates the runtime process for the forms server could not be initialized. Error Component ID: oracle.oats.scripting.modules.formsLT.api

Error Message	Description
	Error Code ID: CONN_ERROR_FAILED_TO_ INITIALIZE_PROCESS
Connection Error: Runtime process has unexpectedly terminated.	Indicates an unexpected runtime process termination. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_PROCESS_ STARTUP_TERMINATED
Connection Error: Session migration in progress.	Indicates the JSessionID you are using to post forms navigations to the server is incorrect. EBS/Forms module plays back a script which is a combination of HTTP/Forms navigations. After playing the HTTP navigations successfully a JSessionID captured from the navigation right before the Forms Connect statement is used as the URL to send Forms navigations to the server. If this URL is not correlated correctly you will see the error string. The most common way to fix this issue is to revert script to recorded and re-correlate it. This should automatically correlate the Connect statement. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_SESSION_ MIGRATION_IN_PROGRESS
Connection Error: Cannot connect to OID server.	Indicates a failure connecting to the Oracle Internet Directory server. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_OID_ CONNECTION_ERROR
Connection Error: The user does not have proper credentials for OID.	Indicates the user does not have proper authentication credentials to connect to the Oracle Internet Directory server. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_OID_ AUTHENTICATION_ERROR
Connection Error: SSO user information is invalid.	Indicates the user Single Sign On authentication credentials are not valid. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_INVALID_SSO
Connection Error: Multiple Sessions are disallowed in this transaction.	Indicates that multiple session are not allows for the transaction attempted. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_MULTIPLE_ SESSIONS

Error Message	Description
Connection Error: Could not create the runtime process.	Indicates a failure to create a runtime process on the Forms server. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: CONN_ERROR_RUNTIME_PROCESS_CREATION_ERROR

D.5.2 I/O Errors

Error Message	Description
Unexpected exception occurred while serializing a Forms Message.	Indicates an error occurred while transferring an Oracle Forms message between the client and the server. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: IO_ERROR_MESSAGE_SERIALIZATION_EXCEPTION
Failed to read more than 10 server messages.	Indicates failed to read server message after ten tries. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: IO_ERROR_FAILED_TO_READ_TEN_DATA_STREAMS
Forms input/output exception occurred.	Indicates an unexpected input/output exception occurred. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: IO_ERROR
Forms encryption exception occurred.	Indicates an encryption exception occurred. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: IO_ERROR_ENCRYPTION_EXCEPTION
Communication with server is broken. Check nohup logs for detailed error.	Indicates a communication problem with the server. Run the nohup (no hang up) command an view thee error log. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: IO_ERROR_COMM_BROKEN
Throttling requested with a non-integer value "{0}".	Indicates Bandwidth throttling (limiting quantity of data) was requested using and non-integer value. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: IO_ERROR_THROTTLING_NUMBER_FORMAT_EXCEPTION
Failed to read message from server.	Indicates an exception occurred when trying to read an Oracle Forms message from the server.

Error Message	Description
	Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: IO_ERROR_MESSAGE_READ_ EXCEPTION
Failed to read terminal message from server.	Indicates an exception occurred when trying to read an Oracle Forms terminal message from the server. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: IO_ERROR_TERMINAL_ MESSAGE_READ_EXCEPTION

D.5.3 Match Errors

Error Message	Description
Text Matching Test "{0}" failed. Failed to match "{1}".	Indicates a Text Matching Test failure and shows the test name and the text to match. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: MATCH_ERROR_TEXT_FAILED_ TO_MATCH
StatusBar Text Matching Test "{0}" failed. Failed to match "{1}".	Indicates a Status bar Text Matching Test failure and shows the test name and the text to match. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: MATCH_ERROR_STATUSBAR_ FAILED_TO_MATCH

D.5.4 Component Not Found Errors

Error Message	Description
Component "{0}" does not exist.	Indicates an Oracle Forms component does not exist. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: COMPONENT_DOES_NOT_EXIST
Component with handler ID "{0}" does not exist.	Indicates an Oracle Forms component with the specified handler ID does not exist. Error Component ID: oracle.oats.scripting.modules.formsLT.api Error Code ID: MATCH_ERROR_STATUSBAR_ FAILED_TO_MATCH
Component with handler ID "{0}" does not exist. Last seen alert message: "{1}".	Indicates an Oracle Forms component with the specified handler ID does not exist and shows the last alert message. Error Component ID: oracle.oats.scripting.modules.formsLT.api

Error Message	Description
	Error Code ID: COMPONENT_ID_DOES_NOT_ EXIST_WITH_STATUS

D.5.5 Playback Errors

Error Message	Description
Control "{0}" not initialized.	Indicates that control between client and server could not be initialized. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: CONTROL_INITIALIZE_ERROR
Cannot send forms messages. Not connected to forms server.	Indicates no connection to the Oracle Forms server. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: NOT_CONNECTED_ERROR
Invalid state exception.	Indicates an invalid state. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: INVALID_STATE_EXCEPTION
Invalid state exception. Server sent the client a CREATE message for a component that was already created. Component handler ID is "{0}".	Indicates an invalid state because the server sent the client a CREATE message for a component that was already created and shows the handler ID. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: INVALID_STATE_EXCEPTION_ COMPONENT_ALREADY_EXISTS
Malformed Forms message exception. Forms could not parse the message, "{0}".	Indicates an incorrectly formed Oracle Forms message stream and could not parse the message, "<Message... />" XML. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: MALFORMED_MESSAGE_ EXCEPTION
Unexpected error occurred. {0}.	Indicates any other exception error. Error Component ID:oracle.oats.scripting.modules.formsLT.api Error Code ID: UNEXPECTED_ERROR

D.6 Shared Data Error Messages

This section lists the error messages for the Shared Data Module.

D.6.1 Shared Data Exceptions

Error Message	Description
CreateDataException	Indicates a failure when attempting to create a new queue. Error Component ID: oracle.oats.scripting.modules.sharedData.api Error Code ID: CREATE_QUEUE_EXCEPTION
CreateDataException	Indicates a failure when attempting to create a new hash map. Error Component ID: oracle.oats.scripting.modules.sharedData.api Error Code ID: CREATE_MAP_EXCEPTION
OATSCredentialException	Indicates a failure when the OATS credential is incorrect or not provided. Error Component ID: oracle.oats.scripting.modules.sharedData.api Error Code ID: OATS_CREDENTIAL_EXCEPTION
PutDataException	Indicates a failure when adding items to a queue. Error Component ID: oracle.oats.scripting.modules.sharedData.api Error Code ID: PUT_TO_QUEUE_EXCEPTION
PutDataException	Indicates a failure when adding items to a hash map. Error Component ID: oracle.oats.scripting.modules.sharedData.api Error Code ID: PUT_TO_MAP_EXCEPTION
SharedDataServiceInitException	Indicates a failure when initializing the service. Error Component ID: oracle.oats.scripting.modules.sharedData.api Error Code ID: SHARED_DATA_INITIALIZE_EXCEPTION
RequestDataException	Indicates a failure when retrieving data/info from a queue. Error Component ID: oracle.oats.scripting.modules.sharedData.api Error Code ID: REQUEST_DATA_FROM_QUEUE_EXCEPTION
RequestDataException	Indicates a failure when retrieving data/info from a hash map. Error Component ID: oracle.oats.scripting.modules.sharedData.api Error Code ID: REQUEST_DATA_FROM_MAP_EXCEPTION

D.7 Siebel Error Messages

This section lists the error messages for the Siebel Module.

D.7.1 Siebel Exceptions

Error Message	Description
Siebel content failure. {0}	Indicates a failure when attempting to validate the contents of a Siebel page. Error Component ID: oracle.oats.scripting.modules.siebel.api Error Code ID: SIEBEL_CONTENT_FAILURE

D.8 Web Error Messages

This section lists the error messages for the Web Functional Test Module.

D.8.1 Web Service Exceptions

Error Message	Description
Invalid object path: {0}.	Indicates the object identification path is invalid, where {0} is the full path to the object. Error Component ID: oracle.oats.scripting.modules.webdom.api Error Code ID: ERR_INVALID_PATH
Object not found: {0}.	Indicates the object identified by {0} was not found. 1. Select **OpenScript Preferences** from the **View** menu. 2. Expand OpenScript and the Playback section. 3. Select Web Functional and increase the **Object Timeout** value. -or- Verify that object path is correct. Some `window.close()` actions may be triggered by a previous action and playback of a previous action may close the window automatically. You may need to remove the `window.close()` manually from the Java Code. Error Component ID: oracle.oats.scripting.modules.webdom.api Error Code ID: ERR_OBJECT_NOT_FOUND

Error Message	Description
Timeout occurred waiting for page to load.	Indicates a timeout occurred waiting for any page or specific page. 1. Select **OpenScript Preferences** from the **View** menu. 2. Expand OpenScript and the Playback section. 3. Select Web Functional and increase the **Object Timeout** value. -or- Change `web.browser("<path>").waitForPage(null)` to a specific duration. For example, `web.browser("<path>").waitForPage(120)` means wait for 120 seconds. Modify the *<path>* to a specific path or set it to use wait for any page. Error Component ID: oracle.oats.scripting.modules.webdom.api Error Code ID: ERR_TIMEOUT_WAITING_FOR_PAGE
Cannot playback the {0} action on this element.	Indicates the action is not supported by the element identified by {0}. Error Component ID: oracle.oats.scripting.modules.webdom.api Error Code ID: INVALID_ACTION_ON_ELEMENT
Failed to playback: {0}.	Indicates a failure to execute an action on the object identified by {0}. Error Component ID: oracle.oats.scripting.modules.webdom.api Error Code ID: FAIL_TO_PLAYBACK
Failed to get a response after sending the message: {0}.	Indicates a failure to get a playback result from the browser after sending the request identified by {0}. Error Component ID: oracle.oats.scripting.modules.webdom.api Error Code ID: FAIL_TO_GET_RESULT
Launch browser timeout after {0} seconds.	Indicates a failure to find a browser within the duration of 'Startup Timeout' setting ({0} seconds) in the browser preferences. 1. Select **OpenScript Preferences** from the **View** menu. 2. Expand OpenScript and the General section. 3. Select Browsers and increase the **Startup Timeout** value. -or- Check if the "Oracle Application Testing Suite Helper Service" is running, or restart it. -or- Close all browsers (IE, FF), run *<installdir>*\OpenScript\UninstallBrowserHelpers.bat and InstallBrowserHelpers.bat.

Error Message	Description
	Error Component ID: oracle.oats.scripting.modules.webdom.api Error Code ID: FAIL_TO_GET_RESULT
Fail to build message: {0}.	Indicates a failure to build a message specified by {0}. The attributes or Identification of a custom DOM element is not correct. Error Component ID: oracle.oats.scripting.modules.webdom.api Error Code ID: ERR_BUILD_MESSAGE

E
Troubleshooting

This chapter provides information and possible solutions to known issues in OpenScript.

E.1 Installation

In some instances, the OpenScript Internet Explorer and/or Firefox browser plug-ins fail to install properly. You can manually install and uninstall Internet Explorer and/or Firefox browser plug-ins using batch files provided with OpenScript to resolve the issue.

The following batch files are located in <*installdir*>\OpenScript

- InstallBrowserHelpers.bat
- UninstallBrowserHelpers.bat

Usage: *file*.bat *installation path* [install | uninstall] [IE | FF | both]

The following example shows the usage and default values:

```
InstallBrowserHelpers.bat C:\OracleATS\OpenScript\ install both
```

> **Note:** On Windows 2000 systems, the InstallBrowserHelpers.bat installation does not detect that the .NET Framework 2.0 is installed and will attempt to reinstall it, which may cause an installation failure for the .NET Framework 2.0. However, the .NET Framework 2.0 is installed during the initial OpenScript installation and should be present on the system. Check Add/Remove Programs in the Control Panel to verify the .NET Framework 2.0 installation.

See the remarks in the batch files for additional information.

E.2 OpenScript Script Execution in Oracle Test Manager

The following additional steps are required in order to run the following types of OpenScript scripts from Oracle Test Manager:

- Siebel Functional
- Oracle Forms Functional
- Web Functional scripts that rely on system input events, such as key press or mouse click

It is necessary to run these scripts using an interactive desktop of a named Windows user account that is always logged in.

1. For Siebel and Oracle Forms, the named user's account must have visited the Siebel or Oracle Forms site at least once to ensure that all necessary ActiveX controls and plug-ins are installed in the named user's browser.
2. On the Oracle Test Manager agent machine that will run the scripts, stop the "Oracle Agent Starter Service" and configure it to start manually.
3. On the Oracle Test Manager agent machine, login as the named Windows user account that will run the scripts. From a command prompt, run:

   ```
   C:\OracleATS\agentmanager\bin\AgentManagerService.exe -c
     C:\OracleATS\agentmanager\bin\AgentManagerService.conf
   ```

   where C:\OracleATS is the OATS installed folder.
4. The named user account must remain logged into the system at all times that scripts will be run.

E.3 Manual Installation of Firefox Extension

If Firefox is upgraded or newly installed after installing OpenScript, it may be necessary to install the OpenScript Firefox extension into it in order to record from it. To install the OpenScript Firefox extension, run the OpenScript Diagnosis Tool, which will automatically detect the missing Firefox extension and install it. To run the OpenScript Diagnosis Tool, select **OpenScript Diagnosis Tool** from the **Help** menu.

E.4 Installation of Security Certificate in Internet Explorer

The Internet Explorer browser reports **There is a problem with this website's security certificate** when recording load test scripts for HTTPS sites. This occurs because the OpenScript proxy is between the browser and the Web Server so the browser does not get the SSL certificate directly from the website. The browser gets the OpenScript proxy's SSL certificate instead.

This issue affects all scripts that record with the Proxy, which are:

- HTTP Load
- Siebel Load
- Oracle EBS/Forms Load
- Oracle Fusion/ADF Load
- Flex Load
- Web/HTTP Load
- Web Services

The Internet Explorer browser will display the warning every time you record unless you explicitly tell the browser to trust the OpenScript certificate.

Caution: Following these steps may have security implications. The machines where the OpenScript product is installed should be strictly used for testing.

To install the OpenScript certificate:

1. Create a load test script project and start recording the website. The Certificate Error: Navigation Blocked screen appears with the message **There is a problem with this website's security certificate**.
2. Click the Continue to this website (not recommended) link. This will navigate to the website with a "Certificate Error" indicated in the a browser address bar.
3. Click the "Certificate Error" indicator box on the right-side of the Internet Explorer address bar.
4. Click "View Certificates" in the Certificate Invalid dialog box.
5. Click "Install Certificate" in the Certificate Information dialog box to open the wizard to install the OpenScript certificate to the Internet Explorer browser.
6. Click **Next** twice at the wizard prompts and then click **Finish**.
7. Click **Yes** in the Security Warning dialog box to install the OpenScript certificate.
8. Click **OK** to close the installation successful message.
9. Select **Internet Options** from the Internet Explorer **Tools** menu.
10. Click the **Advanced** tab and clear the **Warn about certificate address mismatch*** option.
11. Click **OK** to close the Internet Options.

The next time you record scripts, the security certificate warning should no longer appear with the OpenScript certificate accepted in Internet Explorer browser.

F

Third-Party Licenses

This appendix contains licensing information about certain third-party products included with Oracle Application Testing Suite 9.30. Unless otherwise specifically noted, all licenses herein are provided for notice purposes only.

The sections in this appendix describe the following third-party licenses:

- ANTLR 3.2
- Apache 2.0 (Apache BCEL, Apache Jakarta, Apache WSS4J XML, Tomcat, TrueZip, Xalan, XMLBeans)
- Apache POI 3.7
- Cryptix
- dom4j 1.6.1
- Doug Lea License
- GubuSoft Treeview Icons
- Apache 1.1 (IBM XML Parser)
- Intel Utilities
- JaWin
- JDOM
- John McTainsh Utility
- "Generic MIT" (MIT HTML Parser)
- OpenSSH [IGNORE PARTS WHERE CODE ISN'T USED]
- Pradeep Sahu Job Scheduler
- Tanuki Java Service Wrapper
- The Legion Of The Bouncy Castle
- TidyCom
- Tim Taylor Utility
- Vincent Rijmen's AES Encryption
- Xstream
- Zlib

ANTLR 3.2

Copyright (c) 2003-2007, Terence Parr

All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: -Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. -Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. -Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Apache 2.0 (Apache BCEL, Apache Jakarta, Apache WSS4J XML, Tomcat, TrueZip, Xalan, XMLBeans)

Apache License

Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain

to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

Apache POI 3.7

Copyright 2009 The Apache Software Foundation

This product includes software developed by The Apache Software Foundation (http://www.apache.org/).

This product contains the DOM4J library (http://www.dom4j.org).

Copyright 2001-2005 (C) MetaStuff, Ltd. All Rights Reserved.

This product contains parts that were originally based on software from BEA.

Copyright (c) 2000-2003, BEA Systems, <http://www.bea.com/>.

This product contains W3C XML Schema documents. Copyright 2001-2003 (c) World Wide Web Consortium (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University)

This product contains the Piccolo XML Parser for Java

(http://piccolo.sourceforge.net/). Copyright 2002 Yuval Oren.

This product contains the chunks_parse_cmds.tbl file from the vsdump program.

Copyright (C) 2006-2007 Valek Filippov (frob@df.ru)

All recipients must receive a copy of the Apache 2.0 license (see the Appendix below for attachment of the license http://www.apache.org/licenses/LICENSE-2.0.html);

All distributed source code, documentation, and configuration files must retain all copyright, patent, trademark and attribution notices contained in the Apache materials;

If the Apache code retains a "NOTICE" text file, then Oracle must include in distributions of such Apache code the "NOTICE" text file in at least one of the following locations: (i) within a"NOTICE" text file, (ii) within the source code or documentation if distributed with the Apache code, (iii) within a display generated by the distributed code in recognizable third party notice locations; and

Cryptix

"Copyright (c) 1995-2005 The Cryptix Foundation Limited. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the copyright notice, this list of conditions and the following

disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE CRYPTIX FOUNDATION LIMITED AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CRYPTIX FOUNDATION LIMITED OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

dom4j 1.6.1

"Copyright 2001-2005 (C) MetaStuff, Ltd. All Rights Reserved.

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name "DOM4J" must not be used to endorse or promote products derived from this Software without prior written permission of MetaStuff, Ltd. For written permission, please contact dom4j-info@metastuff.com.

4. Products derived from this Software may not be called "DOM4J" nor may "DOM4J" appear in their names without prior written permission of MetaStuff, Ltd. DOM4J is a registered trademark of MetaStuff, Ltd.

5. Due credit should be given to the DOM4J Project -http://www.dom4j.org

THIS SOFTWARE IS PROVIDED BY METASTUFF, LTD. AND CONTRIBUTORS "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL METASTUFF, LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

Doug Lea License

"The Java Software technologies are Copyright © 1994-2000 Sun Microsystems, Inc. All rights reserved. This software is provided"AS IS", without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN MICROSYSTEMS, INC. OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTIAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN MICROSYSTEMS, INC. HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility."

GubuSoft Treeview Icons

"Copyright (C) 2006 Conor O'Mahony (gubusoft@gubusoft.com). All rights reserved. This application includes the TreeView script. You are not authorized to download and/or use the TreeView source code from this application for your own purposes. For your own FREE copy of the TreeView script, please visit the http://www.treeview.net Web site. THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE."

Apache 1.1 (IBM XML Parser)

"This product includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright © 2000-2003 The Apache Software Foundation. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the notice"Copyright © 2000-2003 The Apache Software Foundation. All rights reserved.", this list of conditions and the disclaimer below. 2. Redistributions in binary form must reproduce the notice"Copyright © 2000-2003 The Apache Software Foundation. All rights reserved.", this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:"This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. Neither the component name nor Apache Software Foundation may be used to endorse or promote products derived from the software without specific prior written permission. 5. Products derived from the software may not be called"Apache", nor may"Apache" appear in their name, without prior written permission. THIS SOFTWARE IS PROVIDED"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF ERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

Intel Utilities

"Copyright © 1996 Intel Corporation. All Rights Reserved. Permission is granted to use, copy and distribute this software and its documentation for any purpose and without fee, provided, that the above copyright notice and this statement appear in all copies. Intel makes no representations about the suitability of this software for any purpose. This software is provided "AS IS." Intel specifically disclaims all warranties, express or implied, and all liability, including consequential and other indirect damages, for the use of this software, including liability for infringement of any proprietary rights, and including the warranties of merchantability and fitness for a particular purpose. Intel does not assume any responsibility for any errors which may appear in this software nor any responsibility to update it., this list of conditions and the disclaimer below in the documentation and/or other materials provided with the distribution."

JaWin

"This product includes software developed by the DevelopMentor OpenSource Project (http://www.develop.com/OpenSource). Copyright (c) 2001 DevelopMentor. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowlegement: "This product includes software developed by the DevelopMentor OpenSource Project (http://www.develop.com/OpenSource)." Alternately, this acknowlegement may appear in the software itself if and wherever such third-party acknowlegements normally appear. 4. The name "DevelopMentor" may not be used to name, endorse, or promote products derived from this software without prior written permission. For written permission, please contact opensource@develop.com. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL DEVELOPMENTOR OPENSOURCE OR ITS ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

JDOM

"This product includes software developed by the JDOM Project (http://www.jdom.org/). Copyright (C) 2000-2004 Jason Hunter & Brett McLaughlin. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution. 3. The name "JDOM" must not be used to endorse or promote products derived from this software

without prior written permission. For written permission, please contact <request_AT_jdom_DOT_org>. 4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management <request_AT_jdom_DOT_org>. In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the JDOM Project (http://www.jdom.org/)." Alternatively, the acknowledgment may be graphical using the logos available at http://www.jdom.org/images/logos. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

John McTainsh Utility

"Copyright 1999 © John McTainsh."

"Generic MIT" (MIT HTML Parser)

"Copyright © <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

OpenSSH [IGNORE PARTS WHERE CODE ISN'T USED]

Part 1: Tatu Ylonen

"Copyright (c) 1995 Tatu Ylonen <ylo@cs.hut.fi>, Espoo, Finland. All rights reserved/"

Part 2: CORE SDI

"Cryptographic attack detector for ssh - source code. Copyright (c) 1998 CORE SDI S.A., Buenos Aires, Argentina. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that this copyright notice is retained. THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES ARE DISCLAIMED. IN NO EVENT SHALL CORE SDI S.A. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR

MISUSE OF THIS SOFTWARE. Ariel Futoransky <futo@core-sdi.com>
http://www.core-sdi.com"

Part 3: David Mazieres

"Copyright 1995, 1996 by David Mazieres <dm@lcs.mit.edu>. Modification and redistribution in source and binary forms is permitted provided that due credit is given to the author and the OpenBSD project by leaving this copyright notice intact."

Part 4: Vincent Rijmen (see separate summary below)

Part 5: UC Berkeley (BSD)

"Copyright (c) 1983, 1990, 1992, 1993, 1995 The Regents of the University of California. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THE SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS "AS-IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDINGBUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS AND SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, ARISING IN ANY WAY OUT OF THE USE OF THE SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

Part 6: BSD-Type License

NOTE: Remaining components of the software are provided under a standard 2-term BSD license with the following names as copyright holders:

- Markus Friedl
- Theo de Raadt
- Niels Provos
- Dug Song
- Aaron Campbell
- Damien Miller
- Kevin Steves
- Daniel Kouril
- Wesley Griffin
- Per Allansson
- Nils Nordman
- Simon Wilkinson

"Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source

code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

Pradeep Sahu Job Scheduler

Redistributions of source code or binary form must retain the notice"You are free to use the code, and modify. Provided you don't remove this comments. It will be great if you can provide your feedback."

Tanuki Java Service Wrapper

"Copyright © 1999, 2006 Tanuki Software, Inc. Permission is hereby granted, free of charge, to any person obtaining a copy of the Java Service Wrapper and associated documentation files (the"Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED"AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright © 2001 Silver Egg Technology. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the"Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED"AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

NOTE

Applications which are distributed with the Wrapper must include the license in a file called license-wrapper.txt. The file should be located in a location that is obvious to the user. Furthermore, the Wrapper may not be modified in a way which suppresses the copyright banner displayed on startup.

"Copyright © 1999, 2006 Tanuki Software, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the Java Service Wrapper and associated documentation files (the"Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED"AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

Portions of the Software have been derived from source code developed by Silver Egg Technology under the following license:

BEGIN Silver Egg Technology License ----------

Copyright © 2001 Silver Egg Technology

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the"Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED"AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

The Legion Of The Bouncy Castle

Copyright (c) 2000 - 2009 The Legion Of The Bouncy Castle
(http://www.bouncycastle.org)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit

persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

TidyCom

"Copyright © 1998-2000 World Wide Web Consortium."

Tim Taylor Utility

"Copyright © 2005 Tim Taylor Consulting <http://tool-man.org/> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the"Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED"AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE."

Vincent Rijmen's AES Encryption

"rijndael-alg-fst.c. @version 3.0 (December 2000). Optimised ANSI C code for the Rijndael cipher (now AES). @author Vincent Rijmen vincent.rijmen@esat.kuleuven.ac.be @author Antoon Bosselaers antoon.bosselaers@esat.kuleaven.ac.be @author Paulo Barreto paulo.barreto@terra.com.br This code is hereby places in the public domain. THE SOFTWARE IS PROVIDED BY THE AUTHORS"AS IS", AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMTED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

Xstream

"Copyright (c) 2003-2006, Joe Walnes. Copyright (c) 2006-2007, XStream Committers. All rights reserved. Redistribution and use in source and binary forms, with or

without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of XStream nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE."

Zlib

"Copyright (c) 1995-2005 Jean-loup Gailly and Mark Adler. This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions: 1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required. 2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software. 3. This notice may not be removed or altered from any source distribution. Jean-loup Gailly jloup@gzip.org. Mark Adler madler@alumni.caltech.edu"

Index

A

abort script programmatically, 3-45
Action Message Format, 12-1
Additional Arguments setting, 2-9
ADF
  creating new project, 3-4, 3-5
ADF Functional Test module
  configuring the server, 9-2
  key features, 9-2
  setting record preferences, 2-26, 2-29
  using, 9-1
  verifying compression settings, 9-3
ADF Functional Test scripts
  application programming interface, 9-8
  playing back scripts, 9-6
  playing back scripts with iterations, 9-6
  recording scripts, 9-3, 9-5
ADF Load Test scripts
  playing back scripts, 10-6
Adobe Flex (AMF)
  creating new project, 3-5
Adobe Flex (AMF) Load Test module
  Correlation Library, 12-5
  key features, 12-1
  setting Correlation preferences, 12-5
  using, 12-1
Adobe Flex (AMF) Load Test scripts
  adding Adobe Flex (AMF) actions, 12-4
  modifying, 12-4
  playing back scripts, 12-3
  playing back scripts with iterations, 12-3
  recording scripts, 12-2
  using Adobe Flex (AMF) Load Test API, 12-5
Adobe Flex Functional Test module
  key features, 11-1
  prerequisites, 11-2
  using, 11-1
Adobe Flex Functional Test scripts
  action dialog box, 11-7
  adding Flex actions, 11-6
  modifying, 11-6
  object identification, 11-4
  playing back scripts, 11-4, 11-5
  playing back scripts with iterations, 11-5
  recording scripts, 11-3
  using Flex Functional Test API, 11-10
Advance to Next Record setting, 4-10
alias, 4-4, 4-5
All Records setting, 4-12
AllowAnonUsers setting, 18-3
Apache AXIS parsers, 2-33
assets
  adding to scripts, 3-33
Assets Editor, 1-16
Attachments
  adding to Web Services, 17-6
authentication, 6-21
Automatically (when page is out of date)
    setting, 2-16

B

binary coding exceptions, D-1
Binary Decode Failed setting, 2-11
binary file data
  posting, 6-16
Block Scenario Script
  creating new project, 3-4
Block Scenarios module
  about, 22-1
  adding blocks, 22-3
  adding child blocks, 22-5
  adding child scripts, 22-6
  adding script assets, 22-2
  adding scripts, 22-4
  creating projects, 22-2
  deleting blocks and scripts, 22-7
  editing block and script settings, 22-6
  modifying scenarios, 22-3
  moving blocks and scripts, 22-6
  using, 22-1
boundary, 6-30
Breakpoint View, 1-5, 1-21
breakpoints
  adding, 3-47
browser exceptions, D-5
Browsers preferences, 2-4
By Shuffling setting, 4-11

C

Cache download pages setting, 2-16
callFunction statement
  adding to script, 3-20
  inputting data from files, 3-25
  using list selection, 3-25
  using toList, 3-23
  using toMap, 3-24
Certificates
  configuring in OpenScript, 6-13
  exporting from IE, 6-12
  using, 6-12
certificates
  installing on Internet Explorer browser, E-2
chaining scripts, 3-43
Charset setting, 4-3
Check for newer versions of cached pages setting, 2-16
Child Script Failed setting, 2-11
Clear cache each iteration setting, 2-16
clearing cache, 5-10
clearing cookies, 5-10
clearing Shared Data hash maps, 21-8
clearing Shared Data queues, 21-6
Client Certificate Keystore Error setting, 2-12
Client-Side Digital Certificates
  configuring in OpenScript, 6-13
  exporting from IE, 6-12
  using, 6-12
Close browser after playback setting, 2-20
command line compiler options, C-2
command line settings, A-1, B-1, C-1
  agent general, A-2
  agent settings, A-2
  browser, A-10
  chain proxy, B-3
  compression, A-12
  connections, A-13
  Download Manager, A-15
  error recovery, A-20
  functional test, A-15
  headers, A-12
  HTTP, A-11
  Oracle EBS/Forms functional test, A-16
  other, A-13
  proxy, A-11
  proxy general, B-2
  proxy logging, B-3
  proxy security, B-4
  proxy settings, B-1
  reserved keywords, A-2
  Shared Data Service, A-17
  Siebel load test, A-17
  specifying, A-1, B-1
  Web functional test, A-18
command line tools
  options, C-2
command line tools interface
  using, C-1
command-line asset updater options, C-3
comments
  adding to script results, 3-38
Component Not Found Errors, D-13
Connect errors, D-10
Console View, 1-4, 1-5, 1-17
cookies
  adding to script, 6-25
  removing from script, 6-26
correlation, 6-27
Create Variable Failed setting, 2-11
creating Shared Data hash maps, 21-6
creating Shared Data queues, 21-4
CSV Loading Error setting, 2-13

D

Data Driven Testing
  See also parameterization, 4-1
data input parameterization, 4-2
data input sources, 4-2
data parameterization
  definition, 1-2
data parameterization GUI view, 4-2
Data Table View, 1-4, 1-5, 1-18
Data tables
  accessing from child scripts, 4-22, 4-23
  adding rows and columns, 4-18
  adding worksheets, 4-19
  changing data during playback, 4-17
  deleting rows and columns, 4-18
  deleting worksheets, 4-19
  enabling, 4-14
  entering data manually, 4-15
  exporting spreadsheet files, 4-17, 4-21, 4-22
  exporting worksheets, 4-22
  getting cell values, 4-17
  getting row and column counts, 4-19
  getting worksheet counts, 4-19
  importing spreadsheet files, 4-16, 4-21
  importing worksheets, 4-21
  setting first row policy, 4-14
  setting next and previous rows, 4-20
  using, 4-13
  using edit menu, 4-15
  using worksheet menu, 4-15
databank variables
  substituting, 6-23
databanks
  advance records settings, 4-10
  configuring, 4-3
  creating and editing, 4-6
  getting records, 4-7, 4-10
  loading dynamically, 4-8
  maximum iterations settings, 4-12
  next record settings, 4-11
  out of records settings, 4-11
  range settings, 4-12
  settings, 4-10
  using, 4-2, 4-10
Database Capture files

importing, 3-14
Database Name or Database SID setting, 4-4
Databases
adding SQL Query test, 20-5
calling database procedure statement, 20-7
connecting, 20-4
defining, 20-3
executing SQL statement, 20-4
getting data from CSV files, 20-2
getting data from text files, 20-1
getting data from XML files, 20-3
getting values, 20-3
querying SQL statement, 20-4
debug logging
enabling, 3-51
Debug logging setting, 2-10
Debug View, 1-21
debugging scripts, 3-46
adding breakpoints, 3-47
adding Java exception breakpoint, 3-49
adding views, 3-46
inspecting variables, 3-50
pausing and resuming scripts, 3-49
Declaration View, 1-5, 1-21
delay
adding to script, 3-19
destroying Shared Data hash maps, 21-8
destroying Shared Data queues, 21-6
Details View, 1-4, 1-5, 1-16
Developer Perspective, 1-4
available options, 1-6
Breakpoint View, 1-5, 1-21
Console View, 1-5, 1-17
Data Table View, 1-5
Debug View, 1-5, 1-21
Declaration View, 1-5, 1-21
Details View, 1-5, 1-16
Error Log View, 1-5
Navigator View, 1-5, 1-21
Object Details View, 1-5
Package Explorer View, 1-5, 1-21
Problems View, 1-5, 1-17
Properties View, 1-5, 1-17
Results View, 1-5, 1-17
Script Variables View, 1-5
Script View, 1-5, 1-15
Treeview Breakpoint View, 1-5
Variables View, 1-5, 1-21
DIME, 17-7
Direct Internet Message Encapsulation, 17-7

E

Each Iteration of Script setting, 4-11
Each Occurance setting, 4-10
EBS/Forms Functional Test module
setting playback preferences, 2-17, 2-18
setting record preferences, 2-27
EBS/Forms Load Test module
setting record preferences, 2-28
Element node not found with xpath setting, 2-12
elf-contained zip file, 3-11
EnableAutomation setting, 18-3
enabling Shared Data service, 21-2
encode strings, 6-30, 6-31
encoding
resetting, 6-15
encryption preferences, 2-4
Encryption Service Not Initialized setting, 2-11
End-User Monitoring (EUM), 8-2
Error Log View, 1-4, 1-5
error recovery
adding to script, 3-39
Flex load testing (AMF), A-21
functional testing, A-21
general, A-20
HTTP, A-21
Oracle EBS/Forms functional testing, A-21
Oracle EBS/Forms load testing, A-21
setting preferences, 2-10
utilities, A-22
Web functional testing, A-22
Every visit to the page setting, 2-16
exceptions, 2-10
adding breakpoints, 3-49
execute code, 3-51
export
playback settings, 3-17
scripts, 3-10

F

Failure to create DOM object setting, 2-12
File Not Found setting, 2-11
Finish section
definition, 1-2
Flex (AMF) Load Test, 12-1
Flex (AMF) Load Test module
Correlation Library, 12-5
key features, 12-1
setting Correlation preferences, 12-5
using, 12-1
Flex (AMF) Load Test scripts
adding Adobe Flex (AMF) actions, 12-4
modifying, 12-4
playing back scripts, 12-3
recording scripts, 12-2
using Adobe Flex (AMF) Load Test API, 12-5
folders
managing, 3-3
for statement
adding to script, 3-19
Form Server Connect Failed setting, 2-12
Forms Component Not Found setting, 2-12
Forms Contents Match Failed setting, 2-12
Forms End-User Performance Monitoring, 8-2
Forms Input/Output Communication Error setting, 2-12
Forms Playback Error setting, 2-12
Function Failed setting, 2-11

function libraries, 3-27
about, 3-27
calling functions from, 3-31
converting scripts to, 3-32
creating, 3-30
FAQs, 3-27
function statement
adding to script, 3-20
inputting data from files, 3-25
using List<String>, 3-23
using Map<String, String>, 3-24
using Select List, 3-25

G

General preferences, 2-2

H

Has Error control statement
adding to script, 3-41
Hide browser during playback setting, 2-20
High-Interactivity components, 18-2
Host setting, 4-4
HTML Parsing Error setting, 2-12
HTML Test Failed setting, 2-13
HTML tests, 5-18
HTTP error messages, D-7
HTTP exceptions, D-7
HTTP Get Navigation, 6-29
HTTP module
about, 6-1
key features, 6-1
setting playback preferences, 2-13, 6-13
setting record preferences, 2-22, 6-11
using, 6-1
HTTP scripts
adding a DOM variable, 6-20
adding authentication, 6-21
adding cookies, 6-25
adding Get Navigation, 6-29
adding Multipart Post Navigation, 6-30
adding Post Navigation, 6-29
adding server response tests, 6-23
adding text matching tests, 6-22
adding user agent, 6-26
adding variables, 6-20
adding XML Post Navigation, 6-31
deleting cookie, 6-26
deleting variables from, 6-21
finding variable in, 6-21
modifying, 6-17
playing back, 6-13, 6-14
recording, 6-11, 6-12
using HTTP API, 6-32
viewing playback results, 6-14
HTTP service exceptions, D-7, D-8
Hyperion
creating new project, 3-5
Hyperion Load Test module
Correlation Library, 13-3
key features, 13-1
setting Correlation preferences, 13-3
using, 13-1
Hyperion Load Test scripts
playing back scripts, 13-2
playing back scripts with iterations, 13-2
recording scripts, 13-1

I

import
Database Capture file, 3-14
RUEI User Session log, 3-17
scripts, 3-12
Initialize section
definition, 1-2
inspect variable, 3-50
installation, 1-3
troubleshooting, E-1
Invalid HTTP Response Code setting, 2-12
Invalid URL setting, 2-12
I/O Errors, D-12
iterating scripts, 4-10, 6-14
iterations, 4-9, 6-14
iterations setting, 4-12

J

Java Code Editor, 1-7, 1-15
finish(), 1-15, 1-16
initialize(), 1-15, 1-16
run(), 1-15, 1-16
Java Code Script
creating new project, 3-5
Java Exception Breakpoint, 3-49
JD Edwards EnterpriseOne Functional Test scripts
capturing grid attributes, 14-5
modifying, 14-5
playing back scripts, 14-4
JD Edwards Functional Test module
key features, 14-1
JD Edwards Functional Test scripts
recording scripts, 14-1
JD Edwards Load Test module
Correlation Library, 15-3
key features, 15-1
setting Correlation preferences, 15-3
using, 15-1
JD Edwards Load Test scripts
playing back scripts, 15-2
playing back scripts with iterations, 15-2
recording scripts, 15-1
JDE EnterpriseOne Functional Test module
adding JDE object identifiers, 14-2
editing JDE object identifiers, 14-2
setting preferences, 14-2
using, 14-1
JDE EnterpriseOne Functional Test scripts
adding Fusion/ADF actions, 14-5

application programming interface, 14-6
playing back scripts with iterations, 14-5
recording scripts, 14-4
JRE Plug-in Security Dialogs
suppressing on play back, 2-27
JVM Arguments setting, 2-10

K

Keep the Same Record setting, 4-11
keyboard shortcut
setting preferences, 2-5

L

large data
setting preferences, 2-7
log message
adding to script, 3-19
Loop Over Range setting, 4-11

M

Match Errors, D-13
Maximum In-Memory Cache Size setting, 2-16
Maximum Iterations setting, 4-12
Maximum JVM Heap Size setting, 2-10
Menu options
Edit, 1-6
File, 1-5
Help, 1-11
Navigate, 1-11
Project, 1-12
Run, 1-9
Script, 1-7
Search, 1-7
Tools, 1-10
View, 1-9
Window, 1-12
module error messages, D-1
modules, 12-1
Adobe Flex (AMF) Load Test, 12-1
Adobe Flex Functional Test, 11-1
Block Scenarios, 22-1
HTTP, 6-1
Hyperion Load Test, 13-1
JD Edwards Functional Test, 14-1
JD Edwards Load Test, 15-1
Oracle EBS/Forms Functional Test, 7-1
Oracle EBS/Forms Load Test, 8-1
Oracle Fusion/ADF Functional Test, 9-1
Oracle Fusion/ADF Load Test, 10-1
PeopleSoft Load Test, 16-1
Shared Data, 21-1
Siebel Functional Test, 18-1
Siebel Load Test, 19-1
Utilities, 20-1
Web Functional Test, 5-1
MTOM, 17-7
Multipart Post Navigation, 6-30

N

navigation
adding, 5-9, 6-27
adding browser navigation, 5-9
adding HTTP Get, 6-29
adding HTTP Post, 6-29
adding multipart Post, 6-30
adding XML Post, 6-31
Navigator View, 1-5, 1-21
nodes
moving in a script, 3-44
Nonce, 17-6

O

Object Details View, 1-4, 1-5, 1-19
adding a table test, 5-24
adding object tests, 5-24
saving a path to a lilbrary, 5-24
using, 5-23
viewing the object path, 5-24
Object Enumeration
setting preferences, 2-20
object identification
editing libraries, 5-27
setting preferences, 2-25, 2-26, 2-27, 2-28, 2-29, 2-32
x,y offset, 5-22
x,y position, 5-22
object identifiers
adding/editing, 5-3
Object Not Found setting, 2-13
Object Test Failed setting, 2-11
ODBC Driver, 4-5
Offset (x,y), 5-22
OpenScript
Breakpoint View, 1-21
Console View, 1-17
Correlation interface, 1-2
Data Table View, 1-18
Databanking, 1-2
Debug View, 1-21
Declaration View, 1-21
definition, 1-1
Details View, 1-16
Developer Perspective, 1-4
installing, 1-3
Java Code View, 1-2
menu options, 1-5
Navigator and Package Explorer Views, 1-21
Object Details View, 1-19
preferences, 1-3
Problems View, 1-17
Properties View, 1-2, 1-17
Results View, 1-17
Script Variables View, 1-20
starting, 1-3
tool bar, 1-13
tree view, 1-2
Treeview Breakpoint View, 1-20

Variables View, 1-21
OpenScript Workbench, 1-4
Operation Invocation Error setting, 2-11
options
Substitute Variable, 4-3
Oracle ADF Functional Test module
adding Oracle Forms object identifiers, 9-4
editing Oracle Forms object identifiers, 9-4
setting record preferences, 2-26, 2-29, 9-4
Oracle EBS/Forms
creating new project, 3-3, 3-5
Oracle EBS/Forms Functional Test Error Messages, D-8
Oracle EBS/Forms Functional Test module
about, 7-1, 11-1
adding Oracle Forms object identifiers, 7-3
editing Oracle Forms object identifiers, 7-3
event-driven recording, 7-5
key features, 7-1
prerequisites, 7-2
setting playback preferences, 2-17, 2-18, 7-7
setting record preferences, 2-27, 7-3
using, 7-1
Oracle EBS/Forms Functional Test scripts
adding EBS/Forms actions, 7-8
modifying, 7-8
playing back scripts, 7-7
playing back scripts with iterations, 7-7
recording scripts, 7-2, 7-4
using Forms Functional Test API, 7-8
Oracle EBS/Forms Load Test error messages, D-10
Oracle EBS/Forms Load Test module
Correlation Library, 8-9
key features, 8-1
prerequisites, 8-2
setting Correlation preferences, 8-8
setting playback preferences, 8-4
setting record preferences, 2-28, 8-3
using, 8-1
Oracle EBS/Forms Load Test scripts
adding Forms actions, 8-6
analyzing Message Logs, 8-13
converting Forms actions to XML, 8-7
debugging using the Message Log, 8-12
modifying, 8-6
playing back scripts, 8-4
playing back scripts with iterations, 8-5
recording scripts, 8-3
troubleshooting, 8-11
troubleshooting ifError messages, 8-14
using Forms Load Test API, 8-8
Oracle Forms Error setting, 2-12
Oracle Fusion/ ADF Load Test module
key features, 10-1
Oracle Fusion/ADF Functional Test scripts
adding Fusion/ADF actions, 9-7
modifying, 9-7
Oracle Fusion/ADF Load Test module
Correlation Library, 10-7
setting Correlation preferences, 10-6
using, 10-1
Oracle Fusion/ADF Load Test scripts
application programming interface, 10-7
playing back scripts, 10-5
playing back scripts with iterations, 10-6
recording scripts, 10-2
Oracle JD Edwards
creating new project, 3-6
Oracle JD Edwards EnterpriseOne
creating new project, 3-4
Oracle Load Testing
playing back HTTP scripts, 6-15
Oracle Thin JDBC Driver, 4-4

P

Package Explorer View, 1-5, 1-21
page title tests, 5-17
parameterization, 4-1, 6-27
Parameterize URLs menu option, 3-38
password digest, 17-6
Password setting, 4-5
password text, 17-6
PeopleSoft
creating new project, 3-6
PeopleSoft Load Test module
Correlation Library, 16-3
key features, 16-1
setting Correlation preferences, 16-3
using, 16-1
PeopleSoft Load Test scripts
playing back scripts, 16-2
playing back scripts with iterations, 16-2
recording scripts, 16-1
platform error messages, D-4
Playback Error setting, 2-11
Playback Errors, D-14
Playback Failed setting, 2-13
playback HTTP scripts
using iterations, 4-9, 6-14
using Oracle Load Testing, 6-15
Playback preferences, 2-8
Action Settings, 2-18
Capture, 2-19
Compression, 2-13
Connections, 2-14
Debug, 2-9
Download Manager, 2-15
Error Handling, 2-9
Event Timeout, 2-17
General, 2-8
Headers, 2-14
Miscellaneous, 2-16, 2-20, 2-22
Object Identification, 2-20
Object Timeout, 2-19
setting, 6-13
SSL, 2-15
System, 2-10, 2-13
playback results
comparing, 6-15

playback settings
exporting, 3-17
Port setting, 4-4
Position (x,y), 5-22
post data variables, 6-25
Post Navigation, 6-29
preferences, 2-1
ADF functional test, 2-26, 2-29
Browsers, 2-4
EBS/Forms functional test, 2-17, 2-18, 2-27
EBS/Forms load test, 2-28
encryption, 2-4
error recovery, 2-10
general, 2-8
General category, 2-2
HTTP, 2-13, 2-22
keyboard shortcut settings, 2-5
Large Data settings, 2-7
playback, 6-13
Playback category, 2-8
Record category, 2-22
repository settings, 2-7
setting, 2-1
setting project, 2-40
setting Siebel Correlation, 19-14
Siebel functional test, 2-30
Step Group category, 2-34
Web functional test, 2-19, 2-31
Web Services, 2-33
Problems View, 1-4, 1-5, 1-17
project preferences, 2-40
properties
assets, 1-15
correlation, 3-14
modules, 3-14
setting for scripts, 3-13
step groups, 3-14
Properties View, 1-4, 1-5, 1-17

R

Randomly setting, 4-11
Range settings, 4-12
record ADF Functional Test, 9-3
record Adobe Flex (AMF) Load Test, 12-2
record Adobe Flex Functional Test, 11-3
record HTTP script, 6-12
record Hyperion Load Test, 13-1
record JD Edwards Functional Test, 14-1
record JD Edwards Load Test, 15-1
record Oracle EBS/Forms Load Test, 8-3
record Oracle Forms Functional Test, 7-2
record Oracle Fusion/ADF Load Test, 10-2
record PeopleSoft Load Test, 16-1
Record preferences, 2-22
Certificates, 2-25, 2-33
EBS/Forms load test, 2-28
General, 2-22, 2-27, 2-31, 2-33
Object Identification, 2-25, 2-26, 2-27, 2-28, 2-29, 2-32
Parser Tools, 2-33
Proxy Configuration, 2-33
Proxy Settings, 2-24
URL Filters, 2-25
record Web Functional Test, 5-2
recorded results
comparing, 6-15
recording scripts, 3-7
Remove Unchanging Variables menu option, 3-37
Replace URLs setting, 2-9
repository
creating, 3-1, 3-2
definition, 3-1
managing, 3-3
setting preferences, 2-7
Response Time Error setting, 2-12, 2-13
Result Object
adding to script, 3-42
Results View, 1-4, 1-5, 1-17
toolbar buttons, 1-18
resume script programmatically, 3-45
RUEI User Session logs
importing, 3-17
Run section
definition, 1-2
runScript statement
adding to script, 3-34

S

script
commands, 1-2
steps, 1-2
script creation exceptions, D-2
script databanks
using, 4-2
script exceptions, D-1
script project
creating, 3-3
script service exceptions, D-4
Script Variable View, 1-4, 1-5
script variables
using, 6-18
Script Variables View, 1-20
Script View, 1-5, 6-17
Assets, 1-16
Java Code, 1-15
Tree View, 1-15
Scripting Workbench, 1-1
scripts
aborting and resuming, 3-45
adding assets, 3-33
creating, 3-1
creating from templates, 3-13
debugging, 3-46
exporting, 3-10
importing, 3-10, 3-12
managing, 3-3
migrating, 3-12
modifying, 3-1, 3-18

opening existing, 3-9
pausing and resuming, 3-49
recording, 3-7
setting properties, 3-13
storing, 3-1
using as dedicated function libraries, 3-27
security certificate, E-2
Security Extensions
adding to Web Services, 17-5
Security SOAP Messages with Attachments, 17-7
segment parser exceptions, D-3
Segment Parser Failed setting, 2-11
Select Next Record setting, 4-11
Sequentially setting, 4-11
server response tests, 5-11, 6-23
set variable
adding to script, 3-35
setting Shared Data connection parameters, 21-3
setting Shared Data password, 21-3
settings
exporting, 3-17
Shared Data error messages, D-14
Shared Data module
about, 21-1
basic scenarios, 21-2
clearing hash maps, 21-8
clearing queues, 21-6
creating hash maps, 21-6
creating queues, 21-4
destroying hash maps, 21-8
destroying queues, 21-6
enabling, 21-2
getting data from hash maps, 21-7
getting data from queues, 21-5
inserting data into hash maps, 21-6
inserting data into queues, 21-4
key features, 21-1
setting connection parameters, 21-3
setting password encryption, 21-3
using the Shared Data API, 21-8
Siebel
creating new project, 3-4, 3-6
Siebel Correlation Library, 19-15
Siebel error messages, D-16
Siebel exceptions, D-15, D-16
Siebel Functional Test module
enabling Siebel test automation, 18-3
High-Interactivity components, 18-2
key features, 18-1
setting browser options, 18-4
setting up Siebel environment, 18-2
Standard-Interactivity applications, 18-2
testing Siebel applications, 18-2
using, 18-1
Siebel Functional Test scripts
adding Siebel actions, 18-7
creating Siebel scripts, 18-3
determining a Siebel component type, 18-5
handling non-standard dialog boxes, 18-8
modifying scripts, 18-7
recording Siebel functional test scripts, 18-5
setting record preferences, 18-6
starting the Siebel application, 18-4
using Siebel Functional Test API, 18-10
Siebel Load Test module
about, 19-1
key features, 19-1
SOAP Message Transmission Optimization Mechanism, 17-7
Socket Timeout setting, 2-17
Solve Variable Failed setting, 2-12, 2-13
Specific Records setting, 4-12
SQL Execute Error setting, 2-13
SQL Validation Row Count Error setting, 2-13
SSL certificate, E-2
SSL exceptions, D-5
Standard-Interactivity applications, 18-2
Starting Record setting, 4-12
Status Bar Test Error setting, 2-12
Step Group preferences, 2-34
ADF load test, 2-34
basic module, 2-34
Flex (AMF) load test, 2-35
Forms functional test, 2-36
Forms load test, 2-37
HTTP, 2-36
Siebel functional test, 2-38, 2-39
Siebel load test, 2-39
step groups
adding to script, 3-18
definition, 3-18
Stop the User setting, 4-11
SWA, 17-7
SWECmd=AutoOn, 18-4
synchronization points
adding to scripts, 3-35

T

Table Test Failed setting, 2-11
TCP exceptions, D-5
test cases
adding HTML test, 5-18
adding object tests, 5-13
adding page title test, 5-17
adding server response test, 5-11, 6-23
adding table tests, 5-15
adding text matching, 5-12, 6-22
adding XML test, 5-19
Test Modules, 1-1
Tester Perspective, 1-4
adding views, 3-46
Console View, 1-4, 1-17
Data Table View, 1-4
Details View, 1-4, 1-16
Error Log View, 1-4
Object Details View, 1-4
Problems View, 1-4, 1-17
Properties View, 1-4, 1-17
Results View, 1-4, 1-17

Script Variables View, 1-4
Script View, 1-4, 1-15
Treeview Breakpoint View, 1-4
text file encoding
changing, 3-46
Text Matching Failed setting, 2-11, 2-12
text matching tests
adding, 5-12, 6-22
Title Test Failed setting, 2-13
Tree View, 6-17
Finish section, 1-15
Initialize section, 1-15
Run section, 1-15
Treeview Breakpoint View, 1-4, 1-5, 1-20
troubleshooting, E-1

U

Unexpected Script Error setting, 2-11
URL
encoding exceptions, D-4
parameterizing, 3-38
setting filter preferences, 2-25
Use XPath setting, 2-20
Use XPath with Smart Match setting, 2-20
user agent
adding to script, 6-26
Username setting, 4-5
Utilities module
getting database values, 20-3
key features, 20-1
using, 20-1
using XPath generator, 20-8
working with CSV files, 20-2
working with text files, 20-1
working with XML files, 20-3

V

variable
deleting from script, 6-21
finding in script, 6-21
inspecting, 3-50
removing unchanged, 3-37
variable exceptions, D-4
Variable Not Found setting, 2-11
variable scope, 3-36
Variables View, 1-5, 1-21
verifying actions, 3-41
views
adding, 3-46

W

Wait for Page Timeout setting, 2-13
watch variable, 3-50
Web
creating new project, 3-4
Web error messages, D-16
Web Functional Test module
about, 5-1
adding Web object identifiers, 5-3
editing Web object identifiers, 5-3
key features, 5-2
setting playback preferences, 2-19, 5-7
setting record preferences, 2-31, 5-2
using, 5-1
Web Functional Test scripts
adding browser navigation, 5-9
adding HTML tests, 5-18
adding image object tests, 5-16
adding object libraries to scripts, 5-10
adding object tests, 5-13
adding page title tests, 5-17
adding server response tests, 5-11
adding table tests, 5-15
adding text matching tests, 5-12
adding wait for page, 5-21
adding web actions, 5-9
adding XML tests, 5-19
editing object libraries, 5-27
inspecting object paths, 5-22
modifying, 5-8
playing back scripts, 5-7
playing back scripts with iterations, 5-7
recording scripts, 5-2, 5-6
setting properties, 5-25
substituting databank variables, 5-25
using Web Functional Test API, 5-26
Web Service exceptions, D-16
Web Services module
adding WSDL files, 17-2
key features, 17-1
setting record preferences, 2-33, 17-8, 17-9
using, 17-1
Web Services scripts
adding attachments, 17-6
adding methods to scripts, 17-2
adding post navigation, 17-4
adding security extensions, 17-5
adding Text Matching tests, 17-4
creating new project, 3-5
creating scripts using WSDL Manager, 17-2
editing method parameters, 17-3
modifying scripts, 17-3
recording scripts, 17-8
Web/HTTP
creating new project, 3-6
When Out of Records setting, 4-11
When Script Requests a Record setting, 4-10
Workbench
Developer Perspective, 1-4
overview, 1-4
Tester Perspective, 1-4
workspaces
creating, 3-1
definition, 3-1
WSDL Manager
adding WSDL files, 17-2

X

XML file data
  posting, 6-16
  reading files, 20-3
XML Parsing Error setting, 2-13
XML Post Navigation, 6-31
XML Test Failed setting, 2-12
XML tests, 5-19
XPath Generator, 20-8

Z

Zero Length Downloads setting, 2-12