Oracle® Functional Testing

OpenScript Programmer's Reference

Release 12.2.0.1

E37832-01

November 2012

ORACLE®

Oracle Functional Testing OpenScript Programmer's Reference Release 12.2.0.1

E37832-01

Primary Author: Rick Santos

Contributing Author: Kevin Gehrke.

Contributor:

Contents

5 Oracle EBS/Forms Load Module

17 DataTable Utility Module

18 SharedData Module

19 Utilities Module

20 Web Services Module

Index

List of Tables

Preface

Welcome to the Oracle OpenScript Programmers Reference Guide. Oracle OpenScript is an extensible, standards-based test automation platform designed to test the next generation of Web applications. This guide explains the Oracle OpenScript scripting Application Programming Interface (API) used to creating or modifying test scripts used for testing Web applications.

Audience

This guide is intended for Web Testing Quality Assurance (QA) engineers who will be modifying OpenScript scripts using the Java-based OpenScript scripting API. OpenScript is based upon the Eclipse development environment. This guide assumes that the users of this guide have Java programming experience and are knowledgeable in using the Eclipse development environment.

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:

- Part I, "Introduction" introduces the OpenScript Programmers Reference and provides basic information for using the OpenScript scripting language and Application Programming Interface (API).
- Part II, "Load Testing Modules API Reference" provides a complete listing and reference for the methods in the OpenScript Load Testing Modules Application Programming Interface (API).
- Part III, "Functional Testing Modules API Reference" provides a complete listing and reference for the methods in the OpenScript Functional Testing Modules Application Programming Interface (API).
- Part IV, "Utility Modules API Reference" provides a complete listing and reference for the methods in the OpenScript Utilities Modules Application Programming Interface (API).

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.

Part I

Introduction

This Part of the OpenScript Programmer's Reference introduces the OpenScript Programmers Reference and provides basic information for using the OpenScript scripting language and Application Programming Interface (API).

This Part contains the following chapters:

1
Introduction

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 About the OpenScript Scripting API

This guide provides reference information for OpenScript users who wish to modify script projects in the Java code view. This guide provides basic scripting information and detailed command reference information for the Application Programming Interfaces (APIs) for the technology and utility modules supported by the OpenScript Java scripting language.

The OpenScript scripting API is divided into modules based upon load testing scripts, functional testing scripts and utilities.

Load testing scripts use protocol automation to simulate users creating load against a Web server. The HTTP load testing module of the OpenScript scripting API is the basic load testing module and is used to create load testing scripts. Other supported load testing modules are used with the HTTP module to provide support for specific Web technologies, such as Oracle EBS/Forms or Oracle Fusion/ADF web applications.

Functional testing scripts automate browser actions to simulate a user performing actions within a web application. The Web functional testing module of the OpenScript scripting API is the basic functional testing module and is used to create functional testing scripts. Other supported functional testing modules are used with the Web module to provide support for specific Web technologies, such as Oracle EBS/Forms or Oracle Fusion/ADF web applications.

2

OpenScript Scripting Basics

This chapter explains the procedures for creating and modifying scripts in the OpenScript code view.

2.1 About Oracle OpenScript

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.

2.2 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.

2.3 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.

2.3.1 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. The Assets view tab shows the script assets (databanks, jar files, Object libraries and child scripts) that have been added to the script.

- **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.

- **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.

2.3.2 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.

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

2.3.3.1 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.

2.3.3.2 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.

2.3.3.3 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.

2.3.4 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.

2.3.5 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.

2.3.6 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 the *Oracle OpenScript User's Guide* 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.
- 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.

2.3.7 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 The *Oracle OpenScript User's Guide* 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 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 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.

2.3.8 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.

2.3.9 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.

2.3.10 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.

2.3.11 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.

2.3.12 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.

2.4 Creating a Script Project

You must create a script project to generate the basic structure that you can then customize. The type of script project you create depends upon the type of testing the

script is intended to perform: Load Testing (Protocol Automation) or Functional Testing (Browser/GUI Automation) and the Web application that will be tested.

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 (see the *Oracle OpenScript User's Guide* for additional information):

   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.
   - **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.
   - **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.
   - **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.
   - **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.
   - **Web**: This option lets you create a new script for automated functional testing of Web applications at the browser/gui level.

   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.
   - **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.
   - **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.
   - **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.

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.
- **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.
- **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.
- **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.
- **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.
- **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.
- **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.
- **Web/HTTP**: This option lets you create a new script for load testing of Web Applications at the HTTP protocol level.

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.
- **Object Library**: This option lets you create a new Object Library or open an existing Object Library.

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 2.7.9, "Using a Script as a Dedicated Function Library" for additional information about creating a function library.

2.4.1 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.

In many cases, it is more convenient to initially record a script against the Web application-under-test and modify the script Java code than it is to create the script in script Jave code.

See Chapter 3 in the *Oracle OpenScript User's Guide* for additional information about recording scripts.

2.5 Creating Functional Test Scripts

A script project creates the basic structure of an OpenScript script. Initially, the script project includes only the Initialize, Run, and Finish script nodes with the underlying Java code. You use the recording capabilities of OpenScript to record page navigations and generate the Java code for the script.

2.5.1 Creating a Web Functional Test Script Project

To create a functional test script project:

1. Select **New** from the **File** menu.
2. Select script type under the **Functional Testing (Browser/GUI Automation)** folder.
3. Click **Next**.
4. Set the Repository/workspace.
5. Enter a script name.
6. Click **Finish**. OpenScript creates the script project and shows the Initialize, Run, and Finish nodes in the Script View.

The resulting script code appears as follows in the Java Code view:

```
import oracle.oats.scripting.modules.basic.api.*;
import oracle.oats.scripting.modules.browser.api.*;
import oracle.oats.scripting.modules.functionalTest.api.*;
import oracle.oats.scripting.modules.utilities.api.*;
import oracle.oats.scripting.modules.utilities.api.sql.*;
import oracle.oats.scripting.modules.utilities.api.xml.*;
import oracle.oats.scripting.modules.utilities.api.file.*;
import oracle.oats.scripting.modules.webdom.api.*;

public class script extends IteratingVUserScript {
  @ScriptService oracle.oats.scripting.modules.utilities.api.UtilitiesService utilities;
  @ScriptService oracle.oats.scripting.modules.browser.api.BrowserService browser;
  @ScriptService oracle.oats.scripting.modules.functionalTest.api.FunctionalTestService ft;
  @ScriptService oracle.oats.scripting.modules.webdom.api.WebDomService web;

  public void initialize() throws Exception {
  }

  /**
   * Add code to be executed each iteration for this virtual user.
   */

  public void run() throws Exception {

  }

  public void finish() throws Exception {
  }
}
```

The @ScriptService list will vary depending upon the type of functional test script project created. The script services are used to access the API for each particular service. The following table lists the service names and utilities for the types of functional test script projects:

Table 2–1 Functional Test Script API Service Names

Functional Test Script Type	Service Name
Adobe Flex	`flexFT.[method]`
Applet utilities	`applet.[method]`
Browser utilities	`browser.[method]`
Datatable utilities	`datatable.[method]`

Table 2–1 (Cont.) Functional Test Script API Service Names

Functional Test Script Type	Service Name
Databank utilities	`databank.[method]`
Functional test	`ft.[method]`
Oracle EBS/Forms	`forms.[method]`
Oracle Fusion/ADF	`adf.[method]`
Oracle JD Edwards EnterpriseOne	`eone.[method]`
Shared Data utilities	`sharedData.[method]`
Siebel	`siebelFT.[method]`
Utilities (other)	`utilities.[method]`
Web (base scripting for functional tests).	`web.[method]`

2.5.2 Recording a Functional Test Scripts

Recording scripts captures the page navigations and generates the Java code that will be used to drive script playback for testing purposes. Web Functional test scripts record and playback actions performed on browser objects.

To record a functional test script:

1. Select **Record** from the **Script** menu or click the Record toolbar button. OpenScript opens a browser window and the OpenScript toolbar window.
2. Enter the URL of the Web application-under-test into the browser Address line and press **Enter**.
3. When the application loads, navigate the application to record browser actions against the application.
4. Click stop on the OpenScript toolbar to stop recording.
5. Save the script.

2.5.3 Opening and Closing a Browser

Functional test scripts use browser automation to simulate user actions against a Web application. You can use the `browser` service API to launch and close the browser and set the browser type programmatically, as follows

```
browser.launch();
browser.setBrowserType(BrowserType.InternetExplorer);
browser.close();
browser.closeAllBrowsers();
```

Typically, `browser.launch()` is used in the `initialize()` section of the script as the browser only needs to be launched once at the beginning of script playback. `browser.close()` can be used in the `finish()` section to close the browser at the end of script playback.

2.5.4 Navigating Web Pages in Functional Test Scripts

Functional test scripts use the `web.window()` methods to navigate web pages in the browser. Web page navigation is typically performed in the `run()` section of the script. The following example code navigates the browser to the

`http://example.com/products.aspx` page, then waits for the page to load before continuing script execution:

```
web.window("/web:window[@index='0' or @title='about:blank']")
  .navigate("http://example.com/products.aspx");
web.window("/web:window[@index='0' or @title='http://example.com/products.aspx']")
  .waitForPage(null);
```

Once a page is loaded, you can use other API methods to perform actions on the page, for example click a link, as follows:

```
web.link("/web:window[@index='0' or @title=' Products']" +
  "/web:document[@index='0']/web:a[@text='Mail Order Bikes' " +
    "or @href='http://example.com/bikes/Main.aspx' or @index='5']")
  .click();
```

2.5.5 About Object Identification Paths

The OpenScript functional test object identification paths specify how the API methods locate and identify browser objects on which to perform actions.

The object identification paths typically follow a hierachical structure as follows:

```
window
  document
    [form]
      element
```

In order for the OpenScript playback to perform a click action on a particular button, the object identification path specified in the `web.button("`*`path`*`").click` method must include the path to the button object, as follows:

```
web.button(
  "/web:window[@index='0' or @title='Products']" +
  "/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']")
  .click();
```

This path specifies that the button is located in the browser window with index value 0 or the title "Products". Within the window is a web document with index value 0. Within the document is a web form with the ID "loginform" or the name attribute "loginform" or the form with index value 0. Within the form is the web input button with the name attribute "LoginButton" or the value attribute "Login" or the index value 0.

Depending upon the module, a method uses a shorter path from within the context set by a previous method.

```
web.link(30,"/web:window[@index='0' or @title='Oracle Applications Home Page']" +
  "/web:document[@index='0']" +
  "/web:a[@text='Enter' or @href=\"javascript:launchForm(&apos;" +
  "http://example.com:8000/OA_HTML/RF.jsp?function_id=2100&resp_id=52297" +
  "&resp_appl_id=8721&security_group_id=0&lang_code=US&apos;)\" or @index='35']")
  .click();

forms.button(32, "//forms:button[(@name='CALENDAR_OK_0')]")
  .click();
```

See the Object Identification tab in the Record Preferences of the specific functional test module for a list of the object names and attributes.

2.6 Creating a Load Test Script

Load test scripts record and playback navigations performed using the HTTP protocol. Load script are typically used for performing load tests on an application using the Oracle Load Testing application.

2.6.1 Creating a Load Test Script Project

To create a load test script project:

1. Select **New** from the **File** menu.
2. Select script type under the **Load Testing (Protocol Automation)** folder.
3. Click **Next**.
4. Set the Repository/workspace.
5. Enter a script name.
6. Click **Finish**. OpenScript creates the script project and shows the Initialize, Run, and Finish nodes in the Script view.

The resulting script code appears as follows in the Java Code view:

```
import oracle.oats.scripting.modules.basic.api.internal.*;
import oracle.oats.scripting.modules.basic.api.*;
import oracle.oats.scripting.modules.http.api.*;
import oracle.oats.scripting.modules.http.api.HTTPService.*;
import oracle.oats.scripting.modules.utilities.api.*;
import oracle.oats.scripting.modules.utilities.api.sql.*;
import oracle.oats.scripting.modules.utilities.api.xml.*;
import oracle.oats.scripting.modules.utilities.api.file.*;

public class script extends IteratingVUserScript {
  @ScriptService oracle.oats.scripting.modules.utilities.api.UtilitiesService utilities;
  @ScriptService oracle.oats.scripting.modules.http.api.HTTPService http;

  public void initialize() throws Exception {
  }

  /**
   * Add code to be executed each iteration for this virtual user.
   */

  public void run() throws Exception {

  }

  public void finish() throws Exception {
  }
}
```

The @ScriptService list will vary depending upon the type of load test script project created. The script services are used to access the API for each particular service. The following table lists the service names and utilities for the types of load test script projects:

Table 2–2 Load Test Script API Service Names

Load Test Script Type	Service Name
Adobe Flex (AMF)	`amf.[method]`
Databank utilities	`databank.[method]`
Datatable utilities	`datatable.[method]`
Oracle EBS/Forms	`nca.[method]`
Oracle Fusion/ADF	`adfload.[method]`
Shared Data utilities	`sharedData.[method]`
Siebel	`siebel.[method]`
Web/HTTP (base scripting for load tests).	`http.[method]`
Utilities (other)	`utilities.[method]`

2.6.2 Recording a Load Test Script

To record an load test script:

1. Select **Record** from the **Script** menu or click the Record toolbar button. OpenScript opens a browser window and the OpenScript toolbar window.
2. If asked if you want to clear the cache, click **Yes**.
3. Enter the URL of the Web application under test into the browser's Address bar and press **Enter**.
4. When the application loads, navigate the application to record the HTTP protocol traffic between the web browser and the web server.
5. Click stop on the OpenScript toolbar to stop recording.
6. Save the script.

2.6.3 Setting the User Agent and Language Type

Load test scripts use protocol automation to send requests and receive responses from the web server. A browser instance is not required. However, the request header typically sent by a browser. The request header specifies the type of request and sets various settings, such as the host name, Accept settings, Connection setting, User Agent settings, etc. The following example shows a GET request heading:

```
GET /Stocks HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0;
SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC
6.0; .NET4.0C; .NET4.0E; InfoPath.2)
Host: example.com
Accept: text/html, image/gif, image/jpeg, */*
Accept-Language: en-us
Connection: Keep-Alive
Accept-Encoding: gzip, deflate
```

By specifying the user-agent string, you can simulate other types of browsers during load script playback. You can use the `http.setUserAgent` method to set the user agent settings, as follows:

```
http.setUserAgent("Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; " +
  "WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; " +
  ".NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; InfoPath.2)");
```

Typically, `http.setUserAgent()` method used in the `initialize()` section of the script as the setting is set once at the beginning of script playback.

The request Accept-Language setting is used to specify which languages are acceptable for the response. The setting is set using the `http.setAcceptLanguage()` method. The String parameter specifies the ISO-639 language abbreviation and optionally a ISO-3166 country code, as follows:

```
http.setAcceptLanguage("en");
-or-
http.setAcceptLanguage("en-us");
```

Typically, `http.setAcceptLanguage()` method used in the `initialize()` section of the script as the setting is set once at the beginning of script playback.

2.6.4 Navigating Web Pages in Load Test Scripts

Load test scripts use the `http.get()` and `http.post()` methods to navigate web pages. Web page navigation is typically performed in the `run()` section of the script. The following example code send and HTTP request to the server to for the Web document `http://example.com/default.asp`:

```
http.get(4, "http://example.com/default.asp", null, null, true,
  "ASCII", "ASCII");
```

You can use other API methods to perform actions on the response, for example, extract values from the document, as follows:

```
{
 http.solveXPath("web.formaction.loginform", ".//FORM[@name='loginform']" +
  "/@action", "default.asp", 0, EncodeOptions.None);
 http.solveXPath("web.input.login", ".//INPUT[@name='login']" +
  "/@value", "ta287", 0, EncodeOptions.None);
 http.solveXPath("web.input.password", ".//INPUT[@name='password']" +
  "/@value", "ta", 0, EncodeOptions.None);
http.solveXPath("web.input.LoginButton", ".//INPUT[@name='LoginButton']" +
  "/@value", "Login", 0, EncodeOptions.None);
}
```

The `http.post()` method is used to pass postdata to the Web server, as follows:

```
http.post(10,"http://example.com/{{web.formaction.loginform,default.asp}}",
  null, http.postdata(http.param("login","{{web.input.login,ta287}}"),
    http.param("password", "{{web.input.password,ta}}"),
    http.param("LoginButton","{{web.input.LoginButton,Login}}")),
  null, true, "ASCII", "ASCII");
```

2.7 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.

2.7.1 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 in the Java code:

1. Open or create a script project.
2. Select the Java Code view and add the `beginStep()` and `endStep()` methods as follows:

```
beginStep("Step Group 1", 10);
{
        /**
         * Add code to be executed for the step group.
         */
       info("Step Group 1");
}
endStep();
```

   The first parameter is a String specifying the step group name. The second parameter is an optional value specifying the amount of time to wait in milliseconds before beginning the step. The delay value will be adjusted based on the VUser's current delay settings.
3. Add code to the step group.

2.7.2 Adding a Delay to a Script

To add a delay to a script:

1. Open or create a script project.
2. Select the Java Code view and add the `think()` method with a Double time value in seconds as follows:

```
{
think(10.0);
}
```

2.7.3 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 Java Code view and add the `info()`, `warn()`, or `fail()` method with a String specifying the message, as follows:

```
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.

Warning messages report the error as a warning and continue script execution. Fail messages report the error as failure and stop script execution.

2.7.4 Encrypting and Decrypting Data

To add encrypt and decrypt data in a script:

1. Open or create a script project.
2. Select the Java Code view and add the `encrypt()` and `decrypt()` methods with a String specifying the text to encrypt or decrypt, as follows:

```
String encrypted = encrypt("encrypt this string");
```

```
  info(encrypted);
String decrypted = decrypt(encrypted);
  info(decrypted);
```

2.7.5 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**.

The password must be the same on all machines that will edit or playback this script.

2.7.6 Running a Child Script from Within a Script

You can add a method to run another child script from within the current script using the getScript("alias").run(iterationCount) method.

1. Open or create a script project.
2. Select the Java Code view and add the `getScript("alias").run(iterationCount)` method with a String specifying the alias of the script to run and the iteration count, as follows:

```
getScript("alias").run(1);
```

You must also add the child script to the current script as a script asset to run the child script..

2.7.7 Calling a Function from a Child Script from Within a Script

You can add a method to run another child script from within the current script using the getScript("alias").run(iterationCount) method.

1. Open or create a script project.
2. Select the Java Code view and add the `getScript("alias").callFunction("functionName", "args")` method with a String specifying the alias of the script to run, the function and the function arguments, as follows:

```
getScript("alias").callFunction("functionName", "args");
```

You must also add the child script to the current script as a script asset to call the function in the child script..

2.7.8 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 2.7.9, "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. In the Java Code view, add the `public void` *`function name`* 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 {
```

3. 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:

In the Java Code view, add the `getVariables().set()` statement 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}}");
```

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:

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 2.7.9, "Using a Script as a Dedicated Function Library" for addition information about dedicated function libraries.

2.7.8.1 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.

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"));
```

2.7.8.2 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.

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"));
```

2.7.8.3 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.

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");
```

2.7.8.4 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'}}"));
```

2.7.9 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. Code Assist or Content Assist provides you with a list of suggested completions for partially entered strings. In the Java editor press Ctrl+Space or select Content Assist from the **Edit** menu.

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.

2.7.9.1 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 2.7.8, "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 2.7.9.2, "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 disadvantages 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. Any saved script can be a template script.

   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 "Opening Existing Scripts" in Chapter 2 of the *Oracle OpenScript User's Guide* 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.

2.7.9.2 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 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** menu and then select **Other** from the **Add** sub menu.
10. Expand the **Script Functions** and **Local Script** nodes.
11. Select **[New Function]** and click **OK**.
12. Enter a function name and description.
13. 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.
14. Repeat step 14 for each argument to add to the function.
15. Click **OK** to add the function to the script.
16. 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.
17. Repeat steps 10 through 17 to add additional functions to the library script. See also Section 2.7.8, "Adding a Function to a Script" for additional information about adding functions to a script.
18. Save the library script.

2.7.9.3 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 **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 2.7.8, "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.

2.7.10 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 2.7.9, "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.

2.7.11 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 Assets tab of the script view 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 asset to add from a repository.
5. 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.

6. Click **OK** to add the asset to the script properties.
7. 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.

2.7.12 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. In the Java Code view, add the `syncPointWait("name");` method to the script code, as follows:

```
syncPointWait("MySyncPoint");
```

3. Save the script in OpenScript.
4. 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.

2.7.13 Setting and Evaluating Script Variables

To add a Set Variable to a script:

1. Open or create a script project.
2. In the Java Code view, add the `getVariables().set("variable name", "value");` method to the script code:

```
getVariables().set("sVar_MyVar", "My_Value");
```

If you want to evaluate the variable with a value from an OpenScript transform variable (i.e. a variable value contained in `{{}}` syntax), use the `eval()` method as follow:

```
http.solve("varTitle", "<TITLE>(.+)</TITLE>", "Page Title Error", false,
Source.Html, 0);

getVariables().set("sVar_MyVar", eval("{{varTitle}}"));
//or
```

```
String title = eval("{{varTitle}}");
info("Title = " + title);
//or
info ("Title = {{varTitle}}");
```

2.7.13.1 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)
```

2.7.14 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");
```

2.7.15 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 error recovery action.
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 Pause 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);
```

2.7.15.1 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)

2.7.15.2 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
```

2.7.15.3 Actions

The following are the possible values for *`action`* in the Java code statements:

```
Fail
Ignore
Warn
ReportErrorAndContinue
Pause
```

2.7.16 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.

2.7.16.1 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 2.7.15, "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 2.7.16.2, "Adding a Has Error Control Statement" for additional information.

2.7.16.2 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 2.7.16.3, "Adding a Result Object Message" for additional information.

2.7.16.3 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}}");
    }
    ```

2.7.16.4 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`

2.7.17 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

2.7.17.1 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**.

2.7.17.2 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.

2.7.17.3 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);
getScript("Web2").run(1);
```

2.7.18 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);
}
```

2.8 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 for HTTP protocol load test scripts:

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.

2.8.1 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.
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, 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 2.8.4, "Playing Back Scripts With Iterations" for additional information about playing back scripts with databanks.

2.8.2 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.

2.8.3 Getting Databank Records

To get Databank records to use with a script:

1. Open or create a script project.
2. Click the **Assets** tab.
3. Select **Databanks**.
4. Click **Add** and select the CSV, text or database you want to get databank records from.
5. In the Java Code view, highlight the text you want to replace with a databank record column.
6. Right-click the highlighted text and select **Substitute Variable**.

7. If necessary, expand the Databanks node and select the Databank field you want to use as the input parameter data.
8. Click **Finish**.
9. 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}}");
```

10. You will also see a line added to the section in which the parameter was substituted, for example:

```
getDatabank("customer").getNextDatabankRecord();
```

In the Java Code view, a `getDatabank("databank alias").method();` can 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);
```

2.8.3.1 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.

2.8.3.1.1 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.

2.8.3.1.2 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 of the *Oracle OpenScript User's Guide* 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}}");
}
```

2.8.3.1.3 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 );
```

2.8.3.1.4 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);
}
```

2.8.3.1.5 Getting the First Record

The following example uses the `getFirstRecord()` method to get the first record in the "customer" databank:

```
getDatabank("customer").getFirstRecord();
```

2.8.3.1.6 Getting the Last Record

The following example uses the `getLastRecord()` method to get the last record in the "customer" databank:

```
getDatabank("customer").getLastRecord();
```

2.8.4 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 2.8.1, "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.

2.8.4.1 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.

2.8.4.2 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.

2.9 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.

2.9.1 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**.

   In the Java Code view, the datatable service is added to the script code as follows:

   ```
   @ScriptService oracle.oats.scripting.modules.datatable.api.DataTableService
   datatable;
   ```

5. Select **Data Table** from the **View** menu to show the view/edit Data Table tab view. The runtime Data Table is accessed using the Data Table API.

2.9.2 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");
```

2.9.3 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.

To load data from an Excel spreadsheet file:

1. Enable the Data Table service for the functional test script.
2. In the Java Code view, add the `importSheet()` method, as follows:

```
datatable.importSheet("C:\\OracleATS\\Book1.xls", "Sheet1", "Sheet1", false);
```

2.9.4 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. In the Java Code view, add the `exportSheet()` method, as follows:

```
datatable.exportSheet("C:\\OracleATS\\Book1.xls", "Sheet1", "Sheet1",
  ExportMode.CREATE);
```

When exporting Data Tables to Excel files, you can set the export policy: Create or Merge. Create will completely recreate the destination file. The resulting file will contain only the exported sheets from the Data Table. Merge will add exported sheets to the existing file, overwriting existing sheets only if the sheet is named as an exported sheet from the Data Table.

2.9.5 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.

2.9.5.1 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")`.

2.9.5.1.1 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(cellValue1);
```

2.9.5.1.2 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);
```

2.9.5.1.3 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");
```

2.9.5.1.4 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");
```

2.9.5.2 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.

2.9.5.2.1 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);
```

2.9.5.2.2 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");
```

2.9.5.2.3 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);
```

2.9.5.2.4 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);
```

2.9.5.3 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.

2.9.5.3.1 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");
```

2.9.5.3.2 Deleting Worksheets

The following example script code deletes the worksheet named "Sheet1" from the Data Table:

```
datatable.deleteSheet("Sheet1");
```

2.9.5.4 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.

2.9.5.4.1 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);
```

2.9.5.4.2 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);
```

2.9.5.4.3 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);
```

2.9.5.5 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.

2.9.5.5.1 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)
```

2.9.5.5.2 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);
```

2.9.5.6 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.

2.9.5.6.1 Setting the Next Row

The following example script code sets the next row of the current sheet in the Data Table:

```
datatable.setNextRow();
```

2.9.5.6.2 Setting the Previous Row

The following example script code sets the previous row of the current sheet in the Data Table:

```
datatable.setPreviousRow();
```

2.9.5.7 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.

2.9.5.7.1 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");
```

2.9.5.7.2 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);
```

2.9.5.7.3 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");
```

2.9.5.7.4 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);
```

2.9.5.8 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.

2.9.5.8.1 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.

2.9.5.8.2 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.

2.10 Using the Shared Data Service

This section describes how to enable and use the Shared Data Service.

2.10.1 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.

2.10.2 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. See Section 2.7.5, "Setting the Password Encryption" for additional information about setting the encryption password. You use the `sharedData` class in the Java code view to create manipulate message queues and hash maps.

2.10.3 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.

2.10.4 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.

2.10.5 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);
```

2.10.6 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.

2.10.7 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");
```

2.10.8 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");
```

2.10.9 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.

2.10.10 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 Section 2.10.5, "Inserting Data into a Shared Data Queue" 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);
```

2.10.11 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.

2.10.12 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");
```

2.10.13 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");
```

2.11 Using The Utilities API

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.

2.11.1 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);
```

2.11.2 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);
}
```

2.11.3 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"));
}
```

2.11.4 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 the *Oracle OpenScript User's Guide* 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");
```

2.11.4.1 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 2.11.4, "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.

2.11.4.2 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");
```

2.11.5 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)
   ```

2.12 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.

2.12.1 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**.

2.12.2 Adding Breakpoints to a Script

You can add breakpoints to the Java Code to halt script execution at a specific line of code in the script.

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. You can also select **Toggle Breakpoint** from the right-click shortcut menu. 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.

2.12.3 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.

2.12.4 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.

2.12.5 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).

2.13 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.

Part II

Load Testing Modules API Reference

This Part of the OpenScript Programmer's Reference provides a complete listing and reference for the methods in the OpenScript Load Testing Modules Application Programming Interface (API).

Each chapter in this part contains alphabetical listings and detailed command references for the methods in each class.

This Part contains the following chapters:

3

Adobe Flex (AMF) Load Module

This chapter provides a complete listing and reference for the methods in the OpenScript AmfService Class of AMF Load Module Application Programming Interface (API).

3.1 AmfService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript AmfService API.

3.1.1 Alphabetical Command Listing

The following table lists the AmfService API methods in alphabetical order.

Table 3–1 List of AmfService Methods

Method	Description
amf.assertText	Search the AMF contents of all documents in all browser windows for the specified text pattern.
amf.post	Post an AMF request to the AMF server identified by recID, description, and urlPath.
amf.solve	Parses a value from the most recent navigation's contents and store it as a variable.
amf.solveXpath	Extract a value from the last retrieved response using XPath notation.
amf.verifyText	Search the AMF contents of all documents in all browser windows for the specified text pattern.

The following sections provide detailed reference information for each method and enum in the AmfService Class of AMF Load Module Application Programming Interface.

amf.assertText

Search the AMF contents of all documents in all browser windows for the specified text pattern.

If the test fails, always fail the script unless the text matching test error recovery setting specifies a different action such as Warn or Ignore.

Format

The amf.assertText method has the following command format(s):

amf.assertText(*testName*, *textToAssert*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

textToAssert
a String specifying the text to match on the page, or not match on the page if TextPresence says TextPresence.FailIfPresent.

sourceType
an AmfSource enum specifying where to match the text, i.e. in the XML contents or HTTP response headers.

textPresence
a TestPresence enum specifying either PassIfPresent or FailIfPresent, depending on if you want the test to pass or fail if the text to match is present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a regular expression, wildcard, or exact match.

Throws

MatchException
if the assertion fails.

AbstractScriptException
on any other failure when attempting to assert the text.

Example

Adds Text Matching tests for Response Header and XML Content.

`myTextMatchingTest1` passes if the text to match string is present in the XML Content and matches exactly.

`myTextMatchingTest2` passes if the text to match string is present in the Response header and matches the Regular Expression.

`myTextMatchingTest3` fails if the text to match string is present in the XML Content and matches the Wildcard.

```
amf.assertText("myTextMatchingTest1", "match this text string",
 AmfSource.Content, TextPresence.PassIfPresent,
```

```
 MatchOption.Exact);
amf.assertText("myTextMatchingTest2", "jsessionid=(.+?)(?:\\\"|&)",
 AmfSource.ResponseHeader, TextPresence.PassIfPresent,
 MatchOption.RegEx);
amf.assertText("myTextMatchingTest3", "match this *",
 AmfSource.Content, TextPresence.FailIfPresent,
 MatchOption.Wildcard);
```

amf.post

Post an AMF request to the AMF server identified by recID, description, and urlPath.

Reads the XML `postMessage` as a String, serializes it into an ActionScript (AMF) object and then makes a post to the `urlPath` provided.

Format

The amf.post method has the following command format(s):

amf.post(*description*, *urlPath*, *postMessage*);

amf.post(*recId*, *description*, *urlPath*, *postMessage*);

Command Parameters

description
a String specifying a human-readable description of the operation which will be invoked on the server. Used for reporting and script display purposes only.

urlPath
a String specifying the AMF endpoint on the server where all AMF requests are posted. For example: `"http://server:8080/todolist-web/messagebroker/amf"`

postMessage
a String specifying an XML string containing the AMF message which will be posted to the urlPath.

recId
the ID of a previously recorded navigation, used for comparison purposes. May be null.

Throws

AbstractScriptException
on any exception posting data to the server.

Example

Posts the xml postMessage as an ActionScript (AMF) object to the specified URL.

```
amf.post(10, "Click the button", "http://server-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.examp.Test.getNumber\" response=\"/1\">" +
  " <list><item type=\"double\">123.456</item></list>" +
  "</body>" +
 "</bodies>" +
"</amf>");
```

amf.solve

Parses a value from the most recent navigation's contents and store it as a variable.

This method honors the "Solve Variable Failed" error recovery setting. If "Solve Variable Failed" error recovery is not set to fail, then the script will continue running even if a non-optional variable cannot be solved.

Format

The amf.solve method has the following command format(s):

amf.solve(*varName*, *pattern*, *errorMsg*, *isOptional*, *sourceType*, *resultIndex*, *encodeOption*);

Command Parameters

varName
a String specifying then name of the variable to create. Must not be `null`.

pattern
a Regular expression pattern specifying what to extract from the most recent navigation's contents. May contain a transform expression. Must not be `null`.

errorMsg
a String specifying an optional error message to display if the pattern cannot be solved. If `null`, a meaningful error message is automatically generated.

isOptional
True if the pattern does not have to be solved. If the pattern cannot be solved, the method quietly returns.

sourceType
an AmfSource enum specifying where to match the text, i.e. in the XML contents or HTTP response headers.

resultIndex
an index value specifying the 0-based index of the specific result to retrieve if the pattern matches more than one value. If `null`, all results will be added into the variables collection.

encodeOption
an EncodeOptions enum specifing the encoding or decoding operation to perform on the variable's value after solving the variable. A `null` value is equivalent to `EncodeOptions.None`.

Throws

Exception
if an error parsing the value.

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

SolveException
if the given pattern cannot be matched to the previous contents and the pattern is not optional.

Example

Parse a value from the most recent navigation's specified source using a Regular Expression and store it in a variable. An optional error message returns if the pattern cannot be solved. The pattern is optional and does not have to be solved. Includes a result index value of 0 to specify it should retrieve the first match result.

```
amf.solve("flex.dsid",
 "<entry key=\"DSId\" type=\"string\">(.+?)</entry>",
 "DSId is a required field", true, AmfSource.Content,
 0, EncodeOptions.URLEncode);
```

amf.solveXpath

Extract a value from the last retrieved response using XPath notation.

The value will be stored to the given variable.

Format

The amf.solveXpath method has the following command format(s):

amf.solveXpath(*varName*, *xpath*, *lastValue*, *resultIndex*, *encodeOption*);

Command Parameters

varName
a String specifying the variable name where the value will be stored. Must not be `null`.

xpath
a String specifying the XPath to the value to parse from the last retrieved contents. Must not be `null`.

lastValue
an optional String specifying the recorded or last known value of the value about to be parsed. May be `null`. For informational purposes only.

resultIndex
an index value specifying the 0-based index of the specific result to retrieve if the XPath returns multiple results. A `null` value specifies that all results should be returned.

encodeOption
an EncodeOptions enum specifing the encoding or decoding operation to perform on the variable's value after solving the variable. A `null` value is equivalent to `EncodeOptions.None`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Extracts a value from the browser's last retrieved contents using XPath notation and stores it in the specified script variable. Encode or decode using the specified options.

```
amf.solveXpath("amf.formaction.loginform",
 ".//FORM[@name='loginform']/@action",
 "default.asp", 0, EncodeOptions.None);
info("Login form = {{amf.formaction.loginform}}");
```

amf.verifyText

Search the AMF contents of all documents in all browser windows for the specified text pattern.

This method will never cause a script to fail, regardless of the text matching test error recovery settings.

Use amf.assertText to make the script fail when the test fails.

When playing back a script in Oracle Load Testing, always use Assertions; failed Verifications are NOT reported in OLT.

Format

The amf.verifyText method has the following command format(s):

amf.verifyText(*testName*, *textToVerify*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

textToVerify
a String specifying the text to match on the page, or not match on the page if TextPresence says TextPresence.FailIfPresent.

sourceType
an AmfSource enum specifying where to match the text, i.e. in the XML contents or HTTP response headers.

textPresence
a TestPresence enum specifying either PassIfPresent or FailIfPresent, depending on if you want the test to pass or fail if the text to match is present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a regular expression, wildcard, or exact match.

Throws

MatchException
if the assertion fails.

AbstractScriptException
on any other failure when attempting to assert the text.

Example

Adds a Verify only, never fail Text Matching tests for Response Header and XML Content.

`myTextMatchingTest1` passes if the text to match string is present in the XML Content and matches exactly.

`myTextMatchingTest2` passes if the text to match string is present in the Response header and matches the Regular Expression.

myTextMatchingTest3 fails if the text to match string is present in the XML Content and matches the Wildcard.

```
amf.verifyText("myTextMatchingTest1", "match this text string",
 AmfSource.Content, TextPresence.PassIfPresent,
 MatchOption.Exact);
amf.verifyText("myTextMatchingTest2", "jsessionid=(.+?)(?:\\\"|&)",
 AmfSource.ResponseHeader, TextPresence.PassIfPresent,
 MatchOption.RegEx);
amf.verifyText("myTextMatchingTest3", "match this *",
 AmfSource.Content, TextPresence.FailIfPresent,
 MatchOption.Wildcard);
```

4

Oracle Fusion/ADF Load Module

This chapter provides a complete listing and reference for the methods in the OpenScript AdfLoadService Class of ADF Load Module Application Programming Interface (API).

4.1 AdfLoadService ENUM Reference

The following section provides an alphabetical listing of the enums in the OpenScript AdfLoadService API.

4.1.1 Alphabetical Enum Listing

The following table lists the AdfLoadService Enums in alphabetical order.

Table 4–1 List of AdfLoadService Enums

Enum	Description
VariableType	Specifies the tag types of the ADF response content for the `solveGroupAdf()` method.

4.2 AdfLoadService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript AdfLoadService API.

4.2.1 Alphabetical Command Listing

The following table lists the AdfLoadService API methods in alphabetical order.

Table 4–2 List of AdfLoadService Methods

Method	Description
adfload.getAdfVariable	Constructor for an ADF variable.
adfload.solveGroupAdf	Extracts multiple ADF variables from the ADF Rich Response Content and stores them in the variables collection.

The following sections provide detailed reference information for each method and enum in the AdfLoadService Class of ADF Load Module Application Programming Interface.

adfload.getAdfVariable

Constructor for an ADF variable.

Format

The adfload.getAdfVariable method has the following command format(s):

adfload.getAdfVariable(*name*, *xpath*, *variableValue*, *segmentIndex*, *variableIndex*, *variableType*, *encodeOption*);

Command Parameters

name
a String spcifying the variable name. Must not be `null`.

xpath
a String specifying the XPath which is used to locate the value of an ADF variable from the last retrieved contents. Must not be `null`.

variableValue
a String specifying last value of the variable. Must not be `null`.

segmentIndex
an int specifying the index of the segment in which variable appears. Must not be `null`.

variableIndex
an int specifying the index of the appearance that matches the XPath in a segment. Must not be `null`.

variableType
an enum specifying the type of ADF Variable, such as VariableType.RICHCOMPONENT or VariableType.INPUT. Must not be `null`.

encodeOption
an EncodeOptions enum specifying the encoding or decoding operation to perform on the variable's value after solving the variable. A `null` value is equivalent to `EncodeOptions.None`.

Returns

an AdfVariable Object. Must not be `null`.

Example

Constructor for an ADF variable using properties of an ADF variable.

```
adfload.getAdfVariable("adf_option_0_9",".//OPTION[@id='opt']/@value",
 "2",0, 0, VariableType.OPTION, EncodeOptions.None);
```

adfload.solveGroupAdf

Extracts multiple ADF variables from the ADF Rich Response Content and stores them in the variables collection.

Format

The adfload.solveGroupAdf method has the following command format(s):

adfload.solveGroupAdf(*adfVariables*);

Command Parameters

adfVariables
one or more AdfVariable objects representing multiple ADF variables. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of adfload script.

Example

Extracts two ADF variables from the ADF Rich Response Content and stores them in the variables collection.

```
adfload.solveGroupAdf(adfload.getAdfVariable("adf_comp_ptemppt_soc1",
     "//AdfRichSelectOneChoice[@immediate='true']" +
     "/@fullId","ptemp:pt_soc1", -1, -1, VariableType.RICHCOMPONENT,
         EncodeOptions.None),
    adfload.getAdfVariable("adf_option_0_9", ".//OPTION[@id='opt']" +
     "/@value", "2",0, 0, VariableType.OPTION, EncodeOptions.None));
```

VariableType

The VariableType has the following values:

Table 4–3 List of VariableType Values

Value	Description
LINK	LINK tag.
FORMACTION	FORMACTION tag.
FRAMESRC	FRAMESRC tag.
INPUT	INPUT tag.
TEXTAREA	TEXTAREA tag.
OPTION	OPTION tag.
RICHCOMPONENT	RICHCOMPONENT tag.

5

Oracle EBS/Forms Load Module

This chapter provides a complete listing and reference for the methods in the OpenScript FormsService Class of Forms Load Module Application Programming Interface (API).

5.1 FormsService ENUM Reference

The following section provides an alphabetical listing of the enums in the OpenScript FormsService API.

5.1.1 Alphabetical Enum Listing

The following table lists the FormsService Enums in alphabetical order.

Table 5–1 List of FormsService Enums

Enum	Description
NcaSource	Specify which part of the contents to find the source data.

5.2 FormsService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript FormsService API.

5.2.1 Alphabetical Command Listing

The following table lists the FormsService API methods in alphabetical order.

Table 5–2 List of FormsService Methods

Method	Description
nca.alertDialog	Identifies an Oracle Forms alertDialog object by its handler name.
nca.application	Initializes the Oracle Forms Application specified by its handler name.
nca.assertStatusBarText	Searchs the status bar contents for the specified text pattern.
nca.assertText	Searches the text of selected source type for the specified text pattern.
nca.blockScroller	Identifies an Oracle Forms blockScroller object by its handler name.
nca.button	Identifies an Oracle Forms Button object by its handler name.

Table 5–2 (Cont.) List of FormsService Methods

Method	Description
nca.cancelQueryDialog	Identifies an Oracle Forms cancelQueryDialog object by its recorded ID and handler name.
nca.canvas	Identifies an Oracle Forms canvas object by its recorded ID and handler name.
nca.cfmOLE	Identifies an Oracle Forms cfmOLE object by its recorded ID and handler name.
nca.cfmVBX	Identifies an Oracle Forms cfmVBX object by its recorded ID and handler name.
nca.checkBox	Identifies an Oracle Forms CheckBox object by its handler name.
nca.choiceBox	Identifies an Oracle Forms choiceBox object by its handler name.
nca.comboBox	Identifies an Oracle Forms comboBox object by its recorded ID and handler name.
nca.connect	Connects to an Oracle Forms server.
nca.disconnect	Disconnects from Oracle Forms Server.
nca.displayErrorDialog	Identifies an Oracle Forms displayErrorDialog object by its recorded ID and handler name.
nca.displayList	Identifies an Oracle Forms displayList object by its recorded ID and handler name.
nca.editBox	Identifies an Oracle Forms editBox object by its handler name.
nca.editorDialog	Identifies an Oracle Forms editorDialog object by its recorded ID and handler name.
nca.flexWindow	Identifies an Oracle Forms flexWindow object by its handler name.
nca.genericClient	Identifies an Oracle Forms genericClient object by its handler name.
nca.getLastKnownContents	Returns the string version of contents for the specified ContentSource that was last retrieved for this virtual user.
nca.getStatusBarText	Gets the contents of the last known status bar text.
nca.helpDialog	Identifies an Oracle Forms helpDialog object by its handler name.
nca.image	Identifies an Oracle Forms image object by its recorded ID and handler name.
nca.infoBox	Identifies an Oracle Forms infoBox object by its handler name.
nca.jContainer	Identifies an Oracle Forms jContainer object by its handler name.
nca.list	Identifies an Oracle Forms PopList object by its handler name.
nca.listOfValues	Identifies an Oracle Forms List Of Values object by its handler name.
nca.logon	Identifies an Oracle Forms Logon Dialog object by its handler name.
nca.menuParametersDialog	Identifies an Oracle Forms menuParametersDialog object by its recorded ID and handler name.
nca.popupHelp	Identifies an Oracle Forms popupHelp object by its recorded ID and handler name.

Table 5–2 (Cont.) List of FormsService Methods

Method	Description
nca.promptList	Identifies an Oracle Forms promptList object by its handler name.
nca.radioButton	Identifies an Oracle Forms RadioButton object by its handler name.
nca.registerProperty	Registers a property for any object.
nca.responseBox	Identifies an Oracle Forms responseBox object by its handler name.
nca.send	Sends an Oracle Forms Message to the Oracle Forms server.
nca.sendMessages	Sends raw Oracle Forms Messages to the Oracle Forms server.
nca.sendTerminal	Sends an Oracle Forms Terminal Message request to the Oracle Forms server.
nca.solve	Parses a value from the source of latest action and store it as a variable.
nca.statusBar	Identifies an Oracle Forms statusBar object by its recorded ID and handler name.
nca.tab	Identifies an Oracle Forms TabControl object by its handler name.
nca.tableBox	Identifies an Oracle Forms tableBox object by its recorded ID and handler name.
nca.textField	Creates a reference to a text field object in the Forms applet by handler name.
nca.timer	Identifies an Oracle Forms timer object by its handler name.
nca.tree	Identifies an Oracle Forms Tree object by its handler name.
nca.treeList	Identifies an Oracle Forms TreeList object by its handler name.
nca.unRegisterProperty	Unregister a property that was previously registered using `registerProperty`.
nca.verifyStatusBarText	Searchs the status bar contents for the specified text pattern and performs a verify only matching test.
nca.verifyText	Searches the text of selected source type for the specified text pattern performing a verify only text match.
nca.window	Identifies an Oracle Forms FormWindow object by its handler name.

The following sections provide detailed reference information for each method and enum in the FormsService Class of Forms Load Module Application Programming Interface.

nca.alertDialog

Identifies an Oracle Forms alertDialog object by its handler name.

Format

The nca.alertDialog method has the following command format(s):

nca.alertDialog(*handlerName*);

nca.alertDialog(*recid*, *handlerName*);

Command Parameters

recid
Optional. The ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the alertDialog. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTAlertDialog object.

Example

Performs an action on an Oracle Forms alertDialog object specified by its recorded ID and handler name.

```
nca.window(436, "GROUPS_DETAIL").activate("");
nca.alertDialog(437, "Forms").clickButton(2, "12");
nca.textField(438, "GROUPS_EXPENDITURE_GROUP_0").select(0, 4, 4, "1");
```

nca.application

Initializes the Oracle Forms Application specified by its handler name.

Format

The nca.application method has the following command format(s):

nca.application(*handlerName*);

nca.application(*recid*, *handlerName*);

Command Parameters

recid
Optional. The ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the Form Application. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTRunForm object.

Example

Performs an action on an Oracle Forms Application specified by its recorded ID and handler name.

```
nca.application(161, "Oracle Applications").sendMessage(
 "<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
   "mHandlerClassId=\"1\" mHandlerId=\"1\">\r\n" +
 <Property actionString=\"HEARTBEAT\" action=\"517\" " +
   "type=\"null\" />\r\n</Message>\r\n", "1");
```

nca.assertStatusBarText

Searchs the status bar contents for the specified text pattern.

If the test fails, always fail the script unless the status bar text test error recovery setting specifies a different action such as Warn or Ignore. If any part of the status bar text matches `textToAssert`, the assertion succeeds.

Format

The nca.assertStatusBarText method has the following command format(s):

nca.assertStatusBarText(*testName*, *textToAssert*, *matchOption*);

nca.assertStatusBarText(*recId*, *testName*, *textToVerify*, *matchOption*);

Command Parameters

recId
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

testName
a String specifying the test name.

textToVerify
a String specifying the Text to match on the page, or not match on the page if TextPresence is set as PassIfPresent.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a regular expression, wildcard, or exact match.

textToAssert
a String specifying the Text to match on the page, or not match on the page if TextPresence is set as PassIfPresent.

Throws

AbstractScriptException
on any other failure when attempting to assert the text.

MatchException
if the assertion fails.

Example

Verifies the specified text matches exactly.

```
nca.assertStatusBarText(161, "Assert StatusBar Text" +
 "text value", "FRM-40400: Transaction complete: " +
 "1 records applied and saved.", MatchOption.Exact);
```

nca.assertText

Searches the text of selected source type for the specified text pattern.

If the test fails, always fail the script unless the text matching test error recovery setting specifies a different action such as Warn or Ignore.

If any part of the source text matches `textToAssert`, the assertion succeeds.

Format

The nca.assertText method has the following command format(s):

nca.assertText(*testName*, *textToAssert*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

textToAssert
a String specifying the Text to match on the page, or not match on the page if TextPresence is set as PassIfPresent.

sourceType
a NcaSource enum specifying where to match the text, i.e. in the Object Details, Request and Response, or Status Bar text.

textPresence
a TestPresence enum specifying either PassIfPresent or FailIfPresent, depending on if you want the test to pass or fail if the text to match is present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a regular expression, wildcard, or exact match.

Throws

MatchException
if the assertion fails.

AbstractScriptException
on any other failure when attempting to assert the text.

Example

Adds Text Matching tests for Statu Bar text.

```
nca.assertText("myTextMatchingTest", "match this text string",
 NcaSource.StatusBarText, TextPresence.PassIfPresent, MatchOption.Exact);
```

nca.blockScroller

Identifies an Oracle Forms blockScroller object by its handler name.

Format

The nca.blockScroller method has the following command format(s):

nca.blockScroller(*handlerName*);

nca.blockScroller(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the blockScroller. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTBlockScroller object.

Example

Performs an action on an Oracle Forms blockScroller object specified by its recorded ID and handler name.

```
nca.textField(571, "PROJECT_OPTIONS_OPTION_NAME_DISP_5")
 .select(0, 0, 0, "");
nca.textField(572, "PROJECT_OPTIONS_OPTION_NAME_DISP_6")
 .setFocus("1");
nca.application(573, "Oracle Applications")
 .sendMessage("<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
   "mHandlerClassId=\"1\" mHandlerId=\"1\">\r\n" +
   <Property actionString=\"HEARTBEAT\" action=\"517\" " +
   " type=\"null\" />\r\n</Message>\r\n", "1");
nca.textField(574, "PROJECT_OPTIONS_OPTION_NAME_DISP_6")
 .select(0, 27, 27, "");
nca.blockScroller(575, "COMPONENT_ID_478").scrollTo(0, 8, "1");
nca.blockScroller(575, "COMPONENT_ID_478").scrollPageUp(0, 1);
nca.blockScroller(575, "COMPONENT_ID_478").scrollPageDown(0, 1);
nca.textField(576, "PROJECT_OPTIONS_OPTION_NAME_DISP_7")
 .select(0, 18, 18, "");
nca.textField(577, "PROJECT_OPTIONS_OPTION_NAME_DISP_6")
 .clearFocus("");
nca.textField(578, "PROJECT_OPTIONS_OPTION_NAME_DISP_6")
 .select(0, 0, 0, "");
nca.textField(579, "PROJECT_OPTIONS_OPTION_NAME_DISP_7")
 .setFocus("");
```

nca.button

Identifies an Oracle Forms Button object by its handler name.

Format

The nca.button method has the following command format(s):

nca.button(*handlerName*);

nca.button(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the Button. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTButton object.

Example

Performs an action on an Oracle Forms Button object specified by its recorded ID and handler name.

```
nca.button(130, "FOLDER_QF_NEW_HEADER_0").setFocus("1");
nca.button(131, "FOLDER_QF_NEW_HEADER_0").click();
```

nca.cancelQueryDialog

Identifies an Oracle Forms cancelQueryDialog object by its recorded ID and handler name.

Format

The nca.cancelQueryDialog method has the following command format(s):

nca.cancelQueryDialog(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the cancelQueryDialog. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTCancelQueryDialog object.

Example

Identifies an Oracle Forms cancelQueryDialog object specified by its recorded ID and handler name.

```
nca.cancelQueryDialog(169, "cancelQueryDialog");
```

nca.canvas

Identifies an Oracle Forms canvas object by its recorded ID and handler name.

Format

The nca.canvas method has the following command format(s):

nca.canvas(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the canvas. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTCanvas object.

Example

Identifies an Oracle Forms canvas object specified by its recorded ID and handler name.

```
nca.canvas(169, "canvas");
```

nca.cfmOLE

Identifies an Oracle Forms cfmOLE object by its recorded ID and handler name.

Format

The nca.cfmOLE method has the following command format(s):

nca.cfmOLE(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the cfmOLE. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTCfmOLE object.

Example

Identifies an Oracle Forms cfmOLE object specified by its recorded ID and handler name.

```
nca.cfmOLE(169, "cfmole");
```

nca.cfmVBX

Identifies an Oracle Forms cfmVBX object by its recorded ID and handler name.

Format

The nca.cfmVBX method has the following command format(s):

nca.cfmVBX(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the cfmVBX. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTCfmVBX object.

Example

Identifies an Oracle Forms cfmVBX object specified by its recorded ID and handler name.

```
nca.cfmVBX(169, "cfmVBX");
```

nca.checkBox

Identifies an Oracle Forms CheckBox object by its handler name.

Format

The nca.checkBox method has the following command format(s):

nca.checkBox(*handlerName*);

nca.checkBox(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the CheckBox. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTCheckBox object.

Example

Performs an action on an Oracle Forms CheckBox object specified by its recorded ID and handler name.

```
nca.checkBox(289, "GROUPS_ALLOW_NEGATIVE_TR_0").setFocus();
nca.checkBox(291, "GROUPS_ALLOW_NEGATIVE_TR_0").check(true);
nca.checkBox(297, "GROUPS_ALLOW_NEGATIVE_TR_0").clearFocus();
```

nca.choiceBox

Identifies an Oracle Forms choiceBox object by its handler name.

Format

The nca.choiceBox method has the following command format(s):

nca.choiceBox(*handlerName*);

nca.choiceBox(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the choiceBox. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTChoiceBoxMessenger object.

Example

Performs an action on an Oracle Forms choiceBox object specified by its recorded ID and handler name.

```
nca.button(264, "GROUPS_SUBMIT_0").click("1");
nca.button(265, "GROUPS_SUBMIT_0").sendMessage(
 "<Message mActionString=\"MSG_GET\" mActionCode=\"4\" " +
   "mHandlerClassId=\"261\" mHandlerId=\"289\">\r\n" +
   <Property actionString=\"VISIBLERECT\" action=\"155\" " +
   "type=\"java.awt.Rectangle\" topleftx=\"0.0\" " +
   toplefty=\"0.0\" width=\"124.0\" height=\"28.0\"/>\r\n" +
 </Message>\r\n", "3");
nca.textField(266, "GROUPS_OPERATING_UNIT_0").select(0, 17, 17, "");
nca.button(267, "GROUPS_SUBMIT_0").clearFocus("");
nca.textField(268, "GROUPS_OPERATING_UNIT_0").setFocus("");
nca.window(269, "GROUPS_DETAIL").deactivate("11");
String alertMsg=nca.choiceBox(270, "Error").getAlertMessage();
info("alert message:"+alertMsg);
nca.choiceBox(270, "Error").clickButton(0);
nca.textField(271, "GROUPS_OPERATING_UNIT_0").select(0, 17, 17, "");
```

nca.comboBox

Identifies an Oracle Forms comboBox object by its recorded ID and handler name.

Format

The nca.comboBox method has the following command format(s):

nca.comboBox(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the comboBox. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTComboBox object.

Example

Identifies an Oracle Forms comboBox object specified by its recorded ID and handler name.

```
nca.comboBox(169, "comboBox");
```

nca.connect

Connects to an Oracle Forms server.

If already connected to an Oracle Forms server, this method quietly returns.

Format

The nca.connect method has the following command format(s):

nca.connect(*connMode*, *formsServerHost*, *serverPort*, *formsUrl*);

nca.connect(*recId*, *connMode*, *formsServerHost*, *serverPort*, *formsUrl*);

Command Parameters

recId
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

connMode
a ConnectMode enum specifing if the Forms server expects to use Socket or HTTP connection mode.

formsServerHost
a String specfying the server host. For `ConnectMode.Socket`, specify the host name or IP of the Oracle Forms server to connect to. For `ConnectMode.HTTP`, this parameter is ignored and should be `null`.

serverPort
a value specifying the server port. For `ConnectMode.Socket`, specify the port number on the Oracle Forms server to connect to. For `ConnectMode.HTTP`, this parameter is ignored and should be `0`.

formsUrl
a String specifying the URL. For `ConnectMode.HTTP`, specify the Oracle Forms URL to which the Forms applet posts messages. This URL is specified in the Oracle Forms Applet loading web page. The Oracle Forms Recorder automatically records this URL and correlates it appropriately. For `ConnectMode.Socket`, this parameter is ignored and should be `null`.

Throws

AbstractScriptException
on a failure to connect to the Oracle Forms server.

Example

Connects to the specified Oracle Forms server.

```
nca.connect(90, ConnectMode.HTTP, null, 0, "http://myserver.com:8002
{{formsload.url,/forms/lservlet;jsessionid=" +
8c57166455fa6173d468e0ae4e09966886262ac3b5b3.e38ObheNbx4Ne38Qe0}}");
```

nca.disconnect

Disconnects from Oracle Forms Server.

If not connected to the Forms server, this method quietly returns.

IMPORTANT: No matter how Forms statements are programmed, `nca.disconnect()` must be the last statement to disconnect from the Forms server. In mulitple iterations if `nca.connect()` is called in each iteration, `nca.disconnect()` must be called before the end of each iteration.

Format

The nca.disconnect method has the following command format(s):

```
nca.disconnect( );
```

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Disconnects from Oracle Forms Server.

```
nca.disconnect();
```

nca.displayErrorDialog

Identifies an Oracle Forms displayErrorDialog object by its recorded ID and handler name.

Format

The nca.displayErrorDialog method has the following command format(s):

nca.displayErrorDialog(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the displayErrorDialog. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTDisplayErrorDialog object.

Example

Identifies an Oracle Forms displayErrorDialog object specified by its recorded ID and handler name.

```
nca.comboBox(169, "comboBox");
```

nca.displayList

Identifies an Oracle Forms displayList object by its recorded ID and handler name.

Format

The nca.displayList method has the following command format(s):

nca.displayList(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the displayList. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTDisplayList object.

Example

Identifies an Oracle Forms displayList object specified by its recorded ID and handler name.

```
nca.displayList(169, "displayList");
```

nca.editBox

Identifies an Oracle Forms editBox object by its handler name.

Format

The nca.editBox method has the following command format(s):

nca.editBox(*handlerName*);

nca.editBox(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the editBox. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTEditBoxMessenger object.

Example

Performs an action on an Oracle Forms editBox object specified by its recorded ID and handler name.

```
nca.window(123, "GROUPS_DETAIL").activate("21");
nca.textField(124, "GROUPS_OPERATING_UNIT_0").select(0, 17, 17, "");
nca.window(125, "GROUPS_DETAIL").selectMenu(
 "HELP.ABOUT_ORACLE_APPLICATIONS", "1");
nca.window(126, "GROUPS_DETAIL").deactivate("1");
nca.editBox(127, "About Oracle Applications").clickOk("");
nca.textField(128, "GROUPS_OPERATING_UNIT_0").select(0, 17, 17, "");
```

nca.editorDialog

Identifies an Oracle Forms editorDialog object by its recorded ID and handler name.

Format

The nca.editorDialog method has the following command format(s):

nca.editorDialog(*handlerName*);

nca.editorDialog(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the editorDialog. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTEditorDialog object.

Example

Performs an action on an Oracle Forms editorDialog object specified by its recorded ID and handler name.

```
nca.textField(485, "GROUPS_EXPENDITURE_GROUP_0")
 .sendMessage("<Message mActionString=\"MSG_GET\" " +
   "mActionCode=\"4\" mHandlerClassId=\"257\" mHandlerId=\"272\">\r\n" +
   "<Property actionString=\"VISIBLERECT\" action=\"155\" " +
   "type=\"java.awt.Rectangle\" topleftx=\"0.0\" toplefty=\"0.0\" "
   "width=\"173.0\" height=\"24.0\"/>\r\n</Message>\r\n", "3");
nca.window(486, "GROUPS_DETAIL").selectMenu("EDIT.EDIT_FIELD", "1");
nca.window(487, "GROUPS_DETAIL").deactivate("1");
nca.editorDialog(488, "Editor").setText("ABCD");
nca.textField(489, "GROUPS_EXPENDITURE_GROUP_0").setCursorPosition(4, "");
```

nca.flexWindow

Identifies an Oracle Forms flexWindow object by its handler name.

Format

The nca.flexWindow method has the following command format(s):

nca.flexWindow(*handlerName*);

nca.flexWindow(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the flexWindow. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTFlexWindowMessenger object.

Example

Performs an action on an Oracle Forms flexWindow object specified by its recorded ID and handler name.

```
nca.textField(141, "WORK_ORDER_PARAMETERS_0").setFocus("2");
nca.listOfValues(142, "Reporting Level").displayRows(1, 2, "1");
nca.listOfValues(143, "Reporting Level").selectByIndex(1, "1");
nca.application(144, "Oracle Applications").sendMessage(
 "<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
   "mHandlerClassId=\"1\" mHandlerId=\"1\">\r\n" +
   "<Property actionString=\"HEARTBEAT\" action=\"517\" " +
   "type=\"null\" />\r\n</Message>\r\n","1");
nca.flexWindow(145, "Parameters").clickLOV(1, 0, "122");
nca.listOfValues(146, "Reporting Context").displayRows(1, 11, "1");
nca.listOfValues(147, "Reporting Context").clickCancel("1");
nca.flexWindow(148, "Parameters").focusField(2, 0, "1");
nca.flexWindow(149, "Services Accounting Flex").clickCancel("1");
nca.flexWindow(150, "Parameters").focusField(3, 0, "1");
nca.flexWindow(151, "Services Accounting Flex").clickClear("1");
```

nca.genericClient

Identifies an Oracle Forms genericClient object by its handler name.

Format

The nca.genericClient method has the following command format(s):

nca.genericClient(*handlerName*);

nca.genericClient(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the genericClient. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTGenericClientMessenger object.

Example

Performs an action on an Oracle Forms genericClient object specified by its recorded ID and handler name.

```
nca.genericClient(96, "COMPONENT_ID_54").sendGenericClientMessage();
nca.genericClient(96, "COMPONENT_ID_54").sendGenericClientMessage("", "1");
```

nca.getLastKnownContents

Returns the string version of contents for the specified ContentSource that was last retrieved for this virtual user.

Format

The nca.getLastKnownContents method has the following command format(s):

nca.getLastKnownContents(*sourceType*);

Command Parameters

sourceType
an NcaSource Enum specifying the portion of the contents to return as a String. If `null`, assumes NcaSource.ObjectDetails.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

String version of the contents last retrieved by the client. Returns null if contents are null.

Example

Gets the specified last known contents.

```
String details = nca.getLastKnownContents(NcaSource.ObjectDetails);
info("Object Details " + details);
String reqresp = nca.getLastKnownContents(NcaSource.RequestsAndResponses);
info("Requests and Responses " + reqresp);
String statusbar = nca.getLastKnownContents(NcaSource.StatusBarText);
info("StatusBarText " + statusbar);
```

nca.getStatusBarText

Gets the contents of the last known status bar text.

This is useful when user wants to grab the last known value of the status bar within the script.

Format

The nca.getStatusBarText method has the following command format(s):

nca.getStatusBarText( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

String value for the last known text value of the status bar. If the server has never updated the status bar, this method returns an empty string. This method never returns `null`.

Example

Gets the contents of the last known status bar text.

```
String statusBarText1=nca.getStatusBarText();
info("exist status bar text:"+statusBarText1);
```

nca.helpDialog

Identifies an Oracle Forms helpDialog object by its handler name.

Format

The nca.helpDialog method has the following command format(s):

nca.helpDialog(*handlerName*);

nca.helpDialog(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the helpDialog. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTHelpDialog object.

Example

Performs an action on an Oracle Forms helpDialog object specified by its recorded ID and handler name.

```
nca.window(121, "GROUPS_DETAIL").activate("");
nca.textField(122, "GROUPS_OPERATING_UNIT_0").select(0, 17, 17, "");
nca.helpDialog(123, "Keys").clickCancel("1");
```

nca.image

Identifies an Oracle Forms image object by its recorded ID and handler name.

Format

The nca.image method has the following command format(s):

nca.image(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the image. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTImage object.

Example

Identifies an Oracle Forms image object specified by its recorded ID and handler name.

```
nca.image(169, "image");
```

nca.infoBox

Identifies an Oracle Forms infoBox object by its handler name.

Format

The nca.infoBox method has the following command format(s):

nca.infoBox(*handlerName*);

nca.infoBox(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the infoBox. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTInfoBoxMessenger object.

Example

Performs an action on an Oracle Forms infoBox object specified by its recorded ID and handler name.

```
nca.window(1033, "GROUPS_DETAIL").activate("1");
nca.window(1034, "GROUPS_DETAIL").selectMenu("HELP.RECORD_HISTORY",
 "1");
nca.window(1035, "GROUPS_DETAIL").deactivate("1");
nca.infoBox(1036, "About This Record").clickOk();
```

nca.jContainer

Identifies an Oracle Forms jContainer object by its handler name.

Format

The nca.jContainer method has the following command format(s):

nca.jContainer(*handlerName*);

nca.jContainer(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the jContainer. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTJavaContainer object.

Example

Performs an action on an Oracle Forms jContainer object specified by its recorded ID and handler name.

```
nca.window(9078, "FOLDER_INCIDENT_TRACKING").activate("2");
nca.textField(9079, "CREATE_TASK_PLANNED_END_DATE_UI_0")
 .select(0, 20, 20, "");
nca.button(9080, "CALENDAR_OK_0").clearFocus("");
nca.textField(9081, "CREATE_TASK_PLANNED_END_DATE_UI_0")
 .setFocus("1");
nca.window(9082, "Oracle Applications - Main Window")
 .resize(1364, 768, "1");
nca.textField(9083, "CREATE_TASK_PLANNED_END_DATE_UI_0")
 .clearFocus("");
nca.jContainer(9084, "SR_TASK_GRID_GRID_ITEM_0")
 .setFocus("");
nca.jContainer(9085, "SR_TASK_GRID_GRID_ITEM_0")
 .sendMessage("<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
   "mHandlerClassId=\"271\" mHandlerId=\"496\">\r\n" +
   "<Property actionString=\"EVENT\" action=\"398\" " +
   "type=\"oracle.forms.engine.Message\" >\r\n" +
   "<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
   "mHandlerClassId=\"271\" mHandlerId=\"496\">\r\n" +
   "<Property actionString=\"EVENTNAME\" action=\"399\" " +
   "type=\"java.lang.String\" value=\"focusChanged\"/>\r\n" +
   "<Property actionString=\"EVENT_ARGNAME\" action=\"400\" " +
   "type=\"java.lang.String\" value=\"currentColumn\"/>\r\n" +
```

```
   "<Property actionString=\"EVENT_ARGVALUE\" action=\"401\" " +
   "type=\"java.lang.String\" value=\" \"/>\r\n" +
   "<Property actionString=\"EVENT_ARGNAME\" action=\"400\" " +
   "type=\"java.lang.String\" value=\"currentRow\"/>\r\n" +
   "<Property actionString=\"EVENT_ARGVALUE\" action=\"401\" " +
   "type=\"java.lang.String\" value=\"2\"/>\r\n" +
   </Message>\r\n</Property>\r\n</Message>\r\n", "12");
nca.timer(9086, "COMPONENT_ID_1216").expired("1");
nca.window(9087, "FOLDER_INCIDENT_TRACKING").deactivate("1");
```

nca.list

Identifies an Oracle Forms PopList object by its handler name.

Format

The nca.list method has the following command format(s):

nca.list(*handlerName*);

nca.list(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the PopList. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTPopListItem object.

Example

Performs an action on an Oracle Forms PopList object specified by its recorded ID and handler name.

```
nca.list(120, "GROUPS_SYSTEM_LINKAGE_FUNCTION_0")
 .setFocus();
nca.list(121, "GROUPS_SYSTEM_LINKAGE_FUNCTION_0")
 .sendMessage("<Message mActionString=\"MSG_UPDATE\" " +
   "mActionCode=\"2\" mHandlerClassId=\"263\" " +
   "mHandlerId=\"274\">\r\n<Property actionString=\"MOUSEDOWN\" " +
   "action=\"180\" type=\"oracle.forms.engine.Message\" >\r\n" +
 "<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
   "mHandlerClassId=\"263\" mHandlerId=\"274\">\r\n" +
 "<Property actionString=\"MOUSEDOWN_POS\" action=\"185\" " +
   "type=\"java.awt.Point\" pointx=\"185.0\" " +
   "pointy=\"11.0\"/>\r\n" +
 "<Property actionString=\"MOUSEDOWN_MOD\" action=\"186\" " +
   "type=\"java.lang.Byte\" value=\"16\"/>\r\n" +
 "</Message>\r\n</Property>\r\n</Message>\r\n", "1");
```

nca.listOfValues

Identifies an Oracle Forms List Of Values object by its handler name.

Format

The nca.listOfValues method has the following command format(s):

nca.listOfValues(*handlerName*);

nca.listOfValues(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the List Of Values. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTListValuesDialog object.

Example

Performs an action on an Oracle Forms List Of Values object specified by its recorded ID and handler name.

```
nca.listOfValues(133, "Customers").searchByText("cus", "1");
nca.listOfValues(134, "Customers").displayRows(1, 1, "1");
```

nca.logon

Identifies an Oracle Forms Logon Dialog object by its handler name.

Format

The nca.logon method has the following command format(s):

nca.logon(*handlerName*);

nca.logon(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the Logon Dialog. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTLogonDialog object.

Example

Performs an action on an Oracle Forms Logon Dialog object specified by its recorded ID and handler name.

```
nca.logon(133, "Logon").connect("username", "password", "database");
```

nca.menuParametersDialog

Identifies an Oracle Forms menuParametersDialog object by its recorded ID and handler name.

Format

The nca.menuParametersDialog method has the following command format(s):

nca.menuParametersDialog(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the menuParametersDialog. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTMenuParametersDialog object.

Example

Identifies an Oracle Forms menuParametersDialog object specified by its recorded ID and handler name.

```
nca.menuParametersDialog(169, "menu");
```

NcaSource

The NcaSource has the following values:

Table 5–3 List of NcaSource Values

Value	Description
ObjectDetails	Retrieve the contents of Object Details.
StatusBarText	Retrieve the contents of StatusBarText.
RequestsAndResponses	Retrieve the contents of the last sent and received request/response pairs.

nca.popupHelp

Identifies an Oracle Forms popupHelp object by its recorded ID and handler name.

Format

The nca.popupHelp method has the following command format(s):

nca.popupHelp(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the popupHelp. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTPopupHelp object.

Example

Identifies an Oracle Forms popupHelp object specified by its recorded ID and handler name.

```
nca.popupHelp(169, "popupHelp");
```

nca.promptList

Identifies an Oracle Forms promptList object by its handler name.

Format

The nca.promptList method has the following command format(s):

nca.promptList(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the tpromptList. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTPromptList object.

Example

Identifies an Oracle Forms promptList object specified by its recorded ID and handler name.

```
nca.promptList(172, "promptList");
```

nca.radioButton

Identifies an Oracle Forms RadioButton object by its handler name.

Format

The nca.radioButton method has the following command format(s):

nca.radioButton(*handlerName*);

nca.radioButton(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the RadioButton. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTRadioButton object.

Example

Performs an action on an Oracle Forms RadioButton object specified by its recorded ID and handler name.

```
nca.radioButton(724, "JOBS_QF_WHICH_JOBS_ALL_MY_JOBS_0").setFocus("1");
nca.radioButton(725, "JOBS_QF_WHICH_JOBS_ALL_MY_JOBS_0")
 .sendMessage("<Message mActionString=\"MSG_UPDATE\" " +
   "mActionCode=\"2\" mHandlerClassId=\"267\" " +
   "mHandlerId=\"375\">\r\n<Property actionString=\"MOUSEUP\" " +
   "action=\"181\" type=\"oracle.forms.engine.Message\" >\r\n" +
   "<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
   "mHandlerClassId=\"267\" mHandlerId=\"375\">\r\n" +
   "<Property actionString=\"MOUSEDOWN_POS\" action=\"185\" " +
   "type=\"java.awt.Point\" pointx=\"1.0\" pointy=\"8.0\"/>\r\n" +
   "<Property actionString=\"MOUSEDOWN_MOD\" action=\"186\" " +
   "type=\"java.lang.Byte\" value=\"16\"/>\r\n" +
   "</Message>\r\n</Property>\r\n</Message>\r\n", "");
nca.radioButton(726, "JOBS_QF_WHICH_JOBS_ALL_MY_JOBS_0").select("");
nca.radioButton(727, "JOBS_QF_WHICH_JOBS_ALL_MY_JOBS_0").unselect("");
```

nca.registerProperty

Registers a property for any object.

The property will be sent to the server in subsequent client request(s) in the following cases:

- If the server asks the client for a specific property, the client responds with the property using a `MSG_GET` type message.
- Whenever a client sends a `MSG_UPDATE` type message for an object, the client will send any registered properties in the message.

If a property is already registered, this method will overwrite its existing value.

If a property does not exist, this method will add the property.

Format

The nca.registerProperty method has the following command format(s):

nca.registerProperty(*name*, *value*, *handlerId*, *options*);

Command Parameters

name
a String specifying the name of the property to be registered.

value
a Property value in its Object form. The type of the object is inferred automatically when the property value is submitted to the server. For example, if a boolean `true` value is specified, then the property will be submitted to the server using type "java.lang.boolean".

handlerId
the specific object for which this property should be registered. The object does NOT have to exist at the time this method is invoked.

options
a RegisterPropertyOptions enum specifying the scenario in which to register the custom property, such as only when sending a `MSG_GET message`, or when sending a `MSG_UPDATE` message.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

NullPointerException
if the name is `null`.

Example

Example 1 Sends a WINSYS_REQUIREDVA_LIST property when the application is initialized.

Example 2 Registers a blank custom TITLE property for a window in case the server asks for the title later in the script.

```
//Example 1
nca.registerProperty("WINSYS_REQUIREDVA_LIST", "true", 1,
 RegisterPropertyOptions.RegisterForMSG_UPDATE);
nca.application("ORACLE_APPLICATIONS").initialize(...);
//Example 2
nca.registerProperty("TITLE", "", 6,
 RegisterPropertyOptions.RegisterForMSG_GET);
```

nca.responseBox

Identifies an Oracle Forms responseBox object by its handler name.

Format

The nca.responseBox method has the following command format(s):

nca.responseBox(*handlerName*);

nca.responseBox(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the responseBox. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTResponseBoxMessenger object.

Example

Performs an action on an Oracle Forms responseBox object specified by its recorded ID and handler name.

```
nca.window(263, "GROUPS_DETAIL").activate("2");
nca.textField(264, "GROUPS_OPERATING_UNIT_0").select(0, 17, 17, "");
nca.textField(265, "GROUPS_OPERATING_UNIT_0").setFocus("1");
nca.textField(266, "GROUPS_OPERATING_UNIT_0").select(0, 17, 17, "");
nca.window(267, "GROUPS_DETAIL").selectMenu(
 "PREFERENCES.CHANGE_PASSWORD", "1");
nca.textField(268, "GROUPS_OPERATING_UNIT_0").clearFocus("");
nca.window(269, "GROUPS_DETAIL").deactivate("1");
nca.responseBox(270, "Password Update").validate("F", 0, "sdfas,,",
 "1");
nca.responseBox(271, "Password Update").validate("F", 1,
 "sdfas,as,", "1");
nca.responseBox(272, "Password Update").validate("F", 2,
 "sdfas,as,as", "1");
nca.responseBox(273, "Password Update").validate("O", 2,
 "sdfas,as,as", "");
nca.responseBox(273, "Password Update").clickOk("", 1);
```

nca.send

Sends an Oracle Forms Message to the Oracle Forms server.

Format

The nca.send method has the following command format(s):

nca.send(*strMessage*);

nca.send(*recId*, *strMessage*);

Command Parameters

recId
Optionally specify a recorded element ID. Only used for comparison purposes in the results. May be `null`.

strMessage
a String specifying the Oracle Forms Message object in XML format.

Throws

AbstractScriptException
on any exception while sending the message to the Oracle Forms server.

Example

Sends the specified Oracle Forms Message.

```
nca.send(20,
 "<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
 "mHandlerClassId=\"1\" mHandlerId=\"1\">\r\n" +
 "<Property actionString=\"INITIAL_VERSION\" action=\"268\" " +
 "type=\"java.lang.Integer\" value=\"1012000\"/>\r\n" +
 "<Property actionString=\"INITIAL_RESOLUTION\" action=\"263\" " +
 "type=\"java.awt.Point\" pointx=\"96.0\" pointy=\"96.0\"/>\r\n" +
 "<Property actionString=\"INITIAL_DISP_SIZE\" action=\"264\" " +
 "type=\"java.awt.Point\" pointx=\"1364.0\" pointy=\"768.0\"/>\r\n" +
 "<Property actionString=\"INITIAL_CMDLINE\" action=\"265\" " +
 "type=\"java.lang.String\" value=\"server module=" +
 "{{formsload.module_1,/d1/oracle01/EBS/VIS/apps/apps_st/" +
 "appl/fnd/12.0.0/forms/US/FNDSCSGN}} fndnam=APPS record=names  " +
 "config='{{formsload.config_1,VIS}}' " +
 "icx_ticket='{{formsload.icx_ticket_11i_3,.jACpSETjvCOkvXLRX3StGA..}}' " +
 "resp='ONT/ORDER_MGMT_SUPER_USER' secgrp='STANDARD' " +
 "start_func='ONT_OEXOEORD_SUMMARY' other_params=''\"/>\r\n" +
 "<Property actionString=\"INITIAL_COLOR_DEPTH\" action=\"266\" " +
 "type=\"java.lang.Integer\" value=\"256\"/>\r\n" +
 "<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
 "type=\"java.lang.Integer\" value=\"0\"/>\r\n" +
 "<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
 "type=\"java.lang.Integer\" value=\"255\"/>\r\n" +
 "<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
 "type=\"java.lang.Integer\" value=\"65535\"/>\r\n" +
 "<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
 "type=\"java.lang.Integer\" value=\"4210752\"/>\r\n" +
 "<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
```

```
"type=\"java.lang.Integer\" value=\"8421504\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"65280\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"12632256\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16711935\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16762880\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16756655\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16711680\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16777215\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16776960\"/>\r\n" +
"<Property actionString=\"FONT_NAME\" action=\"383\" " +
"type=\"java.lang.String\" value=\"dialog\"/>\r\n" +
"<Property actionString=\"FONT_SIZE\" action=\"377\" " +
"type=\"java.lang.Integer\" value=\"900\"/>\r\n" +
"<Property actionString=\"FONT_STYLE\" action=\"378\" " +
"type=\"java.lang.Byte\" value=\"0\"/>\r\n" +
"<Property actionString=\"FONT_WEIGHT\" action=\"379\" " +
"type=\"java.lang.Byte\" value=\"0\"/>\r\n" +
"<Property actionString=\"INITIAL_SCALE_INFO\" action=\"267\" " +
"type=\"java.awt.Point\" pointx=\"7.0\" pointy=\"21.0\"/>\r\n" +
"<Property actionString=\"WINSYS_REQUIREDVA_LIST\" action=\"291\" " +
"type=\"java.lang.Boolean\" value=\"true\"/>\r\n" +
"<Property actionString=\"SERVER_USER_PARAMS\" action=\"510\" " +
"type=\"java.lang.String\" value=\"NLS_LANG='AMERICAN_AMERICA' " +
 "FORMS_USER_DATE_FORMAT='DD-MON-RRRR' " +
 "FORMS_USER_DATETIME_FORMAT='DD-MON-RRRR HH24:MI:SS' " +
 "NLS_DATE_LANGUAGE='AMERICAN' NLS_SORT='BINARY' " +
 "NLS_NUMERIC_CHARACTERS='.,'\"/>\r\n" +
"<Property actionString=\"DEFAULT_LOCAL_TZ\" action=\"530\" " +
"type=\"java.lang.String\" value=\"Asia/Shanghai\"/>\r\n" +
"</Message>\r\n");
```

nca.sendMessages

Sends raw Oracle Forms Messages to the Oracle Forms server.

Format

The nca.sendMessages method has the following command format(s):

nca.sendMessages(*description*, *strMessages*);

nca.sendMessages(*recId*, *description*, *strMessages*);

Command Parameters

recId
Optionally specify a recorded element ID. Only used for comparison purposes in the results. May be `null`.

description
Optional a String specifying a human-readable description explaining what this message does. May be `null`.

strMessages
String containing Oracle Forms Messages in XML format.

Throws

AbstractScriptException
on any exception while sending the message to the Oracle Forms server.

Example

Sends the specified Oracle Forms Messages.

```
nca.sendMessages(20, "Message Description"
 "<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
 "mHandlerClassId=\"1\" mHandlerId=\"1\">\r\n" +
 "<Property actionString=\"INITIAL_VERSION\" action=\"268\" " +
 "type=\"java.lang.Integer\" value=\"1012000\"/>\r\n" +
 "<Property actionString=\"INITIAL_RESOLUTION\" action=\"263\" " +
 "type=\"java.awt.Point\" pointx=\"96.0\" pointy=\"96.0\"/>\r\n" +
 "<Property actionString=\"INITIAL_DISP_SIZE\" action=\"264\" " +
 "type=\"java.awt.Point\" pointx=\"1364.0\" pointy=\"768.0\"/>\r\n" +
 "<Property actionString=\"INITIAL_CMDLINE\" action=\"265\" " +
 "type=\"java.lang.String\" value=\"server module=" +
 "{{formsload.module_1,/d1/oracle01/EBS/VIS/apps/apps_st/" +
 "appl/fnd/12.0.0/forms/US/FNDSCSGN}} fndnam=APPS record=names  " +
 "config='{{formsload.config_1,VIS}}' " +
 "icx_ticket='{{formsload.icx_ticket_11i_3,.jACpSETjvCOkvXLRX3StGA..}}' " +
 "resp='ONT/ORDER_MGMT_SUPER_USER' secgrp='STANDARD' " +
 "start_func='ONT_OEXOEORD_SUMMARY' other_params=''\"/>\r\n" +
 "<Property actionString=\"INITIAL_COLOR_DEPTH\" action=\"266\" " +
 "type=\"java.lang.Integer\" value=\"256\"/>\r\n" +
 "<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
 "type=\"java.lang.Integer\" value=\"0\"/>\r\n" +
 "<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
 "type=\"java.lang.Integer\" value=\"255\"/>\r\n" +
 "<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
```

```
"type=\"java.lang.Integer\" value=\"65535\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"4210752\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"8421504\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"65280\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"12632256\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16711935\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16762880\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16756655\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16711680\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16777215\"/>\r\n" +
"<Property actionString=\"WINSYS_COLOR_ADD\" action=\"284\" " +
"type=\"java.lang.Integer\" value=\"16776960\"/>\r\n" +
"<Property actionString=\"FONT_NAME\" action=\"383\" " +
"type=\"java.lang.String\" value=\"dialog\"/>\r\n" +
"<Property actionString=\"FONT_SIZE\" action=\"377\" " +
"type=\"java.lang.Integer\" value=\"900\"/>\r\n" +
"<Property actionString=\"FONT_STYLE\" action=\"378\" " +
"type=\"java.lang.Byte\" value=\"0\"/>\r\n" +
"<Property actionString=\"FONT_WEIGHT\" action=\"379\" " +
"type=\"java.lang.Byte\" value=\"0\"/>\r\n" +
"<Property actionString=\"INITIAL_SCALE_INFO\" action=\"267\" " +
"type=\"java.awt.Point\" pointx=\"7.0\" pointy=\"21.0\"/>\r\n" +
"<Property actionString=\"WINSYS_REQUIREDVA_LIST\" action=\"291\" " +
"type=\"java.lang.Boolean\" value=\"true\"/>\r\n" +
"<Property actionString=\"SERVER_USER_PARAMS\" action=\"510\" " +
"type=\"java.lang.String\" value=\"NLS_LANG='AMERICAN_AMERICA' " +
 "FORMS_USER_DATE_FORMAT='DD-MON-RRRR' " +
 "FORMS_USER_DATETIME_FORMAT='DD-MON-RRRR HH24:MI:SS' " +
 "NLS_DATE_LANGUAGE='AMERICAN' NLS_SORT='BINARY' " +
 "NLS_NUMERIC_CHARACTERS='.,'\"/>\r\n" +
"<Property actionString=\"DEFAULT_LOCAL_TZ\" action=\"530\" " +
"type=\"java.lang.String\" value=\"Asia/Shanghai\"/>\r\n" +
"</Message>\r\n");
```

nca.sendTerminal

Sends an Oracle Forms Terminal Message request to the Oracle Forms server.

Format

The nca.sendTerminal method has the following command format(s):

nca.sendTerminal(*responseCode*);

nca.sendTerminal(*recId*, *responseCode*);

Command Parameters

recId
Optionally specify a recorded element ID. Only used for comparison purposes in the results. May be `null`.

responseCode
Specify the type of terminal message to send, i.e. `1`, `2`, or `3`.

Throws

AbstractScriptException
on any exception while sending the terminal message to the Oracle Forms server.

Example

Sends an Oracle Forms Terminal Message `1`.

```
nca.sendTerminal(90, 1);
```

nca.solve

Parses a value from the source of latest action and store it as a variable.

Format

The nca.solve method has the following command format(s):

nca.solve(*name*, *pattern*, *errorMessage*, *isOptional*, *source*, *resultIndex*, *encoding*);

Command Parameters

name
a String specifying the Name of the variable to create. Must not be `null`.

pattern
a String specifying the Regular expression specifying what to extract from the most recent navigation's contents. May contain a transform expression. Must not be `null`.

errorMessage
an optional String specifying the error message to display if the pattern cannot be solved. If `null`, a meaningful error message is automatically generated.

isOptional
a boolean specifying if pattern must be solved or not. Set to True if the pattern does not have to be solved. If the pattern cannot be solved, the method quietly returns.

source
an NcaSource enum specifying which contents to return as a String.

resultIndex
an index value that specifies the 0-based index of the specific result to retrieve if the pattern matches more than one value. If `null`, all results will be added into the variables collection.

encoding
an EncodeOptions enum specifing the encoding or decoding operation to perform on the variable's value after solving the variable. A `null` value is equivalent to `EncodeOptions.None`.

Throws

Exception
if the object is not found.

Example

Parse a value from the specified source using a Regular Expression and store it in a variable. An optional error message returns if the pattern cannot be solved. The pattern is optional and does not have to be solved. Includes a result index value of 0 to specify it should retrieve the first match result and ecoding or decode using the specified option.

```
nca.solve("varStatusBar0", "FRM-(.+): Transaction complete: (.+) records " +
 "applied and saved.", "Text not Found", false,
 NcaSource.StatusBarText, 0, EncodeOptions.None);
```

nca.statusBar

Identifies an Oracle Forms statusBar object by its recorded ID and handler name.

Format

The nca.statusBar method has the following command format(s):

nca.statusBar(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the statusBar. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTStatusBar object.

Example

Identifies an Oracle Forms statusBar object specified by its recorded ID and handler name.

```
nca.statusBar(173, "statusBar");
```

nca.tab

Identifies an Oracle Forms TabControl object by its handler name.

Format

The nca.tab method has the following command format(s):

nca.tab(*handlerName*);

nca.tab(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the TabControl. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTTabControl object.

Example

Performs an action on an Oracle Forms TabControl object specified by its recorded ID and handler name.

```
nca.tab(119, "SUMMARY_TABS").setFocus("");
nca.tab(120, "SUMMARY_TABS").selectByIndex(9, "1");
nca.tab(121, "SUMMARY_TABS").clearFocus("");
```

nca.tableBox

Identifies an Oracle Forms tableBox object by its recorded ID and handler name.

Format

The nca.tableBox method has the following command format(s):

nca.tableBox(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the tableBox. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTTableBox object.

Example

Identifies an Oracle Forms tableBox object specified by its recorded ID and handler name.

```
nca.tableBox(171, "tablebox");
```

nca.textField

Creates a reference to a text field object in the Forms applet by handler name.

The object is not actually found until performing some action against the object.

Format

The nca.textField method has the following command format(s):

nca.textField(*handlerName*);

nca.textField(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the text field. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a new TFormsLTTextField object.

Example

Performs an action on an Oracle Forms text field object specified by its recorded ID and handler name.

```
nca.window(307, "NAVIGATOR").activate("");
nca.window(308, "NAVIGATOR").setVisible(true, "");
nca.textField(325, "GROUPS_OPERATING_UNIT_0").setFocus();
String text=nca.textField(327, "GROUPS_OPERATING_UNIT_0").getText();
info("text: " + text);
nca.textField(327, "GROUPS_OPERATING_UNIT_0").clickButton();
```

nca.timer

Identifies an Oracle Forms timer object by its handler name.

Format

The nca.timer method has the following command format(s):

nca.timer(*handlerName*);

nca.timer(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the timer. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTApplicationTimer object.

Example

Performs an action on an Oracle Forms timer object specified by its recorded ID and handler name.

```
nca.window(9078, "FOLDER_INCIDENT_TRACKING").activate("2");
nca.textField(9079, "CREATE_TASK_PLANNED_END_DATE_UI_0")
 .select(0, 20, 20, "");
nca.button(9080, "CALENDAR_OK_0").clearFocus("");
nca.textField(9081, "CREATE_TASK_PLANNED_END_DATE_UI_0")
 .setFocus("1");
nca.window(9082, "Oracle Applications - Main Window")
 .resize(1364, 768, "1");
nca.textField(9083, "CREATE_TASK_PLANNED_END_DATE_UI_0")
 .clearFocus("");
nca.jContainer(9084, "SR_TASK_GRID_GRID_ITEM_0")
 .setFocus("");
nca.jContainer(9085, "SR_TASK_GRID_GRID_ITEM_0")
 .sendMessage("<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
   "mHandlerClassId=\"271\" mHandlerId=\"496\">\r\n" +
   "<Property actionString=\"EVENT\" action=\"398\" " +
   "type=\"oracle.forms.engine.Message\" >\r\n" +
   "<Message mActionString=\"MSG_UPDATE\" mActionCode=\"2\" " +
   "mHandlerClassId=\"271\" mHandlerId=\"496\">\r\n" +
   "<Property actionString=\"EVENTNAME\" action=\"399\" " +
   "type=\"java.lang.String\" value=\"focusChanged\"/>\r\n" +
   "<Property actionString=\"EVENT_ARGNAME\" action=\"400\" " +
   "type=\"java.lang.String\" value=\"currentColumn\"/>\r\n" +
```

```
   "<Property actionString=\"EVENT_ARGVALUE\" action=\"401\" " +
   "type=\"java.lang.String\" value=\" \"/>\r\n" +
   "<Property actionString=\"EVENT_ARGNAME\" action=\"400\" " +
   "type=\"java.lang.String\" value=\"currentRow\"/>\r\n" +
   "<Property actionString=\"EVENT_ARGVALUE\" action=\"401\" " +
   "type=\"java.lang.String\" value=\"2\"/>\r\n" +
   </Message>\r\n</Property>\r\n</Message>\r\n", "12");
nca.timer(9086, "COMPONENT_ID_1216").expired("1");
nca.window(9087, "FOLDER_INCIDENT_TRACKING").deactivate("1");
```

nca.tree

Identifies an Oracle Forms Tree object by its handler name.

Format

The nca.tree method has the following command format(s):

nca.tree(*handlerName*);

nca.tree(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the Tree. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTTreeItem object.

Example

Performs an action on an Oracle Forms Tree object specified by its recorded ID and handler name.

```
nca.tree(334, "APPTREE_NAV_TREE_NAVIGATOR_0").setFocus("");
nca.tree(335, "APPTREE_NAV_TREE_NAVIGATOR_0").expandNode(1, "1");
```

nca.treeList

Identifies an Oracle Forms TreeList object by its handler name.

Format

The nca.treeList method has the following command format(s):

nca.treeList(*handlerName*);

nca.treeList(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the TreeList. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTTListItem object.

Example

Performs an action on an Oracle Forms TreeList object specified by its recorded ID and handler name.

```
nca.treeList(124, "NAVIGATOR_LIST_0").setFocus("1");
nca.treeList(125, "NAVIGATOR_LIST_0").selectByIndex(1);
```

nca.unRegisterProperty

Unregister a property that was previously registered using `registerProperty`.

If the specified property does not exist, this method does not do anything.

Format

The nca.unRegisterProperty method has the following command format(s):

nca.unRegisterProperty(*name*, *handlerId*);

Command Parameters

name
a String specifying the name of the property previously registered using `registerProperty`.

handlerId
a value specifying the object for which this property was registered.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

NullPointerException
if the name is `null`.

Example

Unregisters the previously registered property.

```
nca.unRegisterProperty("TITLE", 6);
```

nca.verifyStatusBarText

Searchs the status bar contents for the specified text pattern and performs a verify only matching test.

If the test fails, report a warning.

This method will never cause a script to fail, regardless of the text matching test error recovery settings. Use nca.assertStatusBarText to make the script fail when the test fails.

When playing back a script in Oracle Load Testing, always use Assertions; failed Verifications are NOT reported in OLT. If any part of the status bar text matches `textToVerify`, the verification succeeds.

Format

The nca.verifyStatusBarText method has the following command format(s):

nca.verifyStatusBarText(*testName*, *textToVerify*, *matchOption*);

nca.verifyStatusBarText(*recId*, *testName*, *textToVerify*, *matchOption*);

Command Parameters

recId
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

testName
a String specifying the test name.

textToVerify
a String specifying the Text to match on the page, or not match on the page if TextPresence is set as PassIfPresent.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a regular expression, wildcard, or exact match.

Throws

AbstractScriptException
on any failure when attempting to verify the text.

Example

Verifies the specified text matches exactly.

```
nca.verifyStatusBarText(151, "Verify StatusBar Text" +
 "text value", "FRM-40400: Transaction complete: " +
 "1 records applied and saved.", MatchOption.Exact);
```

nca.verifyText

Searches the text of selected source type for the specified text pattern performing a verify only text match.

If the test fails, report a warning. This method will never cause a script to fail, regardless of the text matching test error recovery settings. Use nca.assertText to make the script fail when the test fails.

When playing back a script in Oracle Load Testing, always use Assertions; failed Verifications are NOT reported in OLT. If any part of the source text matches `textToVerify`, the verification succeeds.

Format

The nca.verifyText method has the following command format(s):

nca.verifyText(*testName*, *textToVerify*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

textToVerify
a String specifying the Text to match on the page, or not match on the page if TextPresence is set as PassIfPresent.

sourceType
a NcaSource enum specifying where to match the text, i.e. in the Object Details, Request and Response, or Status Bar text.

textPresence
a TestPresence enum specifying either PassIfPresent or FailIfPresent, depending on if you want the test to pass or fail if the text to match is present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a regular expression, wildcard, or exact match.

Throws

AbstractScriptException
on any failure when attempting to verify the text.

Example

Adds a Verify only, never fail, Text Matching tests for Statu Bar text.

```
nca.verifyText("myTextMatchingTest", "match this text string",
 NcaSource.StatusBarText, TextPresence.PassIfPresent, MatchOption.Exact);
```

nca.window

Identifies an Oracle Forms FormWindow object by its handler name.

Format

The nca.window method has the following command format(s):

nca.window(*handlerName*);

nca.window(*recid*, *handlerName*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

handlerName
a String specifying the handler name of the FormWindow. Must not be `null`.

Throws

Exception
if the object is not found.

Returns

a new TFormsLTFormWindow object.

Example

Performs an action on an Oracle Forms FormWindow object specified by its recorded ID and handler name.

```
nca.window(102, "NAVIGATOR").activate("");
nca.registerProperty("TITLE", "", 102, RegisterPropertyOptions.RegisterForMSG_
GET);
nca.unRegisterProperty("title", 102);
nca.textField(103, "NAVIGATOR_TYPE_0").setFocus("22");
nca.window(105, "NAVIGATOR").resize(586, 451, "22222");
nca.window(106, "FIND_ORDER_NUMBER_0").setCursorPosition(0, "");
nca.window(107, "CONFIGURATOR_PROCESSING").move(288, 192, "");
```

6

Web/HTTP Load Module

This chapter provides a complete listing and reference for the methods in the OpenScript HTTPService Class of HTTP Load Module Application Programming Interface (API).

6.1 HTTPService ENUM Reference

The following section provides an alphabetical listing of the enums in the OpenScript HTTPService API.

6.1.1 Alphabetical Enum Listing

The following table lists the HTTPService Enums in alphabetical order.

Table 6–1 List of HTTPService Enums

Enum	Description
EncodeOptions	When solving variables using `solve()`, specify an additional encoding/decoding transformation to apply the variable's value.
Source	For the `find()` and `solve()` functions, specify which part of the browser response in which to look.

6.2 HTTPService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript HTTPService API.

6.2.1 Alphabetical Command Listing

The following table lists the HTTPService API methods in alphabetical order.

Table 6–2 List of HTTPService Methods

Method	Description
http.addAuthentication	Add a username and password to use when authenticating a URL that requires NTLM or Basic authentication.
http.addCookie	Adds a cookie to the cookie jar.
http.addGlobalAssertText	Registers a text assertion test to run every time a page is requested.
http.addGlobalVerifyText	Registers a text verification test to run every time a page is requested.

Table 6–2 (Cont.) List of HTTPService Methods

Method	Description
http.addValidator	Adds a class to provide custom validation for all browser responses.
http.assertResponseTime	Tests the server response time for the previous request.
http.assertText	Searches the HTML contents of the specified document XPath for the specified text pattern.
http.assertTitle	Searches the `<title>` tag in the HTML content for the specified title pattern, reporting a failure if the test fails.
http.assertXPath	Extracts a value from the browser's last retrieved contents using XPath notation.
http.beginConcurrent	Begins a concurrent section used to make multiple HTTP requests at the same time by the same Virtual User using a different HTTP connection.
http.clearCookies	Clears the Cookies during playback.
http.element	Finds the element from the browser's DOM tree using XPath notation.
http.endConcurrent	Ends the specified concurrent section waiting the specified amount of time for requests to finish.
http.form	Finds the form element from the browser's DOM tree using XPath notation.
http.frame	Finds the frame element from the browser's DOM tree using XPath notation.
http.get	Requests a web page using an HTTP GET method.
http.getBrowser	Gets a Thin browser for making HTTP requests.
http.getHtmlContent	Gets the HTML content of a document which is specified by the XPath.
http.getLastBrowserResponse	Gets the last retrieved response from this Virtual User's browser.
http.getLastResponseContents	Returns a String version of a given browser response from the specified source type.
http.getResponseHeaders	Gets the headers of a response and its ancestors that is received by a window specified by the XPath.
http.getSettings	Gets the settings for the HTTP service.
http.getValidatorList	Returns all the validators added by `addValidator`.
http.header	Convenience method for creating a new `Header` object.
http.headers	Convenience method to create a new `Header` array.
http.javaScriptPath	Builds the JavaScript path(s) to use to extract JavaScript strings using http.solveGroupJavaScript.
http.link	Finds the link element from the browser's DOM tree using XPath notation.
http.loadKeystore	Loads the keystore file using the specified password.
http.multipartPost	Requests a web page using a multipart HTTP POST method with querystring, postdata, headers, boundary, URL-encoding, and character sets.
http.navigate	Navigates to a web page using the specified ID, URL, additional headers, redirect, character set, method, and input stream.

Table 6–2 (Cont.) List of HTTPService Methods

Method	Description
http.param	Convenience method to create a new File Parameter object.
http.post	Requests a web page using an HTTP POST method with querystrings, postdata, headers, URL encoding, and character sets.
http.postdata	Convenience method to return a postdata `ParamCollection`.
http.querystring	Convenience method to create a query string `ParamCollection` object.
http.removeCookie	Removes the cookie named *`cookieName`* from the cookie jar.
http.removeGlobalTextValidator	Removes a global text assertion or verification test that was previously added using the http.addGlobalAssertText.
http.removeValidator	Removes a previously added validator.
http.setAcceptLanguage	Specifies the `Accept-Language` header that the browser should use when sending HTTP requests.
http.setUserAgent	Specifies the `User-Agent` header that the browser should use when sending HTTP requests.
http.solve	Parses a value from the document found by XPath and stores it as a variable.
http.solveCookieHeader	Extracts the "Set-Cookie" header value which matches the cookie key from the last response header and stores it in the specified script variable.
http.solveGroupJavaScript	Extracts a JavaScript string from a document by XPath using a String and stores it in the script's variables collection.
http.solveHeader	Extracts the specified header value using from the last request's response header acccording to header name.
http.solveRedirectNavs	Parses a value from the re-directed navigation's contents and stores it as a variable.
http.solveRefererHeader	Extracts the "Referer" header value using from the last request's response header using XPath notation.
http.solveXPath	Extracts a value from the specified document contents using XPath notation.
http.text	Convenience method to return a `ParamCollection` object.
http.urlEncode	URL-encodes a given string using a specific character set excludng the specified character array.
http.validate	Validates the browser result for correctness.
http.verifyResponseTime	Tests the server response time for the previous request without failing.
http.verifyText	Searches the HTML contents in the document specified by XPath notation for the specified text pattern.
http.verifyTitle	Searches the `<title>` tag in HTML content for the specified title pattern, reporting a warning if the test fails.
http.verifyXPath	Verify the text of a DOM element in the browser's last retrieved contents.
http.window	Finds the window element from browser's DOM tree using the window index.
http.xmlPost	Posts an XML string to a web server using the HTTP POST method.

The following sections provide detailed reference information for each method and enum in the HTTPService Class of HTTP Load Module Application Programming Interface.

http.addAuthentication

Add a username and password to use when authenticating a URL that requires NTLM or Basic authentication.

When the browser requests a URL that returns a 401 or 407 "Unauthorized" HTTP response code, the browser will authenticate using a username and password in its authentication list whose corresponding URL matches the unauthorized URL. All querystring parameters in the URL are ignored. The protocol of the URL is also ignored. The URL is case-sensitive.

If multiple authentication entries are added for similar looking URLs, the URL that matches the most successive characters will be chosen. The order in which authentication entries are added to the list and searched is arbitrary.

Format

The http.addAuthentication method has the following command format(s):

http.addAuthentication(*url*, *username*, *password*);

Command Parameters

url
a String specifying the URL for this authentication information. May contain {{ }} syntax for transforming.

username
a String specifying the username. May contain {{ }} syntax for transforming. Must not be `null`. For NTLM-based authentication, username may be preceded by a domain and a single backslash. For example, in Java code, type: `oracle\\user1`. Note that two backslashes are typed because Java requires a backslash to be escaped with a backslash.

password
a String specifying the password. May contain {{ }} syntax for transforming. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Sets username and encrypted password authentication for `myServer.com`. See also the {@link oracle.oats.scripting.modules.basic.api.IteratingVUserScript.decrypt} and {@link oracle.oats.scripting.modules.basic.api.IteratingVUserScript.encrypt} methods in the Basic module API.

```
http.addAuthentication("http://myServer.com", "myUsername",
 decrypt("38eHEwYhPOOG9ljr70eF6A=="));
```

http.addCookie

Adds a cookie to the cookie jar. The browser will submit this cookie in subsequent HTTP requests.

This method is identical to http.getBrowser().getCookieJar().addCookie(String, String).

This method URL-encodes the value specified in the cookie string using the specified character set. If no URL-encoding is desired, specify a `null` character set.

The *`cookieString`* is formatted like a Set-Cookie header as if it were sent by the server. The name, value, and other optional attributes are extracted from *`cookieString`* and set into the Cookie object.

If an empty or null domain is specified in *`cookieString`*, this cookie will apply to all domains.

If no path is specified in *`cookieString`* then this cookie will apply to all paths.

If a cookie with the specified name/domain/path (case-sensitive) does not yet exist, then it will be added.

If a cookie with the specified name/domain/path (case-sensitive) exists and its value and attributes are also the same, this method does nothing.

If a cookie with the specified name/domain/path (case-sensitive) exists but its value or attributes differ, then the new cookie will replace the old one.

Format

The http.addCookie method has the following command format(s):

http.addCookie(*cookieString*, *charset*);

Command Parameters

cookieString
a String specifying the Response cookie header. May contain {{ }} syntax for transforming. Must not be `null`. The "Set-Cookie:" portion of the header is optional, and so are any additional attributes.

Example 1: Set-Cookie: cookieFavoriteColor=Blue; path=/; expires=Thursday, 11-Dec-2008 11:15:00 GMT; domain=www.oracle.com

Example 2: client_user=user123

charset
a String specifying the Character set to use when URL-encoding the value part of the cookie. May contain {{ }} syntax for transforming. May be `null`. If `null`, this method will not URL-encode the value.

Throws

UnsupportedEncodingException
if *`charset`* is not a supported character encoding.

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Adds a custom cookie to the cookie jar using the specified character set.

```
http.addCookie("JSESSIONID=JLCTLkmMxvqfL6J7h", "ASCII");
```

http.addGlobalAssertText

Registers a text assertion test to run every time a page is requested.

The text assertion test searches the HTML contents of all documents in all browser windows for the specified text pattern.

If the test fails, the script will always fail and stop running, unless the text matching test error recovery setting specifies a different action such as Warn or Ignore.

The assertion is performed for all requests made by this Virtual User, including requests made inside concurrent sections. The assertion is not performed for resources requested by the Download Manager.

This method supports result code verification methods including `getLastResult()` and `getLastError()`.

To unregister a global text assertion, use http.removeGlobalTextValidator.

Format

The http.addGlobalAssertText method has the following command format(s):

http.addGlobalAssertText(*testName*, *textToAssert*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying a descriptive name of the test being applied. This name may be used by a subsequent call to the http.removeGlobalTextValidator. Must not be `null`.

textToAssert
a String specifying th e text to match on the page, or not match on the page if `TextPresence` is set to `TextPresence.PassIfPresent`. Must not be `null`.

sourceType
a Source enum specifying where to match the text: the HTML contents or HTTP response headers. Must not be `null`.

textPresence
a TextPresence enum specifying either `PassIfPresent` or `FailIfPresent`, depending on if you want the text to be present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a Regular Expression, Wildcard, or exact match.

Example

Adds a Global Text Matching test for Display Content, Response Header, and Source HTML respectively.

`globalTextMatchingTest1` passes if the text to match string is present in the Display Content and matches exactly.

`globalTextMatchingTest2` passes if the text to match string is present in the Response header and matches the Regular Expression.

`globalTextMatchingTest3` fails if the text to match string is present in the Source HTML and matches the Wildcard.

```
http.addGlobalAssertText("globalTextMatchingTest1",
 "match this text string", Source.DisplayContent,
 TextPresence.PassIfPresent, MatchOption.Exact);
http.addGlobalAssertText("globalTextMatchingTest2",
 "jsessionid=(.+?)(?:\\\"|&)", Source.ResponseHeader,
 TextPresence.PassIfPresent, MatchOption.RegEx);
http.addGlobalAssertText("globalTextMatchingTest3",
 "match this *", Source.Html,
 TextPresence.FailIfPresent, MatchOption.Wildcard);
```

http.addGlobalVerifyText

Registers a text verification test to run every time a page is requested.

The text verification test searches the HTML contents of all documents in all browser windows for the specified text pattern.

If the test fails, a warning is always reported, irrespective of current Error Recovery settings.

The verification is performed for all requests made by this Virtual User, including requests made inside concurrent sections. The verification is not performed for resources requested by the Download Manager.

This method supports result code verification methods including `getLastResult()` and `getLastError()`.

To unregister a global text verification, use http.removeGlobalTextValidator.

Format

The http.addGlobalVerifyText method has the following command format(s):

http.addGlobalVerifyText(*testName*, *textToAssert*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying a descriptive name of the test being applied. This name may be used by a subsequent call to http.removeGlobalTextValidator. Must not be `null`.

textToAssert
A String specifying the text to match on the page, or not match on the page if `TextPresence` is set to `TextPresence.PassIfPresent`. Must not be `null`.

sourceType
a Source enum specifying where to match the text: the HTML contents or HTTP response headers. Must not be `null`.

textPresence
a TextPresence enum specifying either `PassIfPresent` or `FailIfPresent`, depending on if you want the text to be present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a Regular Expression, Wildcard, or exact match.

Example

Adds a Global Text Matching test for Display Content, Response Header, and Source HTML respectively.

`globalTextMatchingTest1` passes if the text to match string is present in the Display Content and matches exactly.

`globalTextMatchingTest2` passes if the text to match string is present in the Response header and matches the Regular Expression.

`globalTextMatchingTest3` fails if the text to match string is present in the Source HTML and matches the Wildcard.

```
http.addGlobalVerifyText("globalTextMatchingTest1",
 "match this text string", Source.DisplayContent,
 TextPresence.PassIfPresent, MatchOption.Exact);
http.addGlobalVerifyText("globalTextMatchingTest2",
 "jsessionid=(.+?)(?:\\\"|&)", Source.ResponseHeader,
 TextPresence.PassIfPresent, MatchOption.RegEx);
http.addGlobalVerifyText("globalTextMatchingTest3",
 "match this *", Source.Html,
 TextPresence.FailIfPresent, MatchOption.Wildcard);
```

http.addValidator

Adds a class to provide custom validation for all browser responses.

Custom IValidator instances are invoked after HTTPService performs its own validation.

Format

The http.addValidator method has the following command format(s):

http.addValidator(*validator*);

Command Parameters

validator
a Custom validator object.

Example

Adds a custom validator to test a response.

```
import oracle.oats.scripting.modules.basic.api.exceptions
   .AbstractScriptException;
public class script extends IteratingVUserScript {
 @ScriptService oracle.oats.scripting.modules.utilities.api.
   UtilitiesService utilities;
 @ScriptService oracle.oats.scripting.modules.http.api.HTTPService http;
 public void initialize() throws Exception {
 }
 public void run() throws Exception {
 //define databanks and variables
 getDatabank("navigation_info").getNextDatabankRecord();
 String link = eval("{{navigation_info.link}}");
 LinkValidator linkValidator = null;
 //[script navigation code...]
 //add a validator
 http.addValidator(linkValidator = new LinkValidator(link));
 //[additional script navigation code...]
 }
 public void finish() throws Exception {
 }
//define class for custom validator action
 class LinkValidator implements IValidator {
  @ScriptService oracle.oats.scripting.modules.utilities
    .api.UtilitiesService utilities;
  @ScriptService oracle.oats.scripting.modules.http.api.HTTPService http;
 private String link;
 public LinkValidator(String link) {
   this.link = link;
  }
  public void validate(Response response) {
   String contents = response.getContents();
   if (!contents.contains(link))
    try {
     info("Not found link '" + link + "'");
    } catch (AbstractScriptException e) {
     e.printStackTrace();
    }
```

```
    else
     try {
      info("Found link '" + link + "'");
     } catch (AbstractScriptException e) {
      e.printStackTrace();
     }
    }
   }
}
```

See Also

http.removeValidator and http.getValidatorList

http.assertResponseTime

Tests the server response time for the previous request.

This method does nothing if the Response already has an exception set.

If the test fails, always fail the script unless the server response test error recovery setting specifies a different action such as Warn or Ignore.

Subclasses may override this method if they need to modify how the HTTP Service itself validates server response times.

Format

The http.assertResponseTime method has the following command format(s):

http.assertResponseTime(*testName*, *minTime*, *maxTime*);

Command Parameters

testName
a String specifying a descriptive name describing what request is being tested. The name may appear in results and counter names.

minTime
a double specifying the minimum server response time (inclusive).

maxTime
a double specifying maximum server response time (inclusive).

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

ResponseTimeException
if no previous response to validate, or if the response time does not fall within the given range and stopIterationOnFailure is `true`.

Example

Adds Server Response test with a minimum of 1 second and a maximum of 15 seconds.

```
http.assertResponseTime("myServerResponseTest", 1.0, 15.0);
```

http.assertText

Searches the HTML contents of the specified document XPath for the specified text pattern.

If the test fails, always fail the script unless the text matching test error recovery setting specifies a different action such as Warn or Ignore.

Format

The http.assertText method has the following command format(s):

http.assertText(*testName*, *textToAssert*, *sourceType*, *textPresence*, *matchOption*);

http.assertText(*testName*, *documentXPath*, *textToAssert*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

textToAssert
a String specifying the text to match on the page, or not match on the page if `TextPresence` is set to `TextPresence.PassIfPresent`.

sourceType
a Source enum specifying where to match the text: the HTML contents or HTTP response headers.

textPresence
a TestPresence enum specifying either `PassIfPresent` or `FailIfPresent`, depending on if you want the test to pass or fail if the text to match is present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a Regular Expression, Wildcard, or exact match.

documentXPath
a String specifying the XPath to the document that contains the contents.

Throws

MatchException
if the assertion fails. AbstractScriptException on any other failure when attempting to assert the text.

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Exception
if the assertion fails.

Example

Adds Text Matching tests in the specified document for Display Content, Response Header and Source HTML respectively.

myTextMatchingTest1 passes if the text to match string is present in the Display Content and matches exactly.

myTextMatchingTest2 passes if the text to match string is present in the Response header and matches the Regular Expression.

myTextMatchingTest3 fails if the text to match string is present in the Source HTML and matches the Wildcard.

```
http.assertText("myTextMatchingTest1", "window[@index='0']",
 "match this text string", Source.DisplayContent,
 TextPresence.PassIfPresent, MatchOption.Exact);
http.assertText("myTextMatchingTest2", "window[@index='0']",
 "jsessionid=(.+?)(?:\\\"|&)", Source.ResponseHeader,
 TextPresence.PassIfPresent, MatchOption.RegEx);
http.assertText("myTextMatchingTest3", "window[@index='0']",
 "match this *", Source.Html,
 TextPresence.FailIfPresent, MatchOption.Wildcard);
```

http.assertTitle

Searches the <title> tag in the HTML content for the specified title pattern, reporting a failure if the test fails.

If the test fails, always fail the script unless the text matching test error recovery setting specifies a different action such as Warn or Ignore.

Format

The http.assertTitle method has the following command format(s):

http.assertTitle(*title*, *matchOption*);

http.assertTitle(*documentXPath*, *title*, *matchOption*);

Command Parameters

title
a String specifying the text pattern specifying the expected title.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a Regular Expression, Wildcard, or exact match.

documentXPath
a String specifying the XPath to the document that contains the contents.

Throws

Exception
on any exception while navigating or transforming the data.

MatchException
if the assertion fails.

AbstractScriptException
on any other failure when attempting to assert the text.

Example

Adds a Verify only, never fail Text Title tests for the HTML document title using exact match, Regular Expression match, and Wildcard match respectively.

```
http.assertTitle("window[@index='0']", "Summary Report", MatchOption.Exact);
http.assertTitle("window[@index='0']", "Summary Report For (.+?)",
MatchOption.RegEx);
http.assertTitle("window[@index='0']", "Summary Report For ???",
MatchOption.Wildcard);
```

http.assertXPath

Extracts a value from the browser's last retrieved contents using XPath notation. Compare this value to the `textToVerify` parameter.

If the test fails, always fail the script unless the text matching test error recovery setting specifies a different action such as Warn or Ignore.

When playing back a script in Oracle Load Testing, always use Assertions; failed Verifications are *not* reported in the Oracle Load Test controller.

Format

The http.assertXPath method has the following command format(s):

http.assertXPath(*testName*, *xpath*, *resultIndex*, *textToVerify*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

xpath
a String specifying the XPath describing which value to parse from the last retrieved contents. Must not be `null`.

resultIndex
a 0-based index value specifying the specific result to retrieve if the XPath returns multiple results. A `null` value specifies that all results should be returned.

textToVerify
a String specifying the Text to match, or not match with the XPath result if `TextPresence` is set to `TextPresence.PassIfPresent`.

textPresence
a TestPresence enum specifying either `PassIfPresent` or `FailIfPresent`, depending on if you want the test to pass or fail if the text to match is present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a Regular Expression, Wildcard, or exact match.

Throws

AbstractScriptException
on any failure when attempting to verify the text.

Example

Adds Text Matching tests using XPath notation.

`myTextMatchingTest1` passes if the text to match string is present and matches exactly.

`myTextMatchingTest2` passes if the text to match string is present and matches the entire string exactly.

`myTextMatchingTest3` passes if the text to match string is present and matches the Regular Expression.

`myTextMatchingTest4` passes if the text to match string is present and matches the entire string from the Regular Expression.

`myTextMatchingTest5` fails if the text to match string is present and matches the Wildcard.

```
http.assertXPath("myTextMatchingTest1", ".//input[@name='j_id_id3']" +
 "/@value", 0, "match this text string",
 TextPresence.PassIfPresent, MatchOption.Exact);
http.assertXPath("myTextMatchingTest2", ".//input[@name='j_id_id3']" +
 "/@value", 1, "match this text string",
 TextPresence.PassIfPresent, MatchOption.ExactEntireString);
http.assertXPath("myTextMatchingTest3", ".//input[@name='j_id_id3']" +
 "/@value", 0, "jsessionid=(.+?)(?:\\\"|&)",
 TextPresence.PassIfPresent, MatchOption.RegEx);
http.assertXPath("myTextMatchingTest4", ".//input[@name='j_id_id3']" +
 "/@value", 1, "jsessionid=(.+?)(?:\\\"|&)",
 TextPresence.PassIfPresent, MatchOption.RegExEntireString);
http.assertXPath("myTextMatchingTest5", ".//input[@name='j_id_id3']" +
 "/@value", 0, "match this *",
 TextPresence.FailIfPresent, MatchOption.Wildcard);
```

http.beginConcurrent

Begins a concurrent section used to make multiple HTTP requests at the same time by the same Virtual User using a different HTTP connection.

The maximum number of concurrent requests allowed by one Virtual User at any time is determined by the Maximum Connections playback setting.

When inside a concurrent section, the script will not wait for the completion of any HTTP requests that are started. Use http.endConcurrent to end a concurrent section and wait for all concurrent HTTP requests in the given section to finish.

While inside a concurrent section, responses for individual requests cannot be determined. For example, calling http.getLastBrowserResponse will throw a {@link oracle.oats.scripting.modules.http.api.exceptions.ConcurrentException}

Invoking this method more than once on the same `concurrentSectionId` is harmless and does not begin a new concurrent section with the existing ID.

Note: The API getLastResult and the "result" variables will not work reliably inside HTTP script beginConcurrent/endConcurrent sections.

Format

The http.beginConcurrent method has the following command format(s):

http.beginConcurrent(*concurrentSectionId*);

Command Parameters

concurrentSectionId
a String specifying a unique identifier for the concurrent section. Multiple concurrent sections may be started and stopped independently of each other by providing a different name. Must not be `null`.

Throws

Exception
on any failure.

Example

Begins a concurrent section with the ID `mySessionID_0001`.

```
http.beginConcurrent("mySessionID_0001");
```

See Also

http.endConcurrent

http.clearCookies

Clears the Cookies during playback.

Format

The http.clearCookies method has the following command format(s):

http.clearCookies( );

Example

Clears the browser Cookies added within the run section of a given script.

Cookies added to the browser in the initialize section will be preserved forever unless the playback setting "Preserve Cookies between Iterations" is set to `false`.

```
http.clearCookies();
```

http.element

Finds the element from the browser's DOM tree using XPath notation.

The element instance will be returned.

Format

The http.element method has the following command format(s):

http.element(*xpathToElement*);

Command Parameters

xpathToElement
a String specifying the XPath of the element describing which value to parse from the DOM tree. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the element's reference.

Example

Finds the form element with ID 'frmTest' from the browser's DOM tree.

```
boolean element_xpath_exists = http.element("window[@index='0']" +
 "/frame[@name='right']//form[@id='frmTest' " +
 "and @name='frmTest']")
 .exists();
if(element_xpath_exists){
 info("WebHTTPElement exists");
}
else{
 warn("WebHTTPElement does not exist");
}
```

EncodeOptions

The EncodeOptions has the following values:

Table 6–3 List of EncodeOptions Values

Value	Description
None	Leaves the input string alone; do not apply any additional encoding/decoding transformations.
URLEncode	URL-encodes a string. For example, the string `abc/xyz` becomes `abc%2Fxyz`.
URLDecode	URL-decodes a string. For example, the string `abc%2Fxyz` becomes `abc/def`.
XMLEncode	Encodes any characters that should be escaped for XML serialization. For example, the string `abc<xyz` becomes `abc<xyz`.
XMLDecode	Decodes any characters that were escaped for XML serialization. For example, the string `abc<xyz` becomes `abc<xyz`.
RemoveNewlines	Strips all newlines.
ConvertNewlinesToCRLF	Replaces \0A (not followed by \0D) with \0D\0A.
ConvertNewlinesToLF	Replaces \0D\0A with \0A.
EncodeURIComponent	Encodes special characters. In addition, it encodes the following characters: , / ? : @ & = + $ #

http.endConcurrent

Ends the specified concurrent section waiting the specified amount of time for requests to finish.

It waits the specified maximum amount of time for all pending HTTP requests in the specified concurrent section to finish.

Format

The http.endConcurrent method has the following command format(s):

http.endConcurrent(*concurrentSectionId*);

http.endConcurrent(*concurrentSectionId*, *timeoutSeconds*);

Command Parameters

concurrentSectionId
a String specifying the ID of the concurrent section that was already started using http.beginConcurrent. If `null`, this method will stop the most recently started concurrent section.

timeoutSeconds
a long specifying the maximum number of seconds allocated to wait for the concurrent section to finish. Specify Long.MAX_VALUE to wait forever. This method will return immediately after the requests finish, even if the timeout has not been reached.

Throws

Exception
on any exception that occurs when executing the concurrent requests.

ConcurrentException
if a timeout occurs while waiting for the concurrent requests to finish.

InterruptedException
if the thread is interrupted while waiting for the concurrent requests to finish.

Example

Ends a concurrent section previously started with the ID `mySessionID_0001`. It waits up to 15 seconds for all pending HTTP requests in the specified concurrent section to finish.

```
http.endConcurrent("mySessionID_0001", 15);
```

See Also

http.beginConcurrent

http.form

Finds the form element from the browser's DOM tree using XPath notation.

The element instance will be returned.

Format

The http.form method has the following command format(s):

http.form(*recId*, *xpathToForm*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

xpathToForm
a String specifying the XPath of the form element describing which value to parse from the DOM tree. Must not be `null`.

Throws

AbstractScriptException
on any exception while transforming `xpathToForm`.

Returns

The form element's reference.

Example

Gets the form named 'mainForm' under the first window.

```
boolean form_exists = http.form(142, "window[@index='0']" +
 "//form[@action='{{web.link.research," +
 "http://myServer.com/Ticker.asp}}']")
 .exists();
String form_exists_str = String.valueOf(form_exists);
if(form_exists){
 info("WebHTTPForm exists: + form_exists_str");
}
else{
 warn("WebHTTPForm exists: + form_exists_str");
}
```

http.frame

Finds the frame element from the browser's DOM tree using XPath notation.

The element instance will be returned.

Format

The http.frame method has the following command format(s):

http.frame(*recId*, *xpathToFrame*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

xpathToFrame
a String specifying the XPath of the frame element describing which value to parse from the DOM tree. Must not be `null`.

Throws

AbstractScriptException
on any exception while transforming *`xpathToFrame`*.

Returns

The frame element's reference.

Example

Gets the frame named 'right' under the first window.

```
http.frame(192, "window[@index='0']" +
 "/frame[@name='right']")
 .get("http://myServer.com/FramenumSet.htm",
 null, null, true, "ASCII", "ASCII");
```

http.get

Requests a web page using an HTTP GET method.

Format

The http.get method has the following command format(s):

http.get(*recid*, *urlPath*);

http.get(*recid*, *urlPath*, *querystring*);

http.get(*recid*, *urlPath*, *querystring*, *encode*);

http.get(*recid*, *urlPath*, *querystring*, *headers*, *encode*, *reqCharset*, *respCharset*);

Command Parameters

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

urlPath
a String specifying the URL to request, excluding query string data. May contain {{ }} syntax for transforming. Must not be `null`.

querystring
an Object specifying the Query string data, either as a String or `ParamCollection`. May be `null`.

headers
a headers array specifying additional headers to add/remove from the request before submitting it to the server.

encode
a boolean specifying the parameter encoding. Set to `true` to URL-encode the querystring parameters before submitting them to the sever.

reqCharset
a String specifying the character set to use if URL-encoding any of the request parameters.

respCharset
a String specifying the character set to use when reading the response contents.

Throws

Exception
on any exception while navigating or transforming the data.

Example

Requests the specified web pages using an HTTP GET method. Expanded examples add a query string parameter `myQueryString` and header actions.

`http.get(30[...]` requests the specified web page without parameters, headers, or URL-encoding.

`http.get(31[...]` requests the specified web page without URL-encoded parameters and will always add the header using ASCII character sets for the request and response headers.

`http.get(32[...]` requests the specified web page with URL-encoded parameters and will always add the header using ASCII character sets for the request and response headers.

`http.get(33[...]` requests the specified web page with URL-encoded parameters and will add the header only if it is not set using UTF-8 character sets for the request and response headers.

`http.get(34[...]` requests the specified web page with URL-encoded parameters and will add the header only if it is not set and add it to all navigations after using OracleBinary character sets for the request and response headers.

```
http.get(30, "http://myServer.com/", null, null, false, "ASCII", "ASCII");
http.get(31, "http://myServer.com/", http.querystring(
 http.param("myQueryString", "my String Value")),
 http.headers(http.header("myheaderName", "myHeaderValue",
   Header.HeaderAction.Add)), false, "ASCII", "ASCII");
http.get(32, "http://myServer.com/", http.querystring(
 http.param("myQueryString", "my String Value")),
 http.headers(http.header("myheaderName", "myHeaderValue",
   Header.HeaderAction.Add)), true, "ASCII", "ASCII");
http.get(33, "http://myServer.com/", http.querystring(
 http.param("myQueryString", "my String Value")),
 http.headers(http.header("myheaderName", "myHeaderValue",
   Header.HeaderAction.SetIfNotSet)), true, "UTF-8", "UTF-8");
http.get(34, "http://myServer.com/", http.querystring(
 http.param("myQueryString", "my String Value")),
 http.headers(http.header("myheaderName", "myHeaderValue",
   Header.HeaderAction.GlobalSetIfNotSet)),
   true, "OracleBinary", "OracleBinary");
```

See Also

http.querystring for creating a parameter collection.

http.getBrowser

Gets a Thin browser for making HTTP requests.

Format

The http.getBrowser method has the following command format(s):

http.getBrowser( );

Returns

a Thin browser object for making HTTP requests.

Example

Gets a Thin browser object using an OpenScript script variable and a Java variable.

```
//using script variable
getVariables().set("browserId", http.getBrowser().getBrowserId());
info("The browser ID string using script var: {{browserId}}");
//using Java variable
ThinBrowser browser = http.getBrowser();
info("The browser ID is: " + browser.getBrowserId());
```

http.getHtmlContent

Gets the HTML content of a document which is specified by the XPath.

If the XPath refers to a window, the document is the document in the window. If the XPath refers to a frame, the document is the frame's content document. If the XPath refers to an element, the document is the owner document of the element.

Format

The http.getHtmlContent method has the following command format(s):

http.getHtmlContent(*xpath*);

Command Parameters

xpath
a String specifying the XPath to the window, document, frame, or element containing the HTML content.

Returns

the HTML content string.

Example

Gets the HTML contents of the document in the window with the index of 0.

```
//using Script variable
getVariables().set("htmlContents",
 http.getHtmlContent("/window[@index=0]"));
info("HTML Contents Type in script var = {{htmlContents}}" );
//using Java variable
String htmlContents = http.getHtmlContent("/window[@index=0]");
info("HTML Contents = " + htmlContents);
```

http.getLastBrowserResponse

Gets the last retrieved response from this Virtual User's browser.

The result of this method is unpredictable if more than one navigation occurs at the same time for the same Virtual User. For example, when running multiple concurrent requests using http.beginConcurrent, the last response contents are indeterminate.

Regardless of how many threads are being run, this method will only ever return a response for the current Virtual User that owns this HTTP service instance. This method will never return a response from a different running Virtual User.

Format

The http.getLastBrowserResponse method has the following command format(s):

http.getLastBrowserResponse( );

Returns

this Virtual User's browser's last retrieved response. May be `null`.

Example

Gets the last retrieved response from this Virtual User's browser.

```
//using script variable
getVariables().set("contentType",
 http.getLastBrowserResponse().getContentType());
info("Content Type in script var = {{contentType}}" );
//using Java variable
String contentType = http.getLastBrowserResponse().getContentType();
info("Content Type = " + contentType);
```

http.getLastResponseContents

Returns a String version of a given browser response from the specified source type.

Format

The http.getLastResponseContents method has the following command format(s):

http.getLastResponseContents( );

http.getLastResponseContents(*sourceType*);

http.getLastResponseContents(*lastBrowserResponse*, *sourceType*);

Command Parameters

sourceType
the Source enum specifying which contents to return as a String. If `null`, assumes `Source.Html`.

lastBrowserResponse
a Response object from which to return the last browser contents.

Returns

String version of the given browser response. Returns `null` when the last browser response is `null`.

Example

Get the contents of the Response Header, Display HTML and source HTML last retrieved by the browser for this Virtual User.

```
//using script variable
getVariables().set("lastRespHeader",
 http.getLastResponseContents(http.getLastBrowserResponse(),
 Source.ResponseHeader));
info("Last Response Header in script var = {{lastRespHeader}}");
//using Java variable
String lastRespHeader = http.getLastResponseContents(
 http.getLastBrowserResponse(), Source.ResponseHeader);
info("Last Response Header = " + lastRespHeader);
//using script variable
getVariables().set("lastDisplayConts", http.getLastResponseContents(
 http.getLastBrowserResponse(), Source.DisplayContent));
info("Last Display Contents in script var = {{lastDisplayConts}}");
//using Java variable
String lastDisplayConts = http.getLastResponseContents(
 http.getLastBrowserResponse(), Source.DisplayContent);
info("Last Display Contents = " + lastDisplayConts);
//using script variable
getVariables().set("lastHtml", http.getLastResponseContents(
 http.getLastBrowserResponse(), Source.Html));
info("Last HTML Contents in script var = {{lastHtml}}");
//using Java variable
String lastHtml = http.getLastResponseContents(
 http.getLastBrowserResponse(), Source.Html);
info("Last HTML Contents = " + lastHtml);
```

http.getResponseHeaders

Gets the headers of a response and its ancestors that is received by a window specified by the XPath.

If the XPath refers to a frame, the window is the frame's window. If the XPath refers to an element, the window is the window of the element's owner document.

Format

The http.getResponseHeaders method has the following command format(s):

http.getResponseHeaders(*xpath*);

Command Parameters

xpath
a String specifying the XPath to the window, frame, or element containing the response headers.

Returns

the response headers String.

Example

Gets the HTML contents of the document in the window with the index of 0.

```
//using Script variable
getVariables().set("respHeader",
 http.getResponseHeaders("/window[@index=0]"));
info("Response Headers in script var = {{respHeader}}" );
//using Java variable
String respHeader = http.getResponseHeaders("/window[@index=0]");
info("Response Headers = " + respHeader);
```

http.getSettings

Gets the settings for the HTTP service.

Format

The http.getSettings method has the following command format(s):

http.getSettings( );

Returns

all settings pertaining to the HTTP Service.

Example

Gets the settings for the HTTP service.

```
import java.util.List;
import oracle.oats.scripting.modules.http.api.Settings;
//[...]
info("Callling http.getSettings()...");
Settings settings = http.getSettings();
info("Connection Timeout: " + settings.getConnectTimeout());
info("Proxy Host: " + settings.getProxyHost());
info("Proxy Password: " + settings.getProxyPassword());
info("Proxy Port: " + settings.getProxyPort());
info("Proxy User Name: " + settings.getProxyUsername());
info("Socket Timeout: " + settings.getSocketTimeout());
info("Speed Emulation (Bytes/sec): " +
   settings.getSpeedEmulationBytesPerSecond());
info("User Agent: " + settings.getUserAgent());
info("Cache Setting: " + settings.getCacheSetting());
info("Global Headers: " + settings.getGlobalHeaders());
List <String> resources = http.getSettings().getIgnoredResources();
if (resources != null) {
 for (int i=0; i<resources.size(); i++)
   info("IgnoredResources: " + resources.get(i));
 }
else info("IgnoredResources is null");
List <String> urls = http.getSettings().getIgnoredUrls();
if (urls != null) {
 for (int i=0; i<urls.size(); i++)
   info("IgnoredUrl: " + urls.get(i));
 }
else info("IgnoredUrl is null");
info("Preserve Connections: " + settings.getPreserveConnections());
info("Preserve Cookies: " + settings.getPreserveCookies());
info("Preserve Variables: " + settings.getPreserveVariables());
info("Proxy Enabled: " + settings.getProxyEnabled());
info("Use Cache: " + settings.getUseCache());
```

http.getValidatorList

Returns all the validators added by `addValidator`.

Format

The http.getValidatorList method has the following command format(s):

http.getValidatorList( );

Returns

an ArrayList copy of the validators or `null` when the validator list is empty.

Example

Gets an ArrayList copy of the validators and writes the simple name for each validator to the results.

```
import java.util.List;
//[...]
List<IValidator> list = http.getValidatorList();
  for (int i = 0; i < list.size(); i++) {
   IValidator validator = list.get(i);
   info("Validator Name: " + validator.getClass()
      .getSimpleName());
   if (validator.getClass().getSimpleName()
      .equals("LinkValidator")) {
    info("Removing LinkValidator");
   http.removeValidator(validator);
   } else if
    (validator.getClass().getSimpleName()
      .equals("CategoryValidator")) {
    info("Removing CategoryValidator");
   http.removeValidator(validator);
   } else
    info("Other validator: "
      + validator.getClass().getSimpleName());
  }
```

See Also

http.addValidator and http.removeValidator

http.header

Convenience method for creating a new `Header` object.

Format

The http.header method has the following command format(s):

http.header(*name*, *value*, *action*);

Command Parameters

name
a String specifying the header name. May contain {{ }} syntax for transforming. Must not be `null`.

value
a String specifying the header value. May contain {{ }} syntax for transforming. Must not be `null`.

action
a HeaderAction object, such as remove or add the header.

Throws

AbstractScriptException
on any exception while transforming the name and value.

Returns

a new Header object, with the name and value transformed.

Example

Creates new headers using the specified header names, values, and actions.

```
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));
```

See Also

http.get and http.post Methods.

http.headers

Convenience method to create a new `Header` array.

Format

The http.headers method has the following command format(s):

http.headers(*headers*);

Command Parameters

headers
one or more Header objects.

Returns

a new `Header` array.

Example

Creates a new header collection using the specified header names, values, and actions.

```
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));
```

See Also

http.get and http.post Methods.

http.javaScriptPath

Builds the JavaScript path(s) to use to extract JavaScript strings using http.solveGroupJavaScript.

Format

The http.javaScriptPath method has the following command format(s):

http.javaScriptPath(*name*, *scriptLanguage*, *scriptIndex*, *literalIndex*, *literalType*);

Command Parameters

name
a String specifying the variable name to use.

scriptLanguage
an Integer specifying the the script language. 1 to parse JavaScript, 0 to parse VB Script.

scriptIndex
an Integer specifying the 0-based index of the script block in the HTML contents to parse.

literalIndex
an Integer specifying the 0-based index of the dynamic JavaScript literal inside the block to extract.

literalType
an Integer specifying the data type. Specify '0' to parse a String from JavaScript, or '1' to parse a number.

Returns

the javascript path.

Example

Extracts a JavaScript String from the page and stores it in the variable "varJavaScript" in the script's variables collection. It parses the first script block in the HTML contents for the third the dynamic JavaScript literal as a String from JavaScript.

```
http.solveGroupJavaScript("window[@index='1']",
http.javaScriptPath("varJavaScript", 1, 1, 2, 0));
info("Java Script = {{varJavaScript}}");
```

See Also

http.solveGroupJavaScript

http.link

Finds the link element from the browser's DOM tree using XPath notation.

The element instance will be returned.

Format

The http.link method has the following command format(s):

http.link(*recId*, *xpathToLink*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

xpathToLink
a String specifying the XPath of the link element describing which value to parse from the DOM tree. Must not be `null`.

Throws

AbstractScriptException
on any exception while transforming *`xpathToLink`*.

Returns

the link element's reference.

Example

Clicks the link with the text 'research a company' under the first window.

```
http.link(130, "window[@index='0']//a[@text='research a company']")
 .click();
```

http.loadKeystore

Loads the keystore file using the specified password.

Allows the current virtual user to use a client-side certificate when connecting to any secure website.

Specify a path to a Java keystore file containing the certificate that the Virtual User should use for as long as it is running. The Virtual User will only ever use one keystore at a time when creating secure connections.

Calling this method more than once for the same Virtual User will cause subsequent secure connections to use the most recently loaded keystore.

To stop using a keystore for new connections, call this method with a `null` or empty value for the *`keystoreFilePath`*.

Format

The http.loadKeystore method has the following command format(s):

http.loadKeystore(*keystoreFilePath*, *keystorePassword*);

Command Parameters

keystoreFilePath
a String specifying the Absolute path to a certificate keystore file on the machine where the script will be played back. The keystore file should be generated using Java's `keytool.exe` program. If the path is `null`, an empty string, or the file does not exist, no keystore will be used.

keystorePassword
a String specifying the password used to read certificates from the keystore.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

KeystoreLoadException
if the keystore cannot be loaded for any reason.

Example

Loads the keystore file using the specified password.

```
http.loadKeystore("C:\\keys\\DummyClientKeyFile.jks", "WebAS");
```

http.multipartPost

Requests a web page using a multipart HTTP POST method with querystring, postdata, headers, boundary, URL-encoding, and character sets.

This method formats the post data using a multi-part format.

Format

The http.multipartPost method has the following command format(s):

http.multipartPost(*recid*, *urlPath*, *postdata*);

http.multipartPost(*recid*, *urlPath*, *querystring*, *postdata*, *headers*, *boundary*);

http.multipartPost(*recid*, *urlPath*, *querystring*, *postdata*, *headers*, *reqCharset*, *respCharset*);

http.multipartPost(*recid*, *urlPath*, *querystring*, *postdata*, *headers*, *boundary*, *encode*, *reqCharset*, *respCharset*);

Command Parameters

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

urlPath
a String specifying ther URL to request, excluding query string data. May contain {{ }} syntax for transforming. Must not be `null`.

postdata
a `ParamCollection` of the postdata to POST to the URL's host. Must not be `null`.

querystring
a `ParamCollection` of query string data parameters. May be `null`.

headers
a Header array of additional headers to add/remove from the request before submitting it to the server.

boundary
a String specifying the Multi-part post boundary string as it would appear in the header of the multi-part post.

For example, if the header is:

```
Content-Type: multipart/form-data;
boundary=---------------------------7d82f120260a68
```

Specify `boundary` as:

```
---------------------------7d82f120260a68
```

Note that the boundary string submitted in the body of the multi-part post contains two additional "-" characters.

reqCharset
a String specifying the character set to use if URL-encoding any of the request parameters.

respCharset
a String specifying the character set to use when reading the response contents.

encode
a boolean specifying the parameter encoding. Set to `true` to URL-encode the querystring parameters before submitting them to the sever.

Throws

Exception
on any exception while navigating or transforming the data.

Example

Requests a web page using a multipart HTTP POST method. This method formats the post data using a multipart format with query strings, postdata, headers, Boundary, URL-encoding, and character sets.

```
http.multipartPost(15, "http://myServer.com",
 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:\\OracleATS\\OFT\\DataBank\\mydata.csv",
     "C:\\OracleATS\\OFT\\DataBank\\mydata.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)),
 "--------------------------7d82f120260a68",
 true, "ASCII", "ASCII");
```

See Also

http.querystring and http.postdata for creating parameter collections.

http.navigate

Navigates to a web page using the specified ID, URL, additional headers, redirect, character set, method, and input stream. Only Multipart Post and XmlPost would call this method.

Format

The http.navigate method has the following command format(s):

http.navigate(*id*, *url*);

http.navigate(*id*, *url*, *method*);

http.navigate(*url*, *postdata*, *method*);

http.navigate(*url*, *postdata*, *followRedirects*, *method*);

http.navigate(*id*, *url*, *postdata*, *followRedirects*, *method*);

http.navigate(*id*, *url*, *postdata*, *additionalHeaders*, *followRedirects*, *method*);

http.navigate(*id*, *url*, *postdata*, *additionalHeaders*, *followRedirects*, *method*);

http.navigate(*id*, *url*, *postdata*, *additionalHeaders*, *followRedirects*, *respCharset*, *method*);

http.navigate(*id*, *url*, *postdata*, *additionalHeaders*, *followRedirects*, *respCharset*, *method*);

http.navigate(*id*, *url*, *additionalHeaders*, *followRedirects*, *respCharset*, *method*, *postStream*);

Command Parameters

id
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

url
a String specifying the URL to request. Must not be `null`.

method
an HTTPMethod enum specifing the HTTP request header method: GET or POST.

postdata
a String specifying the Postdata to POST to the URL's host. May be `null`.

followRedirects
a boolean specifying if redirects are followed. Set to `false` to specify that the browser should not follow any redirects, such as a 302 response.

additionalHeaders
a Headers object specifying additional headers to add to the request.

respCharset
a String specifying the character set to use when reading the response contents.

postStream
an InputStream containing postdata to POST to URL. Should be OpenScriptInputStream.

Throws

Exception
on any exception while navigating.

Example

Navigates to the specified URL using the specified headers, character set, POST method, and poststream. Does not follow redirects.

```
String filePath = "C:\\ADF_Load.xml";
String fileName = "ADFLoad";
Headers myheaders = new Headers();
myheaders.add("POST / HTTP/1.1", "");
myheaders.add("Accept-Encoding", "gzip, deflate");
ParamCollection postdata =  http.postdata(
 http.param("__EVENTTARGET","btnUploadTheFile"),
 http.param("__EVENTARGUMENT",""),
 http.param("__VIEWSTATE","dDwxNDc5ODAzNTs7PkPGUbP9tEJUQkJkazuBM8/TR8"),
 http.param("uplTheFile",filePath,fileName,"text/xml"));
InputStream postStream = postdata == null ? null :
 buildMultiPartPostStream(postdata.getParams(),
 "---------------------------7da1482d461390", "UTF8");
http.navigate(11, "http://myServer.com/mockups/uploadfile.aspx",
 myheaders, false, "UTF8", HTTPMethod.POST, postStream);
//[...]
protected InputStream buildMultiPartPostStream(Param params[], String boundary,
String reqCharset) throws AbstractScriptException
{
 if (boundary == null)
   boundary = MultiPartUtil.MULTIPART_HEADER_BOUNDARY_STRING;
 List<InputStream> inList = new ArrayList<InputStream>();
 ByteArrayOutputStream bis = new ByteArrayOutputStream();
 //sb.append("--");
 bis.write(45);
 bis.write(45);
 byte[] bBoundary = boundary.getBytes();
 try {
 bis.write(bBoundary);
 for (Param param : params) {
   //sb.append("\r\n")
   bis.write(13);
   bis.write(10);
   inList.add(new ByteArrayInputStream(bis.toByteArray()));
   bis.reset();
   inList.add(param.toStream(reqCharset));
   //sb.append("\r\n--").append(boundary)
   bis.write(13);
   bis.write(10);
   bis.write(45);
   bis.write(45);
   bis.write(bBoundary);
 }
   bis.write("--\r\n".getBytes());
 inList.add(new ByteArrayInputStream(bis.toByteArray()));
 bis.close();
}
catch (IOException e) {
 http.getCurrentVUSer().handleException(new ScriptException(e));
}
```

```
return new OpenScriptInputStream(inList);
}
```

http.param

Convenience method to create a new File Parameter object.

Format

The http.param method has the following command format(s):

http.param(*value*);

http.param(*name*, *value*);

http.param(*name*, *value*, *fileName*, *contentType*);

Command Parameters

name
a String specifying the parameter name.

value
a String specifying the parameter value.

fileName
a String specifying the full path to a file whose contents will be submitted to the server.

contentType
a String specifying the Content-type string to appear in the multi-part post parameter element when submitting the file contents to the server.

Returns

a new `Param` object with the given name and value.

Example

Specifies the file name and content type parameters as postdata for the multipart post navigation.

```
http.multipartPost(24, "http://myServer.com/", http.querystring(
 http.param("PostName","PostValue")), http.postdata(
   http.param("MyFileParameterName",
     "C:\\myfile.txt", "C:\\myfile.txt", "ASCII")),
 null, "", false, "ASCII", "ASCII");
```

See Also

http.get and http.post, and http.multipartPost Methods.

http.post

Requests a web page using an HTTP POST method with querystrings, postdata, headers, URL encoding, and character sets.

Format

The http.post method has the following command format(s):

http.post(*recid*, *urlPath*, *postdata*);

http.post(*recid*, *urlPath*, *postdata*, *encode*);

http.post(*recid*, *urlPath*, *querystring*, *postdata*, *headers*, *encode*, *reqCharset*, *respCharset*);

Command Parameters

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

urlPath
a String specifying the URL to request, excluding query string data. May contain {{ }} syntax for transforming. Must not be `null`.

postdata
a `ParamCollection` specifying the data to POST to the URL's host. Must not be `null`.

encode
a boolean specifying the parameter encoding. Set to `true` to URL-encode the querystring parameters using the agent machine's default character set before submitting them to the sever.

querystring
a `ParamCollection` specifying the Query string data parameter collection. May be `null`.

headers
a Header array specifying the Additional headers to add/remove from the request before submitting it to the server.

reqCharset
a String specifying the character set to use if URL-encoding any of the request parameters.

respCharset
a String specifying the character set to use when reading the response contents.

Throws

Exception
on any exception while navigating or transforming the data.

Example

Specifies a POST navigation with querystrings, postdata, headers, URL encoding, and character sets.

```
http.post(12, "http://myServer.com",
 http.querystring(http.param("QueryString1", "QueryValue1"),
   http.param("QueryString2", "QueryValue2"),
   http.param("QueryString3", "QueryValue3")),
 http.postdata(http.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");
```

See Also

http.querystring and http.postdata for creating parameter collections.

http.postdata

Convenience method to return a postdata `ParamCollection`.

Format

The http.postdata method has the following command format(s):

http.postdata(*params*);

Command Parameters

params
a `Param` object collection specifying all parameters to add to the collection.

Returns

a new `ParamCollection` object from the given `Param` objects.

Example

Specifies a POST navigation with postdata parameters.

```
http.post(9, "http://myServer.com", http.postdata(
 http.param("PostString1", "PostValue1"),
 http.param("PostString2", "PostValue2"),
 http.param("PostString3", "PostValue3")));
```

See Also

http.get and http.post, and http.multipartPost Methods. http.param for creating a `Param` object.

http.querystring

Convenience method to create a query string `ParamCollection` object.

Format

The http.querystring method has the following command format(s):

http.querystring(*params*);

Command Parameters

params
a `Param` object collection specifying all parameters to add to the collection.

Returns

a new `ParamCollection` object from the given Param objects.

Example

Specifies a POST navigation with querystring parameters.

```
http.post(12, "http://myServer.com",
 http.querystring(http.param("QueryString1", "QueryValue1"),
   http.param("QueryString2", "QueryValue2"),
   http.param("QueryString3", "QueryValue3")),
 null, null, true, "ASCII", "ASCII");
```

See Also

get, post, and multipartPost Methods. http.param for creating a Param object.

http.removeCookie

Removes the cookie named *cookieName* from the cookie jar.

This method removes all Cookies named *cookieName* for all domains and paths.

This method is identical to http.getBrowser().getCookieJar().removeCookie(String).

Format

The http.removeCookie method has the following command format(s):

http.removeCookie(*cookieName*);

Command Parameters

cookieName
a String specifying the case-sensitive name of the cookie. May contain {{ }} syntax for transforming. Must not be `null`.

For example: sessionid

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Removes the specified cookie from the cookie jar.

```
http.removeCookie("sessionid");
```

http.removeGlobalTextValidator

Removes a global text assertion or verification test that was previously added using the http.addGlobalAssertText.

If the specified test was not previously added, this method does nothing. No error is thrown.

Format

The http.removeGlobalTextValidator method has the following command format(s):

http.removeGlobalTextValidator(*testName*);

Command Parameters

testName
a String specifying the name of the test previously added by http.addGlobalAssertText.

Returns

`true` if the test was removed, otherwise `false`.

Example

Removes the global text matching test named "GlobalTextMatch".

```
boolean wasRemoved = http.removeGlobalTextValidator("GlobalTextMatch");
if(wasRemoved){
   info("Global Test Removed: " + wasRemoved);
}
else{
   info("Global Test Removed: " + wasRemoved);
}
```

http.removeValidator

Removes a previously added validator.

Format

The http.removeValidator method has the following command format(s):

http.removeValidator(*validator*);

Command Parameters

validator
a Validator previously added using http.addValidator

Returns

`true` if the validator was removed or `false` if the validator was not found in the list of validators previously added.

Example

Removes a previously added validator.

```
import java.util.List;
//[...]
List<IValidator> list = http.getValidatorList();
  for (int i = 0; i < list.size(); i++) {
   IValidator validator = list.get(i);
   info("Validator Name: " + validator.getClass()
      .getSimpleName());
   if (validator.getClass().getSimpleName()
      .equals("LinkValidator")) {
    info("Removing LinkValidator");
   http.removeValidator(validator);
   } else if
    (validator.getClass().getSimpleName()
      .equals("CategoryValidator")) {
    info("Removing CategoryValidator");
   http.removeValidator(validator);
   } else
    info("Other validator: "
      + validator.getClass().getSimpleName());
  }
```

See Also

http.addValidator and http.getValidatorList.

http.setAcceptLanguage

Specifies the `Accept-Language` header that the browser should use when sending HTTP requests.

This can be used to emulate different language settings in a browser.

Format

The http.setAcceptLanguage method has the following command format(s):

http.setAcceptLanguage(*acceptLanguage*);

Command Parameters

acceptLanguage
a String specifying the value of the `Accept-Language` header.

Example

Specifies the `Accept-Language` header that the browser should use when sending HTTP requests.

```
http.setAcceptLanguage("en-us");
```

http.setUserAgent

Specifies the `User-Agent` header that the browser should use when sending HTTP requests.

This can be used to emulate different browsers.

Format

The http.setUserAgent method has the following command format(s):

http.setUserAgent(*userAgent*);

Command Parameters

userAgent
a String specifying the `User-Agent` header value. May contain {{ }} syntax for transforming. For example, `Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)`

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Sets the browser Request Header `User-Agent` string.

```
http.setUserAgent("Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; " +
 ".NET CLR 1.1.4322; InfoPath.1; .NET CLR 2.0.50727; " +
 ".NET CLR 3.0.04506.30; .NET CLR 3.0.04506.648;" +
 ".NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)");
```

http.solve

Parses a value from the document found by XPath and stores it as a variable.

Format

The http.solve method has the following command format(s):

http.solve(*varName*, *pattern*);

http.solve(*varName*, *pattern*, *errorMsg*);

http.solve(*varName*, *pattern*, *errorMsg*, *isOptional*);

http.solve(*varName*, *pattern*, *errorMsg*, *isOptional*, *sourceType*);

http.solve(*varName*, *pattern*, *errorMsg*, *isOptional*, *sourceType*, *resultIndex*);

http.solve(*varName*, *pattern*, *errorMsg*, *isOptional*, *sourceType*, *resultIndex*, *encodeOption*);

http.solve(*varName*, *documentXPath*, *pattern*, *errorMsg*, *isOptional*, *sourceType*, *resultIndex*, *encodeOption*);

Command Parameters

varName
a String specifying the name of the script variable to create. Must not be `null`.

pattern
a String specifying the Regular Expression pattern specifying what to extract from the most recent navigation's contents. May contain a transform expression. Must not be `null`.

errorMsg
an optional String specifying the error message to display if the pattern cannot be solved. If `null`, a meaningful error message is automatically generated.

isOptional
a boolean specifying if the pattern must be solved or not. Set to `true` if the pattern does not have to be solved. If the pattern cannot be solved, the method quietly returns.

sourceType
the Source enum specifying which contents to return as a String. If `null`, assumes `Source.Html`.

resultIndex
a 0-based index value specifying the specific result to retrieve if the pattern matches more than one value. If `null`, all results will be added into the variables collection.

encodeOption
an EncodeOptions enum specifying the encoding or decoding operation to perform on the variable's value after solving the variable. A `null` value is equivalent to `EncodeOptions.None`.

documentXPath
a String specifying the XPath to the document containing the target html against which to solve the variable. Optional, may be `null`. If `null`, the variable will be solved against the last retrieved contents.

Throws

Exception
on any exception while navigating or transforming the data.

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

SolveException
if the given pattern cannot be matched to the previous contents and the pattern is not optional.

Example

Parses a value from the most recent navigation's contents using a Regular Expression and stores it as a variable.

```
http.solve("solvetitle", "window[@index='0']", "<TITLE>(.+)</TITLE>",
 "Cannot Solve Pattern", true, Source.Html, 0, EncodeOptions.None);
http.solve("solvetitle", "window[@index='0']", "<TITLE>(.+)</TITLE>",
 "Cannot Solve Pattern", true, Source.Html, 0, EncodeOptions.URLEncode);
http.solve("solvetitle", "window[@index='0']", "<TITLE>(.+)</TITLE>",
 "Cannot Solve Pattern", true, Source.Html, 0, EncodeOptions.URLDecode);
http.solve("solvetitle", "window[@index='0']", "<TITLE>(.+)</TITLE>",
 "Cannot Solve Pattern", true, Source.Html, 0, EncodeOptions.XMLEncode);
http.solve("solvetitle", "window[@index='0']", "<TITLE>(.+)</TITLE>",
 "Cannot Solve Pattern", true, Source.Html, 0, EncodeOptions.XMLDecode);
http.solve("solvetitle", "window[@index='0']", "<TITLE>(.+)</TITLE>",
 "Cannot Solve Pattern", true, Source.Html, 0,
 EncodeOptions.ConvertNewlinesToCRLF);
http.solve("solvetitle", "window[@index='0']", "<TITLE>(.+)</TITLE>",
 "Cannot Solve Pattern", true, Source.Html, 0, EncodeOptions.ConvertNewlinesToLF);
http.solve("solvetitle", "window[@index='0']", "<TITLE>(.+)</TITLE>",
 "Cannot Solve Pattern", true, Source.Html, 0, EncodeOptions.RemoveNewlines);
http.solve("solvetitle", "window[@index='0']", "<TITLE>(.+)</TITLE>",
 "Cannot Solve Pattern", true, Source.Html, 0, EncodeOptions.EncodeURIComponent);
```

http.solveCookieHeader

Extracts the "Set-Cookie" header value which matches the cookie key from the last response header and stores it in the specified script variable.

Format

The http.solveCookieHeader method has the following command format(s):

http.solveCookieHeader(*varName*, *key*, *lastValue*, *encodeOption*);

Command Parameters

varName
a String specifying the variable name where the value will be stored. Must not be `null`.

key
a String specifying the 0-based key of the cookie header to look for. Must not be `null`.

lastValue
an optional String specifying the recorded or last known value of the value about to be parsed. May be `null`. For informational purposes only.

encodeOption
an EncodeOptions enum specifying the encoding or decoding operation to perform on the variable's value after solving the variable, . A `null` value is equivalent to `EncodeOptions.None`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

SolveException
if the referer cannot be solved.

Example

Extract the first (key value 0) "Set-Cookie" header value from the Repsonse header (last known value of "Color=Blue") with no encoding.

```
http.solveCookieHeader("varCookieHead", "0", "Color=Blue",
 EncodeOptions.None);
info("Cookie Header = {{varCookieHead}}");
```

http.solveGroupJavaScript

Extracts a JavaScript string from a document by XPath using a String and stores it in the script's variables collection.

Format

The http.solveGroupJavaScript method has the following command format(s):

http.solveGroupJavaScript(*paths*);

http.solveGroupJavaScript(*path*);

http.solveGroupJavaScript(*documentXPath*, *paths*);

http.solveGroupJavaScript(*documentXPath*, *path*);

Command Parameters

paths
a String Array specifying the JavaScript paths to extract.

path
a String path specifying which JavaScript to extract. Path is a ":" separated list of arguments in the form `varName:vbOrJava:blkIndex:literalIndex:literal`, as follows:

- `varName`: the name of the variable to add to the script's variables collection.
- `vbOrJava`: 1 to parse JavaScript, 0 to parse VB Script.
- `blkIndex`: 0-based index of the script block in the HTML contents to parse.
- `literalIndex`: 0-based index of the dynamic JavaScript literal inside the block to extract.
- `literal`: Specify '0' to parse a String from JavaScript, or '1' to parse a number.

documentXPath
a String specifying the document that provides the contents.

Throws

SolveException
if the specified JavaScript path identifiers cannot be found.

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Extracts a JavaScript String from the page and stores it in the variable "varJavaScript" in the script's variables collection. It parses the first script block in the HTML contents for the third the dynamic JavaScript literal as a String from JavaScript.

```
http.solveGroupJavaScript("window[@index='0']", "varJavaScript:1:0:2:0");
info("Java Script = {{varJavaScript}}");
```

http.solveHeader

Extracts the specified header value using from the last request's response header acccording to header name.

The value will be stored in the specified variable.

Format

The http.solveHeader method has the following command format(s):

http.solveHeader(*varName*, *headerName*, *lastValue*);

Command Parameters

varName
a String specifying the variable name where the value will be stored. Must not be `null`.

headerName
a String specifying the header name.

lastValue
an optional String specifying the recorded or last known value of the value. May be `null`. For informational purposes only.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

SolveException
if the header cannot be solved.

Example

Extract the specified header value from the last request's response header and store it in the specified script variable.

```
http.solveHeader("varHeader", "X-ORACLE-BPMUI-CSRF",
 "EC92521CFE4D4D8FDEBC3EA6AD36EFADCF87A2B29794A325D7044B14CF8D14597BD62732CEBE1F75
C71FB3BF679F6B9E1B00705432C7137A3D64FCCE8571C060");
info("CSRF = {{varHeader}}");
```

http.solveRedirectNavs

Parses a value from the re-directed navigation's contents and stores it as a variable.

Always lookup from the latest redirected navigation. A SolveException is thrown if there is nothing found for all the re-directed navigations or no re-directed navigation at all.

This API can only be used manually in HTTP scripts.

This method honors the "Solve Variable Failed" error recovery setting. If "Solve Variable Failed" error recovery is not set to fail, then the script will continue running even if a non-optional variable cannot be solved.

Format

The http.solveRedirectNavs method has the following command format(s):

http.solveRedirectNavs(*varName*, *pattern*, *errorMsg*, *isOptional*, *sourceType*, *resultIndex*, *encodeOption*);

Command Parameters

varName
a String specifying the name of the variable to create. Must not be `null`.

pattern
a String specifying the Regular Expression pattern specifying what to extract from the most recent navigation's contents. May contain a transform expression. Must not be `null`.

errorMsg
an optional String specifying the error message to display if the pattern cannot be solved. If `null`, a meaningful error message is automatically generated.

isOptional
a boolean specifying if the pattern must be solved or not. Set to `true` if the pattern does not have to be solved. If the pattern cannot be solved, the method quietly returns.

sourceType
the Source enum specifying which contents to return as a String. If `null`, assumes Source.Html.

resultIndex
a 0-based index value specifying the specific result to retrieve if the pattern matches more than one value. If `null`, all results will be added into the variables collection.

encodeOption
an EncodeOptions enum specifying the encoding or decoding operation to perform on the variable's value after solving the variable. A `null` value is equivalent to `EncodeOptions.None`.

Throws

Exception
if redirected navigation cannot be solved.

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

SolveException
if the given pattern cannot be matched to the previous contents and the pattern is not optional.

Example

Parses a value from the most recent re-directed navigation's specified source using a Regular Expression and stores it in a variable. An optional error message returns if the pattern cannot be solved. The pattern is optional and does not have to be solved. Includes a result index value of 0 to specify it should retrieve the first match result and encode or decode using the specified options.

```
http.solveRedirectNavs("varTitle0", "<TITLE>(.+)</TITLE>",
 "Title not Found", true, Source.Html, 0, EncodeOptions.None);
info("Page Title = {{varTitle}}");
```

http.solveRefererHeader

Extracts the "Referer" header value using from the last request's response header using XPath notation.

The value will be stored in the specified variable.

Format

The http.solveRefererHeader method has the following command format(s):

http.solveRefererHeader(*varName*, *lastValue*);

http.solveRefererHeader(*varName*, *documentXPath*, *lastValue*);

Command Parameters

varName
a String specifying the variable name where the value will be stored. Must not be `null`.

lastValue
an optional String specifying the recorded or last known value of the value about to be parsed. May be `null`. For informational purposes only.

documentXPath
a String specifying the XPath to the document that provides the contents.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

SolveException
if the referer cannot be solved.

Example

Extract the "Referer" header value from the last request's response header and store it in the specified script variable.

```
http.solveRefererHeader("varRefHeader", "window[@index='0']",
 "http://myServer.com/");
info("Referer = {{varRefHeader}}");
```

http.solveXPath

Extracts a value from the specified document contents using XPath notation.

This API handles the following two rules:

- Web DOM rule: Combine *documentXPath* with *xpath* to create a full XPath, and find the element in the DOM jBrowser.
- XML Dom rule: Use *documentXPath* to find the document in DOM jBrowser, then use *xpath* to find the element in the document.

Format

The http.solveXPath method has the following command format(s):

http.solveXPath(*varName*, *xpath*, *lastValue*, *resultIndex*, *encodeOption*);

http.solveXPath(*varName*, *documentXPath*, *xpath*, *lastValue*, *resultIndex*, *encodeOption*);

Command Parameters

varName
a String specifying the variable name where the value will be stored. Must not be `null`.

xpath
a String specifying the XPath describing which value to parse from the last retrieved contents. Must not be `null`.

lastValue
an optional String specifying the recorded or last known value of the value about to be parsed. May be `null`. For informational purposes only.

resultIndex
a 0-based index value specifying the specific result to retrieve if the XPath returns multiple results. A `null` value specifies that all results should be returned.

encodeOption
an EncodeOptions enum specifying the encoding or decoding operation to perform on the variable's value after solving the variable. A `null` value is equivalent to `EncodeOptions.None`.

documentXPath
a String specifying the XPath to the document that provides the contents.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

HTMLParserException
on any failure to parse the HTML.

SolveException
if the XPath cannot be solved.

Example

Extracts a value from the browser's last retrieved contents using XPath notation and stores it in the specified script variable. Encode or decode using the specified options.

```
http.solveXPath("web.formaction.loginform", "/window[@index=0]",
 ".//FORM[@name='loginform']/@action", "default.asp", 0,
 EncodeOptions.None);
info("Login form = {{web.formaction.loginform}}");
http.solveXPath("web.formaction.loginform", "/window[@index=0]",
 ".//FORM[@name='loginform']/@action", "default.asp", 0,
 EncodeOptions.URLEncode);
http.solveXPath("web.formaction.loginform", "/window[@index=0]",
 ".//FORM[@name='loginform']/@action", "default.asp", 0,
 EncodeOptions.URLDecode);
http.solveXPath("web.formaction.loginform", "/window[@index=0]",
 ".//FORM[@name='loginform']/@action", "default.asp", 0,
 EncodeOptions.XMLEncode);
http.solveXPath("web.formaction.loginform", "/window[@index=0]",
 ".//FORM[@name='loginform']/@action", "default.asp", 0,
 EncodeOptions.XMLDecode);
http.solveXPath("web.formaction.loginform", "/window[@index=0]",
 ".//FORM[@name='loginform']/@action", "default.asp", 0,
 EncodeOptions.ConvertNewlinesToCRLF);
http.solveXPath("web.formaction.loginform", "/window[@index=0]",
 ".//FORM[@name='loginform']/@action", "default.asp", 0,
 EncodeOptions.ConvertNewlinesToLF);
http.solveXPath("web.formaction.loginform", "/window[@index=0]",
 ".//FORM[@name='loginform']/@action", "default.asp", 0,
 EncodeOptions.RemoveNewlines);
http.solveXPath("web.formaction.loginform", "/window[@index=0]",
 ".//FORM[@name='loginform']/@action", "default.asp", 0,
 EncodeOptions.EncodeURIComponent);
```

Source

The Source has the following values:

Table 6–4 List of Source Values

Value	Description
Html	Raw HTML, including tags.
ResponseHeader	HTTP response header.
DisplayContent	Received HTML content, excluding markup tags.
RequestHeader	HTTP request header.

http.text

Convenience method to return a `ParamCollection` object.

Format

The http.text method has the following command format(s):

http.text(*text*);

Command Parameters

text
a String specifying the raw text data to include in the parameter collection.

Returns

a new `ParamCollection` object from the specified text.

Example

Returns a `ParamCollection` from the specified text.

```
ParamCollection collection = http.text("abc");
if(collection.getText().equals("abc")){
   info("passed!");
}
else{
   warn("failed!");
}
```

http.urlEncode

URL-encodes a given string using a specific character set excludng the specified character array.

For example, the string, 'the file "abc" is in rootetc', would be encoded as: the+file+%22abc%22+is+in+%5Croot%5Cetc

Format

The http.urlEncode method has the following command format(s):

http.urlEncode(*s*);

http.urlEncode(*s*, *charset*);

http.urlEncode(*s*, *charset*, *noEncodeList*);

Command Parameters

s
a String specifying the string to encode.

charset
a String specifying the character set to use to encoded the string. May be `null`.

noEncodeList
a char[] array of characters not to encode. May be `null`.

Throws

URLEncodingException
if any characters cannot be encoded.

Returns

a URL-encoded String.

Example

Encodes the specified string using the specified character set and excludes the specified character list.

```
char[] noEncodeList = {'&', '\\', '?', '"', '*'};
//using script variable
getVariables().set("urlEncoded", HTTPService
 .urlEncode("the file \"abc\" is in \\root\\etc\\*.*",
   "ASCII", noEncodeList));
info("Encoded Text in script var = {{urlEncoded}}");
//using Java variable
String urlEncoded = HTTPService
 .urlEncode("the file \"abc\" & test?s in \\root\\etc\\*.*",
   "ASCII", noEncodeList);
info("The Encoded String is: " + urlEncoded);
```

http.validate

Validates the browser result for correctness.

This method does nothing if the Response already has an exception set.

If a content failure is found, this method sets an error for the response by calling `oracle.oats.scripting.modules.http.api.Response.setException(AbstractScriptException)`.

Scripts may add custom validation handlers to provide additional content validation, using http.addValidator.

Subclasses may override this method if they need to modify how the HTTP Service itself validates all browser responses.

Format

The http.validate method has the following command format(s):

http.validate(*r*);

Command Parameters

r
a Browser Response object. May be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Gets the last browser browser response for validation.

```
Response r = http.getLastBrowserResponse().getRequest()
 .getLastRequest().getResponse();
r.setResponseCode(600);
http.validate(r);
if (r.getException() != null&&r.getException().getMessage()
   .trim().equals("HTTP response code: 600  OK") ) {
 info("validate passed!");
}
else{
 warn("validate failed!");
}
```

See Also

http.addValidator

http.verifyResponseTime

Tests the server response time for the previous request without failing.

This method does nothing if the Response already has an exception set.

This method will never cause a script to fail, regardless of the server response test error recovery settings.

Use http.assertResponseTime to make the script fail when the test fails.

When playing back a script in Oracle Load Testing, always use Assertions; failed Verifications are *not* reported in the Oracle Load Test controller.

Subclasses may override this method if they need to modify how the HTTP Service itself validates server response times.

Format

The http.verifyResponseTime method has the following command format(s):

http.verifyResponseTime(*testName*, *minTime*, *maxTime*);

Command Parameters

testName
an optional String specifying a descriptive name describing what request is being tested. The name may appear in results and counter names.

minTime
a double specifying the minimum server response time (inclusive).

maxTime
a double specifying the maximum server response time (inclusive).

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

ResponseTimeException
if no previous response to validate, or if the response time does not fall within the given range *and* stopIterationOnFailure is `false`.

Example

Adds a Verify only, never fail Server Response test with a minimum of 1 second and a maximum of 15 seconds.

```
http.verifyResponseTime("myServerResponseTest", 1.0, 15.0);
```

http.verifyText

Searches the HTML contents in the document specified by XPath notation for the specified text pattern.

If the test fails, report a warning.

This method will never cause a script to fail, regardless of the text matching test error recovery settings.

Use http.assertText to make the script fail when the test fails.

When playing back a script in Oracle Load Testing, always use Assertions; failed Verifications are *not* reported in the Oracle Load Test controller.

Format

The http.verifyText method has the following command format(s):

http.verifyText(*testName*, *textToVerify*, *sourceType*, *textPresence*, *matchOption*);

http.verifyText(*testName*, *documentXPath*, *textToVerify*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

textToVerify
a String specifying the text to match on the page, or not match on the page if `TextPresence` is set to `TextPresence.PassIfPresent`.

sourceType
a Source enum specifying where to match the text: the HTML contents or HTTP response headers.

textPresence
a TestPresence enum specifying either `PassIfPresent` or `FailIfPresent`, depending on if you want the test to pass or fail if the text to match is present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a Regular Expression, Wildcard, or exact match.

documentXPath
a String specifying the XPath to the document that contains the contents.

Throws

AbstractScriptException
on any failure when attempting to verify the text.

Exception
on any exception while navigating or transforming the data.

Example

Adds a Verify only, never fail Text Matching tests for Display Content, Response Header, and Source HTML respectively.

`myTextMatchingTest1` passes if the text to match string is present in the Display Content and matches exactly.

`myTextMatchingTest2` passes if the text to match string is present in the Response Header and matches the Regular Expression.

`myTextMatchingTest3` fails if the text to match string is present in the Source HTML and matches the Wildcard.

```
http.verifyText("myTextMatchingTest1", "window[@index='0']",
 "match this text string", Source.DisplayContent,
 TextPresence.PassIfPresent, MatchOption.Exact);
http.verifyText("myTextMatchingTest2", "window[@index='0']",
 "jsessionid=(.+?)(?:\\\"|&)", Source.ResponseHeader,
 TextPresence.PassIfPresent, MatchOption.RegEx);
http.verifyText("myTextMatchingTest3", "window[@index='0']",
 "match this *", Source.Html,
 TextPresence.FailIfPresent, MatchOption.Wildcard);
```

http.verifyTitle

Searches the <title> tag in HTML content for the specified title pattern, reporting a warning if the test fails.

Format

The http.verifyTitle method has the following command format(s):

http.verifyTitle(*title*, *matchOption*);

http.verifyTitle(*documentXPath*, *title*, *matchOption*);

Command Parameters

title
a String specifying the text pattern specifying the expected title.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a Regular Expression, Wildcard, or exact match.

documentXPath
a String specifying the XPath to the document that contains the contents.

Throws

Exception
on any exception while navigating or transforming the data.

MatchException
if the assertion fails.

AbstractScriptException
on any other failure when attempting to assert the text.

Example

Adds Text Title tests for the HTML document title using exact match, Regular Expression match, and Wildcard match respectively.

```
http.verifyTitle("window[@index='0']", "Summary Report", MatchOption.Exact);
http.verifyTitle("window[@index='0']", "Summary Report For (.+?)",
MatchOption.RegEx);
http.verifyTitle("window[@index='0']", "Summary Report For ???",
MatchOption.Wildcard);
```

http.verifyXPath

Verify the text of a DOM element in the browser's last retrieved contents.

Compare this value to the *textToVerify* parameter. If the test fails, report a warning.

This method will never cause a script to fail, regardless of the XPath test error recovery settings. Use http.assertXPath to make the script fail when the test fails.

When playing back a script in Oracle Load Testing, always use Assertions; failed Verifications are *not* reported in the Oracle Load Test controller.

Format

The http.verifyXPath method has the following command format(s):

http.verifyXPath(*testName*, *xpath*, *resultIndex*, *textToVerify*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

xpath
a String specifying the XPath describing which value to parse from the last retrieved contents. Must not be `null`.

resultIndex
a 0-based index value specifying the specific result to retrieve if the XPath returns multiple results. A `null` value specifies that all results should be returned.

textToVerify
a String specifying the text to match, or not match with the XPath result if `TextPresence` is set to `TextPresence.PassIfPresent`.

textPresence
a TestPresence enum specifying either `PassIfPresent` or `FailIfPresent`, depending on if you want the test to pass or fail if the text to match is present or not.

matchOption
a MatchOption enum specifying how the text to match should be searched on the page, such as using a Regular Expression, Wildcard, or exact match.

Throws

AbstractScriptException
on any failure when attempting to verify the text.

Example

Adds Text Matching tests using XPath notation.

`myTextMatchingTest1` passes if the text to match string is present and matches exactly.

`myTextMatchingTest2` passes if the text to match string is present and matches the entire string exactly.

myTextMatchingTest3 passes if the text to match string is present and matches the Regular Expression.

myTextMatchingTest4 passes if the text to match string is present and matches the entire string from the Regular Expression.

myTextMatchingTest5 fails if the text to match string is present and matches the Wildcard.

```
http.verifyXPath("myTextMatchingTest1", ".//input[@name='j_id_id3']" +
 "/@value", 0, "match this text string",
 TextPresence.PassIfPresent, MatchOption.Exact);
http.verifyXPath("myTextMatchingTest2", ".//input[@name='j_id_id3']" +
 "/@value", 1, "match this text string",
 TextPresence.PassIfPresent, MatchOption.ExactEntireString);
http.verifyXPath("myTextMatchingTest3", ".//input[@name='j_id_id3']" +
 "/@value", 0, "jsessionid=(.+?)(?:\\\"|&)",
 TextPresence.PassIfPresent, MatchOption.RegEx);
http.verifyXPath("myTextMatchingTest4", ".//input[@name='j_id_id3']" +
 "/@value", 1, "jsessionid=(.+?)(?:\\\"|&)",
 TextPresence.PassIfPresent, MatchOption.RegExEntireString);
http.verifyXPath("myTextMatchingTest5", ".//input[@name='j_id_id3']" +
 "/@value", 0, "match this *",
 TextPresence.FailIfPresent, MatchOption.Wildcard);
```

http.window

Finds the window element from browser's DOM tree using the window index.

The element instance will be returned.

Format

The http.window method has the following command format(s):

http.window(*recId*, *xpathToWindow*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

xpathToWindow
a String specifying the XPath of the window element describing which value to parse from the DOM tree. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the window element's reference.

Example

Finds the window element from browser's DOM tree indexed as 0.

```
boolean http_window_exists = http.window(184, "window[" +
 "@index='0']").exists();
if(http_window_exists){
 info("WebHTTPWindow exists: " + http_window_exists);
}
else{
 warn("WebHTTPWindow exists: " + http_window_exists);
}
```

http.xmlPost

Posts an XML string to a web server using the HTTP POST method.

Format

The http.xmlPost method has the following command format(s):

http.xmlPost(*recid*, *urlPath*, *querystring*, *postdata*, *headers*, *encode*, *reqCharset*, *respCharset*);

Command Parameters

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

urlPath
a String specifying the URL to request, excluding query string data. May contain {{ }} syntax for transforming. Must not be `null`.

querystring
a `ParamCollection` specifying the query string data. May be `null`.

postdata
a String specifying the XML data to POST to the URL's host. Must not be `null`.

headers
a Header array of additional headers to add/remove from the request before submitting it to the server.

encode
a boolean specifying the parameter encoding. Set to `true` to URL-encode the querystring parameters before submitting them to the sever.

reqCharset
a String specifying the character set to use if URL-encoding any of the request parameters.

respCharset
a String specifying the character set to use when reading the response contents.

Throws

Exception
on any exception while navigating or transforming the data.

Example

Specifies an XML POST navigation with querystrings, postdata, headers, URL encoding, and Character sets.

```
http.xmlPost(0, "http://myServer.com",
 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");
```

See Also

http.querystring for creating a parameter collection.

7

Siebel Load Module

This chapter provides a complete listing and reference for the methods in the OpenScript SiebelService Class of Siebel Load Module Application Programming Interface (API).

7.1 SiebelService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript SiebelService API.

7.1.1 Alphabetical Command Listing

The following table lists the SiebelService API methods in alphabetical order.

Table 7–1 List of SiebelService Methods

Method	Description
siebel.getSettings	Gets the Siebel Settings object.
siebel.solve	Parse a Siebel value from the most recent navigation's contents and store it as a variable.

The following sections provide detailed reference information for each method and enum in the SiebelService Class of Siebel Load Module Application Programming Interface.

siebel.getSettings

Gets the Siebel Settings object.

Format

The siebel.getSettings method has the following command format(s):

siebel.getSettings( );

Returns

Settings pertaining to the Siebel Service.

Example

Gets the Siebel Settings object and check validation enabled.

```
import oracle.oats.scripting.modules.siebel.api.SiebelSettings;
//[...]
SiebelSettings settings = siebel.getSettings();
settings.setIsSiebelValidationEnabled(true);
boolean isvalidationEnabled = settings.getIsSiebelValidationEnabled();
info("isvalidationEnabled = " + isvalidationEnabled);
```

siebel.solve

Parse a Siebel value from the most recent navigation's contents and store it as a variable.

This method honors the "Solve Variable Failed" error recovery setting. If "Solve Variable Failed" error recovery is not set to fail, then the script will continue running even if a non-optional variable cannot be solved.

Format

The siebel.solve method has the following command format(s):

siebel.solve(*varName*, *pattern*, *errorMsg*, *isOptional*, *sourceType*);

Command Parameters

varName
a String specifying the name of the variable to create. Must not be `null`.

pattern
a String specifying the Siebel pattern specifying what to extract from the most recent navigation's contents. May contain a transform expression. Must not be `null`.

errorMsg
a String specifying an optional error message to display if the pattern cannot be solved. If `null`, a meaningful error message is automatically generated.

isOptional
a Boolean specifying `true` if the pattern does not have to be solved. If the pattern cannot be solved, the method quietly returns.

sourceType
a Source enum specifying the part of the most recent navigation's contents (i.e. body or headers) against which to solve the pattern. If `null`, this method will assume `Source.Html`.

Throws

Exception
if any failure.

SolveException
If the given pattern cannot be matched to the previous contents and the pattern is not optional.

Example

Parse a Siebel value from the most recent navigation's contents and store it as a variable.

```
http.post(107,
  "http://siebel005.us.oracle.com/callcenter_enu/start.swe",
  null, http.postdata(http.param("SWEC", "{
}"),
   http.param("SWERPC", "1"),
   http.param("SWENeedContext", "false"),
```

```
    http.param("SWEView", "Account List View"),
    http.param("SWECmd", "GotoView"),
    http.param("SWEKeepContext", "1")), null, true,
   "UTF8", "UTF8");
{
 siebel.solve(
  "_Siebel_CORRLIB_FIELD_1294336896147__S_BC1_S17_R01_F01_#_3 scripts co",
  "_Siebel_CORRLIB_FIELD__TEXT", null, false,
  Source.Html);
 siebel.solve(
  "_Siebel_CORRLIB_FIELD_1294336896147__S_BC1_S17_R01_F04_#_Qualified",
  "_Siebel_CORRLIB_FIELD__TEXT", null, false,
  Source.Html);
 siebel.solve(
  "_Siebel_CORRLIB_FIELD_1294336896147__S_BC1_S17_R01_F03_#_",
  "_Siebel_CORRLIB_FIELD__PHONE", null, false,
  Source.Html);
 siebel.solve(
  "_Siebel_CORRLIB_FIELD_1294336896147__S_BC1_S17_R01_F05_#_",
  "_Siebel_CORRLIB_FIELD__TEXT", null, false,
  Source.Html);
 siebel.solve(
  "_Siebel_CORRLIB_FIELD_1294336896147__S_BC1_S17_R01_F02_#_abc",
  "_Siebel_CORRLIB_FIELD__TEXT", null, false,
  Source.Html);
 siebel.solve(
  "_Siebel_CORRLIB_ROWID_1294336896147__S_BC1_S17_R01_FID_#_1-J8TZ",
  "_Siebel_CORRLIB_ROWID", null, false,
  Source.Html);
}
http.get(111,"http://siebel005.us.oracle.com/callcenter_enu/start.swe",
  http.querystring(
   http.param("SWECmd","GetViewLayout"),
   http.param("SWEView","Account List View"),
   http.param("SWEVI", ""),
   http.param("SWEVLC",
    "{{SWEVLC,0-1_Siebel+Universal+Agent_37%7c1119321220%7c0_0_19221_00_L}}")),
  null, true, "UTF8", "UTF8");
```

Part III

Functional Testing Modules API Reference

This Part of the OpenScript Programmer's Reference provides a complete listing and reference for the methods in the OpenScript Functional Testing Modules Application Programming Interface (API).

Each chapter in this part contains alphabetical listings and detailed command references for the methods in each class.

This Part contains the following chapters:

8 Applet Module

This chapter provides a complete listing and reference for the methods in the OpenScript AppletService Class of Applet Module Application Programming Interface (API).

8.1 AppletService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript AppletService API.

8.1.1 Alphabetical Command Listing

The following table lists the AppletService API methods in alphabetical order.

Table 8–1 List of AppletService Methods

Method	Description
applet.appWindow	Identifies a top window object that doesn't have an owner, by its path.
applet.button	Identifies a button object by its path.
applet.checkBox	Identifies a check box object by its path.
applet.comboBox	Identifies a combo box object by its path.
applet.dcmLayoutEditor	Identifies an OcgleDcm layout editor object by its path.
applet.dialog	Identifies a dialog object by its path.
applet.dtree	Identifies a DTree object by its path.
applet.expansionTree	Identifies an expansion tracking tree by its path.
applet.grid	Identifies a grid object by its path.
applet.infiniteScrollBar	Identifies a tree browser object by its path.
applet.javaObject	Identifies a component object by its path if this component could not be recognized as a specific type of object.
applet.radioButton	Identifies a radio button object by its path.
applet.scrollBar	Identifies a scroll bar object by its path.
applet.tab	Identifies a tab bar object by its path.
applet.textField	Identifies a text field object by its path.
applet.timeBrowser	Identifies a time browser object by its path.
applet.timeline	Identifies a time line object by its path.

Table 8–1 (Cont.) List of AppletService Methods

Method	Description
applet.toolBar	Identifies a tool bar object by its path.
applet.treeBrowser	Identifies a tree browser object by its path.

The following sections provide detailed reference information for each method and enum in the AppletService Class of Applet Module Application Programming Interface.

applet.appWindow

Identifies a top window object that doesn't have an owner, by its path.

Format

The applet.appWindow method has the following command format(s):

applet.appWindow(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the window object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the window object.

Example

Performs an action on a window object specified by its path.

```
applet.appWindow(399, "/applet:TJavaAppWindow[@text='Oracle Applications' " +
 "or @posIndex='1']")
 .selectMenu("File|Close");
```

applet.button

Identifies a button object by its path.

Format

The applet.button method has the following command format(s):

applet.button(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the button object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the button object.

Example

Performs an action on a button object specified by its path.

```
applet.button(2967, "/applet:TJavaDialog[@posIndex='0']" +
 "/applet:TJavaButton[@text=' OK ' or @posIndex='0']")
 .click();
```

applet.checkBox

Identifies a check box object by its path.

Format

The applet.checkBox method has the following command format(s):

applet.checkBox(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the check box object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the check box object.

Example

Performs an action on a check box object specified by its path.

```
applet.checkBox(8956, "/applet:TJavaDialog[@text='Source Questions - " +
  "Create Conditional Blocks' or @posIndex='0']" +
 "/applet:TJavaCheckbox[@text='AE_ANY1' or @posIndex='0']")
 .setCheck(true);
```

applet.comboBox

Identifies a combo box object by its path.

Format

The applet.comboBox method has the following command format(s):

applet.comboBox(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the combo box object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the text field object.

Example

Performs an action on a combo box object specified by its path.

```
applet.comboBox(8092, "/applet:TJavaDcmLayoutEditor[" +
  "@text='AE_IND SUBSET 1 LAYOUT 1 Adverse Events' or @posIndex='4']" +
 "/applet:TJavaToolbar[@posIndex='0']" +
 "/applet:TJavaCombobox[@posIndex='0']")
 .selectItem("200%");
```

applet.dcmLayoutEditor

Identifies an OcgleDcm layout editor object by its path.

Format

The applet.dcmLayoutEditor method has the following command format(s):

applet.dcmLayoutEditor(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the layout editor object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the OcgleDcm layout editor object.

Example

Performs an action on an Ocgle DCM layout editor object specified by its path.

```
applet.dcmLayoutEditor(2962, "/applet:TJavaDcmLayoutEditor[@text='AE_IND " +
 "SUBSET 1 LAYOUT 1 Adverse Events' or @posIndex='4']")
 .selectMenu("File|Validate");
```

applet.dialog

Identifies a dialog object by its path.

Format

The applet.dialog method has the following command format(s):

applet.dialog(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the dialog object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the dialog object.

Example

Performs an action on a dialog object specified by its path.

```
applet.dialog(1097,"/applet:TJavaDcmLayoutEditor[@text='AE_IND SUBSET 1 " +
  "LAYOUT 1 Adverse Events' or @posIndex='4']" +
 "/applet:TJavaDialog[@text='Validation' or @posIndex='-1']")
 .close();
```

applet.dtree

Identifies a DTree object by its path.

Format

The applet.dtree method has the following command format(s):

applet.dtree(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the DTree object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the DTree object.

Example

Performs an action on a DTree object specified by its path.

```
applet.dtree(110, "/applet:TJavaWindow[@text='Edit Schedule' " +
  or @posIndex='4']" +
 "/applet:TJavaDTree[@posIndex='0']")
 .selectNode("49010");
```

applet.expansionTree

Identifies an expansion tracking tree by its path.

Format

The applet.expansionTree method has the following command format(s):

applet.expansionTree(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the expansion tracking tree object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the expansion tracking tree object.

Example

Performs an action on an expansion tracking tree object specified by its path.

```
applet.expansionTree(3284,"/applet:TJavaWindow[@text='Flow Workstation  (M1)' " +
  "or @posIndex='3']" +
 "/applet:TJavaExpansionTree[@posIndex='0']")
 .expandNode("Vision Pad:CASE|Assemblies");
```

applet.grid

Identifies a grid object by its path.

Format

The applet.grid method has the following command format(s):

applet.grid(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the grid object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the grid object.

Example

Performs an action on a grid object specified by its path.

```
applet.grid(3490, "/applet:TJavaWindow[@text='Edit Schedule' " +
  "or @posIndex='5']" +
 "/applet:TJavaGrid[@posIndex='0']")
 .editCell(3, 1);
```

applet.infiniteScrollBar

Identifies a tree browser object by its path.

Format

The applet.infiniteScrollBar method has the following command format(s):

applet.infiniteScrollBar(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the tree browser object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the tree browser object.

Example

Performs an action on a tree browser object specified by its path.

```
applet.infiniteScrollBar(152, "/applet:TJavaWindow[@text='Job Scheduling
Workbench' " +
  or @posIndex='3']
 "/applet:TJavaInfiniteScrollBar[@posIndex='0']")
 .scroll(-349);
```

applet.javaObject

Identifies a component object by its path if this component could not be recognized as a specific type of object. Gives a default representation to the component object.

Format

The applet.javaObject method has the following command format(s):

applet.javaObject(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the scroll bar object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the component object.

Example

Performs an action on a component object specified by its path.

```
applet.javaObject(6315, "/applet:TJavaObject[@posIndex='4']")
 .mouseClick();
```

applet.radioButton

Identifies a radio button object by its path.

Format

The applet.radioButton method has the following command format(s):

applet.radioButton(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the radio button object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the radio button object.

Example

Performs an action on a radio button object specified by its path.

```
applet.radioButton(533, "/applet:TJavaWindow[@text='Filter' " +
  "or @posIndex='4']" +
 "/applet:TJavaRadioButton[@text='Date' or @posIndex='8']")
 .select();
```

applet.scrollBar

Identifies a scroll bar object by its path.

Format

The applet.scrollBar method has the following command format(s):

applet.scrollBar(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the scroll bar object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the scroll bar object.

Example

Performs an action on a scroll bar object specified by its path.

```
applet.scrollBar(6658, "/applet:TJavaDcmLayoutEditor[@text='AE_IND SUBSET 1 " +
  "LAYOUT 1 Adverse Events' or @posIndex='4']" +
 "/applet:TJavaScrollbar[@posIndex='0']")
 .setPosition(0);
```

applet.tab

Identifies a tab bar object by its path.

Format

The applet.tab method has the following command format(s):

applet.tab(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the tab bar object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the tab bar object.

Example

Performs an action on a tab bar object specified by its path.

```
applet.tab(725,"/applet:TJavaWindow[@text='Flow Workstation  (M1)' " +
  "or @posIndex='3']" +
 "/applet:TJavaTabBar[posIndex=0 and tClass='TJavaTabBar']")
 .select("Properties");
```

applet.textField

Identifies a text field object by its path.

Format

The applet.textField method has the following command format(s):

applet.textField(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the text field object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the text field object.

Example

Performs an action on a text field object specified by its path.

```
applet.textField(7945, "/applet:TJavaDcmLayoutEditor[@text='AE_IND SUBSET 1 " +
  "LAYOUT 1 Adverse Events' or @posIndex='4']" +
 "/applet:TJavaTextfield[@posIndex='31']")
 .setSelectedRange(0, 0);
```

applet.timeBrowser

Identifies a time browser object by its path.

Format

The applet.timeBrowser method has the following command format(s):

applet.timeBrowser(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the time browser object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the time browser object.

Example

Performs an action on a time browser object specified by its path.

```
applet.timeBrowser(145, "/applet:TJavaWindow[@text='Job Scheduling Workbench' " +
  "or @posIndex='3']" +
 "/applet:TJavaTimeBrowser[@posIndex='0']")
 .move("243645 : XA1000", "07-DEC-2006 17:03:00", "05-DEC-2006 15:45:55");
```

applet.timeline

Identifies a time line object by its path.

Format

The applet.timeline method has the following command format(s):

applet.timeline(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the time line object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the time line object.

Example

Performs an action on a time line object specified by its path.

```
applet.timeline(186, "/applet:TJavaWindow[@text='Job Scheduling Workbench' " +
  "or @posIndex='3']" +
 "/applet:TJavaTimeline[@posIndex='0']")
 .selectPopupMenu("Months");
```

applet.toolBar

Identifies a tool bar object by its path.

Format

The applet.toolBar method has the following command format(s):

applet.toolBar(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the tool bar object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the tool bar object.

Example

Performs an action on a tool bar object specified by its path.

```
applet.toolBar(7086, "/applet:TJavaDcmLayoutEditor[@text='AE_IND SUBSET 1 " +
  LAYOUT 1 Adverse Events' or @posIndex='4']" +
 "/applet:TJavaToolbar[@posIndex='0']")
 .clickItemByLabel("Center items horizontally");
```

applet.treeBrowser

Identifies a tree browser object by its path.

Format

The applet.treeBrowser method has the following command format(s):

applet.treeBrowser(*recId*, *xPath*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

xPath
a String specifying the path to identify the tree browser object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the tree browser object.

Example

Performs an action on a tree browser object specified by its path.

```
applet.treeBrowser(220, "/applet:TJavaWindow[@text='Job Scheduling Workbench' " +
  "or @posIndex='3']" +
 "/applet:TJavaTreeBrowser[@posIndex='0']")
 .selectNode("NC132 : CM52295|:10");
```

9

Adobe Flex Functional Module

This chapter provides a complete listing and reference for the methods in the OpenScript FlexFTService Class of Flex Functional Module Application Programming Interface (API).

9.1 FlexFTService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript FlexFTService API.

9.1.1 Alphabetical Command Listing

The following table lists the FlexFTService API methods in alphabetical order.

Table 9–1 List of FlexFTService Methods

Method	Description
flexFT.accordion	Creates a Flex Accordion element.
flexFT.alert	Creates a Flex Alert element.
flexFT.application	Creates a Flex Application element.
flexFT.areaChart	Creates a Flex AreaChart element.
flexFT.areaSeries	Creates a Flex AreaSeries element.
flexFT.barChart	Creates a Flex BarChart element.
flexFT.barSeries	Creates a Flex BarSeries element.
flexFT.box	Creates a Flex Box element.
flexFT.bubbleSeries	Creates a Flex BubbleSeries element.
flexFT.button	Creates a Flex Button element.
flexFT.buttonBar	Creates a Flex ButtonBar element.
flexFT.cartesianChart	Creates a Flex CartesianChart element.
flexFT.checkbox	Creates a Flex CheckBox element.
flexFT.colorPicker	Creates a Flex ColorPicker element.
flexFT.columnChart	Creates a Flex ColumnChart element.
flexFT.columnSeries	Creates a Flex ColumnSeries element.
flexFT.combobox	Creates a Flex ComboBox element.
flexFT.dataGrid	Creates a Flex DataGrid element.

Table 9–1 (Cont.) List of FlexFTService Methods

Method	Description
flexFT.dateChooser	Creates a Flex DateChooser element.
flexFT.dateField	Creates a Flex DateField element.
flexFT.dividedBox	Creates a Flex DividedBox element.
flexFT.lineChart	Creates a Flex LineChart element.
flexFT.lineSeries	Creates a Flex LineSeries element.
flexFT.linkBar	Creates a Flex LinkBar element.
flexFT.list	Creates a Flex List element.
flexFT.menu	Creates a Flex Menu element.
flexFT.menubar	Creates a Flex MenuBar element.
flexFT.numericStepper	Creates a Flex NumericStepper element.
flexFT.panel	Creates a Flex Panel element.
flexFT.pieChart	Creates a Flex PieChart element.
flexFT.pieSeries	Creates a Flex PieSeries element.
flexFT.plotSeries	Creates a Flex PlotSeries element.
flexFT.popupButton	Creates a Flex PopUpButton element.
flexFT.progressBar	Creates a Flex ProgressBar element.
flexFT.radioButton	Creates a Flex RadioButton element.
flexFT.scrollbar	Creates a Flex ScrollBar element.
flexFT.slider	Creates a Flex Slider element.
flexFT.toggleButtonBar	Creates a Flex ToggleButtonBar element.
flexFT.tree	Creates a Flex Tree element.

The following sections provide detailed reference information for each method and enum in the FlexFTService Class of Flex Functional Module Application Programming Interface.

flexFT.accordion

Creates a Flex Accordion element.

Format

The flexFT.accordion method has the following command format(s):

flexFT.accordion(*path*);

flexFT.accordion(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new Accordion.

Example

Performs an action on Flex Accordion element.

```
String accordion="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='explorer' " +
   "and @label='' " +
   "and @automationIndex='index:-1' " +
   "and @automationName='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @automationName='index:0']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @automationName='index:1']" +
  "/flex:panel[@automationClassName='FlexPanel' " +
   "and @className='loaderPanel' " +
   "and @label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='swfLoader' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='AccordionExample' " +
   "and @label='' " +
   "and @automationIndex='index:1' " +
```

```
   "and @automationName='containers/AccordionExample.swf' " +
   "and @id='null']" +
  "/flex:panel[@automationClassName='FlexPanel' " +
   "and @className='mx.containers.Panel' " +
   "and @label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='Accordion%20Container%20Example' " +
   "and @id='null']" +
  "/flex:accordion[@automationClassName='FlexAccordion' " +
   "and @className='mx.containers.Accordion' " +
   "and @label='' " +
   "and @automationIndex='index:1' " +
   "and @automationName='accordion' " +
   "and @id='accordion']";
flexFT.accordion(10, accordion).change("Accordion Button for Panel 2");
try{
 String result=flexFT.accordion(15, accordion).getSelectedChild();
 if (!result.equals("Accordion Button for Panel 2")) {
  warn("getSelectedChild returned " + result);
 }
 }catch(Exception exp) {warn("getSelectedChild exception: "
  + exp.getMessage());}
try{
 Integer result=flexFT.accordion(20, accordion).getSelectedIndex();
 if (result !=1) {
  warn("getSelectedIndex returned " + result);
 }
 }catch(Exception exp) {warn("getSelectedIndex exception: "
  + exp.getMessage());}
```

flexFT.alert

Creates a Flex Alert element.

Format

The flexFT.alert method has the following command format(s):

flexFT.alert(*path*);

flexFT.alert(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new Alert.

Example

Performs an action on a Flex Alert element.

```
String Alert="/web:window[@index='0']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:alert[@automationName='Color%20Selection' " +
   "and @automationClassName='FlexAlert' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
   "and @className='mx.controls.Alert' " +
   "and @id='null']";
try{
 String result=flexFT.alert(10, Alert).getText();
 if (!result.equals("Select a color:")) {
  warn("getText returned " + result);
 }
 }catch(Exception exp) {warn("getText exception: "
   + exp.getMessage());}
```

flexFT.application

Creates a Flex Application element.

Format

The flexFT.application method has the following command format(s):

flexFT.application(*path*);

flexFT.application(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new Application.

Example

Performs an action on Flex Application element.

```
String application="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='explorer' " +
   "and @label='' " +
   "and @automationIndex='index:-1' " +
   "and @automationName='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @automationName='index:0']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @automationName='index:1']" +
  "/flex:panel[@automationClassName='FlexPanel' " +
   "and @className='loaderPanel' " +
   "and @label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='swfLoader' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='SimpleApplicationExample' " +
   "and @label='' and @automationIndex='index:1' " +
   "and @automationName='core/SimpleApplicationExample.swf' " +
```

```
   "and @id='null']";
flexFT.application(10, application).click(KeyModifier.None);
try{
 String result=flexFT.application(15, application).getUrl();
 if (!result.equals("http://example.com/core/SimpleApplicationExample.swf")) {
  warn("getUrl returned " + result);
 }
 }catch(Exception exp) {warn("getUrl exception: "
  + exp.getMessage());}
```

flexFT.areaChart

Creates a Flex AreaChart element.

Format

The flexFT.areaChart method has the following command format(s):

flexFT.areaChart(*path*);

flexFT.areaChart(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new AreaChart.

Example

Performs an action on Flex AreaChart element.

```
String areaChart="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/Line_AreaChartExample.swf' " +
   "and @label='' " +
```

```
   "and @className='Line_AreaChartExample' " +
   "and @id='null']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='LineChart%20and%20AreaChart%20Controls%20Example' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']" +
  "/flex:areaChart[@automationIndex='index:5' " +
   "and @automationClassName='FlexAreaChart' " +
   "and @automationName='Areachart' " +
   "and @className='mx.charts.AreaChart' " +
   "and @id='Areachart']";
flexFT.areaChart(10, areaChart).click(KeyModifier.None);
try{
 String result=flexFT.areaChart(15, areaChart).getType();
 if (!result.equals("overlaid")) {
  warn("getType returned " + result);
 }
 }catch(Exception exp) {warn("getType exception: "
  + exp.getMessage());}
```

flexFT.areaSeries

Creates a Flex AreaSeries element.

Format

The flexFT.areaSeries method has the following command format(s):

flexFT.areaSeries(*path*);

flexFT.areaSeries(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new AreaSeries.

Example

Performs an action on Flex AreaSeries element.

```
String areaSeries="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='charts/Line_AreaChartExample.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
   "and @className='Line_AreaChartExample' " +
   "and @id='null']" +
  "/flex:panel[@automationName='LineChart%20and%20AreaChart%20Controls%20Example'
" +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']" +
  "/flex:areaChart[@automationName='Areachart' " +
   "and @automationClassName='FlexAreaChart' " +
   "and @automationIndex='index:5' " +
   "and @className='mx.charts.AreaChart' " +
   "and @id='Areachart']" +
  "/flex:areaSeries[@automationName='Expenses' " +
   "and @automationClassName='FlexAreaSeries' " +
   "and @automationIndex='index:1' " +
   "and @className='mx.charts.series.AreaSeries' " +
   "and @id='_Line_AreaChartExample_AreaSeries2']";
flexFT.areaSeries(10, areaSeries).clickSeries(2.0);
try{
 String result=flexFT.areaSeries(15, areaSeries).getXField();
 if (!result.equals("")) {
  warn("getXField returned " + result);
 }
 }catch(Exception exp) {warn("getXField exception: "
  + exp.getMessage());}
try{
 String result=flexFT.areaSeries(20, areaSeries).getYField();
 if (!result.equals("Expenses")) {
  warn("getYField returned " + result);
 }
 }catch(Exception exp) {warn("getYField exception: "
  + exp.getMessage());}
```

flexFT.barChart

Creates a Flex BarChart element.

Format

The flexFT.barChart method has the following command format(s):

flexFT.barChart(*path*);

flexFT.barChart(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new BarChart.

Example

Performs an action on Flex BarChart element.

```
String barChart="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/Column_BarChartExample.swf' " +
   "and @label='' " +
```

```
   "and @className='Column_BarChartExample' " +
   "and @id='null']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='ColumnChart%20and%20BarChart%20Controls%20Example' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']" +
  "/flex:barChart[@automationIndex='index:5' " +
   "and @automationClassName='FlexBarChart' " +
   "and @automationName='bar' " +
   "and @className='mx.charts.BarChart' " +
   "and @id='bar']";
flexFT.barChart(10, barChart).click(KeyModifier.None);
try{
 Double result=flexFT.barChart(15, barChart).getBarWidthRatio();
 if (result !=0.65) {
  warn("getBarWidthRatio returned " + result);
 }
 }catch(Exception exp) {warn("getBarWidthRatio exception: "
  + exp.getMessage());}
try{
 Double result=flexFT.barChart(20, barChart).getMaxBarWidth();
 if (result !=null) {
  warn("getMaxBarWidth returned " + result);
 }
 }catch(Exception exp) {warn("getMaxBarWidth exception: "
  + exp.getMessage());}
```

flexFT.barSeries

Creates a Flex BarSeries element.

Format

The flexFT.barSeries method has the following command format(s):

flexFT.barSeries(*path*);

flexFT.barSeries(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new BarSeries.

Example

Performs an action on Flex BarSeries element.

```
String barSeries= "/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/Column_BarChartExample.swf' " +
   "and @label='' " +
```

```
    "and @className='Column_BarChartExample' " +
    "and @id='null']" +
   "/flex:panel[@automationIndex='index:0' " +
    "and @automationClassName='FlexPanel' " +
    "and @automationName='ColumnChart%20and%20BarChart%20Controls%20Example' " +
    "and @label='' " +
    "and @className='mx.containers.Panel' " +
    "and @id='null']" +
   "/flex:barChart[@automationIndex='index:5' " +
    "and @automationClassName='FlexBarChart' " +
    "and @automationName='bar' " +
    "and @className='mx.charts.BarChart' " +
    "and @id='bar']" +
   "/flex:barSeries[@automationIndex='index:1' " +
    "and @automationClassName='FlexBarSeries' " +
    "and @automationName='Silver;Country' " +
    "and @className='mx.charts.series.BarSeries' " +
    "and @id='_Column_BarChartExample_BarSeries2']";
flexFT.barSeries(10, barSeries).clickSeries(1.0);
try{
 String result=flexFT.barSeries(15, barSeries).getXField();
 if (!result.equals("Silver")) {
  warn("getXField returned " + result);
 }
 }catch(Exception exp) {warn("getXField exception: "
  + exp.getMessage());}
try{
 String result=flexFT.barSeries(20, barSeries).getYField();
 if (!result.equals("Country")) {
  warn("getYField returned " + result);
 }
 }catch(Exception exp) {warn("getYField exception: "
  + exp.getMessage());}
```

flexFT.box

Creates a Flex Box element.

Format

The flexFT.box method has the following command format(s):

flexFT.box(*path*);

flexFT.box(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new Box.

Example

Performs an action on Flex Box element.

```
String box="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='explorer' " +
   "and @label='' " +
   "and @automationIndex='index:-1' " +
   "and @automationName='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @automationName='index:0']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @automationName='index:1']" +
  "/flex:panel[@automationClassName='FlexPanel' " +
   "and @className='loaderPanel' " +
   "and @label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='swfLoader' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='SimpleCanvasExample' " +
   "and @label='' " +
   "and @automationIndex='index:1' " +
```

```
   "and @automationName='containers/SimpleCanvasExample.swf' " +
   "and @id='null']" +
  "/flex:panel[@automationClassName='FlexPanel' " +
   "and @className='mx.containers.Panel' " +
   "and @label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='Canvas%20Container%20Example' " +
   "and @id='null']" +
  "/flex:box[@automationClassName='FlexBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:6' " +
   "and @className='mx.containers.VBox' " +
   "and @automationName='index:6']";
flexFT.box(10, box).click(KeyModifier.None);
try{
 String result=flexFT.box(15, box).getDirection();
 if (!result.equals("vertical")) {
  warn("getDirection returned " + result);
 }
 }catch(Exception exp) {warn("getDirection exception: "
  + exp.getMessage());}
```

flexFT.bubbleSeries

Creates a Flex BubbleSeries element.

Format

The flexFT.bubbleSeries method has the following command format(s):

flexFT.bubbleSeries(*path*);

flexFT.bubbleSeries(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new BubbleSeries.

Example

Performs an action on Flex BubbleSeries element.

```
String bubbleSeries="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/BubbleChartExample.swf' " +
   "and @label='' " +
```

```
    "and @className='BubbleChartExample' " +
    "and @id='null']" +
   "/flex:panel[@automationIndex='index:0' " +
    "and @automationClassName='FlexPanel' " +
    "and @automationName='BubbleChart%20Control%20Example' " +
    "and @label='' " +
    "and @className='mx.containers.Panel' " +
    "and @id='null']" +
   "/flex:cartesianChart[@automationIndex='index:0' " +
    "and @automationClassName='FlexCartesianChart' " +
    "and @automationName='bubblechart' " +
    "and @className='mx.charts.BubbleChart' " +
    "and @id='bubblechart']" +
   "/flex:bubbleSeries[@automationIndex='index:0' " +
    "and @automationClassName='FlexBubbleSeries' " +
    "and @automationName='Profit/Expenses/Amount' " +
    "and @className='mx.charts.series.BubbleSeries' " +
    "and @id='_BubbleChartExample_BubbleSeries1']";
flexFT.bubbleSeries(10, bubbleSeries).clickSeries(1.0);
try{
 String result=flexFT.bubbleSeries(15, bubbleSeries).getXField();
 if (!result.equals("Profit")) {
 warn("getXField returned " + result);
  }
 }catch(Exception exp) {warn("getXField exception: "
  + exp.getMessage());}
try{
 String result=flexFT.bubbleSeries(20, bubbleSeries).getYField();
 if (!result.equals("Expenses")) {
  warn("getYField returned " + result);
 }
 }catch(Exception exp) {warn("getYField exception: "
  + exp.getMessage());}
```

flexFT.button

Creates a Flex Button element.

Format

The flexFT.button method has the following command format(s):

flexFT.button(*path*);

flexFT.button(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new Button

Example

Performs an action on a Flex Button element.

```
String Button="/web:window[@index='0']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' a" +
   "nd @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/SimpleAlert.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
   "and @className='SimpleAlert' " +
   "and @id='null']" +
```

```
  "/flex:panel[@automationName='Alert%20Control%20Example' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' and @id='null']" +
  "/flex:button[@automationName='Click%20Me' " +
   "and @automationClassName='FlexButton' " +
   "and @automationIndex='index:6' " +
   "and @label='Click%20Me' " +
   "and @className='mx.controls.Button' " +
   "and @id='null']";
flexFT.button(10, Button).click(KeyModifier.None);
```

flexFT.buttonBar

Creates a Flex ButtonBar element.

Format

The flexFT.buttonBar method has the following command format(s):

flexFT.buttonBar(*path*);

flexFT.buttonBar(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new ButtonBar.

Example

Performs an action on Flex buttonBar element.

```
String ButtonBar="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @id='explorer' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:0' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @label='']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:1' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @label='']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @id='swfLoader' " +
   "and @className='loaderPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:application[@automationName='controls/ButtonBarExample.swf' " +
   "and @id='null' " +
   "and @className='ButtonBarExample' " +
   "and @automationIndex='index:1' " +
```

```
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:panel[@automationName='ButtonBar%20Control%20Example' " +
   "and @id='null' " +
   "and @className='mx.containers.Panel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:buttonBar[@id='null' " +
   "and @automationIndex='index:2' " +
   "and @automationClassName='FlexButtonBar' " +
   "and @automationName='index:2' " +
   "and @className='mx.controls.ButtonBar' " +
   "and @label='']";
flexFT.buttonBar(10, ButtonBar).change("Flash");
{
 think(10.5);
}
flexFT.buttonBar(15, ButtonBar).type("EnterKey",
 KeyModifier.None);
```

flexFT.cartesianChart

Creates a Flex CartesianChart element.

Format

The flexFT.cartesianChart method has the following command format(s):

flexFT.cartesianChart(*path*);

flexFT.cartesianChart(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new CartesianChart.

Example

Performs an action on Flex CartesianChart element.

```
String cartesianChart="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/PlotChartExample.swf' " +
   "and @label='' " +
```

```
   "and @className='PlotChartExample' " +
   "and @id='null']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='PlotChart%20Control%20Example' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']" +
  "/flex:cartesianChart[@automationIndex='index:0' " +
   "and @automationClassName='FlexCartesianChart' " +
   "and @automationName='plot' " +
   "and @className='mx.charts.PlotChart' " +
   "and @id='plot']";
flexFT.cartesianChart(10, cartesianChart).click(KeyModifier.None); *
try{
 String result=flexFT.cartesianChart(15, cartesianChart).getTextAlign() ;
 if (!result.equals("left")) {
  warn("getTextAlign returned " + result);
 }
 }catch(Exception exp) {warn("getTextAlign exception: "
  + exp.getMessage());}
```

flexFT.checkbox

Creates a Flex CheckBox element.

Format

The flexFT.checkbox method has the following command format(s):

flexFT.checkbox(*path*);

flexFT.checkbox(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new CheckBox.

Example

Performs an action on Flex checkBox element.

```
String CheckBox="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @id='explorer' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:0' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @label='']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:1' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @label='']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @id='swfLoader' " +
   "and @className='loaderPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:application[@automationName='controls/CheckBoxExample.swf' " +
   "and @id='null' " +
   "and @className='CheckBoxExample' " +
   "and @automationIndex='index:1' " +
```

```
    "and @label='' " +
    "and @automationClassName='FlexApplication']" +
   "/flex:panel[@automationName='CheckBox%20Control%20Example' " +
    "and @id='null' " +
    "and @className='mx.containers.Panel' " +
    "and @automationIndex='index:0' " +
    "and @label='' " +
    "and @automationClassName='FlexPanel']" +
   "/flex:checkbox[@automationName='milk' " +
    "and @id='milkCB' " +
    "and @className='mx.controls.CheckBox' " +
    "and @automationIndex='index:1' " +
    "and @label='milk' " +
    "and @automationClassName='FlexCheckBox']";
flexFT.checkbox(10, CheckBox).click(KeyModifier.None);
```

flexFT.colorPicker

Creates a Flex ColorPicker element.

Format

The flexFT.colorPicker method has the following command format(s):

flexFT.colorPicker(*path*);

flexFT.colorPicker(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new ColorPicker

Example

Performs an action on a Flex ColorPicker element.

```
String ColorPicker="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/ColorPickerExample.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
    "and @className='ColorPickerExample' " +
    "and @id='null']" +
   "/flex:panel[@automationName='ColorPicker%20Control%20Example' " +
    "and @automationClassName='FlexPanel' " +
    "and @automationIndex='index:0' " +
    "and @label='' " +
    "and @className='mx.containers.Panel' " +
    "and @id='null']" +
   "/flex:colorPicker[@automationName='cp' " +
    "and @automationClassName='FlexColorPicker' " +
    "and @automationIndex='index:1' " +
    "and @className='mx.controls.ColorPicker' " +
    "and @id='cp']";
flexFT.colorPicker(10, ColorPicker).change("#00cc00");
```

flexFT.columnChart

Creates a Flex ColumnChart element.

Format

The flexFT.columnChart method has the following command format(s):

flexFT.columnChart(*path*);

flexFT.columnChart(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new ColumnChart.

Example

Performs an action on Flex ColumnChart element.

```
String columnChart="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/Column_BarChartExample.swf' " +
   "and @label='' " +
   "and @className='Column_BarChartExample' " +
```

```
   "and @id='null']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='ColumnChart%20and%20BarChart%20Controls%20Example' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']" +
  "/flex:columnChart[@automationIndex='index:0' " +
   "and @automationClassName='FlexColumnChart' " +
   "and @automationName='column' " +
   "and @className='mx.charts.ColumnChart' " +
   "and @id='column']";
flexFT.columnChart(10, columnChart).click(KeyModifier.None);
try{
 Double result=flexFT.columnChart(15, columnChart).getColumnWidthRatio();
 if (result !=0.65) {
  warn("getColumnWidthRatio returned " + result);
 }
 }catch(Exception exp) {warn("getColumnWidthRatio exception: "
  + exp.getMessage());}
try{
 Double result=flexFT.columnChart(20, columnChart).getMaxColumnWidth();
 if (result !=null) {
  warn("getMaxColumnWidth returned " + result);
 }
 }catch(Exception exp) {warn("getMaxColumnWidth exception: "
  + exp.getMessage());}
```

flexFT.columnSeries

Creates a Flex ColumnSeries element.

Format

The flexFT.columnSeries method has the following command format(s):

flexFT.columnSeries(*path*);

flexFT.columnSeries(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new ColumnSeries.

Example

Performs an action on Flex ColumnSeries element.

```
String columnSeries="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/Column_BarChartExample.swf' " +
   "and @label='' " +
```

```
    "and @className='Column_BarChartExample' " +
    "and @id='null']" +
   "/flex:panel[@automationIndex='index:0' " +
    "and @automationClassName='FlexPanel' " +
    "and @automationName='ColumnChart%20and%20BarChart%20Controls%20Example' " +
    "and @label='' " +
    "and @className='mx.containers.Panel' " +
    "and @id='null']" +
   "/flex:columnChart[@automationIndex='index:0' " +
    "and @automationClassName='FlexColumnChart' " +
    "and @automationName='column' " +
    "and @className='mx.charts.ColumnChart' " +
    "and @id='column']" +
   "/flex:columnSeries[@automationIndex='index:2' " +
    "and @automationClassName='FlexColumnSeries' " +
    "and @automationName='Country;Bronze' " +
    "and @className='mx.charts.series.ColumnSeries' " +
    "and @id='_Column_BarChartExample_ColumnSeries3']";
flexFT.columnSeries(10, columnSeries).clickSeries(0.0);
try{
 String result=flexFT.columnSeries(15, columnSeries).getXField();
 if (!result.equals("Country")) {
  warn("getXField returned " + result);
 }
 }catch(Exception exp) {warn("getXField exception: "
  + exp.getMessage());}
try{
 String result=flexFT.columnSeries(20, columnSeries).getYField();
 if (!result.equals("Bronze")) {
  warn("getYField returned " + result);
 }
 }catch(Exception exp) {warn("getYField exception: "
  + exp.getMessage());}
```

flexFT.combobox

Creates a Flex ComboBox element.

Format

The flexFT.combobox method has the following command format(s):

flexFT.combobox(*path*);

flexFT.combobox(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new ComboBox.

Example

Performs an action on a Flex ComboBox element.

```
String ComboBox="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/SimpleComboBox.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
   "and @className='SimpleComboBox' " +
   "and @id='null']" +
  "/flex:panel[@automationName='ComboBox%20Control%20Example' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']" +
  "/flex:combobox[@automationName='_SimpleComboBox_ComboBox1' " +
   "and @automationClassName='FlexComboBox' " +
   "and @automationIndex='index:0' " +
   "and @className='mx.controls.ComboBox' " +
   "and @id='_SimpleComboBox_ComboBox1']";
flexFT.combobox(10, ComboBox).open(TriggerEvent.Mouse);
{
 think(0.764);
}
flexFT.combobox(15, ComboBox).select("MasterCard", TriggerEvent.Mouse,
KeyModifier.None);
flexFT.combobox(20, ComboBox).input("zopa");
flexFT.combobox(25, ComboBox).type("Enter", KeyModifier.None);
flexFT.combobox(30, ComboBox).storeCell("var", 1, 1);
if (!eval("{{var}}").equals("Visa")){
 warn("storeCell returned " + eval("{{var}}"));}
{
 think(1.231);
}
```

flexFT.dataGrid

Creates a Flex DataGrid element.

Format

The flexFT.dataGrid method has the following command format(s):

flexFT.dataGrid(*path*);

flexFT.dataGrid(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new DataGrid.

Example

Performs an action on Flex DataGrid element.

```
String DataGrid="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/SimpleDataGrid.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
  "and @className='SimpleDataGrid' " +
  "and @id='null']" +
 "/flex:panel[@automationName='DataGrid%20Control%20Example' " +
  "and @automationClassName='FlexPanel' " +
  "and @automationIndex='index:0' " +
  "and @label='' " +
  "and @className='mx.containers.Panel' " +
  "and @id='null']" +
 "/flex:dataGrid[@automationName='dg' " +
  "and @automationClassName='FlexDataGrid' " +
  "and @automationIndex='index:1' " +
  "and @className='mx.controls.DataGrid' " +
  "and @id='dg']";

flexFT.dataGrid(10, DataGrid).select("*Chris* | 2270 | chris@example.com"
 ,TriggerEvent.Mouse, KeyModifier.None);
think(1);
try{
 String result=flexFT.dataGrid(DataGrid).getCell(1, 1);
 if (!result.equals("Chris")) {
  warn("getCell returned " + result);
 }
 }catch(Exception exp) {warn("getCell exception: "
  + exp.getMessage());}
```

flexFT.dateChooser

Creates a Flex DateChooser element.

Format

The flexFT.dateChooser method has the following command format(s):

flexFT.dateChooser(*path*);

flexFT.dateChooser(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new DateChooser.

Example

Performs an action on Flex DateChooser element.

```
String DateChooser="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @id='explorer' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:0' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @label='']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:1' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @label='']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @id='swfLoader' " +
   "and @className='loaderPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:application[@automationName='controls/DateChooserExample.swf' " +
   "and @id='null' " +
   "and @className='DateChooserExample' " +
   "and @automationIndex='index:1' " +
```

```
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:panel[@automationName='DateChooser%20Control%20Example' " +
   "and @id='null' " +
   "and @className='mx.containers.Panel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:dateChooser[@automationName='dateChooser1' " +
   "and @id='dateChooser1' " +
   "and @className='mx.controls.DateChooser' " +
   "and @automationIndex='index:5' " +
   "and @automationClassName='FlexDateChooser']";
flexFT.dateChooser(10, DateChooser).scroll(DateChooserDetail.nextMonth);
{
 think(1.0);
}
flexFT.dateChooser(15, DateChooser).change("2012-09-12");
```

flexFT.dateField

Creates a Flex DateField element.

Format

The flexFT.dateField method has the following command format(s):

flexFT.dateField(*path*);

flexFT.dateField(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new DateField.

Example

Performs an action on Flex DateField element.

```
String dateField="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @id='explorer' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:0' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @label='']/flex:dividedBox[@id='null'" +
   " and @automationIndex='index:1' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @label='']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @id='swfLoader' " +
   "and @className='loaderPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:application[@automationName='controls/DateFieldExample.swf' " +
   "and @id='null' " +
   "and @className='DateFieldExample' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
   "and @automationClassName='FlexApplication']" +
  "/flex:panel[@automationName='DateField%20Control%20Example' " +
   "and @id='null' " +
   "and @className='mx.containers.Panel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:dateField[@automationName='dateField1' " +
   "and @id='dateField1' " +
   "and @className='mx.controls.DateField' " +
   "and @automationIndex='index:2' " +
   "and @automationClassName='FlexDateField']";
flexFT.dateField(10, dateField).open(TriggerEvent.Mouse);
{
 think(2.0);
}
flexFT.dateField(15, dateField).change("2010-09-12");
```

flexFT.dividedBox

Creates a Flex DividedBox element.

Format

The flexFT.dividedBox method has the following command format(s):

flexFT.dividedBox(*path*);

flexFT.dividedBox(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new DividedBox.

Example

Performs an action on Flex dividedBox element.

```
String dividedBox="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']";
flexFT.dividedBox(10, dividedBox).released(0, 153.0);
flexFT.dividedBox(15, dividedBox).dragged(0, 148.0);
flexFT.dividedBox(20, dividedBox).pressed(0, 148.0);
```

flexFT.lineChart

Creates a Flex LineChart element.

Format

The flexFT.lineChart method has the following command format(s):

flexFT.lineChart(*path*);

flexFT.lineChart(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new LineChart.

Example

Performs an action on Flex LineChart element.

```
String lineChart="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/Line_AreaChartExample.swf' " +
   "and @label='' " +
```

```
    "and @className='Line_AreaChartExample' " +
    "and @id='null']" +
   "/flex:panel[@automationIndex='index:0' " +
    "and @automationClassName='FlexPanel' " +
    "and @automationName='LineChart%20and%20AreaChart%20Controls%20Example' " +
    "and @label='' " +
    "and @className='mx.containers.Panel' " +
    "and @id='null']" +
   "/flex:lineChart[@automationIndex='index:0' " +
    "and @automationClassName='FlexLineChart' " +
    "and @automationName='linechart' " +
    "and @className='mx.charts.LineChart' " +
    "and @id='linechart']";
flexFT.lineChart(10, lineChart).click(KeyModifier.None);
```

flexFT.lineSeries

Creates a Flex LineSeries element.

Format

The flexFT.lineSeries method has the following command format(s):

flexFT.lineSeries(*path*);

flexFT.lineSeries(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new LineSeries.

Example

Performs an action on Flex LineSeries element.

```
String lineSeries="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='charts/Line_AreaChartExample.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
   "and @className='Line_AreaChartExample' " +
   "and @id='null']" +
  "/flex:panel[@automationName='LineChart%20and%20AreaChart%20Controls%20Example'
" +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']" +
  "/flex:lineChart[@automationName='linechart' " +
   "and @automationClassName='FlexLineChart' " +
   "and @automationIndex='index:0' " +
   "and @className='mx.charts.LineChart' " +
   "and @id='linechart']" +
  "/flex:lineSeries[@automationName='Profit' " +
   "and @automationClassName='FlexLineSeries' " +
   "and @automationIndex='index:0' " +
   "and @className='mx.charts.series.LineSeries' " +
   "and @id='_Line_AreaChartExample_LineSeries1']";
flexFT.lineSeries(10, lineSeries).clickSeries(1.0);
try{
 String result=flexFT.lineSeries(15, lineSeries).getXField();
 if (!result.equals("")) {
  warn("getXField returned " + result);
 }
 }catch(Exception exp) {warn("getXField exception: "
  + exp.getMessage());}
try{
 String result=flexFT.lineSeries(20, lineSeries).getYField();
 if (!result.equals("Profit")) {
  warn("getYField returned " + result);
 }
 }catch(Exception exp) {warn("getYField exception: "
  + exp.getMessage());}
```

flexFT.linkBar

Creates a Flex LinkBar element.

Format

The flexFT.linkBar method has the following command format(s):

flexFT.linkBar(*path*);

flexFT.linkBar(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new LinkBar.

Example

Performs an action on Flex linkBar element.

```
String LinkBar="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @id='explorer' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:0' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @label='']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:1' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @label='']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @id='swfLoader' " +
   "and @className='loaderPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:application[@automationName='controls/LinkBarExample.swf' " +
   "and @id='null' " +
   "and @className='LinkBarExample' " +
   "and @automationIndex='index:1' " +
```

```
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:panel[@automationName='LinkBar%20Control%20Example' " +
   "and @id='null' " +
   "and @className='mx.containers.Panel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:linkBar[@automationName='_LinkBarExample_LinkBar1' " +
   "and @id='_LinkBarExample_LinkBar1' " +
   "and @className='mx.controls.LinkBar' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
   "and @automationClassName='FlexLinkBar']";
flexFT.linkBar(10, LinkBar).change("Customer Info");
try{
 Integer result=flexFT.linkBar(15, LinkBar).getSelectedIndex();
 if (result !=1) {
  warn("getSelectedIndex returned " + result);
 }
 }catch(Exception exp) {warn("getSelectedIndex exception: "
  + exp.getMessage());}
```

flexFT.list

Creates a Flex List element.

Format

The flexFT.list method has the following command format(s):

flexFT.list(*path*);

flexFT.list(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new List

Example

Performs an action on Flex List element.

```
String List="/web:window[@index='0' or @title='Adobe Flex 3 Component Explorer']"
+
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/HorizontalListExample.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
   "and @className='HorizontalListExample' " +
   "and @id='null']" +
  "/flex:panel[@automationName='HorizontalList%20Control%20Example' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']" +
  "/flex:list[@automationName='CameraSelection' " +
   "and @automationClassName='FlexList' " +
   "and @automationIndex='index:1' " +
   "and @className='mx.controls.HorizontalList' " +
   "and @id='CameraSelection']";
try{
 String result=flexFT.list(10, List).getCell(1, 1);
 if (!result.equals("Nokia 6630")) {
  warn("getCell returned " + result);
 }
 }catch(Exception exp) {warn("getCell exception: "+ exp.getMessage());}
```

flexFT.menu

Creates a Flex Menu element.

Format

The flexFT.menu method has the following command format(s):

flexFT.menu(*path*);

flexFT.menu(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new Menu.

Example

Performs an action on Flex Menu element.

```
String menu="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='explorer' " +
   "and @label='' " +
   "and @automationIndex='index:-1' " +
   "and @automationName='explorer' " +
   "and @id='explorer']" +
  "/flex:menu[@automationClassName='FlexMenu' " +
   "and @className='mx.controls.Menu' " +
   "and @automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null']";
flexFT.menu(menu).select(10, "MenuItem 1");
```

flexFT.menubar

Creates a Flex MenuBar element.

Format

The flexFT.menubar method has the following command format(s):

flexFT.menubar(*path*);

flexFT.menubar(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new MenuBar.

Example

Performs an action on Flex MenuBar element.

```
String menubar="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='explorer' " +
   "and @label='' " +
   "and @automationIndex='index:-1' " +
   "and @automationName='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @automationName='index:0']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @automationName='index:1']" +
  "/flex:panel[@automationClassName='FlexPanel' " +
   "and @className='loaderPanel' " +
   "and @label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='swfLoader' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='MenuBarExample' " +
   "and @label='' " +
   "and @automationIndex='index:1' " +
```

```
   "and @automationName='controls/MenuBarExample.swf' " +
   "and @id='null']" +
  "/flex:panel[@automationClassName='FlexPanel' " +
   "and @className='mx.containers.Panel' " +
   "and @label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='MenuBar%20Control%20Example' " +
   "and @id='null']" +
  "/flex:menubar[@automationClassName='FlexMenuBar' " +
   "and @className='mx.controls.MenuBar' " +
   "and @automationIndex='index:1' " +
   "and @automationName='_MenuBarExample_MenuBar1' " +
   "and @id='_MenuBarExample_MenuBar1']";
flexFT.menubar(10, menubar).show("Menu1");
try{
 Double result=flexFT.menubar(15, menubar).getSelectedIndex();
 if (result !=0) {
  warn("getSelectedIndex returned " + result);
 }
 }catch(Exception exp) {warn("getSelectedIndex exception: "
  + exp.getMessage());}
flexFT.menubar(20, menubar).hide();
```

flexFT.numericStepper

Creates a Flex NumericStepper element.

Format

The flexFT.numericStepper method has the following command format(s):

flexFT.numericStepper(*path*);

flexFT.numericStepper(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new NumericStepper.

Example

Performs an action on Flex numericStepper element.

```
String numericStepper="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/NumericStepperExample.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
   "and @className='NumericStepperExample' " +
   "and @id='null']" +
  "/flex:panel[@automationName='NumericStepper%20Control%20Example' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']" +
  "/flex:numericStepper[@automationName='index:1' " +
   "and @automationClassName='FlexNumericStepper' " +
   "and @automationIndex='index:1' " +
   "and @className='mx.controls.NumericStepper' " +
   "and @id='null']";
flexFT.numericStepper(10, numericStepper).change(1.0);
think(2.0);
flexFT.numericStepper(15, numericStepper).selectText(0, 1);
```

flexFT.panel

Creates a Flex Panel element.

Format

The flexFT.panel method has the following command format(s):

flexFT.panel(*path*);

flexFT.panel(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new Panel

Example

Performs an action on Flex Panel element.

```
String Panel="/web:window[@index='0' or @title='Adobe Flex 3 Component Explorer']"
+
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/SimpleHRule.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
   "and @className='SimpleHRule' " +
```

```
  "and @id='null']" +
 "/flex:panel[@automationName='HRule%20Control%20Example' " +
  "and @automationClassName='FlexPanel' " +
  "and @automationIndex='index:0' " +
  "and @label='' " +
  "and @className='mx.containers.Panel' " +
  "and @id='myPanel']";
try{
 String result=flexFT.panel(10, Panel).getTitle();
 if (!result.equals("HRule Control Example")) {
  warn("getTitle returned " + result);
 }
 }catch(Exception exp) {warn("getTitle exception: "+ exp.getMessage());}
```

flexFT.pieChart

Creates a Flex PieChart element.

Format

The flexFT.pieChart method has the following command format(s):

flexFT.pieChart(*path*);

flexFT.pieChart(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new PieChart.

Example

Performs an action on Flex PieChart element.

```
String pieChart="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/PieChartExample.swf' " +
   "and @label='' " +
```

```
   "and @className='PieChartExample' " +
   "and @id='null']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='Olympics%202004%20Medals%20Tally%20Panel' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @id='null']/flex:pieChart[@automationIndex='index:0' " +
   "and @automationClassName='FlexPieChart' " +
   "and @automationName='chart' " +
   "and @className='mx.charts.PieChart' " +
   "and @id='chart']";
flexFT.pieChart(10, pieChart).click(KeyModifier.None);
try{
 Double result=flexFT.pieChart(15, pieChart).getInnerRadius();
 if (result !=0) {
  warn("getInnerRadius returned " + result);
 }
 }catch(Exception exp) {warn("getInnerRadius exception: "
  + exp.getMessage());}
```

flexFT.pieSeries

Creates a Flex PieSeries element.

Format

The flexFT.pieSeries method has the following command format(s):

flexFT.pieSeries(*path*);

flexFT.pieSeries(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new PieSeries.

Example

Performs an action on Flex PieSeries element.

```
String pieSeries="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/PieChartExample.swf' " +
   "and @label='' " +
```

```
    "and @className='PieChartExample' " +
    "and @id='null']" +
   "/flex:panel[@automationIndex='index:0' " +
    "and @automationClassName='FlexPanel' " +
    "and @automationName='Olympics%202004%20Medals%20Tally%20Panel' " +
    "and @label='' " +
    "and @className='mx.containers.Panel' " +
    "and @id='null']" +
   "/flex:pieChart[@automationIndex='index:0' " +
    "and @automationClassName='FlexPieChart' " +
    "and @automationName='chart' " +
    "and @className='mx.charts.PieChart' " +
    "and @id='chart']" +
   "/flex:pieSeries[@automationIndex='index:0' " +
    "and @automationClassName='FlexPieSeries' " +
    "and @automationName='_PieChartExample_PieSeries1' " +
    "and @className='mx.charts.series.PieSeries' " +
    "and @id='_PieChartExample_PieSeries1']";
flexFT.pieSeries(10, pieSeries).clickSeries(1.0);
try{
 String result=flexFT.pieSeries(15, pieSeries).getField() ;
 if (!result.equals("Gold")) {
  warn("getField returned " + result);
 }
 }catch(Exception exp) {warn("getField exception: "
  + exp.getMessage());}
```

flexFT.plotSeries

Creates a Flex PlotSeries element.

Format

The flexFT.plotSeries method has the following command format(s):

flexFT.plotSeries(*path*);

flexFT.plotSeries(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new PlotSeries.

Example

Performs an action on Flex PlotSeries element.

```
String plotSeries="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationIndex='index:-1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='explorer' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationName='index:0' " +
   "and @automationIndex='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationName='index:1' " +
   "and @automationIndex='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationIndex='index:0' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationName='swfLoader' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationIndex='index:1' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationName='charts/PlotChartExample.swf' " +
   "and @label='' " +
```

```
    "and @className='PlotChartExample' " +
    "and @id='null']" +
   "/flex:panel[@automationIndex='index:0' " +
    "and @automationClassName='FlexPanel' " +
    "and @automationName='PlotChart%20Control%20Example' " +
    "and @label='' " +
    "and @className='mx.containers.Panel' " +
    "and @id='null']" +
   "/flex:cartesianChart[@automationIndex='index:0' " +
    "and @automationClassName='FlexCartesianChart' " +
    "and @automationName='plot' " +
    "and @className='mx.charts.PlotChart' " +
    "and @id='plot']" +
   "/flex:plotSeries[@automationIndex='index:0' " +
    "and @automationClassName='FlexPlotSeries' " +
    "and @automationName='Expenses;Profit' " +
    "and @className='mx.charts.series.PlotSeries' " +
    "and @id='_PlotChartExample_PlotSeries1']";
flexFT.plotSeries(10, plotSeries).clickSeries(1.0);
try{
 String result=flexFT.plotSeries(15, plotSeries).getXField() ;
 if (!result.equals("Expenses")) {
  warn("getXField returned " + result);
 }
 }catch(Exception exp) {warn("getXField exception: "
  + exp.getMessage());}
try{
 String result=flexFT.plotSeries(20, plotSeries).getYField() ;
 if (!result.equals("Profit")) {
  warn("getYField returned " + result);
 }
 }catch(Exception exp) {warn("getYField exception: "
  + exp.getMessage());}
```

flexFT.popupButton

Creates a Flex PopUpButton element.

Format

The flexFT.popupButton method has the following command format(s):

flexFT.popupButton(*path*);

flexFT.popupButton(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new PopUpButton.

Example

Performs an action on Flex PopupButton element.

```
String popupButton="/web:window[@index='0' " +
  "or @title='Adobe Flex 3 Component Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='explorer' " +
   "and @label='' " +
   "and @automationIndex='index:-1' " +
   "and @automationName='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @automationName='index:0']" +
  "/flex:dividedBox[@automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @id='null' " +
   "and @automationIndex='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @automationName='index:1']" +
  "/flex:panel[@automationClassName='FlexPanel' " +
   "and @className='loaderPanel' " +
   "and @label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='swfLoader' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationClassName='FlexApplication' " +
   "and @className='PopUpButtonExample' " +
   "and @label='' " +
   "and @automationIndex='index:1' " +
```

```
   "and @automationName='controls/PopUpButtonExample.swf' " +
   "and @id='null']" +
  "/flex:panel[@automationClassName='FlexPanel' " +
   "and @className='mx.containers.Panel' " +
   "and @label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='PopUpButton%20Control%20Example' " +
   "and @id='null']" +
  "/flex:popupButton[@automationClassName='FlexPopUpButton' " +
   "and @className='mx.controls.PopUpButton' " +
   "and @label='Put%20in:%20New%20Folder' " +
   "and @automationIndex='index:1' " +
   "and @automationName='popB' and @id='popB']";
flexFT.popupButton(10, popupButton).open(TriggerEvent.Mouse);
{
 think(1.328);
}
flexFT.popupButton(15, popupButton).close(TriggerEvent.Mouse);
```

flexFT.progressBar

Creates a Flex ProgressBar element.

Format

The flexFT.progressBar method has the following command format(s):

flexFT.progressBar(*path*);

flexFT.progressBar(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new ProgressBar.

Example

Performs an action on Flex progressBar element.

```
String ProgressBar="/web:window[@index='0']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@label='' " +
   "and @automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @id='null']" +
  "/flex:dividedBox[@label='' " +
   "and @automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @id='null']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @automationIndex='index:0' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/SimpleProgressBar.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @label='' " +
   "and @className='SimpleProgressBar' " +
   "and @automationIndex='index:1' " +
```

```
   "and @id='null']" +
  "/flex:panel[@automationName='ProgressBar%20Control%20Example' " +
   "and @automationClassName='FlexPanel' " +
   "and @label='' " +
   "and @className='mx.containers.Panel' " +
   "and @automationIndex='index:0' " +
   "and @id='null']" +
  "/flex:progressBar[@automationName='CurrentProgress%200%25' " +
   "and @automationClassName='FlexProgressBar' " +
   "and @label='CurrentProgress%200%25' " +
   "and @className='mx.controls.ProgressBar' " +
   "and @automationIndex='index:2' " +
   "and @id='bar']";
try{
 Double result=flexFT.progressBar(10, ProgressBar).getPercentComplete();
 if (result !=0) {
  warn("getPercentComplete returned " + result);
 }
 }catch(Exception exp) {warn("getPercentComplete exception: "
  + exp.getMessage());}
```

flexFT.radioButton

Creates a Flex RadioButton element.

Format

The flexFT.radioButton method has the following command format(s):

flexFT.radioButton(*path*);

flexFT.radioButton(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new RadioButton.

Example

Performs an action on Flex RadioButton element.

```
String RadioButton="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @id='explorer' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:0' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @label='']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:1' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @label='']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @id='swfLoader' " +
   "and @className='loaderPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:application[@automationName='controls/RadioButtonExample.swf' " +
   "and @id='null' " +
   "and @className='RadioButtonExample' " +
   "and @automationIndex='index:1' " +
```

```
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:panel[@automationName='RadioButton%20Control%20Example' " +
   "and @id='null' " +
   "and @className='mx.containers.Panel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:radioButton[@automationName='1952' " +
   "and @id='option2' " +
   "and @className='mx.controls.RadioButton' " +
   "and @automationIndex='index:2' " +
   "and @label='1952' " +
   "and @automationClassName='FlexRadioButton']";
flexFT.radioButton(RadioButton).click(KeyModifier.None);
try{
 String result=flexFT.radioButton(RadioButton).getGroupName();
 if (!result.equals("year")) {
  warn("getGroupName returned " + result);
 }
 }catch(Exception exp) {warn("getSGroupName exception: "
  + exp.getMessage());}
```

flexFT.scrollbar

Creates a Flex ScrollBar element.

Format

The flexFT.scrollbar method has the following command format(s):

flexFT.scrollbar(*path*);

flexFT.scrollbar(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new ScrollBar.

Example

Performs an action on Flex Scrollbar element.

```
String scrollBar="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/HScrollBarExample.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
    "and @className='HScrollBarExample' " +
    "and @id='null']" +
   "/flex:panel[@automationName='HScrollBar%20Control%20Example' " +
    "and @automationClassName='FlexPanel' " +
    "and @automationIndex='index:0' " +
    "and @label='' " +
    "and @className='mx.containers.Panel' " +
    "and @id='panel']" +
   "/flex:scrollbar[@automationName='bar' " +
    "and @automationClassName='FlexScrollBar' " +
    "and @automationIndex='index:1' " +
    "and @className='mx.controls.HScrollBar' " +
    "and @id='bar']";
flexFT.scrollbar(10, scrollBar).scroll(100,
  ScrollDirection.horizontal,
  ScrollDetail.pageRight);
```

flexFT.slider

Creates a Flex Slider element.

Format

The flexFT.slider method has the following command format(s):

flexFT.slider(*path*);

flexFT.slider(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new Slider.

Example

Performs an action on Flex Slider element.

```
String Slider="/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @className='explorer' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.HDividedBox']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @id='null' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @label='' " +
   "and @className='mx.containers.VDividedBox']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @className='loaderPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@automationName='controls/SimpleImageHSlider.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @automationIndex='index:1' " +
   "and @label='' " +
```

```
    "and @className='SimpleImageHSlider' " +
    "and @id='null']" +
   "/flex:panel[@automationName='HSlider%20Control%20Example' " +
    "and @automationClassName='FlexPanel' " +
    "and @automationIndex='index:0' " +
    "and @label='' " +
    "and @className='mx.containers.Panel' " +
    "and @id='panel']" +
   "/flex:slider[@automationName='hSlider' " +
    "and @automationClassName='FlexSlider' " +
    "and @automationIndex='index:3' " +
    "and @direction='horizontal' " +
    "and @className='mx.controls.HSlider' " +
    "and @id='hSlider']";
flexFT.slider(10, Slider).change(54.0, 0, "track",
 TriggerEvent.Mouse, "");
flexFT.slider(15, Slider).change(52.0, 0, "thumb",
 TriggerEvent.Mouse, "");
```

flexFT.toggleButtonBar

Creates a Flex ToggleButtonBar element.

Format

The flexFT.toggleButtonBar method has the following command format(s):

flexFT.toggleButtonBar(*path*);

flexFT.toggleButtonBar(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new ToggleButtonBar.

Example

Performs an action on Flex toggleButtonBar element.

```
flexFT.toggleButtonBar(19, "/web:window[@index='0' " +
   "or @title='Adobe Flex 3 Component Explorer" +
 "/web:document[@index='0']" +
  "/flex:application[@automationName='explorer' " +
   "and @id='explorer' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @label='' " +
   "and @automationClassName='FlexApplication']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:0' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:0' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @label='']" +
  "/flex:dividedBox[@id='null' " +
   "and @automationIndex='index:1' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @automationName='index:1' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @label='']" +
  "/flex:panel[@automationName='swfLoader' " +
   "and @id='swfLoader' " +
   "and @className='loaderPanel' " +
   "and @automationIndex='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexPanel']" +
  "/flex:application[@automationName='controls/ToggleButtonBarExample.swf' " +
   "and @id='null' " +
   "and @className='ToggleButtonBarExample' " +
   "and @automationIndex='index:1' " +
```

```
 "and @label='' " +
 "and @automationClassName='FlexApplication']" +
"/flex:panel[@automationName='ToggleButtonBar%20Control%20Example' " +
 "and @id='null' and @className='mx.containers.Panel' " +
 "and @automationIndex='index:0' " +
 "and @label='' " +
 "and @automationClassName='FlexPanel']" +
"/flex:toggleButtonBar[@id='null' " +
 "and @automationIndex='index:2' " +
 "and @automationClassName='FlexToggleButtonBar' " +
 "and @automationName='index:2' " +
 "and @className='mx.controls.ToggleButtonBar' " +
 "and @label='']").change("Director");
```

flexFT.tree

Creates a Flex Tree element.

Format

The flexFT.tree method has the following command format(s):

flexFT.tree(*path*);

flexFT.tree(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes.

Returns

new Tree.

Example

Performs an action on Flex progressBar element.

```
flexFT.tree(5, "/web:window[@index='0' or @title='Adobe Flex 3 Component
Explorer']" +
 "/web:document[@index='0']" +
  "/flex:application[@label='' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @id='null']" +
  "/flex:panel[@label='' " +
   "and @className='mx.containers.Panel' " +
   "and @automationIndex='index:0' " +
   "and @automationName='Adobe%20Flex%203%20Component%20Explorer' " +
   "and @automationClassName='FlexPanel' " +
   "and @id='null']" +
  "/flex:tree[@className='mx.controls.Tree' " +
   "and @automationIndex='index:0' " +
   "and @automationName='compLibTree' " +
   "and @automationClassName='FlexTree' " +
   "and @id='compLibTree']")
   .select("Visual Components>General Controls>Tree",
   TriggerEvent.Mouse, KeyModifier.None);
String Tree="/web:window[@index='0' or @title='Adobe Flex 3 Component Explorer']"
+
 "/web:document[@index='0']" +
```

```
  "/flex:application[@label='' " +
   "and @className='explorer' " +
   "and @automationIndex='index:-1' " +
   "and @automationName='explorer' " +
   "and @automationClassName='FlexApplication' " +
   "and @id='explorer']" +
  "/flex:dividedBox[@automationIndex='index:0' " +
   "and @automationName='index:0' " +
   "and @label='' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @id='null']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @label='' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @className='mx.containers.VDividedBox' " +
   "and @id='null']" +
  "/flex:panel[@label='' " +
   "and @className='loaderPanel' " +
   "and @automationIndex='index:0' " +
   "and @automationName='swfLoader' " +
   "and @automationClassName='FlexPanel' " +
   "and @id='swfLoader']" +
  "/flex:application[@label='' " +
   "and @className='TreeExample' " +
   "and @automationIndex='index:1' " +
   "and @automationName='controls/TreeExample.swf' " +
   "and @automationClassName='FlexApplication' " +
   "and @id='null']" +
  "/flex:panel[@label='' " +
   "and @className='mx.containers.Panel' " +
   "and @automationIndex='index:0' " +
   "and @automationName='Tree%20Control%20Example' " +
   "and @automationClassName='FlexPanel' " +
   "and @id='null']" +
  "/flex:dividedBox[@automationIndex='index:1' " +
   "and @automationName='index:1' " +
   "and @label='' " +
   "and @automationClassName='FlexDividedBox' " +
   "and @className='mx.containers.HDividedBox' " +
   "and @id='null']" +
  "/flex:tree[@className='mx.controls.Tree' " +
   "and @automationIndex='index:0' " +
   "and @automationName='myTree' " +
   "and @automationClassName='FlexTree' " +
   "and @id='myTree']";
{
 think(2.0);
}
flexFT.tree(10, Tree).open("Inbox", TriggerEvent.Mouse);
{
 think(1.5);
}
flexFT.tree(15, Tree).select("Inbox>Marketing",
 TriggerEvent.Mouse, KeyModifier.None);
```

10 Functional Test Module

This chapter provides a complete listing and reference for the methods in the OpenScript FunctionalTestService Class of Functional Test Module Application Programming Interface (API).

10.1 FunctionalTestService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript FunctionalTestService API.

10.1.1 Alphabetical Command Listing

The following table lists the FunctionalTestService API methods in alphabetical order.

Table 10–1 List of FunctionalTestService Methods

Method	Description
ft.drag	Perform a drag operation with the mouse.
ft.dragAndDrop	Perform a drag and drop operation with the mouse.
ft.getScreenCapture	Capture the specified region of the screen to a file.
ft.keyDown	Perform key down action on the keys in the form of Key Code.
ft.keyUp	Release key on the keyboard if it is pressed down.
ft.mouseClick	Move the mouse to the specified coordinates on the screen and press the specified mouse button.
ft.mouseDown	Move the mouse to the specified coordinates on the screen and depress the specified mouse button without releasing it.
ft.mouseUp	Move the mouse to the specified coordinates on the screen and release the specified mouse button.
ft.typeCharacters	Simulate typing a string of character keys on the keyboard.
ft.typeKeyCode	Simulate pressing keys on the keyboard by keycodes.
ft.typeKeys	Simulate pressing keys on the keyboard.

The following sections provide detailed reference information for each method and enum in the FunctionalTestService Class of Functional Test Module Application Programming Interface.

ft.drag

Perform a drag operation with the mouse. The mouse button is NOT released after the drag operation finishes.

All coordinates are relative to the entire screen.

Mouse operations are synchronized within the running agent process, but, as of 2.3.0, are NOT synchronized across all running agent processes on the machine. Therefore, if multiple agent processes are running, there is contention for the mouse.

For reliable playback, it is imperative not to move the mouse when this action is being performed during script playback.

Format

The ft.drag method has the following command format(s):

ft.drag(*xDragStart*, *yDragStart*, *xDragEnd*, *yDragEnd*);

Command Parameters

xDragStart
X coordinate where to start the drag operation. Coordinates are relative to the screen, and specified in pixels.

yDragStart
Y coordinate where to start the drag operation. Coordinates are relative to the screen, and specified in pixels.

xDragEnd
X coordinate where to stop the drag operation. Coordinates are relative to the screen, and specified in pixels.

yDragEnd
Y coordinate where to stop the drag operation. Coordinates are relative to the screen, and specified in pixels.

Throws

FTCoreException
if the method fails.

Example

Performs a mouse drag starting at screen coordinate x=100, y=100 and ending at corrdinate x=200, y=200.

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
ft.drag(100, 100, 200, 200);
```

See Also

ft.dragAndDrop

ft.dragAndDrop

Perform a drag and drop operation with the mouse.

All coordinates are relative to the entire screen.

Mouse operations are synchronized within the running agent process, but, as of 2.3.0, are NOT synchronized across all running agent processes on the machine. Therefore, if multiple agent processes are running, there is contention for the mouse.

For reliable playback, it is imperative not to move the mouse when this action is being performed during script playback.

Format

The ft.dragAndDrop method has the following command format(s):

ft.dragAndDrop(*xDragStart*, *yDragStart*, *xDropTo*, *yDropTo*);

Command Parameters

xDragStart
X coordinate where to start the drag operation. Coordinates are relative to the screen, and specified in pixels.

yDragStart
Y coordinate where to start the drag operation. Coordinates are relative to the screen, and specified in pixels.

xDropTo
X coordinate where to release the mouse after dragging the item over its target. Coordinates are relative to thescreen, and specified in pixels.

yDropTo
Y coordinate where to release the mouse after dragging the item over its target. Coordinates are relative to thescreen, and specified in pixels.

Throws

FTCoreException
if the method fails.

Example

Performs a mouse drag and drop starting at screen coordinate x=100, y=100 and ending at corrdinate x=200, y=200.

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
ft.dragAndDrop(100, 100, 200, 200);
```

See Also

ft.drag

ft.getScreenCapture

Capture the specified region of the screen to a file.

The file name must end with `.jpg` or `.png`, not case-sensitive, depending on the output format desired.

Coordinates are relative to the screen, and specified in pixels.

Format

The ft.getScreenCapture method has the following command format(s):

ft.getScreenCapture(*left*, *top*, *width*, *height*, *imgFileName*);

Command Parameters

left
Left (X) corner of the region to capture. Coordinates are relative to the screen, and specified in pixels.

top
Top (Y) corner of the region to capture. Coordinates are relative to the screen, and specified in pixels.

width
Right (X) corner of the region to capture. Coordinates are relative to the screen, and specified in pixels.

height
Bottom (Y) corner of the region to capture. Coordinates are relative to the screen, and specified in pixels.

imgFileName
Local filename to which to write the captured image. Must have write-access to the output file name.Any existing contents in the file will be overwritten. The file name must end with `.jpg` or `.png`,not case-sensitive, depending on the output format desired.

Throws

FTCoreException
if the method fails.

Example

Captures the screen image from coordinates x=0, y=0 to coordinates x=400, y=400.

```
import java.io.File;
import java.io.FileOutputStream;
[...]
String path = getScriptPackage().getScriptPath();
String folder = path.substring(0, path.lastIndexOf("\\"));
String imgName = folder + "\\capturedImg.jpg";
File imgFile = new File (imgName);
[...]
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
```

```
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
web.textBox("/web:window[@index='0' or @title='Test Web Events']" +
 "/web:document[@index='0']" +
 "/web:input_text[@name='Text' or @index='0']")
 .focus();
if(imgFile.exists()){
 if(imgFile.setReadOnly()){
  imgFile.setWritable(true);
 }
 imgFile.delete();
}
ft.getScreenCapture(0, 0, 400, 400, imgName);
if(!imgFile.exists()){
 warn("getScreenCapture() failed.");
}
```

ft.keyDown

Perform key down action on the keys in the form of Key Code.

Format

The ft.keyDown method has the following command format(s):

ft.keyDown(*keyCodes*);

Command Parameters

keyCodes
an int Array specifying the keys to be pressed down.

Throws

FTCoreException
if the method fails.

Example

Simulates pressing down the keys "g" "o" "o" "d".

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
web.textBox("/web:window[@index='0' or @title='Test Web Events']" +
 "/web:document[@index='0']" +
 "/web:input_text[@name='Text' or @index='0']")
 .focus();
ft.keyDown(KeyEvent.VK_G, KeyEvent.VK_O, KeyEvent.VK_O, KeyEvent.VK_D);
```

ft.keyUp

Release key on the keyboard if it is pressed down.

Format

The ft.keyUp method has the following command format(s):

ft.keyUp(*keyCodes*);

Command Parameters

keyCodes
an int Array specifying the keys to be pressed down.

Throws

FTCoreException
if the method fails.

Example

Simulates pressing down the keys "g" "o" "o" "d".

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
web.textBox("/web:window[@index='0' or @title='Test Web Events']" +
 "/web:document[@index='0']" +
 "/web:input_text[@name='Text' or @index='0']")
 .focus();
ft.keyUp(KeyEvent.VK_G, KeyEvent.VK_O, KeyEvent.VK_O, KeyEvent.VK_D);
```

ft.mouseClick

Move the mouse to the specified coordinates on the screen and press the specified mouse button.

This method is different from a DHTML-based click event because it literally moves the mouse before clicking, and it requires the correct window to be in the foreground and visible. All coordinates are relative to the entire screen.

Mouse operations are synchronized within the running agent process, but, as of 2.3.0, are NOT synchronized across all running agent processes on the machine. Therefore, if multiple agent processes are running, there is contention for the mouse.

For reliable playback, it is imperative not to move the mouse when this action is being performed during script playback.

Format

The ft.mouseClick method has the following command format(s):

ft.mouseClick(*x*, *y*, *clickCount*, *rightButton*, *modifiers*);

Command Parameters

x
X coordinate to which to move the mouse. Coordinates are relative to the screen, and specified in pixels.

y
Y coordinate to which to move the mouse. Coordinates are relative to the screen, and specified in pixels.

clickCount
Specifies how many times to click the mouse. For example, specify `2` to double-click, or `1`to single-click.

rightButton
Set to `true` to click the secondary mouse button, `false` to click the primary mouse button.On most systems, the right mouse button is the secondary mouse button.

modifiers
Specify a string of keyboard modifiers to hold down when clicking the mouse. Example values include any combinationof `ALT`, `SHIFT`, and `CTRL`, surrounded by angle brackets. For example, to hold down the Shiftand Control keys, specify the string: `<SHIFT><CTRL>`

Throws

FTCoreException
if the method fails.

Example

Performs a mouse click at screen coordinate x=100, y=100.

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
```

```
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
ft.mouseClick(100, 100, 1, false, "");
ft.mouseClick(100, 100, 1, false, "<SHIFT>");
```

ft.mouseDown

Move the mouse to the specified coordinates on the screen and depress the specified mouse button without releasing it.

This method is different from a DHTML-based onMouseDown event because it literally moves the mouse before clicking, and it requires the correct window to be in the foreground and visible. All coordinates are relative to the entire screen.

Mouse operations are synchronized within the running agent process, but, as of 2.3.0, are NOT synchronized across all running agent processes on the machine. Therefore, if multiple agent processes are running, there is contention for the mouse.

For reliable playback, it is imperative not to move the mouse when this action is being performed during script playback.

Format

The ft.mouseDown method has the following command format(s):

ft.mouseDown(*x*, *y*, *rightButton*);

Command Parameters

x
X coordinate to which to move the mouse. Coordinates are relative to the screen, and specified in pixels.

y
Y coordinate to which to move the mouse. Coordinates are relative to the screen, and specified in pixels.

rightButton
Set to `true` to press down on the secondary mouse button, `false` to press down onthe primary mouse button. On most systems, the right mouse button is the secondary mouse button.

Throws

FTCoreException
if the method fails.

Example

Performs a mouse down at screen coordinate x=500, y=500.

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
ft.mouseDown(500, 500, false);
```

See Also

ft.mouseUp

ft.mouseUp

Move the mouse to the specified coordinates on the screen and release the specified mouse button.

This method is different from a DHTML-based onMouseUp event because it literally moves the mouse before releasing it, and it requires the correct window to be in the foreground and visible. All coordinates are relative to the entire screen.

Mouse operations are synchronized within the running agent process, but, as of 2.3.0, are NOT synchronized across all running agent processes on the machine. Therefore, if multiple agent processes are running, there is contention for the mouse.

For reliable playback, it is imperative not to move the mouse when this action is being performed during script playback.

Format

The ft.mouseUp method has the following command format(s):

ft.mouseUp(*x*, *y*, *rightButton*);

Command Parameters

x
X coordinate to which to move the mouse. Coordinates are relative to the screen, and specified in pixels.

y
Y coordinate to which to move the mouse. Coordinates are relative to the screen, and specified in pixels.

rightButton
Set to `true` to release the secondary mouse button, `false` to release the primarymouse button. On most systems, the right mouse button is the secondary mouse button.

Throws

FTCoreException
if the method fails.

Example

Performs a mouse up at screen coordinate x=500, y=500.

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
ft.mouseUp(500, 500, false);
```

See Also

ft.mouseDown

ft.typeCharacters

Simulate typing a string of character keys on the keyboard.

This API would press the keys directly by the sequence of the characters String.

No function keys like "SHIFT","CTRL","ALT","F1" , etc. would be supported.

Format

The ft.typeCharacters method has the following command format(s):

ft.typeCharacters(*characters*);

Command Parameters

characters
a String specifying the keys to type.

Throws

FTCoreException
if the method fails.

Example

Simulates typing the keys "g" "o" "o" "d".

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
web.textBox("/web:window[@index='0' or @title='Test Web Events']" +
 "/web:document[@index='0']" +
 "/web:input_text[@name='Text' or @index='0']")
 .focus();
ft.typeCharacters("good");
 web.verifyText("tmt", "good", Source.DisplayContent,
 TextPresence.PassIfPresent, MatchOption.Exact);
```

ft.typeKeyCode

Simulate pressing keys on the keyboard by keycodes.

Special keys should be formated as a Java KEYNAME (without prefix VK_) inside < > characters, i.e. <Insert>, <Less>, <SHIFT-A>, <TAB> This method automatically inserts < > around ASCII characters like A-Z, 0-9. <SHIFT-A><TAB><H>ome types: A home.

Format

The ft.typeKeyCode method has the following command format(s):

ft.typeKeyCode(*keyCodes*);

Command Parameters

keyCodes
an int Array specifying the keys to type.

Throws

FTCoreException
if the method fails.

Example

Simulates typing the keys "g" "o" "o" "d".

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
web.textBox("/web:window[@index='0' or @title='Test Web Events']" +
 "/web:document[@index='0']" +
 "/web:input_text[@name='Text' or @index='0']")
 .focus();
ft.typeKeyCode(KeyEvent.VK_G, KeyEvent.VK_O, KeyEvent.VK_O, KeyEvent.VK_D);
{
 web.verifyText("tmt", "good", Source.DisplayContent,
 TextPresence.PassIfPresent, MatchOption.Exact);
}
```

ft.typeKeys

Simulate pressing keys on the keyboard.

Special keys should be formated as a Java KEYNAME (without prefix VK_) inside < > characters, i.e. <Insert>, <Less>, <SHIFT-A>, <TAB> This method automatically inserts < > around ASCII characters like A-Z, 0-9. <SHIFT-A><TAB><H>ome types: A home.

Format

The ft.typeKeys method has the following command format(s):

ft.typeKeys(*keys*);

Command Parameters

keys
a String specifying the keys to type.

Throws

FTCoreException
if the method fails.

Example

Simulates typing the keys "g" "o" "o" "d".

```
web.window(5, "/web:window[@index='0' or @title='Test Web Events']")
 .navigate("http://example.com/Web_Event.html");
web.window(7, "/web:window[@index='0' or @title='Test Web Events']")
 .waitForPage(null);
web.textBox("/web:window[@index='0' or @title='Test Web Events']" +
 "/web:document[@index='0']" +
 "/web:input_text[@name='Text' or @index='0']")
 .focus();
ft.typeKeys("<g>");
ft.typeKeys("<o>");
ft.typeKeys("<o>");
ft.typeKeys("<d>");
{
 web.verifyText("tmt", "good", Source.DisplayContent,
 TextPresence.PassIfPresent, MatchOption.Exact);
}
```

11

Oracle Fusion/ADF Functional Module

This chapter provides a complete listing and reference for the methods in the OpenScript ADFService Class of ADF Module Application Programming Interface (API).

11.1 ADFService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript ADFService API.

11.1.1 Alphabetical Command Listing

The following table lists the ADFService API methods in alphabetical order.

Table 11–1 List of ADFService Methods

Method	Description
adf.calendar	Identifies an ADFCalendar by its recorded ID and path.
adf.carousel	Identifies an ADFDVTGraph by its path.
adf.commandButton	Identifies an ADFCommandButton by its recorded ID and path.
adf.commandImageLink	Identifies an ADFCommandImageLink by its recorded ID and path.
adf.commandLink	Identifies an ADFCommandLink by its recorded ID and path.
adf.commandMenuItem	Identifies an ADFCommandMenuItem by its recorded ID and path.
adf.commandNavigationItem	Identifies an ADFCommandNavigationItem by its recorded ID and path.
adf.commandToolbarButton	Identifies an ADFCommandToolbarButton by its recorded ID and path.
adf.dialog	Identifies an ADFDialog by its recorded ID and path.
adf.gauge	Identifies an ADFDVTGraph by its recorded ID and path.
adf.goMenuItem	Identifies an ADFGoMenuItem by its recorded ID and path.
adf.graph	Identifies an ADFDVTGraph by its recorded ID and path.
adf.inputColor	Identifies an ADFInputColor by its recorded ID and path.
adf.inputComboboxListOfValues	Identifies an ADFInputComboboxListOfValues by its recorded ID and path.
adf.inputDate	Identifies an ADFInputDate by its recorded ID and path.

Table 11–1 (Cont.) List of ADFService Methods

Method	Description
adf.inputFile	Identifies an ADFInputFile by its recorded ID and path.
adf.inputListOfValues	Identifies an ADFInputListOfValues by its recorded ID and path.
adf.inputNumberSlider	Identifies an ADFInputNumberSlider by its recorded ID and path.
adf.inputNumberSpinbox	Identifies an ADFInputNumberSpinbox by its recorded ID and path.
adf.inputRangeSlider	Identifies an ADFInputRangeSlider by its recorded ID and path.
adf.inputText	Identifies an ADFInputText by its recorded ID and path.
adf.menu	Identifies an ADFMenu by its recorded ID and path.
adf.message	Identifies an ADFMessage by its recorded ID and path.
adf.messages	Identifies an ADFMessages by its recorded ID and path.
adf.navigationPane	Identifies an ADFNavigationPane by its recorded ID and path.
adf.noteWindow	Identifies an ADFNoteWindow by its recorded ID and path.
adf.outputFormatted	Identifies an ADFOutputFormat by its recorded ID and path.
adf.outputLabel	Identifies an ADFOutputLabel by its recorded ID and path.
adf.outputText	Identifies an ADFOutputText by its recorded ID and path.
adf.page	Identifies an ADFPage by its recorded ID and path.
adf.panelAccordion	Identifies an ADFPanelAccordion by its recorded ID and path.
adf.panelBox	Identifies an ADFPanelBox by its recorded ID and path.
adf.panelHeader	Identifies an ADFPanelHeader by its recorded ID and path.
adf.panelLabelAndMessage	Identifies an ADFPanelLabelAndMessage by its recorded ID and path.
adf.panelList	Identifies an ADFPanelList by its recorded ID and path.
adf.panelSplitter	Identifies an ADFPanelSplitter by its recorded ID and path.
adf.panelTabbed	Identifies an ADFPanelTabbed by its recorded ID and path.
adf.panelWindow	Identifies an ADFPanelWindow by its recorded ID and path.
adf.progressIndicator	Identifies an ADFProgressIndicator by its recorded ID and path.
adf.query	Identifies an ADFQuery by its recorded ID and path.
adf.quickQuery	Identifies an ADFQuickQuery by its recorded ID and path.
adf.resetButton	Identifies an ADFResetButton by its recorded ID and path.
adf.richTextEditor	Identifies an ADFRichTextEditor by its recorded ID and path.
adf.selectBooleanCheckbox	Identifies an ADFSelectBooleanCheckbox by its recorded ID and path.
adf.selectBooleanRadio	Identifies an ADFSelectBooleanRadio by its recorded ID and path.
adf.selectManyCheckbox	Identifies an ADFSelectManyCheckbox by its recorded ID and path.
adf.selectManyChoice	Identifies an ADFSelectManyChoice by its recorded ID and path.
adf.selectManyListbox	Identifies an ADFSelectManyListbox by its recorded ID and path.

Table 11–1 (Cont.) List of ADFService Methods

Method	Description
adf.selectManyShuttle	Identifies an ADFSelectManyShuttle by its recorded ID and path.
adf.selectOneChoice	Identifies an ADFSelectOneChoice by its recorded ID and path.
adf.selectOneListbox	Identifies an ADFSelectOneListbox by its recorded ID and path.
adf.selectOneRadio	Identifies an ADFSelectOneRadio by its recorded ID and path.
adf.selectOrderShuttle	Identifies an ADFSelectOrderShuttle by its recorded ID and path.
adf.showDetail	Identifies an ADFShowDetail by its recorded ID and path.
adf.showDetailHeader	Identifies an ADFShowDetailHeader by its recorded ID and path.
adf.table	Identifies an ADFTable by its recorded ID and path.
adf.toolbar	Identifies an ADFToolbar by its recorded ID and path.
adf.train	Identifies an ADFTrain by its path.
adf.trainButtonBar	Identifies an ADFTrain by its recorded ID and path.
adf.tree	Identifies an ADFTree by its recorded ID and path.
adf.treeTable	Identifies an ADFTreeTable by its recorded ID and path.
adf.waitForPageLoaded	Wait for ADF page to be fully loaded.

The following sections provide detailed reference information for each method and enum in the ADFService Class of ADF Module Application Programming Interface.

adf.calendar

Identifies an ADFCalendar by its recorded ID and path.

Format

The adf.calendar method has the following command format(s):

adf.calendar(*path*);

adf.calendar(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFCalendar object specified by its recorded ID and path.

```
adf.calendar(11, "/web:window[@index='0' or @title='Calendar Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFCalendar[@id='dmoTpl:cal']")
 .calendarEvent("today", 9, 0, false);
```

adf.carousel

Identifies an ADFDVTGraph by its path.

Format

The adf.carousel method has the following command format(s):

adf.carousel(*path*);

adf.carousel(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFDVTGraph object specified by its recorded ID and path.

```
adf.graph(7, "/web:window[@index='0' or @title='treeTable Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/ADFDVTGauge[@id='dmoTpl:folderTree']")
 .getAttribute("indicateValue");
```

adf.commandButton

Identifies an ADFCommandButton by its recorded ID and path.

Format

The adf.commandButton method has the following command format(s):

adf.commandButton(*path*);

adf.commandButton(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFCommandButton object specified by its recorded ID and path.

```
adf.commandButton(22, "/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();
```

adf.commandImageLink

Identifies an ADFCommandImageLink by its recorded ID and path.

Format

The adf.commandImageLink method has the following command format(s):

adf.commandImageLink(*path*);

adf.commandImageLink(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFCommandImageLink object specified by its recorded ID and path.

```
adf.commandImageLink(6, "/web:window[@index='0' " +
   or @title='commandImageLink Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFCommandImageLink[@id='dmoTpl:cil1' and @text='dmoTpl:cil1']")
 .click();
```

adf.commandLink

Identifies an ADFCommandLink by its recorded ID and path.

Format

The adf.commandLink method has the following command format(s):

adf.commandLink(*path*);

adf.commandLink(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFCommandLink object specified by its recorded ID and path.

```
adf.commandLink(32, "/web:window[@index='0' or @title='Tag Guide']" +
 "/web:ADFCommandLink[@id='tmplt:abcCols:cols:0:secs:2:itms:13:cmd' " +
  "and @text='tmplt:abcCols:cols:0:secs:2:itms:13:cmd']")
 .click();
```

adf.commandMenuItem

Identifies an ADFCommandMenuItem by its recorded ID and path.

Format

The adf.commandMenuItem method has the following command format(s):

adf.commandMenuItem(*path*);

adf.commandMenuItem(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFCommandMenuItem object specified by its recorded ID and path.

```
adf.commandMenuItem(45, "/web:window[@index='0' " +
   or @title='commandMenuItem Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFCommandMenuItem[@text='Radio 1' " +
   "and @id='dmoTpl:commandMenuItem8']")
 .click();
```

adf.commandNavigationItem

Identifies an ADFCommandNavigationItem by its recorded ID and path.

Format

The adf.commandNavigationItem method has the following command format(s):

adf.commandNavigationItem(*path*);

adf.commandNavigationItem(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFCommandNavigationItem object specified by its recorded ID and path.

```
adf.commandNavigationItem(7, "/web:window[@index='0' or @title='treeTable Demo']"
+
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/ADFCommandNavigationItem[@id='dmoTpl:folderTree']");
```

adf.commandToolbarButton

Identifies an ADFCommandToolbarButton by its recorded ID and path.

Format

The adf.commandToolbarButton method has the following command format(s):

adf.commandToolbarButton(*path*);

adf.commandToolbarButton(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFCommandToolbarButton object specified by its recorded ID and path.

```
adf.commandToolbarButton(57, "/web:window[@index='0' " +
   "or @title='commandToolbarButton Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFCommandToolbarButton[@id='dmoTpl:commandToolbarButton3' " +
   "and @text='forward']")
 .clickButton();
```

adf.dialog

Identifies an ADFDialog by its recorded ID and path.

Format

The adf.dialog method has the following command format(s):

adf.dialog(*path*);

adf.dialog(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFDialog object specified by its recorded ID and path.

```
adf.dialog(19, "/web:window[@index='0' or @title='Calendar Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFDialog[@id='dmoTpl:cal:d5' and @title='Create New Activity']")
 .closeDialog();
```

adf.gauge

Identifies an ADFDVTGraph by its recorded ID and path.

Format

The adf.gauge method has the following command format(s):

adf.gauge(*path*);

adf.gauge(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFDVTGraph object specified by its recorded ID and path.

```
adf.graph(7, "/web:window[@index='0' or @title='treeTable Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/ADFDVTGauge[@id='dmoTpl:folderTree']")
 .getAttribute("indicateValue");
```

adf.goMenuItem

Identifies an ADFGoMenuItem by its recorded ID and path.

Format

The adf.goMenuItem method has the following command format(s):

adf.goMenuItem(*path*);

adf.goMenuItem(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFGoMenuItem object specified by its recorded ID and path.

```
adf.goMenuItem(157, "/web:window[@index='0' or @title='goMenuItem Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFGoMenuItem[@id='dmoTpl:gmi7' and @text='shortDesc']")
 .click();
```

adf.graph

Identifies an ADFDVTGraph by its recorded ID and path.

Format

The adf.graph method has the following command format(s):

adf.graph(*path*);

adf.graph(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFDVTGraph object specified by its recorded ID and path.

```
adf.graph(7, "/web:window[@index='0' or @title='treeTable Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/ADFDVTGraph[@id='dmoTpl:folderTree']")
 .getAttribute("seriesCount");
```

adf.inputColor

Identifies an ADFInputColor by its recorded ID and path.

Format

The adf.inputColor method has the following command format(s):

adf.inputColor(*path*);

adf.inputColor(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFInputColor object specified by its recorded ID and path.

```
adf.inputColor(7, "/web:window[@index='0' or @title='InputColor Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFInputColor[@id='dmoTpl:iColor' and @label='Select a color']")
 .setValue("FFFFFF");
```

adf.inputComboboxListOfValues

Identifies an ADFInputComboboxListOfValues by its recorded ID and path.

Format

The adf.inputComboboxListOfValues method has the following command format(s):

adf.inputComboboxListOfValues(*path*);

adf.inputComboboxListOfValues(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFInputComboboxListOfValues object specified by its recorded ID and path.

```
adf.inputComboboxListOfValues(7, "/web:window[@index='0' " +
   "or @title='inputComboboxListOfValues Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFInputComboboxListOfValues[" +
   @id='dmoTpl:idInputComboboxListOfValues' " +
   "and @label='Ename']")
 .showDropDown();
```

adf.inputDate

Identifies an ADFInputDate by its recorded ID and path.

Format

The adf.inputDate method has the following command format(s):

adf.inputDate(*path*);

adf.inputDate(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFInputDate object specified by its recorded ID and path.

```
adf.inputDate(9, "/web:window[@index='0' or @title='inputDate Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFInputDate[@id='dmoTpl:iDate' " +
   "and @label='Select a date: ']")
 .setValue("12/12/2010");
```

adf.inputFile

Identifies an ADFInputFile by its recorded ID and path.

Format

The adf.inputFile method has the following command format(s):

adf.inputFile(*path*);

adf.inputFile(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFInputFile object specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.inputFile(7, "/web:window[@index='0' " +
   "or @title='inputFile Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFInputFile[@id='dmoTpl:testid' " +
   "and @label='File Upload']")
 .getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
  if (key.equals("id") && attributes.get(key).equals("dmoTpl:testid"))
   pass_attr++;
  if (key.equals("value") && attributes.get(key).equals(""))
   pass_attr++;
  if (key.equals("label") && attributes.get(key).equals("File Upload"))
   pass_attr++;
  if (key.equals("required") && attributes.get(key).equals("False"))
   pass_attr++;
  //info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 4)
   info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.inputListOfValues

Identifies an ADFInputListOfValues by its recorded ID and path.

Format

The adf.inputListOfValues method has the following command format(s):

adf.inputListOfValues(*path*);

adf.inputListOfValues(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFInputListOfValues object specified by its recorded ID and path.

```
adf.inputListOfValues(7, "/web:window[@index='0' " +
   "or @title='inputListOfValues Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFInputListOfValues[@id='dmoTpl:idInputText' " +
   "and @label='Ename']")
 .setValue("test", "Click");
```

adf.inputNumberSlider

Identifies an ADFInputNumberSlider by its recorded ID and path.

Format

The adf.inputNumberSlider method has the following command format(s):

adf.inputNumberSlider(*path*);

adf.inputNumberSlider(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFInputNumberSlider object specified by its recorded ID and path.

```
adf.inputNumberSlider(89, "/web:window[@index='0' " +
   or @title='inputNumberSlider Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFInputNumberSlider[@id='dmoTpl:slider12' " +
   "and @label='Vertical Slider']")
 .setValue("300");
```

adf.inputNumberSpinbox

Identifies an ADFInputNumberSpinbox by its recorded ID and path.

Format

The adf.inputNumberSpinbox method has the following command format(s):

adf.inputNumberSpinbox(*path*);

adf.inputNumberSpinbox(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFInputNumberSpinbox object specified by its recorded ID and path.

```
adf.inputNumberSpinbox(7, "/web:window[@index='0' " +
   "or @title='InputNumberSpinbox Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFInputNumberSpinbox[@id='dmoTpl:idInputNumberSpinbox' " +
   "and @label='Label']")
 .setValue("2011");
```

adf.inputRangeSlider

Identifies an ADFInputRangeSlider by its recorded ID and path.

Format

The adf.inputRangeSlider method has the following command format(s):

adf.inputRangeSlider(*path*);

adf.inputRangeSlider(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFInputRangeSlider object specified by its recorded ID and path.

```
adf.inputRangeSlider(113, "/web:window[@index='0' " +
   "or @title='inputRangeSlider Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFInputRangeSlider[@id='dmoTpl:rangeSlider1' " +
   "and @label='Horizontal Slider']")
 .setRange("0", "6");
```

adf.inputText

Identifies an ADFInputText by its recorded ID and path.

Format

The adf.inputText method has the following command format(s):

adf.inputText(*path*);

adf.inputText(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFInputText object specified by its recorded ID and path.

```
adf.inputText(68, "/web:window[@index='0' or @title='dialog Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFInputText[@id='dmoTpl:it1' and @label='* Required']")
 .setValue("1");
```

adf.menu

Identifies an ADFMenu by its recorded ID and path.

Format

The adf.menu method has the following command format(s):

adf.menu(*path*);

adf.menu(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFMenu object specified by its recorded ID and path.

```
adf.menu(44, "/web:window[@index='0' or @title='commandMenuItem Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFMenu[@id='dmoTpl:m1' and @text='File']")
 .clickShow();
```

adf.message

Identifies an ADFMessage by its recorded ID and path.

Format

The adf.message method has the following command format(s):

adf.message(*path*);

adf.message(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFMessage object specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.message(7, "/web:window[@index='0' " +
   "or @title='message Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFMessage[@id='dmoTpl:message1']")
 .getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
 if (key.equals("id") && attributes.get(key).equals("dmoTpl:message1"))
   pass_attr++;
 if (key.equals("message") && attributes.get(key).equals(""))
   pass_attr++;
 if (key.equals("shortDesc") && attributes.get(key).equals(""))
   pass_attr++;
 if (key.equals("messageType") && attributes.get(key).equals("none"))
   pass_attr++;
 // info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 4)
   info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.messages

Identifies an ADFMessages by its recorded ID and path.

Format

The adf.messages method has the following command format(s):

adf.messages(*path*);

adf.messages(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFMessages object specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.messages(7, "/web:window[@index='0' " +
   "or @title='messages Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFMessages[@id='dmoTpl:messages1']").getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
 if (key.equals("id") && attributes.get(key)
   .equals("dmoTpl:messages1"))
   pass_attr++;
 if (key.equals("message") && attributes.get(key)
   .equals("Please read this carefully:"))
   pass_attr++;
 if (key.equals("shortDesc") && attributes.get(key)
   .equals("Sample shortDesc text"))
   pass_attr++;
 if (key.equals("rendererType") && attributes.get(key)
   .equals("oracle.adf.rich.Messages"))
   pass_attr++;
 // info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 4)
   info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.navigationPane

Identifies an ADFNavigationPane by its recorded ID and path.

Format

The adf.navigationPane method has the following command format(s):

adf.navigationPane(*path*);

adf.navigationPane(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFNavigationPane object specified by its recorded ID and path.

```
adf.navigationPane(152, "/web:window[@index='0' " +
   or @title='navigationPane Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFNavigationPane[@id='dmoTpl:barExample']")
 .select("Bar Item 1", "xdc xh5");
```

adf.noteWindow

Identifies an ADFNoteWindow by its recorded ID and path.

Format

The adf.noteWindow method has the following command format(s):

adf.noteWindow(*path*);

adf.noteWindow(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFNoteWindow object specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.noteWindow(
 7, "/web:window[@index='0' " +
   "or @title='noteWindow Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFNoteWindow[@id='dmoTpl:nw1']")
 .getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
 if (key.equals("id") && attributes.get(key).equals("dmoTpl:nw1"))
   pass_attr++;
 if (key.equals("content") && !attributes.get(key).equals(""))
   pass_attr++;
 // info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 2)
   info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.outputFormatted

Identifies an ADFOutputFormat by its recorded ID and path.

Format

The adf.outputFormatted method has the following command format(s):

adf.outputFormatted(*path*);

adf.outputFormatted(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFOutputFormatobject specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.outputLabel(
 7, "/web:window[@index='0' " +
   "or @title='outputLabel Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFOutputFormat[@id='dmoTpl:_OutputLabel']")
 .getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
 if (key.equals("id") && attributes.get(key).equals("dmoTpl:_OutputLabel"))
   pass_attr++;
 if (key.equals("value") && attributes.get(key).equals("Test Output Label"))
   pass_attr++;
 // info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 2)
   info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.outputLabel

Identifies an ADFOutputLabel by its recorded ID and path.

Format

The adf.outputLabel method has the following command format(s):

adf.outputLabel(*path*);

adf.outputLabel(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFOutputLabelobject specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.outputLabel(
 7, "/web:window[@index='0' " +
   "or @title='outputLabel Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFOutputLabel[@id='dmoTpl:_OutputLabel']")
 .getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
 if (key.equals("id") && attributes.get(key).equals("dmoTpl:_OutputLabel"))
   pass_attr++;
 if (key.equals("value") && attributes.get(key).equals("Test Output Label"))
   pass_attr++;
 // info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 2)
   info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.outputText

Identifies an ADFOutputText by its recorded ID and path.

Format

The adf.outputText method has the following command format(s):

adf.outputText(*path*);

adf.outputText(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFOutputText object specified by its recorded ID and path.

```
adf.outputText(7, "/web:window[@index='0' or @title='treeTable Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFOutputText[@id='dmoTpl:folderTree']")
 .getAttribute("value");
```

adf.page

Identifies an ADFPage by its recorded ID and path.

Format

The adf.page method has the following command format(s):

adf.page(*path*);

adf.page(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFPage specified by its recorded ID and path.

```
adf.page(5, "/web:window[@index='0' or @title='Shop Fusion Order Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFPage[@title='Shop Fusion Order Demo']")
 .waitForPage(10000);
```

adf.panelAccordion

Identifies an ADFPanelAccordion by its recorded ID and path.

Format

The adf.panelAccordion method has the following command format(s):

adf.panelAccordion(*path*);

adf.panelAccordion(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFPanelAccordion specified by its recorded ID and path.

```
adf.panelAccordion(8, "/web:window[@index='0' " +
   "or @title='panelAccordion Demo']/" +
 "web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFPanelAccordion[@id='dmoTpl:sampleAccordion']")
 .expand("dmoTpl:doubleFlexPane");
```

adf.panelBox

Identifies an ADFPanelBox by its recorded ID and path.

Format

The adf.panelBox method has the following command format(s):

adf.panelBox(*path*);

adf.panelBox(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFPanelBox specified by its recorded ID and path.

```
adf.panelBox(8, "/web:window[@index='0' or @title='panelBox Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFPanelBox[@id='dmoTpl:panelBoxB3' " +
   "and @text='Highlight Light']")
.expand();
```

adf.panelHeader

Identifies an ADFPanelHeader by its recorded ID and path.

Format

The adf.panelHeader method has the following command format(s):

adf.panelHeader(*path*);

adf.panelHeader(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFPanelHeaderobject specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.panelHeader(
 7, "/web:window[@index='0' " +
   "or @title='outputLabel Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFPanelHeader[@id='dmoTpl:_OutputLabel']")
 .getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
 if (key.equals("id") && attributes.get(key).equals("dmoTpl:_OutputLabel"))
   pass_attr++;
 if (key.equals("value") && attributes.get(key).equals("Test Output Label"))
   pass_attr++;
 // info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 2)
   info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.panelLabelAndMessage

Identifies an ADFPanelLabelAndMessage by its recorded ID and path.

Format

The adf.panelLabelAndMessage method has the following command format(s):

adf.panelLabelAndMessage(*path*);

adf.panelLabelAndMessage(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFPanelLabelAndMessageobject specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.panelLabelAndMessage(
 7, "/web:window[@index='0' " +
   "or @title='outputLabel Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFPanelLabelAndMessage[@id='dmoTpl:_OutputLabel']")
 .getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
 if (key.equals("id") && attributes.get(key).equals("dmoTpl:_OutputLabel"))
   pass_attr++;
 if (key.equals("value") && attributes.get(key).equals("Test Output Label"))
   pass_attr++;
 // info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 2)
   info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.panelList

Identifies an ADFPanelList by its recorded ID and path.

Format

The adf.panelList method has the following command format(s):

adf.panelList(*path*);

adf.panelList(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFPanelList specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.panelList(8, "/web:window[@index='0' " +
   "or @title='panelList Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFPanelList[@id='dmoTpl:panelList']")
 .getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
 if (key.equals("id") && attributes.get(key).equals("dmoTpl:panelList"))
   pass_attr++;
 if (key.equals("maxColumns") && attributes.get(key).equals("3"))
   pass_attr++;
 if (key.equals("rows") && attributes.get(key).equals("2147483647"))
 pass_attr++;
 // info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 3)
 info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.panelSplitter

Identifies an ADFPanelSplitter by its recorded ID and path.

Format

The adf.panelSplitter method has the following command format(s):

adf.panelSplitter(*path*);

adf.panelSplitter(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFPanelSplitter specified by its recorded ID and path.

```
adf.panelSplitter(6, "/web:window[@index='0' " +
   "or @title='panelSplitter Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFPanelSplitter[@id='dmoTpl:outerSplitter']")
 .collapse();
```

adf.panelTabbed

Identifies an ADFPanelTabbed by its recorded ID and path.

Format

The adf.panelTabbed method has the following command format(s):

adf.panelTabbed(*path*);

adf.panelTabbed(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFPanelTabbed specified by its recorded ID and path.

```
adf.panelTabbed(8, "/web:window[@index='0' or @title='panelTabbed Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFPanelTabbed[@id='dmoTpl:ShowOneTab']")
 .selectTab("");
```

adf.panelWindow

Identifies an ADFPanelWindow by its recorded ID and path.

Format

The adf.panelWindow method has the following command format(s):

adf.panelWindow(*path*);

adf.panelWindow(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFPanelWindow specified by its recorded ID and path.

```
adf.commandButton(7, "/web:window[@index='0' or @title='panelWindow Demo']" +
 "/web:document[@index='0']" +
 "/web:ADFCommandButton[@id='dmoTpl:commandButton' " +
   "and @text='Show Window']")
.click();
adf.panelWindow(8, "/web:window[@index='0' or @title='panelWindow Demo']" +
 "/web:document[@index='0']" +
 "/web:ADFPanelWindow[@id='dmoTpl:panelWindow1' " +
   "and @title='Test Window']")
.clickCloseIcon();
```

adf.progressIndicator

Identifies an ADFProgressIndicator by its recorded ID and path.

Format

The adf.progressIndicator method has the following command format(s):

adf.progressIndicator(*path*);

adf.progressIndicator(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFProgressIndicator specified by its recorded ID and path.

```
import java.util.Map;
[...]
Map<String, String> attributes = adf.progressIndicator(
 7, "/web:window[@index='0' or @title='progressIndicator Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFProgressIndicator[@id='dmoTpl:progressIndicator']")
 .getAttributes();
int pass_attr = 0;
for (String key : attributes.keySet()) {
 if (key.equals("id") && attributes.get(key)
     .equals("dmoTpl:progressIndicator"))
   pass_attr++;
 if (key.equals("value") && attributes.get(key).equals("0%"))
   pass_attr++;
 // info("Attribute: " + key + " ; " + "Value: " + attributes.get(key));
}
if (pass_attr == 2)
   info("API getAttributes() passed");
else {
   info(pass_attr + " attributes passed");
   fail("API getAttributes() failed");
}
```

adf.query

Identifies an ADFQuery by its recorded ID and path.

Format

The adf.query method has the following command format(s):

adf.query(*path*);

adf.query(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFQuery specified by its recorded ID and path.

```
adf.query(11, "/web:window[@index='0' or @title='query Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFQuery[@id='dmoTpl:highSalariedClerksQueryId' " +
   "and @headerText='Search']")
 .clickButton("Search");
```

adf.quickQuery

Identifies an ADFQuickQuery by its recorded ID and path.

Format

The adf.quickQuery method has the following command format(s):

adf.quickQuery(*path*);

adf.quickQuery(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFQuickQuery specified by its recorded ID and path.

```
adf.quickQuery(8, "/web:window[@index='0' or @title='quickQuery Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFQuickQuery[@id='dmoTpl:search' and @label='Search']")
 .click();
```

adf.resetButton

Identifies an ADFResetButton by its recorded ID and path.

Format

The adf.resetButton method has the following command format(s):

adf.resetButton(*path*);

adf.resetButton(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFResetButton object specified by its recorded ID and path.

```
adf.resetButton(351,"/web:window[@index='0' or @title='resetButton Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFResetButton[@id='dmoTpl:idResetButton' " +
   "and @text='Rich ResetButton']")
 .click();
```

adf.richTextEditor

Identifies an ADFRichTextEditor by its recorded ID and path.

Format

The adf.richTextEditor method has the following command format(s):

adf.richTextEditor(*path*);

adf.richTextEditor(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFRichTextEditor object specified by its recorded ID and path.

```
adf.richTextEditor(35, "/web:window[@index='0' " +
   "or @title='richTextEditor Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFRichTextEditor[@id='dmoTpl:idRichTextEditor' " +
   "and @label='Rich text value']")
 .setValue("<P dir=ltr style=\"MARGIN-RIGHT: 0px\">Hello World</P>\r\n");
```

adf.selectBooleanCheckbox

Identifies an ADFSelectBooleanCheckbox by its recorded ID and path.

Format

The adf.selectBooleanCheckbox method has the following command format(s):

adf.selectBooleanCheckbox(*path*);

adf.selectBooleanCheckbox(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectBooleanCheckbox object specified by its recorded ID and path.

```
adf.selectBooleanCheckbox(47,"/web:window[@index='0' " +
   "or @title='selectBooleanCheckbox Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectBooleanCheckbox[@id='dmoTpl:idSBC2' " +
   "and @label='Extra Keys']")
 .select();
```

adf.selectBooleanRadio

Identifies an ADFSelectBooleanRadio by its recorded ID and path.

Format

The adf.selectBooleanRadio method has the following command format(s):

adf.selectBooleanRadio(*path*);

adf.selectBooleanRadio(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectBooleanRadio object specified by its recorded ID and path.

```
adf.selectBooleanRadio(8, "/web:window[@index='0' " +
   "or @title='selectBooleanRadio Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectBooleanRadio[@id='dmoTpl:radio1' " +
   "and @label='Age']")
 .select("10-18");
```

adf.selectManyCheckbox

Identifies an ADFSelectManyCheckbox by its recorded ID and path.

Format

The adf.selectManyCheckbox method has the following command format(s):

adf.selectManyCheckbox(*path*);

adf.selectManyCheckbox(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectManyCheckbox object specified by its recorded ID and path.

```
adf.selectManyCheckbox(8, "/web:window[@index='0' " +
   "or @title='selectManyCheckbox Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectManyCheckbox[@id='dmoTpl:targetListbox' " +
   "and @label='Drinks']")
 .select("milk", "4");
```

adf.selectManyChoice

Identifies an ADFSelectManyChoice by its recorded ID and path.

Format

The adf.selectManyChoice method has the following command format(s):

adf.selectManyChoice(*path*);

adf.selectManyChoice(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectManyChoice object specified by its recorded ID and path.

```
adf.selectManyChoice(7, "/web:window[@index='0' " +
   "or @title='selectManyChoice Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectManyChoice[@id='dmoTpl:targetChoice' " +
   "and @label='Drinks']")
 .select("coffee");
```

adf.selectManyListbox

Identifies an ADFSelectManyListbox by its recorded ID and path.

Format

The adf.selectManyListbox method has the following command format(s):

adf.selectManyListbox(*path*);

adf.selectManyListbox(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectManyListbox object specified by its recorded ID and path.

```
adf.selectManyListbox(8, "/web:window[@index='0' " +
   "or @title='selectManyListbox Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectManyListbox[@id='dmoTpl:targetListbox' " +
   "and @label='Drinks']")
 .select("tea;green", "1");
```

adf.selectManyShuttle

Identifies an ADFSelectManyShuttle by its recorded ID and path.

Format

The adf.selectManyShuttle method has the following command format(s):

adf.selectManyShuttle(*path*);

adf.selectManyShuttle(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectManyShuttle object specified by its recorded ID and path.

```
adf.selectManyShuttle(18, "/web:window[@index='0' " +
   "or @title='selectManyShuttle Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectManyShuttle[@id='dmoTpl:manyShuttle' " +
 "and @label='Drinks']")
 .move();
```

adf.selectOneChoice

Identifies an ADFSelectOneChoice by its recorded ID and path.

Format

The adf.selectOneChoice method has the following command format(s):

adf.selectOneChoice(*path*);

adf.selectOneChoice(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectOneChoice object specified by its recorded ID and path.

```
adf.selectOneChoice(127, "/web:window[@index='0' " +
   "or @title='selectOneChoice Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectOneChoice[@id='dmoTpl:targetChoice' " +
   "and @label='Drink']")
 .select("coffee");
```

adf.selectOneListbox

Identifies an ADFSelectOneListbox by its recorded ID and path.

Format

The adf.selectOneListbox method has the following command format(s):

adf.selectOneListbox(*path*);

adf.selectOneListbox(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectOneListbox object specified by its recorded ID and path.

```
adf.selectOneListbox(7, "/web:window[@index='0' " +
   "or @title='selectOneListbox Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectOneListbox[@id='dmoTpl:targetListbox' " +
   "and @label='Drinks']")
 .select("tea;green");
```

adf.selectOneRadio

Identifies an ADFSelectOneRadio by its recorded ID and path.

Format

The adf.selectOneRadio method has the following command format(s):

adf.selectOneRadio(*path*);

adf.selectOneRadio(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectOneRadio object specified by its recorded ID and path.

```
adf.selectOneRadio(7, "/web:window[@index='0' " +
   "or @title='selectOneRadio Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectOneRadio[@id='dmoTpl:targetRadio' " +
   "and @label='Drinks']")
 .select("lemonade");
```

adf.selectOrderShuttle

Identifies an ADFSelectOrderShuttle by its recorded ID and path.

Format

The adf.selectOrderShuttle method has the following command format(s):

adf.selectOrderShuttle(*path*);

adf.selectOrderShuttle(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFSelectOrderShuttle object specified by its recorded ID and path.

```
adf.selectOrderShuttle(11, "/web:window[@index='0' " +
   "or @title='selectOrderShuttle Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFSelectOrderShuttle[@id='dmoTpl:manyShuttle' " +
   "and @label='Drinks']")
 .moveUp();
```

adf.showDetail

Identifies an ADFShowDetail by its recorded ID and path.

Format

The adf.showDetail method has the following command format(s):

adf.showDetail(*path*);

adf.showDetail(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFShowDetail object specified by its recorded ID and path.

```
adf.showDetail(7, "/web:window[@index='0' " +
   "or @title='showDetail Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFShowDetail[@id='dmoTpl:showDetail']")
 .expand();
```

adf.showDetailHeader

Identifies an ADFShowDetailHeader by its recorded ID and path.

Format

The adf.showDetailHeader method has the following command format(s):

adf.showDetailHeader(*path*);

adf.showDetailHeader(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFShowDetailHeader object specified by its recorded ID and path.

```
adf.showDetailHeader(7, "/web:window[@index='0' " +
   "or @title='showDetailHeader Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFShowDetailHeader[@id='dmoTpl:showDetailHeader' " +
   "and @text='Automatic Header']")
 .expand();
```

adf.table

Identifies an ADFTable by its recorded ID and path.

Format

The adf.table method has the following command format(s):

adf.table(*path*);

adf.table(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFTable object specified by its recorded ID and path.

```
String getCell = adf.table(11, "/web:window[@index='0' " +
   "or @title='table Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFTable[@id='dmoTpl:table']")
 .getCell(1, 1);
info("Cell Value = " + getCell);
```

adf.toolbar

Identifies an ADFToolbar by its recorded ID and path.

Format

The adf.toolbar method has the following command format(s):

adf.toolbar(*path*);

adf.toolbar(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFToolbar object specified by its recorded ID and path.

```
adf.toolbar(7, "/web:window[@index='0' " +
   "or @title='toolbar Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFToolbar[@id='dmoTpl:toolbar2']")
 .expand();
```

adf.train

Identifies an ADFTrain by its path.

Format

The adf.train method has the following command format(s):

adf.train(*path*);

adf.train(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFTrain object specified by its recorded ID and path.

```
adf.train(7, "/web:window[@index='0' or @title='train Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFTrain[@id='dmoTpl:train']")
 .clickNode("Current Step: Second Step");
```

adf.trainButtonBar

Identifies an ADFTrain by its recorded ID and path.

Format

The adf.trainButtonBar method has the following command format(s):

adf.trainButtonBar(*path*);

adf.trainButtonBar(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFTrain object specified by its recorded ID and path.

```
adf.trainButtonBar(7, "/web:window[@index='0' " +
   "or @title='trainButtonBar Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFTrainButtonBar[@id='dmoTpl:trainBB']")
 .clickNode("next");
```

adf.tree

Identifies an ADFTree by its recorded ID and path.

Format

The adf.tree method has the following command format(s):

adf.tree(*path*);

adf.tree(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFTree object specified by its recorded ID and path.

```
adf.tree(14, "/web:window[@index='0' or @title='tree Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFTree[@id='dmoTpl:folderTree']")
 .expandCollapse("Expand", "2");
```

adf.treeTable

Identifies an ADFTreeTable by its recorded ID and path.

Format

The adf.treeTable method has the following command format(s):

adf.treeTable(*path*);

adf.treeTable(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an ADFTreeTable object specified by its recorded ID and path.

```
adf.treeTable(7, "/web:window[@index='0' or @title='treeTable Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']" +
 "/web:ADFTreeTable[@id='dmoTpl:folderTree']")
 .expandCollapse("Collapse", "2");
```

adf.waitForPageLoaded

Wait for ADF page to be fully loaded. It would use the object time out value as the default time out value.

Format

The adf.waitForPageLoaded method has the following command format(s):

adf.waitForPageLoaded(*path*);

adf.waitForPageLoaded(*path*, *timeout*);

Command Parameters

path
a String specifying the document path.

timeout
the timeout value (milisecond) of the wait action.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Wait for page fully loaded.

```
adf.waitForPageLoaded("/web:window[@index='0' " +
   "or @title='Shop Fusion Order Demo']" +
 "/web:document[@index='0' or @name='_afr_init_']", 3);
```

12

Oracle JD Edwards EnterpriseOne Functional Module

This chapter provides a complete listing and reference for the methods in the OpenScript EnterpriseOneService Class of EnterpriseOne Module Application Programming Interface (API).

12.1 EnterpriseOneService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript EnterpriseOneService API.

12.1.1 Alphabetical Command Listing

The following table lists the EnterpriseOneService API methods in alphabetical order.

Table 12–1 List of EnterpriseOneService Methods

Method	Description
eone.grid	Identifies an EnterpriseOne Grid Control by its recorded ID and path.

The following sections provide detailed reference information for each method and enum in the EnterpriseOneService Class of EnterpriseOne Module Application Programming Interface.

eone.grid

Identifies an EnterpriseOne Grid Control by its recorded ID and path.

Format

The eone.grid method has the following command format(s):

eone.grid(*path*);

eone.grid(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Performs an action on an EOneGrid object specified by its recorded ID and path.

```
eone.grid(11,
 "/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);
```

13

Oracle EBS/Forms Functional Module

This chapter provides a complete listing and reference for the methods in the OpenScript FormsService Class of Forms Functional Module Application Programming Interface (API).

13.1 FormsService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript FormsService API.

13.1.1 Alphabetical Command Listing

The following table lists the FormsService API methods in alphabetical order.

Table 13–1 List of FormsService Methods

Method	Description
forms.alertDialog	Identifies an Oracle Forms alertDialog object by its path.
forms.appletAdapter	Identifies an Oracle Forms AppletAdapter object by its applet name.
forms.attribute	Construct property data of a single attribute.
forms.attributes	Utility function to return a list of property tests.
forms.blockScroller	Identifies an Oracle Forms blockScroller object by its path.
forms.button	Identifies an Oracle Forms Button object by its path.
forms.calendar	Identifies an Oracle Forms calendar object by its path.
forms.captureScreenshot	Capture the screen shot at runtime.
forms.cell	Constructs the test data of a table cell.
forms.cells	Constructs the list of table cell tests.
forms.checkBox	Identifies an Oracle Forms checkBox object by its path.
forms.choiceBox	Identifies an Oracle Forms choiceBox object by its path.
forms.close	Close the running forms.
forms.comboBox	Identifies an Oracle Forms comboBox object by its path.
forms.editBox	Identifies an Oracle Forms editBox object by its path.
forms.editorDialog	Identifies an Oracle Forms EditorDialog object by its path.
forms.flexWindow	Identifies an Oracle Forms FlexWindow object by its path.

Table 13–1 (Cont.) List of FormsService Methods

Method	Description
forms.getAllObjects	Gets all forms objects.
forms.getStatusBarCount	Gets the bar count shown in the bottom of the forms client.
forms.getStatusBarErrorCode	Gets the error code shown in the status bar of the forms client.
forms.getStatusBarItem	Gets the item shown in the status bar of forms client.
forms.getStatusBarItemCount	Gets the item count shown in the status bar of forms client.
forms.getStatusBarMessage	Gets the message shown in the status bar of forms client.
forms.getStatusBarRecordInfo	Gets the record info shown in the status bar of the forms client.
forms.hGridApp	Identifies an Oracle Forms HGridApp object by its path.
forms.helpDialog	Identifies an Oracle Forms helpDialog object by its path.
forms.imageItem	Identifies an Oracle Forms imageItem object by its path.
forms.infoBox	Identifies an Oracle Forms infoBox object by its path.
forms.list	Identifies an Oracle Forms list object by its path.
forms.listOfValues	Identifies an Oracle Forms listOfValues object by its path.
forms.logonDialog	Identifies an Oracle Forms logonDialog object by its path.
forms.otsHGrid	Identifies an Oracle Forms otsHGrid object by its path.
forms.radioButton	Identifies an Oracle Forms radioButton object by its path.
forms.responseBox	Identifies an Oracle Forms responseBox object by its path.
forms.schedulingDataClient	Identifies an Oracle Forms schedulingDataClient object by its path.
forms.spreadTable	Identifies an Oracle Forms spreadTable object by its path.
forms.statusBar	Identifies an Oracle Forms StatusBar object by its recorded ID and path.
forms.tab	Identifies an Oracle Forms tab object by its path.
forms.tableBox	Identifies an Oracle Forms tableBox object by its path.
forms.textField	Identifies an Oracle Forms textField object by its path.
forms.tree	Identifies an Oracle Forms tree object by its path.
forms.treeList	Identifies an Oracle Forms treeList object by its path.
forms.window	Identifies an Oracle Forms window object by its path.

The following sections provide detailed reference information for each method and enum in the FormsService Class of Forms Functional Module Application Programming Interface.

forms.alertDialog

Identifies an Oracle Forms alertDialog object by its path.

Format

The forms.alertDialog method has the following command format(s):

forms.alertDialog(*path*);

forms.alertDialog(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms alertDialog. For example, `//forms:alertDialog`. Must not be `null`. Since it is only possible to show one alertDialog in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the alertDialog object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms alertDialog.

Example

Performs an action on an Oracle Forms alertDialog object specified by its recorded ID and path.

```
String msg=forms.alertDialog(128, "//forms:alertDialog")
 .getAlertMessage();
info("alertDialog: " + msg);
forms.alertDialog(128, "//forms:alertDialog")
 .clickNo();
```

forms.appletAdapter

Identifies an Oracle Forms AppletAdapter object by its applet name.

Format

The forms.appletAdapter method has the following command format(s):

forms.appletAdapter(*path*);

forms.appletAdapter(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the applet name to identify the forms appletAdater. For example, `//forms:appletAdapter[(@appletName='NAV_CONTROLS_EXPAND_ALL_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms AppletAdapter.

Example

Performs an action on an Oracle Forms appletAdapter object specified by its recorded ID and applet name.

```
forms.appletAdapter(10, "//forms:appletAdapter[" +
 "(@appletName='NAV_CONTROLS_EXPAND_ALL_0')]")
 .execute("value", "prompt");
```

forms.attribute

Construct property data of a single attribute.

Format

The forms.attribute method has the following command format(s):

forms.attribute(*property*, *expectedValue*, *operator*);

Command Parameters

property
a String specifying the property name.

expectedValue
a String specifying the expected value.

operator
a TestOperator enum defining the type and the match approach of the data.

Returns

a PropertyTest object.

Example

Specify the attributes of an object.

```
forms.textField(248, "//forms:textField[" +
 "(@name='STUB_APPLICATION_NAME_0')]")
 .setFocus();
forms.textField(249,"//forms:textField[" +
 "(@name='STUB_APPLICATION_NAME_0')]")
 .setText("abc");
forms.textField("//forms:textField[" +
 "(@name='STUB_APPLICATION_NAME_0')]")
 .verifyAttributes("test1",forms.attributes(
   forms.attribute("enabled", "true",
     TestOperator.StringExact),
   forms.attribute("text", "Sales",
     TestOperator.StringExact),
   forms.attribute("visible", "true",
     TestOperator.StringExact),
   forms.attribute("name", "STUB_APPLICATION_NAME_0",
     TestOperator.StringExact),
   forms.attribute("showing", "true",
     TestOperator.StringExact),
   forms.attribute("prompt", "Application",
     TestOperator.StringExact),
   forms.attribute("editable", "true",
     TestOperator.StringExact)));
```

forms.attributes

Utility function to return a list of property tests.

Format

The forms.attributes method has the following command format(s):

forms.attributes(*tests*);

Command Parameters

tests
an array of PropertyTest attribute objects specifying all the tests included in the forms object test.

Returns

a new PropertyTest[].

Example

Specify the attributes of an object.

```
forms.textField(248, "//forms:textField[" +
 "(@name='STUB_APPLICATION_NAME_0')]")
 .setFocus();
forms.textField(249,"//forms:textField[" +
 "(@name='STUB_APPLICATION_NAME_0')]")
 .setText("abc");
forms.textField("//forms:textField[" +
 "(@name='STUB_APPLICATION_NAME_0')]")
 .verifyAttributes("test1",forms.attributes(
   forms.attribute("enabled", "true",
     TestOperator.StringExact),
   forms.attribute("text", "Sales",
     TestOperator.StringExact),
   forms.attribute("visible", "true",
     TestOperator.StringExact),
   forms.attribute("name", "STUB_APPLICATION_NAME_0",
     TestOperator.StringExact),
   forms.attribute("showing", "true",
     TestOperator.StringExact),
   forms.attribute("prompt", "Application",
     TestOperator.StringExact),
   forms.attribute("editable", "true",
     TestOperator.StringExact)));
```

forms.blockScroller

Identifies an Oracle Forms blockScroller object by its path.

Format

The forms.blockScroller method has the following command format(s):

forms.blockScroller(*path*);

forms.blockScroller(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms blockScroller. For example, `//forms:blockScroller`. Must not be `null`. Without further notice, this method only works properly when only one blockScroller exists in a window.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms blockScroller.

Example

Performs an action on an Oracle Forms blockScroller object specified by its recorded ID and path.

```
int maxNum=forms.blockScroller(50, "//forms:blockScroller")
 .getMaximum();
int minNum=forms.blockScroller50, "//forms:blockScroller")
 .getMinimum();
int value=forms.blockScroller(50, "//forms:blockScroller")
 .getValue();
info("maxNum:"+maxNum+" minNum:"+minNum+" value8:"+value);
forms.blockScroller(50, "//forms:blockScroller")
 .scrollDown();
int scrollDownValue=forms.blockScroller(50, "//forms:blockScroller")
 .getValue();
forms.blockScroller(50, "//forms:blockScroller")
 .scrollUp();
int scrollUpValue=forms.blockScroller(50, "//forms:blockScroller")
 .getValue();
   if(value!=8)warn("scroll to error!");
   if(scrollDownValue!=9)warn("scroll to error!");
   if(scrollUpValue!=8)warn("scroll up error!");
```

forms.button

Identifies an Oracle Forms Button object by its path.

Format

The forms.button method has the following command format(s):

forms.button(*path*);

forms.button(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms button. For example, `//forms:button[(@name='NAV_CONTROLS_EXPAND_ALL_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms Button object.

Example

Performs an action on an Oracle Forms Button object specified by its recorded ID and path.

```
forms.button(32, "//forms:button[(@name='CALENDAR_OK_0')]")
 .setFocus();
String accName1=forms.button(32, "//forms:button[(@name='CALENDAR_OK_0')]")
 .getAccessibleName();
String label1=forms.button(32, "//forms:button[(@name='CALENDAR_OK_0')]")
 .getLabel();
info("accssibleName: " + accName1);
info("Label: " + label1);
```

forms.calendar

Identifies an Oracle Forms calendar object by its path.

Format

The forms.calendar method has the following command format(s):

forms.calendar(*path*);

forms.calendar(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms calendar. For example, `//forms:calendar`. Must not be `null`. Since it is only possible to show one calendar in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the calendar object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms calendar.

Example

Performs an action on an Oracle Forms calendar object specified by its recorded ID and path.

```
forms.textField(111, "//forms:textField[" +
 "(@name='GROUPS_EXPENDITURE_ENDING_DATE_0')]")
 .openDialog();
forms.button(114, "//forms:button[" +
 "(@name='CALENDAR_OK_0')]")
 .setFocus();
forms.calendar(115, "//forms:calendar")
 .enter("06-JAN-2011");
forms.window(118, "//forms:window[" +
 "(@name='GROUPS_DETAIL')]")
 .activate(true);
forms.textField(119, "//forms:textField[" +
 "(@name='GROUPS_EXPENDITURE_ENDING_DATE_0')]")
 .openDialog();
forms.button(122, "//forms:button[" +
 "(@name='CALENDAR_OK_0')]")
 .setFocus();
forms.calendar(123, "//forms:calendar")
 .clickOk();
```

forms.captureScreenshot

Capture the screen shot at runtime. Please refer to {@link #captureScreenshot(Integer,String)}.

Format

The forms.captureScreenshot method has the following command format(s):

forms.captureScreenshot( );

forms.captureScreenshot(*recid*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Captures the screen shot at runtime.

```
forms.captureScreenshot(103);
```

forms.cell

Constructs the test data of a table cell.

Format

The forms.cell method has the following command format(s):

forms.cell(*row*, *column*, *expectedValue*, *operator*);

Command Parameters

row
a 1-based index value specifying the row number.

column
a 1-based index value specifying the column number.

expectedValue
a String specifying the expected value of the property.

operator
a TestOperator enum that defines the type and the match approach of the data.

Returns

a CellTest object with the specified row, column, expectedValue, and operator.

Example

Adds a table test for specifying exact match or wildcard match for individual cells.

```
forms.otsHGrid("//forms:otsHGrid[(@name='ASTEVENR_CONF_HGRID_ITEMS_0')]" +
 .assertCells("MyTableTest",
   forms.cells(forms.cell(1, 1, "Ticker ",TestOperator.StringExact),
     forms.cell(1, 2,"Company ", TestOperator.StringExact),
     forms.cell(2, 1, "ORCL*", TestOperator.StringWildCard),
     forms.cell(2, 2, "Oracle", TestOperator.StringExact)));
```

forms.cells

Constructs the list of table cell tests.

Format

The forms.cells method has the following command format(s):

forms.cells(*tests*);

Command Parameters

tests
a list of table cells.

Returns

a CellTest[] with the specified cells.

Example

Adds a table test for specifying exact match or wildcard match for each cell.

```
forms.otsHGrid("//forms:otsHGrid[(@name='ASTEVENR_CONF_HGRID_ITEMS_0')]" +
 .assertCells("MyTableTest",
   web.cells(forms.cell(1, 1, "Ticker ",TestOperator.StringExact),
     forms.cell(1, 2,"Company ", TestOperator.StringExact),
     forms.cell(2, 1, "ORCL*", TestOperator.StringWildCard),
     forms.cell(2, 2, "Oracle", TestOperator.StringExact)));
```

forms.checkBox

Identifies an Oracle Forms checkBox object by its path.

Format

The forms.checkBox method has the following command format(s):

forms.checkBox(*path*);

forms.checkBox(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms checkBox. For example, `//forms:checkBox[(@name='MTL_SYSTEM_ITEMS_INVENTORY_ITEM_FLAG_MIR_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms checkBox.

Example

Performs an action on an Oracle Forms checkBox object specified by its recorded ID and path.

```
forms.checkBox(35, "//forms:checkBox[" +
 "(@name='JOBS_QF_SHOW_STAGES_0')]")
 .check(true);
forms.checkBox(35, "//forms:checkBox[" +
 "(@name='JOBS_QF_SHOW_STAGES_0')]")
 .verifyAttribute("checkbox status", "state",
  "true", TestOperator.StringExact);
if( !forms.checkBox(35, "//forms:checkBox[" +
 "(@name='JOBS_QF_SHOW_STAGES_0')]")
 .isSelected())warn("check statement error");
String accessibleName=forms.checkBox(35, "//forms:checkBox[" +
 "(@name='JOBS_QF_SHOW_STAGES_0')]")
 .getAccessibleName();
String label=forms.checkBox(35, "//forms:checkBox[" +
 "(@name='JOBS_QF_SHOW_STAGES_0')]")
 .getLabel();
//output accessible name and label
info("check box's acccessible name: " + accessibleName);
info("check box's label: " + label);
```

forms.choiceBox

Identifies an Oracle Forms choiceBox object by its path.

Format

The forms.choiceBox method has the following command format(s):

forms.choiceBox(*path*);

forms.choiceBox(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms choiceBox. For example, `//forms:choiceBox`. Must not be `null`. Since it is only possible to show one choiceBox in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the choiceBox object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms choiceBox.

Example

Performs an action on an Oracle Forms choiceBox object specified by its recorded ID and path.

```
forms.choiceBox(147, "//forms:choiceBox").clickButton("OK");
```

forms.close

Close the running forms. Please refer to forms.close

Format

The forms.close method has the following command format(s):

forms.close( );

forms.close(*recid*);

Command Parameters

recid
the ID of a previously recorded element which is another corresponding element to this, holding all the info recorded.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Closes the running forms.

```
forms.close(144);
```

forms.comboBox

Identifies an Oracle Forms comboBox object by its path.

Format

The forms.comboBox method has the following command format(s):

forms.comboBox(*path*);

forms.comboBox(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms comboBox. For example, `//forms:comboBox[(@name='ALL_WIDGETS_LIST508_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms comboBox.

Example

Performs an action on an Oracle Forms comboBox object specified by its recorded ID and path.

```
//get accessible name and count
String accessName=forms.comboBox(20, "//forms:comboBox[" +
 "(@name='ALL_WIDGETS_LIST508_0')]")
 .getAccessibleName();
int itemCount=forms.comboBox(20, "//forms:comboBox[" +
 "(@name='ALL_WIDGETS_LIST508_0')]")
 .getItemCount();
info("accessible Name: " + accessName);
info("item count: " + itemCount);
forms.comboBox(20, "//forms:comboBox[" +
 "(@name='ALL_WIDGETS_LIST508_0')]")
 .setFocus();
forms.comboBox(20, "//forms:comboBox[" +
 "(@name='ALL_WIDGETS_LIST508_0')]")
 .setText("text");
String itemValue = forms.comboBox(20, "//forms:comboBox[" +
 "(@name='ALL_WIDGETS_LIST508_0')]")
 .getItemValue("");
info(itemValue);
```

forms.editBox

Identifies an Oracle Forms editBox object by its path.

Format

The forms.editBox method has the following command format(s):

forms.editBox(*path*);

forms.editBox(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms editBox. For example, `//forms:calendar`. Must not be `null`. Since it is only possible to show one editBox in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the editBox object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms editBox.

Example

Performs an action on an Oracle Forms editBox object specified by its recorded ID and path.

```
forms.window(65, "//forms:window[(@name='GROUPS_DETAIL')]")
 .selectMenu("Help|About Oracle Applications...");
forms.editBox(68, "//forms:editBox").clickOk();
String text1 = forms.editBox(68, "//forms:editBox")
 .getText();
info("editBox text: " + text1);
```

forms.editorDialog

Identifies an Oracle Forms EditorDialog object by its path.

Format

The forms.editorDialog method has the following command format(s):

forms.editorDialog(*path*);

forms.editorDialog(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms editorDialog. For example, `//forms:editorDialog`. Must not be `null`. Since it is only possible to show one editorDialog in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the editorDialog object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms editorDialog.

Example

Performs an action on an Oracle Forms editorDialog object specified by its recorded ID and path.

```
forms.editorDialog(31, "//forms:editorDialog")
 .setText("4500217");
forms.editorDialog(32, "//forms:editorDialog")
 .clickOk();
```

forms.flexWindow

Identifies an Oracle Forms FlexWindow object by its path.

Format

The forms.flexWindow method has the following command format(s):

forms.flexWindow(*path*);

forms.flexWindow(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms flexWindow. For example, `//forms:flexWindow`. Must not be `null`. Since it is only possible to show one flexWindow in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the flexWindow object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms flexWindow.

Example

Performs an action on an Oracle Forms FlexWindow object specified by its recorded ID and path.

```
forms.textField(156, "//forms:textField[" +
 "(@name='LINES_ACCOUNTING_FLEXFIELD_0')]")
 .openDialog();
forms.flexWindow(159, "//forms:flexWindow")
 .setTextAndClickOk("Account Alias", "",
 "01-000-2225-0000-000");
```

forms.getAllObjects

Gets all forms objects.

Format

The forms.getAllObjects method has the following command format(s):

forms.getAllObjects( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

all objects in XML format.

Example

Gets all forms objects.

```
String objXML = forms.getAllObjects();
info("Forms objects: " + objXML);
```

forms.getStatusBarCount

Gets the bar count shown in the bottom of the forms client.

Format

The forms.getStatusBarCount method has the following command format(s):

forms.getStatusBarCount( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the bar count.

Example

Gets the status bar count.

```
for(int i=0;i<forms.getStatusBarCount();i++)
 for(int j=0;j>forms.getStatusBarItemCount(i);j++)
   {
   info("get status bar "+ i+" item count:"+forms.getStatusBarItemCount(i));
   info("get status bar item("+i+","+j+"):"+forms.getStatusBarItem(i, j));
   }
```

forms.getStatusBarErrorCode

Gets the error code shown in the status bar of the forms client.

Format

The forms.getStatusBarErrorCode method has the following command format(s):

forms.getStatusBarErrorCode( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the error code.

Example

Gets the status bar error code.

```
info("get Status Bar error code: " + forms.getStatusBarErrorCode());
```

forms.getStatusBarItem

Gets the item shown in the status bar of forms client.

Format

The forms.getStatusBarItem method has the following command format(s):

forms.getStatusBarItem(*barIndex*, *barItemIndex*);

Command Parameters

barIndex
the index of the status bar starting from 0.

barItemIndex
the item index of the status bar starting from 0.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the item.

Example

Gets the status bar item.

```
for(int i=0;i<forms.getStatusBarCount();i++)
 for(int j=0;j>forms.getStatusBarItemCount(i);j++)
   {
   info("get status bar "+ i+" item count:"+forms.getStatusBarItemCount(i));
   info("get status bar item("+i+","+j+"):"+forms.getStatusBarItem(i, j));
   }
```

forms.getStatusBarItemCount

Gets the item count shown in the status bar of forms client.

Format

The forms.getStatusBarItemCount method has the following command format(s):

forms.getStatusBarItemCount(*index*);

Command Parameters

index
the index of status bar starting from 0.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the item count.

Example

Gets the status bar item count.

```
for(int i=0;i<forms.getStatusBarCount();i++)
 for(int j=0;j>forms.getStatusBarItemCount(i);j++)
   {
   info("get status bar "+ i+" item count:"+forms.getStatusBarItemCount(i));
   info("get status bar item("+i+","+j+"):"+forms.getStatusBarItem(i, j));
   }
```

forms.getStatusBarMessage

Gets the message shown in the status bar of forms client.

Format

The forms.getStatusBarMessage method has the following command format(s):

forms.getStatusBarMessage( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the message.

Example

Gets the status bar message.

```
info("Status Bar message: " + forms.getStatusBarMessage());
```

forms.getStatusBarRecordInfo

Gets the record info shown in the status bar of the forms client.

Format

The forms.getStatusBarRecordInfo method has the following command format(s):

forms.getStatusBarRecordInfo( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the record info.

Example

Gets the status bar record info.

```
info("get Status Bar record info: " + forms.getStatusBarRecordInfo());
```

forms.helpDialog

Identifies an Oracle Forms helpDialog object by its path.

Format

The forms.helpDialog method has the following command format(s):

forms.helpDialog(*path*);

forms.helpDialog(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms helpDialog. For example, `//forms:helpDialog`. Must not be `null`. Since it is only possible to show one helpDialog in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the helpDialog object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms helpDialog.

Example

Performs an action on an Oracle Forms helpDialog object specified by its recorded ID and path.

```
forms.helpDialog(74, "//forms:helpDialog").clickOk();
```

forms.hGridApp

Identifies an Oracle Forms HGridApp object by its path.

Format

The forms.hGridApp method has the following command format(s):

forms.hGridApp(*path*);

forms.hGridApp(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms HGridApp. For example, `//hGridApp[@name='IEU_HGRID_IEU_HGRID_0']`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms HGridApp.

Example

Performs an action on an Oracle Forms HGridApp object specified by its recorded ID and path.

```
forms.hGridApp(43, "//forms:hGridApp[(@name='IEU_HGRID_IEU_HGRID_0')]")
 .selectNode("Delinquent Transactions|Active");
```

forms.imageItem

Identifies an Oracle Forms imageItem object by its path.

Format

The forms.imageItem method has the following command format(s):

forms.imageItem(*path*);

forms.imageItem(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms imageItem. For example, `//forms:imageItem[(@name='NAV_CONTROLS_EXPAND_ALL_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms imageItem.

Example

Performs an action on an Oracle Forms imageItem object specified by its recorded ID and path.

```
forms.imageItem(100, "//forms:imageItem[" +
 "(@name='NAV_CONTROLS_EXPAND_ALL_0')]")
 .click();
```

forms.infoBox

Identifies an Oracle Forms infoBox object by its path.

Format

The forms.infoBox method has the following command format(s):

forms.infoBox(*path*);

forms.infoBox(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms infoBox. For example, `//forms:infoBox`. Must not be `null`. Since it is only possible to show one infoBox in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the infoBox object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms infoBox.

Example

Performs an action on an Oracle Forms infoBox object specified by its recorded ID and path.

```
forms.infoBox(42, "//forms:infoBox").close();
```

forms.list

Identifies an Oracle Forms list object by its path.

Format

The forms.list method has the following command format(s):

forms.list(*path*);

forms.list(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms list. For example, `//forms:list[(@name='JTF_SORT_COLUMN1_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms list.

Example

Performs an action on an Oracle Forms list object specified by its recorded ID and path.

```
String accessName=forms.list(25, "//forms:list[" +
 "(@name='GROUPS_SYSTEM_LINKAGE_FUNCTION_0')]")
 .getAccessibleName();
int itemCount=forms.list(25, "//forms:list[" +
 "(@name='GROUPS_SYSTEM_LINKAGE_FUNCTION_0')]")
 .getItemCount();
info("accessible Name: " + accessName);
info("item count: " + itemCount);
//getItemValue
forms.list(25, "//forms:list[" +
 "(@name='GROUPS_SYSTEM_LINKAGE_FUNCTION_0')]")
 .selectItem("Usages");
String itemValue=forms.list(25, "//forms:list[" +
 "(@name='GROUPS_SYSTEM_LINKAGE_FUNCTION_0')]")
 .getItemValue();
info(itemValue);
//isItemInList
boolean isItem=forms.list(25, "//forms:list[" +
 "(@name='GROUPS_SYSTEM_LINKAGE_FUNCTION_0')]")
 .isItemInList("Usages");
if(!isItem)warn("Item not in list");
```

```
//isItemSelected
boolean isSelected=forms.list(25, "//forms:list[" +
  "(@name='GROUPS_SYSTEM_LINKAGE_FUNCTION_0')]")
  .isItemSelected("Usages");
if(!isSelected)warn("Item not selected");
```

forms.listOfValues

Identifies an Oracle Forms listOfValues object by its path.

Format

The forms.listOfValues method has the following command format(s):

forms.listOfValues(*path*);

forms.listOfValues(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms listOfValues. For example, `//forms:listOfValues`. Must not be `null`. Since it is only possible to show one listOfValues in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the listOfValues object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms listOfValues.

Example

Performs an action on an Oracle Forms listOfValues object specified by its recorded ID and path.

```
//get item count
int itemcount=forms.listOfValues(20, "//forms:listOfValues")
 .getItemCount();
info("Item count: " + itemcount);
//get text
String text = "";
for(int i=0;i<itemcount;i++)
   {
   text=forms.listOfValues(21, "//forms:listOfValues")
     .getText(i);
   info("Item " + i + "text: " + text);
   }
```

forms.logonDialog

Identifies an Oracle Forms logonDialog object by its path.

Format

The forms.logonDialog method has the following command format(s):

forms.logonDialog(*path*);

forms.logonDialog(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms logonDialog. For example, `//logonDialog`. Must not be `null`. Since it is only possible to show one logonDialog in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the logonDialog object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms logonDialog.

Example

Performs an action on an Oracle Forms logonDialog object specified by its recorded ID and path.

```
forms.logonDialog(10, "//logonDialog")
 .activate(true);
forms.logonDialog(10, "//logonDialog")
 .clickConnect("name", "password", "database");
```

forms.otsHGrid

Identifies an Oracle Forms otsHGrid object by its path.

Format

The forms.otsHGrid method has the following command format(s):

forms.otsHGrid(*path*);

forms.otsHGrid(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms otsHGrid. For example, `//forms:otsHGrid[(@name='ASTEVENR_CONF_HGRID_ITEMS_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms otsHGrid.

Example

Performs an action on an Oracle Forms otsHGrid object specified by its recorded ID and path.

```
if(forms.otsHGrid(10, "//forms:otsHGrid[(@name='ASTEVENR_CONF_HGRID_ITEMS_0')]")
 .exists())
 info("otsHGrid passed!");
else warn("otsHGrid failed!");
```

forms.radioButton

Identifies an Oracle Forms radioButton object by its path.

Format

The forms.radioButton method has the following command format(s):

forms.radioButton(*path*);

forms.radioButton(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms radioButton. For example, `//forms:radioButton[(@name='EXPENDITURES_TRANSACTION_STATUS_REJECTED_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms radioButton.

Example

Performs an action on an Oracle Forms radioButton object specified by its recorded ID and path.

```
String radioLabel=forms.radioButton(24,"//forms:radioButton[" +
 "(@name='JOBS_QF_WHICH_JOBS_MY_UNCOMPLETED_0')]")
 .getLabel();
boolean isSelected=forms.radioButton(24, "//forms:radioButton[" +
 "(@name='JOBS_QF_WHICH_JOBS_MY_UNCOMPLETED_0')]")
 .isSelected();
info("radio label:"+radioLabel+" selected:"+isSelected);
if(isSelected)warn("select error!");
forms.radioButton(24, "//forms:radioButton[" +
 "(@name='JOBS_QF_WHICH_JOBS_MY_UNCOMPLETED_0')]")
 .select();
```

forms.responseBox

Identifies an Oracle Forms responseBox object by its path.

Format

The forms.responseBox method has the following command format(s):

forms.responseBox(*path*);

forms.responseBox(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms responseBox. For example, `//forms:responseBox`. Must not be `null`. Since it is only possible to show one responseBox in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the responseBox object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms responseBox.

Example

Performs an action on an Oracle Forms responseBox object specified by its recorded ID and path.

```
forms.responseBox(154, "//forms:responseBox")
 .setText("Old Password", "abc");
forms.responseBox(155, "//forms:responseBox")
 .setText("New Password", "xyz");
```

forms.schedulingDataClient

Identifies an Oracle Forms schedulingDataClient object by its path.

Format

The forms.schedulingDataClient method has the following command format(s):

forms.schedulingDataClient(*path*);

forms.schedulingDataClient(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms schedulingDataClient. For example, `//forms:schedulingDataClient[(@name='GANTT_GANTTBEAN_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms schedulingDataClient.

Example

Performs an action on an Oracle Forms schedulingDataClient object specified by its recorded ID and path.

```
forms.schedulingDataClient(32, "//forms:schedulingDataClient[" +
 "(@name='GANTT_GANTTBEAN_0')]")
 .clickNext();
forms.schedulingDataClient(33, "//forms:schedulingDataClient[" +
 "(@name='GANTT_GANTTBEAN_0')]")
 .clickPrevious();
forms.schedulingDataClient(34, "//forms:schedulingDataClient[" +
 "(@name='GANTT_GANTTBEAN_0')]")
 .clickAssigneeCell(3, 1);
```

forms.spreadTable

Identifies an Oracle Forms spreadTable object by its path.

Format

The forms.spreadTable method has the following command format(s):

forms.spreadTable(*path*);

forms.spreadTable(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms spreadTable. For example, `//forms:spreadTable[(@name='BLK_INBOX_INBOX_GRID_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms spreadTable.

Example

Performs an action on an Oracle Forms spreadTable object specified by its recorded ID and path.

```
forms.spreadTable(66, "//forms:spreadTable[" +
 "(@name='SR_TASK_GRID_GRID_ITEM_0')]")
 .selectRows(1, 1);
forms.spreadTable(66, "//forms:spreadTable[" +
 "(@name='SR_TASK_GRID_GRID_ITEM_0')]")
 .focusCell(1, 1);
String cellValue=forms.spreadTable(66, "//forms:spreadTable[" +
 "(@name='SR_TASK_GRID_GRID_ITEM_0')]")
 .getCell(1, 2);
forms.spreadTable(66, "//forms:spreadTable[" +
 "(@name='SR_TASK_GRID_GRID_ITEM_0')]")
 .storeCell("testVar", 1, 2);
if(!cellValue.equals("{{testVar}}"))warn("get cell value error");
//test column count and row count
forms.spreadTable(66, "//forms:spreadTable[" +
 "(@name='SR_TASK_GRID_GRID_ITEM_0')]")
 .verifyAttributes("objectTest1",
   forms.attributes(
     forms.attribute("columnCount", "42",
       TestOperator.StringExact),
   forms.attribute("rowCount", "2",
```

```
      TestOperator.StringExact)));
forms.spreadTable(66, "//forms:spreadTable[" +
 "(@name='SR_TASK_GRID_GRID_ITEM_0')]")
 .focusRow(1);
```

forms.statusBar

Identifies an Oracle Forms StatusBar object by its recorded ID and path.

Format

The forms.statusBar method has the following command format(s):

forms.statusBar(*path*);

forms.statusBar(*recid*, *path*);

Command Parameters

path
a String specifying the path to identify the forms statusBar. For example, `//forms:statusBar`. Must not be `null`. Since it is only possible to show one statusBar in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the Status Bar object.

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms StatusBar.

Example

Performs an action on an Oracle Forms statusBar object specified by its recorded ID and path.

```
forms.statusBar(100, "//forms:statusBar")
 .assertText("FormsFT AutoValidation: Verify StatusBar text value",
 "FRM-40400: Transaction complete: 1 records applied and saved.",
 MatchOption.Exact);
```

forms.tab

Identifies an Oracle Forms tab object by its path.

Format

The forms.tab method has the following command format(s):

forms.tab(*path*);

forms.tab(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms tab. For example, `//forms:tab[(@name='ATTRIBUTE_GROUPS_REGIONS')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms tab.

Example

Performs an action on an Oracle Forms tab object specified by its recorded ID and path.

```
forms.tab(485, "//forms:tab[" +
 "(@name='FLOWDOWN_TAB')]")
 .select("Terms & Conditions");
String selectedStr=forms.tab(485, "//forms:tab[" +
 "(@name='FLOWDOWN_TAB')]")
 .getSelectedTabLabel();
if(!selectedStr.equals("Terms & Conditions"))
 warn("getSelectedTabLabel() error");
boolean isItem=forms.tab(485, "//forms:tab[" +
 "(@name='FLOWDOWN_TAB')]")
 .isItemInList("Terms & Conditions");
if(!isItem)
 warn("isItemInList(string) error");
boolean isSelected=forms.tab(485, "//forms:tab[" +
 "(@name='FLOWDOWN_TAB')]")
 .isItemSelected("Terms & Conditions");
if(!isSelected)
 warn("isItemSelected(string) error");
```

forms.tableBox

Identifies an Oracle Forms tableBox object by its path.

Format

The forms.tableBox method has the following command format(s):

forms.tableBox(*path*);

forms.tableBox(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms tableBox. For example, `//forms:tableBox`. Must not be `null`. Since it is only possible to show one tableBox in Oracle Forms at runtime, it is not necessary to specify additional conditions in the path to locate the tableBox object.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms tableBox.

Example

Performs an action on an Oracle Forms tableBox object specified by its recorded ID and path.

```
forms.tableBox(100, "//forms:tableBox")
 .check(1, 1);
forms.tableBox(100, "//forms:tableBox")
 .clickOk();
```

forms.textField

Identifies an Oracle Forms textField object by its path.

Format

The forms.textField method has the following command format(s):

forms.textField(*path*);

forms.textField(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms textField. For example, `//forms:textField[(@name='NAVIGATOR_TYPE_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms TextField.

Example

Performs an action on an Oracle Forms textField object specified by its recorded ID and path.

```
forms.textField(248, "//forms:textField[" +
 "(@name='STUB_APPLICATION_NAME_0')]")
 .setFocus();
forms.textField(249,"//forms:textField[" +
 "(@name='STUB_APPLICATION_NAME_0')]")
 .setText("abc");
forms.textField("//forms:textField[" +
 "(@name='STUB_APPLICATION_NAME_0')]")
 .verifyAttributes("test1",forms.attributes(
   forms.attribute("enabled", "true",
     TestOperator.StringExact),
   forms.attribute("text", "Sales",
     TestOperator.StringExact),
   forms.attribute("visible", "true",
     TestOperator.StringExact),
   forms.attribute("name", "STUB_APPLICATION_NAME_0",
     TestOperator.StringExact),
   forms.attribute("showing", "true",
     TestOperator.StringExact),
   forms.attribute("prompt", "Application",
     TestOperator.StringExact),
```

```
forms.attribute("editable", "true",
  TestOperator.StringExact)));
```

forms.tree

Identifies an Oracle Forms tree object by its path.

Format

The forms.tree method has the following command format(s):

forms.tree(*path*);

forms.tree(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms tree. For example, `//forms:tree[@accessibleName='Navigator']`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms tree.

Example

Performs an action on an Oracle Forms tree object specified by its recorded ID and path.

```
forms.tree(24, "//forms:tree[(@name='APPTREE_NAV_TREE_NAVIGATOR_0')]")
 .selectNode("Personal Shortcuts");
forms.list(25, "//forms:list[(@name='APPTREE_VIEW_BY_VIEW_BY_0')]")
 .setFocus();
forms.tree(26, "//forms:tree[(@name='APPTREE_NAV_TREE_NAVIGATOR_0')]")
 .expandNode("People By Name");
forms.tree(27, "//forms:tree[(@name='APPTREE_NAV_TREE_NAVIGATOR_0')]")
 .selectNode("People By Name|A");
forms.tree(28, "//forms:tree[(@name='APPTREE_NAV_TREE_NAVIGATOR_0')]")
 .expandNode("People By Name|B");
forms.tree(29, "//forms:tree[(@name='APPTREE_NAV_TREE_NAVIGATOR_0')]")
 .collapseNode("People By Name|B");
```

forms.treeList

Identifies an Oracle Forms treeList object by its path.

Format

The forms.treeList method has the following command format(s):

forms.treeList(*path*);

forms.treeList(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms treeList. For example, `//forms:treeList[(@name='NAVIGATOR_LIST_0')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms treeList.

Example

Performs an action on an Oracle Forms treeList object specified by its recorded ID and path.

```
forms.treeList(79, "//forms:treeList[(@name='NAVIGATOR_LIST_0')]")
 .focusItem("Contract Groups");
forms.treeList(80, "//forms:treeList[(@name='NAVIGATOR_LIST_0')]")
 .selectItem("Contract Groups");
```

forms.window

Identifies an Oracle Forms window object by its path.

Format

The forms.window method has the following command format(s):

forms.window(*path*);

forms.window(*recid*, *path*);

Command Parameters

recid
Optional the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

path
a String specifying the path to identify the forms window. For example, `//forms:window[(@name='NAVIGATOR')]`. Must not be `null`.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the forms window.

Example

Performs an action on an Oracle Forms window object specified by its recorded ID and path.

```
forms.window(24, "//forms:window[(@name='STUB')]")
 .activate(true);
forms.window(25, "//forms:window[(@name='STUB')]")
 .clickToolBarButton("Save");
forms.statusBar(26, "//forms:statusBar").assertText(
 "FormsFT AutoValidation: Verify StatusBar text value",
 "FRM-40401: No changes to save.", MatchOption.Exact);
forms.window(27, "//forms:window[(@name='STUB')]")
 .selectMenu("File|Print...");
boolean isMenu=forms.window(27, "//forms:window[" +
 "(@name='STUB')]")
 .isMenuEnabled("File|Print...");
if(!isMenu)warn("isMenuEnabled(string) error");
forms.window(28, "//forms:window[(@name='STUB')]")
 .close();
```

14

Siebel Functional Module

This chapter provides a complete listing and reference for the methods in the OpenScript SiebelFTService Class of Siebel Functional Module Application Programming Interface (API).

14.1 SiebelFTService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript SiebelFTService API.

14.1.1 Alphabetical Command Listing

The following table lists the SiebelFTService API methods in alphabetical order.

Table 14–1 List of SiebelFTService Methods

Method	Description
siebelFT.applet	Creates a Siebel Applet element.
siebelFT.application	Creates a Siebel Application element.
siebelFT.attribute	Construct property data of a single attribute.
siebelFT.attributes	Utility function to return a list of property tests.
siebelFT.button	Creates a Siebel Button element.
siebelFT.calculator	Creates a Siebel Calculator element.
siebelFT.calendar	Creates a Siebel Calendar element.
siebelFT.cell	Construct a test data of a table cell.
siebelFT.cells	Utility function to return a list of cell tests.
siebelFT.currency	Creates a Siebel Currency element.
siebelFT.element	Creates a generic Siebel element.
siebelFT.exists	Checks for the existence of a Siebel control.
siebelFT.list	Creates a Siebel List element.
siebelFT.menu	Creates a Siebel Menu element.
siebelFT.pageTabs	Creates a Siebel PageTabs element.
siebelFT.pdq	Creates a Siebel PDQ element.
siebelFT.richText	Creates a Siebel RichText element.
siebelFT.screen	Creates a Siebel Screen element.

Table 14–1 (Cont.) List of SiebelFTService Methods

Method	Description
siebelFT.screenViews	Creates a Siebel ScreenViews element.
siebelFT.text	Creates a Siebel Text element.
siebelFT.textArea	Creates a Siebel TextArea element.
siebelFT.threadbar	Creates a Siebel Threadbar element.
siebelFT.toolbar	Creates a Siebel Toolbar element.
siebelFT.tree	Creates a Siebel Tree element.

The following sections provide detailed reference information for each method and enum in the SiebelFTService Class of Siebel Functional Module Application Programming Interface.

siebelFT.applet

Creates a Siebel Applet element.

Format

The siebelFT.applet method has the following command format(s):

siebelFT.applet(*path*);

siebelFT.applet(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Applet.

Example

Performs an action on a Siebel Applet element.

```
String SiebApplet="/siebelft:cas[@ClassName='SiebApplication' " +
   "and @RepositoryName='Siebel Universal Agent']" +
  "/siebelft:cas[@ClassName='SiebScreen' " +
   "and @RepositoryName='Opportunities Screen']" +
  "/siebelft:cas[@ClassName='SiebView' " +
   "and @RepositoryName='Opportunity List View']" +
  "/siebelft:cas[@ClassName='SiebApplet' " +
   "and @RepositoryName='Opportunity Form Applet - Child']";
try{
  String result=siebelFT.applet(10, SiebApplet)
   .childPathFromRepositoryName("SiebTextArea", "Description");
  String expected="/siebelft:cas[@ClassName='SiebApplication' " +
    "and @RepositoryName='Siebel Universal Agent']" +
   "/siebelft:cas[@ClassName='SiebScreen' " +
    "and @RepositoryName='Opportunities Screen']" +
   "/siebelft:cas[@ClassName='SiebView' " +
    "and @RepositoryName='Opportunity List View']" +
   "/siebelft:cas[@ClassName='SiebApplet' " +
    "and @RepositoryName='Opportunity Form Applet - Child']" +
   "/siebelft:cas[@ClassName='SiebTextArea' " +
    "and @RepositoryName='Description']";
  if (!result.equals(expected)) {
   warn("childPathFromRepositoryName returned  " + result);
  }
  }catch(Exception exp) {}
```

siebelFT.application

Creates a Siebel Application element.

Format

The siebelFT.application method has the following command format(s):

siebelFT.application(*path*);

siebelFT.application(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Application.

Example

Performs an action on a Siebel Application element.

```
String SiebApplication="/siebelft:cas[@ClassName='SiebApplication' " +
 "and @RepositoryName='Siebel Universal Agent']";
try{
 String expected="/siebelft:cas[@ClassName='SiebApplication' " +
   "and @RepositoryName='Siebel Universal Agent']" +
  "/siebelft:cas[@ClassName='SiebScreen' " +
   "and @RepositoryName='Web Call Center Home Screen']";
 String result=siebelFT.application(10, SiebApplication)
  .childPathFromRepositoryName("SiebScreen", "Web Call Center Home Screen");
 if (!result.equals(expected)) {
  warn("childPathFromRepositoryName returned " + result);

 }
 }catch(Exception exp) {}
```

siebelFT.attribute

Construct property data of a single attribute.

Format

The siebelFT.attribute method has the following command format(s):

siebelFT.attribute(*property*, *expectedValue*, *operator*);

Command Parameters

property
A String specifying the property to test.

expectedValue
a String specifying ther expected value.

operator
a TestOperator that defines the type and the match approach of the data.

Returns

PropertyTest object.

Example

Specifies individual Siebel element attributes.

```
String path="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebMenu' " +
  "and @RepositoryName='SiebMenu']";
siebelFT.menu(path).assertAttributes("attributes",
siebelFT.attributes(siebelFT.attribute("ClassName",
  "SiebMenu", TestOperator.StringExact),
 siebelFT..attribute("RepositoryName", "SiebMenu",
  TestOperator.StringExact),
 siebelFT..attribute("UIName", "Menu",
  TestOperator.StringExact),
 siebelFT..attribute("Count", "60",
  TestOperator.StringExact)));
siebelFT.menu(path).assertAttribute("attribute", "UIName", "Menu",
 TestOperator.StringExact);
try{
 boolean result=siebelFT.element(path).exists();
 if (!result) {
 warn("exists returned  " + result);
 }
 }catch(Exception exp) {}
```

siebelFT.attributes

Utility function to return a list of property tests.

Format

The siebelFT.attributes method has the following command format(s):

siebelFT.attributes(*tests*);

Command Parameters

tests
- array of property tests.

Returns

an array of property tests.

Example

Specifies a set of Siebel element attributes.

```
String path="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebMenu' " +
  "and @RepositoryName='SiebMenu']";
siebelFT.menu(path).assertAttributes("attributes",
siebelFT.attributes(siebelFT.attribute("ClassName",
  "SiebMenu", TestOperator.StringExact),
 siebelFT.attribute("RepositoryName", "SiebMenu",
  TestOperator.StringExact),
 siebelFT.attribute("UIName", "Menu",
  TestOperator.StringExact),
 siebelFT.attribute("Count", "60",
  TestOperator.StringExact)));
siebelFT.menu(path).assertAttribute("attribute", "UIName", "Menu",
 TestOperator.StringExact);
try{
 boolean result=siebelFT.element(path).exists();
 if (!result) {
 warn("exists returned  " + result);
 }
 }catch(Exception exp) {}
```

siebelFT.button

Creates a Siebel Button element.

Format

The siebelFT.button method has the following command format(s):

siebelFT.button(*path*);

siebelFT.button(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Button.

Example

Performs an action on a Siebel Button element.

```
String SiebButton="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Accounts Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Account List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Account Entry Applet']" +
 "/siebelft:cas[@ClassName='SiebButton' " +
  "and @RepositoryName='NewRecord']";
try{
 Boolean result=siebelFT.button(10, SiebButton).isEnabled();
 if (!result) {
  warn("isEnabled() returned " + result);
 }
 }catch(Exception exp) {}

siebelFT.button(15, "/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Accounts Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Account List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Account Entry Applet']" +
 "/siebelft:cas[@ClassName='SiebButton' " +
  "and @RepositoryName='DeleteRecord']")
 .click();
```

siebelFT.calculator

Creates a Siebel Calculator element.

Format

The siebelFT.calculator method has the following command format(s):

siebelFT.calculator(*path*);

siebelFT.calculator(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Calculator.

Example

Performs an action on a Siebel Calculator element.

```
String SiebCurrency="/siebelft:cas[@ClassName='SiebApplication' " +
   "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Opportunity List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Opportunity Form Applet - Child']" +
 "/siebelft:cas[@ClassName='SiebCurrency' " +
  "and @RepositoryName='Revenue']";

String SiebCalculator="/siebelft:cas[@ClassName='SiebApplication' " +
   "and @RepositoryName='Siebel Universal Agent']" +
  "/siebelft:cas[@ClassName='SiebScreen' " +
   "and @RepositoryName='Opportunities Screen']" +
  "/siebelft:cas[@ClassName='SiebView' " +
   "and @RepositoryName='Opportunity List View']" +
  "/siebelft:cas[@ClassName='SiebApplet' " +
   "and @RepositoryName='Opportunity Form Applet - Child']" +
  "/siebelft:cas[@ClassName='SiebCurrency' " +
   "and @RepositoryName='Revenue']" +
  "/siebelft:cas[@ClassName='SiebCalculator' " +
   "and @RepositoryName='Amount']";
siebelFT.currency(SiebCurrency).openPopup();
siebelFT.calculator(10, SiebCalculator).openPopup();
siebelFT.calculator(12, SiebCalculator).clickKey("9");
siebelFT.calculator(14, SiebCalculator).clickKeys("99");
siebelFT.calculator(16, SiebCalculator).processKey("EnterKey");
siebelFT.calculator(18, SiebCalculator).processKey("EnterKey");
try{
```

```
 String result=siebelFT.calculator(20, SiebCalculator).getText();
 if (!result.equals("$999.00")) {
  warn("getText returned " + result);

}
}catch(Exception exp) {}
try{
 Boolean result=siebelFT.calculator(22, SiebCalculator).isEnabled();
 if (!result) {
  warn("isEnabled returned " + result);
 }
 }catch(Exception exp) {}

try{
 Boolean result=siebelFT.calculator(24, SiebCalculator).isOpen();
 if (result) {
  warn("isOpen returned " + result);
 }
 }catch(Exception exp) {}

siebelFT.currency(SiebCurrency).closePopup();
```

siebelFT.calendar

Creates a Siebel Calendar element.

Format

The siebelFT.calendar method has the following command format(s):

siebelFT.calendar(*path*);

siebelFT.calendar(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Calendar.

Example

Performs an action on a Siebel Calendar element.

```
String SiebCalendar="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Opportunity List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Opportunity Form Applet - Child']" +
 "/siebelft:cas[@ClassName='SiebCalendar' " +
  "and @RepositoryName='CloseDate']";
siebelFT.calendar(SiebCalendar).openPopup();
try{
 String result=siebelFT.calendar(SiebCalendar)
  .getCalendarType();
 if (!result.equals("Date")) {
  warn("getCalendarType returned " + result);
 }
 }catch(Exception exp) {}

try{
 String result=siebelFT.calendar(10, SiebCalendar)
  .getDay();
 if (!result.equals("27")) {
  info("getDay returned " + result);
 }
 }catch(Exception exp) {}

try{
 String result=siebelFT.calendar(12, SiebCalendar)
  .getMonth();
```

```
 if (!result.equals("8")) {
  info("getMonth returned " + result);
 }
 }catch(Exception exp) {}

try{
 String result=siebelFT.calendar(14, SiebCalendar)
  .getYear();
 if (!result.equals("2010")) {
  info("getYear returned " + result);
 }
 }catch(Exception exp) {}

try{
 String result="none"+siebelFT.calendar(16, SiebCalendar)
  .getTime();
 if (!result.equals("none")) {
  warn("getTime returned " + result);
 }
 }catch(Exception exp) {}
```

siebelFT.cell

Construct a test data of a table cell.

Format

The siebelFT.cell method has the following command format(s):

siebelFT.cell(*row*, *column*, *expectedValue*, *operator*);

Command Parameters

row
the row value starting from 1.

column
the column value starting from 1.

expectedValue
a String specifying the expected value.

operator
a TestOperator that defines the type and the match approach of the data.

Returns

CellTest.

Example

Specifies individual Siebel cell elements.

```
String SiebThreadbar1="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Activities Screen']" +
 "/siebelft:cas[@ClassName='SiebThreadbar' " +
  "and @RepositoryName='SiebThreadbar']";
siebelFT.threadbar(SiebThreadbar).assertCells("assert",
 siebelFT.cells(
  siebelFT.cell(1, 1,"Opportunity Detail - Activities View0",
   TestOperator.StringExact)));
siebelFT.threadbar(SiebThreadbar).verifyCells("verify",
 siebelFT.cells(
  siebelFT.cell(1, 1,"Opportunity Detail - Activities View0",
   TestOperator.StringExact)));
try{
 String result=siebelFT.threadbar(SiebThreadbar)
  .getActiveThreadItem();
 if (!result.equals("Opportunity Detail - Activities View0")) {
  warn("getActiveThreadItem returned  " + result);
 }
 }catch(Exception exp) {}
```

siebelFT.cells

Utility function to return a list of cell tests.

Format

The siebelFT.cells method has the following command format(s):

siebelFT.cells(*tests*);

Command Parameters

tests
- array of property tests.

Returns

array of cell tests.

Example

Specifies set of Siebel cell elements.

```
String SiebThreadbar1="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Activities Screen']" +
 "/siebelft:cas[@ClassName='SiebThreadbar' " +
  "and @RepositoryName='SiebThreadbar']";
siebelFT.threadbar(SiebThreadbar).assertCells("assert",
 siebelFT.cells(
  siebelFT.cell(1, 1,"Opportunity Detail - Activities View0",
   TestOperator.StringExact)));
siebelFT.threadbar(SiebThreadbar).verifyCells("verify",
 siebelFT.cells(
  siebelFT.cell(1, 1,"Opportunity Detail - Activities View0",
   TestOperator.StringExact)));
try{
 String result=siebelFT.threadbar(SiebThreadbar)
  .getActiveThreadItem();
 if (!result.equals("Opportunity Detail - Activities View0")) {
  warn("getActiveThreadItem returned  " + result);
 }
 }catch(Exception exp) {}
```

siebelFT.currency

Creates a Siebel Currency element.

Format

The siebelFT.currency method has the following command format(s):

siebelFT.currency(*path*);

siebelFT.currency(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Currency.

Example

Performs an action on a Siebel Currency element.

```
String SiebCurrency="/siebelft:cas[@ClassName='SiebApplication' " +
   "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Opportunity List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Opportunity Form Applet - Child']" +
 "/siebelft:cas[@ClassName='SiebCurrency' " +
  "and @RepositoryName='Revenue']";

try{
 String result=siebelFT.currency(5, SiebCurrency)
  .getAmount();
 if (!result.equals("$.00")) {
  info("getAmount returned  " + result);
 }
 }catch(Exception exp) {}

try{
 String result=siebelFT.currency(7, SiebCurrency)
  .getText();
 if (!result.equals("$0.00")) {
  info("getText() returned  " + result);
 }
 }catch(Exception exp) {}

try{
 Boolean result=siebelFT.currency(9, SiebCurrency).isEnabled();
 if (!result) {
```

```
  warn("isEnabled returned " + result);
 }
 }catch(Exception exp) {}

try{
 Boolean result=siebelFT.currency(11, SiebCurrency).isOpen();
 if (result) {
  warn("isOpen returned " + result);
 }
 }catch(Exception exp) {}

siebelFT.currency(14, SiebCurrency).setText("999");
siebelFT.currency(16, SiebCurrency).processKey("EnterKey");
siebelFT.currency(18, SiebCurrency).openPopup();
siebelFT.currency(20, SiebCurrency).cancelPopup();
```

siebelFT.element

Creates a generic Siebel element.

Format

The siebelFT.element method has the following command format(s):

siebelFT.element(*path*);

siebelFT.element(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Element.

Example

CPerforms an action on a generic Siebel element.

```
web.window(1376,"/web:window[@index='0' " +
   "or @title='about:blank']")
 .navigate("http://example.com/callcenter_enu/start.swe?SWECmd=AutoOn");
web.textBox(1378,"/web:window[@index='0' " +
   "or @title='Siebel Call Center']" +
  "/web:document[@index='0']" +
  "/web:form[@name='SWEEntryForm' or @index='0']" +
  "/web:input_text[@rn='_SweUserName' " +
   "and @ot='SiebWebText' and @un='_SweUserName']")
  .setText("sadmin");
{
  think(0.156);
}
web.textBox(1379,"/web:window[@index='0' " +
   "or @title='Siebel Call Center']" +
  "/web:document[@index='0']" +
  "/web:form[@name='SWEEntryForm' or @index='0']" +
  "/web:input_text[@rn='_SweUserName' " +
  "and @ot='SiebWebText' and @un='_SweUserName']")
  .pressTab();
{
  think(0.031);
}
web.textBox(1380,"/web:window[@index='0' " +
   "or @title='Siebel Call Center']" +
  "/web:document[@index='0']" +
  "/web:form[@name='SWEEntryForm' or @index='0']" +
  "/web:input_password[@rn='_SwePassword' " +
  "and @ot='SiebWebPassword' and @un='_SwePassword']")
  .setPassword(decrypt("OvCQi9wcp7lT2jehewJtOQ=="));
```

```
{
  think(1.232);
}
web.image(1381,"/web:window[@index='0' " +
   "or @title='Siebel Call Center']" +
  "/web:document[@index='0']" +
  "/web:img[@alt='Login' or @index='3' " +
   "or @src='http://example.com/callcenter_enu/images/login77_d.gif']")
  .click();

String path="/siebelft:cas[@ClassName='SiebApplication' " +
   "and @RepositoryName='Siebel Universal Agent']" +
  "/siebelft:cas[@ClassName='SiebMenu' " +
   "and @RepositoryName='SiebMenu']";
siebelFT.menu(path).assertAttributes("attributes",siebelFT.attributes(
 siebelFT.attribute("ClassName","SiebMenu",
  TestOperator.StringExact),
 siebelFT.attribute("RepositoryName", "SiebMenu",
  TestOperator.StringExact),
 siebelFT.attribute("UIName", "Menu",
  TestOperator.StringExact),
 siebelFT.attribute("Count", "60",
  TestOperator.StringExact)));

siebelFT.menu(path).assertAttribute("attribute", "UIName", "Menu",
 TestOperator.StringExact);

try{
  boolean result=siebelFT.element(10, path).exists();
  if (!result) {
   warn("exists returned  " + result);
  }
}catch(Exception exp) {}

try{
  String result=siebelFT.element(12, path).getAttribute("RepositoryName");
  if (!result.equals("SiebMenu")) {
   warn("getClassName returned  " + result);
  }
}catch(Exception exp) {}

try{
  String result=siebelFT.element(14, path).getRepositoryName();
  if (!result.equals("SiebMenu")) {
   warn("getRepositoryName returned  " + result);
  }
}catch(Exception exp) {}

try{
  siebelFT.element(16, path).storeAttribute("var", "Count");
  if (!eval("{{var}}").equals("60")) {
   warn("Count returned  " + eval("{{var}}"));
  }
}catch(Exception exp) {}

siebelFT.element(18, path).waitForAttribute("Count", "60", 10);
```

siebelFT.exists

Checks for the existence of a Siebel control.

Format

The siebelFT.exists method has the following command format(s):

siebelFT.exists(*path*);

Command Parameters

path
pathname of the Siebel control.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

`true` if the control exists, otherwise `false`.

Example

Checks if the specified Siebel element exists.

```
try{
 if(siebelFT.exists("/siebelft:cas[@ClassName='SiebApplication' " +
   "and @RepositoryName='Siebel Universal Agent']" +
  "/siebelft:cas[@ClassName='SiebScreen' " +
   "and @RepositoryName='Activities Screen']" +
  "/siebelft:cas[@ClassName='SiebThreadbar' " +
   "and @RepositoryName='SiebThreadbar']")) {
  info("element exists");
 }
 }catch(Exception exp) {}
```

siebelFT.list

Creates a Siebel List element.

Format

The siebelFT.list method has the following command format(s):

siebelFT.list(*path*);

siebelFT.list(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new List.

Example

Performs an action on a Siebel List element.

```
String pdq="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebPDQ' " +
  "and @RepositoryName='SiebPDQ']";
siebelFT.pdq(pdq).select("** All Opportunities");
think(11.13);
siebelFT.list(72,"/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Opportunity List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Opportunity List Applet']" +
 "/siebelft:cas[@ClassName='SiebList' " +
  "and @RepositoryName='SiebList']")
 .activateRow(1);
{
 think(0.031);
}
siebelFT.list(73,"/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Opportunity List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Opportunity List Applet']" +
```

```
"/siebelft:cas[@ClassName='SiebList' " +
 "and @RepositoryName='SiebList']")
.drillDownColumn("Name", 1);
```

siebelFT.menu

Creates a Siebel Menu element.

Format

The siebelFT.menu method has the following command format(s):

siebelFT.menu(*path*);

siebelFT.menu(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Menu.

Example

Performs an action on a Siebel Menu element.

```
siebelFT.menu(256,"/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebMenu' " +
  "and @RepositoryName='SiebMenu']")
 .select("File\\\\File - Logout");
```

siebelFT.pageTabs

Creates a Siebel PageTabs element.

Format

The siebelFT.pageTabs method has the following command format(s):

siebelFT.pageTabs(*path*);

siebelFT.pageTabs(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new PageTabs.

Example

Performs an action on a Siebel PageTabs element.

```
siebelFT.pageTabs(41,"/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebPageTabs' " +
  "and @RepositoryName='SiebPageTabs']")
 .gotoScreen("Opportunities Screen");
```

siebelFT.pdq

Creates a Siebel PDQ element.

Format

The siebelFT.pdq method has the following command format(s):

siebelFT.pdq(*path*);

siebelFT.pdq(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new PDQ.

Example

Performs an action on a Siebel PDQ element.

```
String pdq="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebPDQ' " +
  "and @RepositoryName='SiebPDQ']";
siebelFT.pdq(10, pdq).select("** All Opportunities");
think(11.13);
siebelFT.list(72,"/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Opportunity List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Opportunity List Applet']" +
 "/siebelft:cas[@ClassName='SiebList' " +
  "and @RepositoryName='SiebList']")
 .activateRow(1);
{
 think(0.031);
}
siebelFT.list(73,"/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Opportunity List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Opportunity List Applet']" +
```

```
"/siebelft:cas[@ClassName='SiebList' " +
 "and @RepositoryName='SiebList']")
.drillDownColumn("Name", 1);
```

siebelFT.richText

Creates a Siebel RichText element.

Format

The siebelFT.richText method has the following command format(s):

siebelFT.richText(*path*);

siebelFT.richText(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new RichText.

Example

Performs an action on a Siebel RichText element.

```
String SiebRichText="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Projects Screen (ePS)']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Project Private Note View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Project Private Note Form Applet']" +
 "/siebelft:cas[@ClassName='SiebRichText' " +
  "and @RepositoryName='Note']";
siebelFT.button(249,"/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Projects Screen (ePS)']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Project Private Note View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Project Private Note Form Applet']" +
 "/siebelft:cas[@ClassName='SiebButton' " +
  "and @RepositoryName='NewRecord']")
 .click();
{
 think(6.812);
}
siebelFT.richText(5, SiebRichText).setText("zopa");
try{
 String result=siebelFT.richText(7, SiebRichText).getText();
 if (!result.equals("zopa")) {
  warn("getText returned  " + result);
  }
```

```
 }catch(Exception exp) {}

try{
 Boolean result=siebelFT.richText(9, SiebRichText).isEnabled();
 if (!result) {
  warn("isEnabled returned  " + result);
  }
 }catch(Exception exp) {}
try{
 Boolean result=siebelFT.richText(11, SiebRichText).isRequired();
 if (result) {
  warn("isRequired returned  " + result);
  }
 }catch(Exception exp) {}
```

siebelFT.screen

Creates a Siebel Screen element.

Format

The siebelFT.screen method has the following command format(s):

siebelFT.screen(*path*);

siebelFT.screen(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Screen.

Example

Performs an action on a Siebel Screen element.

```
String SiebScreen="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Accounts Screen']";

try{
 String result=siebelFT.screen(3, SiebScreen)
  .childPathFromRepositoryName("SiebPDQ", "SiebPDQ");
 String expected="/siebelft:cas[@ClassName='SiebApplication' " +
   "and @RepositoryName='Siebel Universal Agent']" +
  "/siebelft:cas[@ClassName='SiebScreen' " +
   "and @RepositoryName='Accounts Screen']" +
  "/siebelft:cas[@ClassName='SiebPDQ' " +
   "and @RepositoryName='SiebPDQ']";
 if (!result.equals(expected)) {
  warn("childPathFromRepositoryName returned  " + result);
 }
 }catch(Exception exp) {}
try{
 Integer result=siebelFT.screen(5, SiebScreen)
  .getClassCount("SiebPDQ");
 if (result<1) {
  warn("getClassCountreturned  " + result);
 }
 }catch(Exception exp) {}

try{
 String result=siebelFT.screen(7, SiebScreen)
  .getRepositoryNameByIndex("SiebPDQ", 0);
 if (!result.equals("SiebPDQ")) {
```

```
 warn("getRepositoryNameByIndex returned  " + result);
}
}catch(Exception exp) {} *
```

siebelFT.screenViews

Creates a Siebel ScreenViews element.

Format

The siebelFT.screenViews method has the following command format(s):

siebelFT.screenViews(*path*);

siebelFT.screenViews(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new ScreenViews.

Example

Performs an action on a Siebel ScreenViews element.

```
String SiebScreenView="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Activities Screen']" +
 "/siebelft:cas[@ClassName='SiebScreenViews' " +
  "and @RepositoryName='SiebScreenViews']";
siebelFT.screenViews(5, SiebScreenView).assertCells("screenView assert",
 siebelFT.cells(siebelFT.cell(2, 1, "Attachments",
   TestOperator.StringExact)));
siebelFT.screenViews(6, SiebScreenView).verifyCells("screenView verify",
 siebelFT.cells(siebelFT.cell(2, 1, "Attachments",
   TestOperator.StringExact)));
try{
 String result=siebelFT.screenViews(7, SiebScreenView)
  .getActiveView();
 if (!result.equals("Activity Attachment View")) {
   warn("getActiveView returned  " + result);
  }
  }catch(Exception exp) {}
try{
 String result=siebelFT.screenViews(8, SiebScreenView)
  .getCell(1, 1);
 if (!result.equals("More Info")) {
   warn("getCell returned  " + result);
  }
  }catch(Exception exp) {}
try{
 Integer result=siebelFT.screenViews(9, SiebScreenView)
  .getL2Count();
 if (result <0) {
```

```
   warn("getL2Count returned  " + result);
  }
  }catch(Exception exp) {}
try{
 Integer result=siebelFT.screenViews(10, SiebScreenView)
  .getL3Count();
 if (result <1) {
   warn("getL3Count returned  " + result);
  }
  }catch(Exception exp) {}
try{
 Integer result=siebelFT.screenViews(11, SiebScreenView)
  ;.getL4Count();
 if (result <0) {
   warn("getL4Count returned  " + result);
  }
  }catch(Exception exp) {}
try{
 String result=siebelFT.screenViews(12, SiebScreenView)
  .getUIName("Activity Detail View", "L3");
 if (!result.equals("More Info")) {
   warn("getUIName returned  " + result);
  }
  }catch(Exception exp) {}
```

siebelFT.text

Creates a Siebel Text element.

Format

The siebelFT.text method has the following command format(s):

siebelFT.text(*path*);

siebelFT.text(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Text.

Example

Performs an action on a Siebel Text element.

```
String SiebText="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Accounts Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Account List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Account Entry Applet']" +
 "/siebelft:cas[@ClassName='SiebText' " +
  "and @RepositoryName='Location']";
siebelFT.text(SiebText).setText("zopa");
siebelFT.text(SiebText).getPopupType();
try{
 String result=siebelFT.text(5, SiebText).getText();
 if (!result.equals("zopa")) {
  warn("getText returned " + result);
 }
 }catch(Exception exp) {}

try{
 Boolean result=siebelFT.text(6, SiebText).isEnabled();
 if (!result) {
  warn("isEnabled returned " + result);
 }
 }catch(Exception exp) {}

try{
 Boolean result=siebelFT.text(7, SiebText).isEncrypted();
 if (result) {
  warn("isEncrypted() returned " + result);
```

```
 }
 }catch(Exception exp) {}

siebelFT.text(8, SiebText).openPopup();
siebelFT.text(9, SiebText).setPassword("abc");
siebelFT.text(10, SiebText).processKey("EnterKey");
```

siebelFT.textArea

Creates a Siebel TextArea element.

Format

The siebelFT.textArea method has the following command format(s):

siebelFT.textArea(*path*);

siebelFT.textArea(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new TextArea.

Example

Performs an action on a Siebel TextArea element.

```
String SiebTextArea="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Opportunity List View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Opportunity Form Applet - Child']" +
 "/siebelft:cas[@ClassName='SiebTextArea' " +
  "and @RepositoryName='Description']";
siebelFT.textArea(5, SiebTextArea).setText("zopa");
try{
 String result=siebelFT.textArea(6, SiebTextArea)
  .getText();
 if (!result.equals("zopa")) {
  warn("getText returned " + result);
 }
 }catch(Exception exp) {}

try{
 Boolean result=siebelFT.textArea(7, SiebTextArea)
  .isEnabled();
 if (!result) {
  warn("isEnabled returned " + result);
 }
 }catch(Exception exp) {}
```

siebelFT.threadbar

Creates a Siebel Threadbar element.

Format

The siebelFT.threadbar method has the following command format(s):

siebelFT.threadbar(*path*);

siebelFT.threadbar(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Threadbar.

Example

Performs an action on a Siebel Threadbar element.

```
String SiebThreadbar="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Activities Screen']" +
 "/siebelft:cas[@ClassName='SiebThreadbar' " +
  "and @RepositoryName='SiebThreadbar']";
siebelFT.threadbar(2, SiebThreadbar).assertCells("assert",
 siebelFT.cells(siebelFT.cell(1, 1,"Opportunity Detail - Activities View0",
  TestOperator.StringExact)));
siebelFT.threadbar(4, SiebThreadbar).verifyCells("verify",
 siebelFT.cells(siebelFT.cell(1, 1,"Opportunity Detail - Activities View0",
  TestOperator.StringExact)));
try{
 String result=siebelFT.threadbar(5, SiebThreadbar)
  .getActiveThreadItem();
 if (!result.equals("Opportunity Detail - Activities View0")) {
  warn("getActiveThreadItem returned  " + result);
 }
 }catch(Exception exp) {}
try{
 String result=siebelFT.threadbar(6, SiebThreadbar)
  .getCell(1, 1);
 if (!result.equals("Opportunity Detail - Activities View0")) {
  warn("getCell returned  " + result);
 }
 }catch(Exception exp) {}
try{
 Integer result=siebelFT.threadbar(7, SiebThreadbar)
  .getCount();
 if (result <1) {
```

```
  warn("getCount returned  " + result);
 }
 }catch(Exception exp) {}
try{
 String result=siebelFT.threadbar(8, SiebThreadbar)
  .getThreadItemByIndex(0);
 if (!result.equals("Opportunity Detail - Activities View0")) {
  warn("getThreadItemByIndex returned  " + result);
 }
 }catch(Exception exp) {}
try{
 String result=siebelFT.threadbar(9, SiebThreadbar)
  .getThreadItems();
 if (!result.equals("Opportunity Detail - Activities View0")) {
  warn("getThreadItems returned  " + result);
 }
 }catch(Exception exp) {}
try{
 Boolean result=siebelFT.threadbar(10, SiebThreadbar)
  .isExists("Opportunity Detail - Activities View0");
 if (!result) {
  warn("isExists returned  " + result);
 }
 }catch(Exception exp) {}

siebelFT.threadbar(11, SiebThreadbar).storeCell("var", 1, 1);
if (!eval("{{var}}").equals("Opportunity Detail - Activities View0")){
 warn("storeCell returned  " + eval("{{var}}"));
}
```

siebelFT.toolbar

Creates a Siebel Toolbar element.

Format

The siebelFT.toolbar method has the following command format(s):

siebelFT.toolbar(*path*);

siebelFT.toolbar(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Toolbar.

Example

Performs an action on a Siebel Toolbar element.

```
siebelFT.toolbar(185,"/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebToolbar' " +
  "and @RepositoryName='HIMain']")
 .click("SiteMap");
```

siebelFT.tree

Creates a Siebel Tree element.

Format

The siebelFT.tree method has the following command format(s):

siebelFT.tree(*path*);

siebelFT.tree(*recid*, *path*);

Command Parameters

path
a String specifying the path of the element.

recid
the ID of a previously recorded navigation, used for comparison purposes. May be `null`.

Returns

a new Tree.

Example

Performs an action on a Siebel Tree element.

```
String SiebTree="/siebelft:cas[@ClassName='SiebApplication' " +
  "and @RepositoryName='Siebel Universal Agent']" +
 "/siebelft:cas[@ClassName='SiebScreen' " +
  "and @RepositoryName='Opportunities Screen']" +
 "/siebelft:cas[@ClassName='SiebView' " +
  "and @RepositoryName='Opportunity Explorer View']" +
 "/siebelft:cas[@ClassName='SiebApplet' " +
  "and @RepositoryName='Opportunity Tree Applet']" +
 "/siebelft:cas[@ClassName='SiebTree' " +
  "and @RepositoryName='SiebTree']";
siebelFT.tree(4, SiebTree).expand("1");
siebelFT.tree(5, SiebTree).select("1.2");
try{
 String result=siebelFT.tree(6, SiebTree)
  .getActiveTreeItem();
 if ((result.length()<1)) {
  warn("getActiveTreeItem returned " + result);
 }
 }catch(Exception exp) {}
try{
 String result=siebelFT.tree(7, SiebTree)
  .getTreeItemName("1.2");
 if (result.length()<1) {
  warn("getTreeItemName returned " + result);
 }
 }catch(Exception exp) {}
try{
 Integer result=siebelFT.tree(8, SiebTree)
  .getChildCount("1");
 if (result <1) {
```

```
  warn("getChildCount returned " + result);
 }
 }catch(Exception exp) {}
try{
 Boolean result=siebelFT.tree(9, SiebTree)
 .isExpanded("1");
 if (!result) {
  warn("isExpanded returned " + result);
 }
 }catch(Exception exp) {}

siebelFT.tree(10, SiebTree).nextPage();
siebelFT.tree(11, SiebTree).previousPage();
siebelFT.tree(12, SiebTree).collapse("1");
```

15

Web Functional Module

This chapter provides a complete listing and reference for the methods in the OpenScript WebDomService Class of Webdom Module Application Programming Interface (API).

15.1 WebDomService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript WebDomService API.

15.1.1 Alphabetical Command Listing

The following table lists the WebDomService API methods in alphabetical order.

Table 15–1 List of WebDomService Methods

Method	Description
web.accButton	Identifies an accessibility button by its path.
web.accCheckBox	Identifies an accessibility checkbox by its path.
web.accComboBox	Identifies an accessibility combobox by its path.
web.accElement	Identifies an accessibility element by its recorded ID and path.
web.accListBox	Identifies an accessibility listbox by its path.
web.accMenu	Identifies an accessibility menu by its path.
web.accRadioButton	Identifies an accessibility radio button by its path.
web.accTextBox	Identifies an accessibility text box by its path.
web.addGlobalAssertText	Registers a text matching (assertion) test to run every time a page is requested.
web.addGlobalVerifyText	Registers a text matching (verification) test to run every time a page is requested.
web.alertDialog	Identifies an alert dialog box by its recorded ID and path.
web.assertErrors	Checks for exceptions.
web.assertText	Searches the HTML contents of all documents in all browser windows for the specified text pattern, reporting a failure if the test fails.
web.attribute	Constructs the property data of a single object attribute.
web.attributes	Constructs the object attributes.
web.button	Identifies a button by its recorded ID and path.

Table 15–1 (Cont.) List of WebDomService Methods

Method	Description
web.cell	Constructs the test data of a table cell.
web.cells	Constructs the list of table cell tests.
web.checkBox	Identifies a checkbox by its recorded ID and path.
web.clearAllCache	Clears all cache files.
web.clearAllPersistentCookies	Clears all persistent Cookies.
web.clearCache	Clears cache files by defined domain name.
web.clearPersistentCookies	Clears persistent Cookies by defined domain name.
web.clearSessionCookies	Clears session cookies in the current opened browser window.
web.confirmDialog	Identifies a confirmation dialog box by its recorded ID and path.
web.customElement	Identifies a custom DOM element by its recorded ID and path.
web.dialog	Identifies a dialog box by its path.
web.document	Identifies a window by its recorded ID and path.
web.element	Identifies an element by its recorded ID and path.
web.exists	Checks if an object exists or not with a specified timeout value.
web.getFocusedWindow	Gets the window with the focus.
web.image	Identifies an image by its recorded ID and path.
web.link	Identifies a link by its recorded ID and path.
web.loginDialog	Identifies a login dialog box by its recorded ID and path.
web.notificationBar	Identifies a notification bar by its path.
web.object	Identifies an object by its recorded ID and path.
web.promptDialog	Identifies a prompt dialog box by its recorded ID and path.
web.radioButton	Identifies a radio button by its recorded ID and path.
web.removeGlobalTextValidator	Removes a global text matching test (either assertion or verification).
web.selectBox	Identifies a select list box by its recorded ID and path.
web.solve	Searches the HTML contents of all documents in all browsers for the specified regular expression pattern.
web.table	Identifies a table by its recorded ID and path.
web.textArea	Identifies a text area by its recorded ID and path.
web.textBox	Identifies a text box by its recorded ID and path.
web.verifyText	Searches the HTML contents of all documents in all browser windows for the specified text pattern, reporting a warning if the test fails.
web.waitForObject	Waits for an object to exist before timing out.
web.window	Identifies a window by its recorded ID and path.
web.xmlDocument	Identifies an xml document by its recorded ID and path.

The following sections provide detailed reference information for each method and enum in the WebDomService Class of Webdom Module Application Programming Interface.

web.accButton

Identifies an accessibility button by its path.

Format

The web.accButton method has the following command format(s):

web.accButton(*path*);

web.accButton(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

an AccButton object using the specified path.

Example

Examples of actions on a button specified by its path.

```
//using script variable
getVariables().set("btnPath", "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accPushButton[@index='1']");
web.accButton(1, "{{btnPath}}").click();
//using Java variable
String btnPath = "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accPushButton[@index='1']";
web.accButton(10, btnPath).click();
```

web.accCheckBox

Identifies an accessibility checkbox by its path.

Format

The web.accCheckBox method has the following command format(s):

web.accCheckBox(*path*);

web.accCheckBox(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

an AccCheckBox object using the specified path.

Example

Examples of actions on a checkbox specified by its recorded ID and path.

```
//using script variable
getVariables().set("cbpath", "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accCheckButton[@index='0']");
web.accCheckBox(1, "{{cbpath}}").check(true);
//using Java variable
String cbpath = "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accCheckButton[@index='0']";
web.accCheckBox(2, cbpath).check(false);
```

web.accComboBox

Identifies an accessibility combobox by its path.

Format

The web.accComboBox method has the following command format(s):

web.accComboBox(*path*);

web.accComboBox(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

an AccComboBox object using the path.

Example

Examples of actions on the combobox specified by its recorded ID and path.

```
//using script variable
getVariables().set("combobox", "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accComboBox[@index='0']");
web.accComboBox(101, "{{combobox}}").selectOptionByText("Bikes");
//using Java variable
String combobox = "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accComboBox[@index='0']";
web.accComboBox((101, combobox).selectOptionByIndex(0);
```

web.accElement

Identifies an accessibility element by its recorded ID and path.

Format

The web.accElement method has the following command format(s):

web.accElement(*path*);

web.accElement(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

an AccElement object using the specified recorded ID and path.

Example

Performs an action on an accessibility element specified by its recorded ID and path.

```
//using script variable
getVariables().set("elmPath", "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='3']" +
 "/web:accClient[@index='0']");
web.accElement(56, "{{elmPath}}").mouseClick(260, 265);
//using Java variable
String elmPath = "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='3']" +
 "/web:accClient[@index='0']";
web.accElement(56, elmPath).mouseClick(260, 265);
```

web.accListBox

Identifies an accessibility listbox by its path.

Format

The web.accListBox method has the following command format(s):

web.accListBox(*path*);

web.accListBox(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

an AccListBox object using the specified path.

Example

Examples of actions on the list box specified by its recorded ID and path.

```
//using script variable
getVariables().set("listbox", "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accList[@index='0']");
web.accListBox(101, "{{listbox}}").selectOptionByText("Bikes");
//using Java variable
String listbox = "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accList[@index='0']";
web.accListBox(101, listbox).selectOptionByIndex(0);
```

web.accMenu

Identifies an accessibility menu by its path.

Format

The web.accMenu method has the following command format(s):

web.accMenu(*path*);

web.accMenu(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

an AccMenu object using the specified path.

Example

Examples of actions on an accessibility menu specified by its recorded ID and path.

```
//using script variable
getVariables().set("menu", "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accMenu[@index='0']");
web.accMenu(101, "{{menu}}").selectNode("File->Open");
//using Java variable
String menu = "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accMenu[@index='0']";
web.accMenu(101, menu).selectNode("File->Open");
```

web.accRadioButton

Identifies an accessibility radio button by its path.

Format

The web.accRadioButton method has the following command format(s):

web.accRadioButton(*path*);

web.accRadioButton(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

an AccRadioButton object using the specified path.

Example

Performs an action on a radio button specified by its recorded ID and path.

```
//using script variable
getVariables().set("radioBtn", "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accRadioButton[@index='0' " +
   "or @name='Search Any']");
web.accRadioButton(96, "{{radioBtn}}").select();
//using Java variable
String radioBtn = "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accRadioButton[@index='0' " +
   "or @name='Search Any']";
web.accRadioButton(96, radioBtn).select();
```

web.accTextBox

Identifies an accessibility text box by its path.

Format

The web.accTextBox method has the following command format(s):

web.accTextBox(*path*);

web.accTextBox(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

an AccTextBox object using the specified recorded ID and path.

Example

Clicks on a text box specified by its recorded ID and path and sets the text in the text box.

```
//using script variable
getVariables().set("tbPath", "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accText[@index='0']");
web.accTextBox(10, "{{tbPath}}").setText("text box text");
//using Java variable
String tbPath = "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='0']" +
 "/web:accText[@index='1']";
web.accTextBox(10, tbPath).setText("text box text");
```

web.addGlobalAssertText

Registers a text matching (assertion) test to run every time a page is requested.

The text matching (assertion) test searches the HTML contents of all documents in all browser windows for the specified text pattern. If the test fails, the script will always fail and stop running, unless the text matching test error recovery setting specifies a different action such as Warn or Ignore.

The assertion is performed for all requests made by this virtual user.

This method supports result code verification methods including `getLastResult()` and `getLastError()`.

To unregister a global text assertion, use web.removeGlobalTextValidator.

Format

The web.addGlobalAssertText method has the following command format(s):

web.addGlobalAssertText(*testName*, *textToAssert*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
A string specifying aescriptive name of the test being applied. This name may be used by a subsequent call to web.removeGlobalTextValidator. Must not be `null`.

textToAssert
a String specifying the text to match on the page, or not match on the page if TextPresence is TextPresence.PassIfPresent. Must not be `null`.

sourceType
a Source enum specifying the location to match the text either in the HTML contents or HTTP response headers. Must not be null.

textPresence
TextPresence enum specifying Either PassIfPresent or FailIfPresent, depending on if you want the text to be present or not.

matchOption
a MatchOption enum specifying either Exact, RegEx, or Wildcard, depending on match type.

Example

Adds a Global Text Matching test for Display Content, Response Header, and Source HTML respectively.

`globalTextMatchingTest1` passes if the text to match string is present in the Display Content and matches exactly.

`globalTextMatchingTest2` passes if the text to match string is present in the Response header and matches the Regular Expression.

`globalTextMatchingTest3` fails if the text to match string is present in the Source HTML and matches the Wildcard.

```
web.addGlobalAssertText("globalTextMatchingTest1",
```

```
 "match this text string", Source.DisplayContent,
 TextPresence.PassIfPresent, MatchOption.Exact);
web.addGlobalAssertText("globalTextMatchingTest2",
 "jsessionid=(.+?)(?:\\\"|&)", Source.ResponseHeader,
 TextPresence.PassIfPresent, MatchOption.RegEx);
web.addGlobalAssertText("globalTextMatchingTest3",
 "match this *", Source.Html, TextPresence.FailIfPresent,
 MatchOption.Wildcard);
```

web.addGlobalVerifyText

Registers a text matching (verification) test to run every time a page is requested.

The text matching (verification) test searches the HTML contents of all documents in all browser windows for the specified text pattern. If the test fails, a warning is always reported, irrespective of current Error Recovery settings.

The verification is performed for all requests made by this virtual user.

This method supports result code verification methods including `getLastResult()` and `getLastError()`.

To unregister a global text verification, use web.removeGlobalTextValidator.

Format

The web.addGlobalVerifyText method has the following command format(s):

web.addGlobalVerifyText(*testName*, *textToAssert*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the descriptive name of the test being applied. This name may be used by a subsequent call to web.removeGlobalTextValidator. Must not be null.

textToAssert
a String specifying the text to match on the page, or not match on the page if TextPresence says TextPresence.PassIfPresent. Must not be null.

sourceType
a Source enum specifying where to match the text either in the HTML contents or HTTP response headers. Must not be null.

textPresence
a TextPresence enum specifying either PassIfPresent or FailIfPresent, depending on if you want the text to be present or not.

matchOption
a MatchOption enum specifying either Exact, RegEx, or Wildcard, depending on match type.

Example

Adds a Global Text Matching test for Display Content, Response Header, and Source HTML respectively.

`globalTextMatchingTest1` passes if the text to match string is present in the Display Content and matches exactly.

`globalTextMatchingTest2` passes if the text to match string is present in the Response header and matches the Regular Expression.

`globalTextMatchingTest3` fails if the text to match string is present in the Source HTML and matches the Wildcard.

```
web.addGlobalVerifyText("globalTextMatchingTest1",
 "match this text string", Source.DisplayContent,
```

```
 TextPresence.PassIfPresent, MatchOption.Exact);
web.addGlobalVerifyText("globalTextMatchingTest2",
 "jsessionid=(.+?)(?:\\\"|&)", Source.ResponseHeader,
 TextPresence.PassIfPresent, MatchOption.RegEx);
web.addGlobalVerifyText("globalTextMatchingTest3",
 "match this *", Source.Html, TextPresence.FailIfPresent,
 MatchOption.Wildcard);
```

web.alertDialog

Identifies an alert dialog box by its recorded ID and path.

Format

The web.alertDialog method has the following command format(s):

web.alertDialog(*path*);

web.alertDialog(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMAlertDialog object using the specified recorded ID and path.

Example

Performs an action on an alert dialog box specified by its recorded ID and path.

```
//using script variable
getVariables().set("dlgPath",
 "/web:dialog_alert[@text='ALERT!' and @index='0']");
web.alertDialog(1, "{{dlgPath}}").clickOk();
//using Java variable
String dlgPath = "/web:dialog_alert[@text='ALERT!' and @index='0']";
web.alertDialog(42, dlgPath).clickOk();
```

web.assertErrors

Checks for exceptions. Manually enter this method into a script to check if an exception occurred.

If there are exception in previous stepResult, just break the script.

Format

The web.assertErrors method has the following command format(s):

```
web.assertErrors( );
```

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Checks for exceptions. If there is an exception in the previous stepResult, cause a break in the script execution.

```
web.assertErrors();
```

web.assertText

Searches the HTML contents of all documents in all browser windows for the specified text pattern, reporting a failure if the test fails.

If the test fails, always fail the script unless the text matching test error recovery setting specifies a different action such as Warn or Ignore.

Format

The web.assertText method has the following command format(s):

web.assertText(*testName*, *pattern*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

pattern
a String specifying the pattern of the test.

sourceType
a Source enum specifying the location to match, either the source HTML or the HTML excluding markup tags.

textPresence
a TextPresence enum specifying either PassIfPresent or FailIfPresent, depending on if you want the text to be present or not.

matchOption
a MatchOption enum specifying either Exact, RegEx, or Wildcard, depending on match type.

Throws

MatchException
if the assertion fails. AbstractScriptException on any other failure when attempting to assert the text.

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Adds Text Matching tests for Display Content and Raw HTML.

`myTextMatchingTest1` passes if the text to match string is present in the Display Content and matches exactly.

`myTextMatchingTest2` passes if the text to match string is present in the source HTML and matches the Regular Expression.

`myTextMatchingTest3` fails if the text to match string is present in the source HTML and matches the Wildcard.

```
web.assertText("myTextMatchingTest1", "match this text string",
```

```
 Source.DisplayContent, TextPresence.PassIfPresent,
 MatchOption.Exact);
web.assertText("myTextMatchingTest2", "match(.+?)(?:\"|&)",
 Source.Html, TextPresence.PassIfPresent,
 MatchOption.RegEx);
web.assertText("myTextMatchingTest3", "match this *",
 Source.Html, TextPresence.FailIfPresent,
 MatchOption.Wildcard);
```

web.attribute

Constructs the property data of a single object attribute.

Format

The web.attribute method has the following command format(s):

web.attribute(*property*, *expectedValue*, *operator*);

Command Parameters

property
a String specifying the name of the property.

expectedValue
a String specifying the expected value of the property.

operator
TestOperator that defines the type and the match approach of the data.

Returns

a PropertyTest object with the specified property, expectedValue, operator

Example

Specifies each individual attribute of an Object Test.

```
getVariables().set("elmPath", "/web:window[@index='0' or @title='Mockups']" +
 "/web:document[@index='0']" +
 "/web:table[@id='toc' or @index='0']");
web.element(122,"{{elmPath}}").assertAttributes("MyObjectTest",
 web.attributes(web.attribute("tag", "TABLE", TestOperator.StringExact),
   web.attribute("id","toc", TestOperator.StringExact),
   web.attribute("className", "toc", TestOperator.StringExact),
   web.attribute("class", "toc", TestOperator.StringExact),
   web.attribute("summary", "Conts", TestOperator.StringExact),
   web.attribute("index", "0", TestOperator.StringExact)));
```

web.attributes

Constructs the object attributes.

Format

The web.attributes method has the following command format(s):

web.attributes(*tests*);

Command Parameters

tests
a list of attributes.

Returns

a PropertyTest[] with the specified attributes.

Example

Specifies the attibutes of an Object Test.

```
getVariables().set("elmPath", "/web:window[@index='0' or @title='Mockups']" +
 "/web:document[@index='0']" +
 "/web:table[@id='toc' or @index='0']");
web.element(122,"{{elmPath}}").assertAttributes("MyObjectTest",
 web.attributes(web.attribute("tag", "TABLE", TestOperator.StringExact),
   web.attribute("id", "toc", TestOperator.StringExact),
   web.attribute("className", "toc", TestOperator.StringExact),
   web.attribute("class", "toc", TestOperator.StringExact),
   web.attribute("summary", "Contents", TestOperator.StringExact),
   web.attribute("index", "0", TestOperator.StringExact)));
```

web.button

Identifies a button by its recorded ID and path.

Format

The web.button method has the following command format(s):

web.button(*path*);

web.button(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMButton object using the specified recorded ID and path.

Example

Examples of actions on a button specified by its recorded ID and path.

```
//using script variable
getVariables().set("btnPath", "/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']");
web.button(1, "{{btnPath}}").click();
web.button(2, "{{btnPath}}").mouseClick("<CTRL>", 1, false);
//using Java variable
String btnPath = "/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']";
web.button(1, btnPath).click();
web.button(2, btnPath).mouseClick("<CTRL>", 1, false);
```

web.cell

Constructs the test data of a table cell.

Format

The web.cell method has the following command format(s):

web.cell(*row*, *column*, *expectedValue*, *operator*);

Command Parameters

row
a 1-based index value specifying the row number.

column
a 1-based index value specifying the column number.

expectedValue
a String specifying the expected value of the property.

operator
a TestOperator enum that defines the type and the match approach of the data.

Returns

a CellTest object with the specified row, column, expectedValue, and operator.

Example

Adds a table test for specifying exact match or wildcard match for individual cells.

```
web.table(14,"/web:window[@title='Ticker List']" +
 "/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.StringWildCard),
     web.cell(2, 2, "Oracle", TestOperator.StringExact)));
```

web.cells

Constructs the list of table cell tests.

Format

The web.cells method has the following command format(s):

web.cells(*tests*);

Command Parameters

tests
a list of table cells.

Returns

a CellTest[] with the specified cells.

Example

Adds a table test for specifying exact match or wildcard match for each cell.

```
web.table(14,"/web:window[@title='Ticker List']" +
 "/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.StringWildCard),
     web.cell(2, 2, "Oracle", TestOperator.StringExact)));
```

web.checkBox

Identifies a checkbox by its recorded ID and path.

Format

The web.checkBox method has the following command format(s):

web.checkBox(*path*);

web.checkBox(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMCheckbox object using the specified recorded ID and path.

Example

Examples of actions on a checkbox specified by its recorded ID and path.

```
//using script variable
getVariables().set("cbpath", "/web:window[@index='0' " +
   "or @title='Chart Portfolio']" +
 "/web:document[@index='0']" +
 "/web:input_checkbox[@id='chkTrack' or @name='chkTrack' " +
   "or @index='0']");
web.checkBox(1, "{{cbpath}}").click();
web.checkBox(2, "{{cbpath}}").check(true);
web.checkBox(3, "{{cbpath}}").dblClick();
web.checkBox(4, "{{cbpath}}").mouseClick(1, 1, "<ALT>", 1,
 false, ClickPosition.Center);
web.checkBox(5, "{{cbpath}}").clickContextMenu("contextMenuItem", 0);
web.checkBox(6, "{{cbpath}}").keyPress("<SPACE>");
//using Java variable
String cbpath = "/web:window[@index='0' " +
   "or @title='Chart Portfolio']" +
 "/web:document[@index='0']" +
  "/web:input_checkbox[@id='chkTrack' or @name='chkTrack' " +
   "or @index='0']";
web.checkBox(1, cbpath).click();
web.checkBox(2, cbpath).check(true);
web.checkBox(3, cbpath).dblClick();
web.checkBox(4, cbpath).mouseClick(1, 1, "<ALT>", 1,
 false, ClickPosition.Center);
```

```
web.checkBox(5, cbpath).clickContextMenu("contextMenuItem", 0);
web.checkBox(6, cbpath).keyPress("<SPACE>");
```

web.clearAllCache

Clears all cache files.

Format

The web.clearAllCache method has the following command format(s):

web.clearAllCache( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Clears all Cache.

```
web.clearAllCache();
```

web.clearAllPersistentCookies

Clears all persistent Cookies.

Format

The web.clearAllPersistentCookies method has the following command format(s):

```
web.clearAllPersistentCookies( );
```

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Clears all Persistent Cookies.

```
web.clearAllPersistentCookies();
```

web.clearCache

Clears cache files by defined domain name.

Format

The web.clearCache method has the following command format(s):

web.clearCache(*domainName*);

Command Parameters

domainName
a String specifying the Domain name of the Web server.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Clears cache files by the specified domain name.

```
web.clearCache("domainName");
```

web.clearPersistentCookies

Clears persistent Cookies by defined domain name.

Format

The web.clearPersistentCookies method has the following command format(s):

web.clearPersistentCookies(*domainName*);

Command Parameters

domainName
a String specifying the Domain name of the Web server.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Clears Persistent Cookies by the specified domain name.

```
web.clearPersistentCookies("domainName");
```

web.clearSessionCookies

Clears session cookies in the current opened browser window.

Format

The web.clearSessionCookies method has the following command format(s):

web.clearSessionCookies( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Clears session cookies in the currently opened browser window.

```
web.clearSessionCookies();
```

web.confirmDialog

Identifies a confirmation dialog box by its recorded ID and path.

Format

The web.confirmDialog method has the following command format(s):

web.confirmDialog(*path*);

web.confirmDialog(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMConfirmDialog object using the specified recorded ID and path.

Example

Performs an action on a open JavaScript dialog box specified by its recorded ID and path.

```
//using script variable
getVariables().set("cfmDialog",
 "/web:dialog_confirm[@text='Continue? OK/Cancel' " +
   "and @index='0']");
web.confirmDialog(46, "{{cfmDialog}}").clickOk();
//using Java variable
String cfmDialog = "/web:dialog_confirm[@text='Continue? OK/Cancel' " +
   "and @index='0']";
web.confirmDialog(46, cfmDialog).clickCancel();
```

web.customElement

Identifies a custom DOM element by its recorded ID and path.

Format

The web.customElement method has the following command format(s):

web.customElement(*path*);

web.customElement(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a ICustomDOMElement object using the specified path.

Example

Performs an action on the custom element specified by its path.

```
getVariables().set("custElmPath", "/web:window[@index='0' " +
   "or @title='Mail Order']" +
 "/web:document[@index='0']" +
 "/web:web:element[@id='btnBikes' or @name='btnBikes' " +
   "or @index='0']");
web.customElement(12, "{{custElmPath}}").assertAttribute("myTest", "id",
 "btnBikes", TestOperator.StringExact);
```

web.dialog

Identifies a dialog box by its path.

Format

The web.dialog method has the following command format(s):

web.dialog(*path*);

web.dialog(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMDialog object using the specified path.

Example

Performs an action on an open dialog box specified by its recorded ID and path.

```
//using script variable
getVariables().set("dlgPath",
 "/web:dialog_unknown[@text='an input box' and @index='0']");
web.dialog(1, "{{dlgPath}}").setText(0, "text input");
web.dialog(2, "{{dlgPath}}").clickButton(0);
//using Java variable
String dlgPath = "/web:dialog_unknown[@text='an input box' and @index='0']";
web.dialog(1, dlgPath).setText(0, "text input");
web.dialog(2, dlgPath).clickButton(0);
```

web.document

Identifies a window by its recorded ID and path.

Format

The web.document method has the following command format(s):

web.document(*path*);

web.document(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMDocument object using the specified recorded ID and path.

Example

Performs an action on an open web document specified by its recorded ID and path.

```
getVariables().set("recHtml", web.document(1,
 "/web:document[@index='0]'").getHTML());
info("The HTML string is: {{recHtml}}");
getVariables().set("recUrl", web.document(2,
 "/web:document[@index='0]'").getRecordedURL());
info("The URL is: {{recUrl}}");
```

web.element

Identifies an element by its recorded ID and path.

Format

The web.element method has the following command format(s):

web.element(*path*);

web.element(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMElement object using the specified recorded ID and path.

Example

Performs an action on an open VB Script dialog box specified by its recorded ID and path.

```
//using script variable
getVariables().set("elmPath", "/web:window[@index='1' " +
   "or @title='VBScript Form']" +
 "/web:document[@index='0']" +
 "/web:table[@index='0']");
web.element(56, "{{elmPath}}").click();
//using Java variable
String elmPath1 = "/web:window[@index='1' " +
   "or @title='VBScript Form']" +
 "/web:document[@index='0']" +
 "/web:table[@index='0']";
web.element(56, elmPath).click();
```

web.exists

Checks if an object exists or not with a specified timeout value.

Format

The web.exists method has the following command format(s):

web.exists(*path*);

web.exists(*path*, *timeout*);

Command Parameters

path
a String specifying the object path.

timeout
a timeout value to find an object, in seconds.

Returns

return `true` if the object exists, otherwise return `false`.

Example

Checks if the specified object exists or not. Timeout after 10 seconds.

```
Boolean doesExist = web.exists("/web:document[@index='0]'", 10);
if (doesExist = true) {
 info("document exists");
}
```

web.getFocusedWindow

Gets the window with the focus.

Format

The web.getFocusedWindow method has the following command format(s):

web.getFocusedWindow( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMBrowser object.

Example

Gets the title of the window that has the focus.

```
//using script variable
getVariables().set("winTitle", web.getFocusedWindow().getTitle());
info("Script: Title is {{winTitle}}");
//using Java variable
String winTitle = web.getFocusedWindow().getTitle();
info("Java: Title is " + winTitle);
```

web.image

Identifies an image by its recorded ID and path.

Format

The web.image method has the following command format(s):

web.image(*path*);

web.image(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMImage object using the specified recorded ID and path.

Example

Examples of actions on an image specified by its recorded ID and path.

```
//using script variable
getVariables().set("imgPath", "/web:window[@index='0' " +
   "or @title='Mail Order']" +
 "/web:document[@index='0']" +
 "/web:input_image[@id='btnBikes' or @name='btnBikes' " +
   "or @src='http://testserver2/images/bikes.jpg' " +
   "or @index='0']");
web.image(82, "{{imgPath}}").click();
//using Java variable
String imgPath = "/web:window[@index='0' or @title='Mail Order']" +
 "/web:document[@index='0']" +
 "/web:input_image[@id='btnBikes' or @name='btnBikes' " +
   "or @src='http://testserver2/images/bikes.jpg' " +
   "or @index='0']";
web.image(82, imgPath).click();
```

web.link

Identifies a link by its recorded ID and path.

Format

The web.link method has the following command format(s):

web.link(*path*);

web.link(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMLink object using the specified recorded ID and path.

Example

Examples of actions on the link specified by its recorded ID and path.

```
//using script variable
getVariables().set("lnkPath", "/web:window[@index='0' " +
   "or @title=' Bikes']" +
 "/web:document[@index='0']" +
 "/web:a[@text='Mail Order Bikes' " +
   "or @href='http://testserver2/bikes/Main.aspx' " +
   "or @index='5']");
web.link(60, "{{lnkPath}}").click();
//using Java variable
String lnkPath = "/web:window[@index='0' or @title=' Bikes']" +
 "/web:document[@index='0']" +
 "/web:a[@text='Mail Order Bikes' " +
   "or @href='http://testserver2/bikes/Main.aspx' " +
   "or @index='5']";
web.link(60, lnkPath).click();
```

web.loginDialog

Identifies a login dialog box by its recorded ID and path.

Format

The web.loginDialog method has the following command format(s):

web.loginDialog(*path*);

web.loginDialog(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMLoginDialog object using the specified recorded ID and path.

Example

Performs an action on the login dialog box specified by its recorded ID and path.

```
getVariables().set("dlgPath", "/web:dialog_unknown[@text='Login!' " +
   "and @index='0']");
//Click OK with username and password. Remember password true.
web.loginDialog(10, "{{dlgPath}}").clickOk("username",
   decrypt("vGXUWvDW/F7E6OSYUjRmsQ=="), true);
//Click Cancel
web.loginDialog(15, "{{dlgPath}}").clickCancel();
```

web.notificationBar

Identifies a notification bar by its path.

Format

The web.notificationBar method has the following command format(s):

web.notificationBar(*path*);

web.notificationBar(*recId*, *path*);

Command Parameters

recId
the ID of a previously recorded navigation, used for comparison purposes.

path
a String specifying the object path.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMNotificationBar object using the specified path.

Example

Performs an action on a notification bar specified by its recorded ID and path.

```
//using script variable
getVariables().set("notiBarPath",
 "/web:window[@index='0' or @title='Download Portfolio']");
web.notificationBar("{{notiBarPath}}").selectOptionBy(3, 1);
web.notificationBar("{{notiBarPath}}").clickButton(2);
//using Java variable
String notiBarPath = "/web:window[@index='0' or @title='Download Portfolio']";
web.notificationBar(notiBarPath).selectOptionBy(3, 1);
web.notificationBar(notiBarPath).clickButton(2);
```

web.object

fies an object by its recorded ID an

Format

The web.object method has the following command format(s):

web.object(*path*);

web.object(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMObject object using the specified recorded ID and path.

Example

Performs an action on the object specified by its recorded ID and path.

```
getVariables().set("objPath", "/web:window[@index='0']" +
 "/web:document[@index='0']" +
 "/web:object[@index='3']");
web.object(14, "{{objPath}}").click();
```

web.promptDialog

Identifies a prompt dialog box by its recorded ID and path.

Format

The web.promptDialog method has the following command format(s):

web.promptDialog(*path*);

web.promptDialog(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMPromptDialog object using the specified recorded ID and path.

Example

Performs an action on the prompt dialog box specified by its recorded ID and path.

```
//using script variable
getVariables().set("dlgPropmt", "/web:dialog_prompt[@text='Script Prompt:' " +
   "and @index='0']");
web.promptDialog(90, "{{dlgPropmt}}").clickOk("input text");
//using Java variable
String dlgPropmt = "/web:dialog_prompt[@text='Script Prompt:' and @index='0']";
web.promptDialog(90, dlgPropmt).clickOk("input text");
```

web.radioButton

Identifies a radio button by its recorded ID and path.

Format

The web.radioButton method has the following command format(s):

web.radioButton(*path*);

web.radioButton(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMRadioButton object using the specified recorded ID and path.

Example

Performs an action on a radio button specified by its recorded ID and path.

```
//using script variable
getVariables().set("radioBtn", "/web:window[@index='0' or @title='Sell']" +
 "/web:document[@index='0']" +
 "/web:form[@index='0']" +
 "/web:input_radio[@id='symbol3' " +
   "or (@name='symbol' and @value='ORCL') or @index='2']");
Boolean isSelected = web.radioButton(96, "{{radioBtn}}")
   .isSelected();
if (isSelected = false) {
 info("radio button not selected");
web.radioButton(96, "{{radioBtn}}").select();
}
//using Java variable
String radioBtn = "/web:window[@index='0' or @title='Sell']" +
 "/web:document[@index='0']/web:form[@index='0']" +
 "/web:input_radio[@id='symbol3'" +
   "or (@name='symbol' and @value='ORCL') or @index='2']");
Boolean isSelected = web.radioButton(96, "{{radioBtn}}")
   .isSelected();
if (isSelected = false) {
 info("radio button not selected");
web.radioButton(96, radioBtn).select();
}
```

web.removeGlobalTextValidator

Removes a global text matching test (either assertion or verification).

This method removes text matching tests that were previously added using web.addGlobalAssertText or web.addGlobalVerifyText.

If the specified test was not previously added, this method does nothing. No error is thrown.

Format

The web.removeGlobalTextValidator method has the following command format(s):

web.removeGlobalTextValidator(*testName*);

Command Parameters

testName
a String specifying the name of the test previously added using web.addGlobalAssertText or web.addGlobalVerifyText.

Returns

true if the test was removed, false if no test removed.

Example

Removes the Global Text Matching test added as "globalTextMatchingTest1".

```
web.removeGlobalTextValidator("globalTextMatchingTest1");
```

web.selectBox

Identifies a select list box by its recorded ID and path.

Format

The web.selectBox method has the following command format(s):

web.selectBox(*path*);

web.selectBox(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMSelect object using the specified recorded ID and path.

Example

Examples of actions on the select list box specified by its recorded ID and path.

```
//using script variable
getVariables().set("selectList", "/web:window[@index='0'" +
   "or @title='downhill bikes home']" +
 "/web:document[@index='0']" +
 "/web:form[@id='Default' or @name='Default' or @index='0']" +
 "/web:select[(@id='lstCatalogCategory' or @name='lstCatalogCategory'" +
   ""or @index='0') and multiple mod 'False']");
web.selectBox(1, "{{selectList}}").selectOptionByText("Bikes");
//using Java variable
String selectList = "/web:window[@index='0'" +
   "or @title='downhill bikes home']" +
 "/web:document[@index='0']" +
 "/web:form[@id='Default' or @name='Default' or @index='0']" +
 "/web:select[(@id='lstCatalogCategory' or @name='lstCatalogCategory'" +
   "or @index='0') and multiple mod 'False']";
web.selectBox(1, selectList).selectOptionByText("Bikes");
//multiple list select using script variable
getVariables().set("selectList", "/web:window[@index='0' " +
   "or @title='downhill bikes home']" +
 "/web:document[@index='0']" +
 "/web:form[@id='Default' or @name='Default' or @index='0']" +
 "/web:select[(@id='lstCatalogCategory' or @name='lstCatalogCategory'" +
   "or @index='0') and multiple mod 'True']");
web.selectBox(1, "{{selectList}}").multiSelectOptionByText("Bikes", "Parts");
```

web.solve

Searches the HTML contents of all documents in all browsers for the specified regular expression pattern.

Found results are stored in the variable named `varName`. If no results are found, and error recovery settings specify that WEBDOM_SOLVE_ERROR should fail, then this method throws a `SolveException`.

Format

The web.solve method has the following command format(s):

web.solve(*varName*, *pattern*, *sourceType*, *resultIndex*);

Command Parameters

varName
a String specifying the name of the variable to create.

pattern
a String specifying the Regular Expression pattern specifying what to extract from each document's HTML contents.

sourceType
a Source emun specifying the location to match, either the raw HTML or the HTML excluding markup tags.

resultIndex
a value specifying which value to match if the pattern matches more than 1 value. Specify the 0-based index of the specific result to retrieve. If `null` is specified, all results will be added into the variables collection. For example, the first result found is stored in `varName`,the second is stored in `varName[1]`, the third in `varName[2]`, [...]

Throws

AbstractScriptException
If no results are found, and error recovery settings specifythat WEBDOM_SOLVE_ ERROR should fail, then this method throws a `SolveException`.

Example

Parse a value from the most recent navigation's specified source using a Regular Expression and store it in a variable. Includes a result index value of 0 to specify it should retrieve the first match result.

```
web.solve("varTitle", "<TITLE>(.+)</TITLE>", Source.Html, 0);
info("Page Title = {{varTitle}}");
```

web.table

Identifies a table by its recorded ID and path.

Format

The web.table method has the following command format(s):

web.table(*path*);

web.table(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMTable object using the specified recorded ID and path.

Example

Performs an action on the table specified by its recorded ID and path.

```
getVariables().set("tblPath", "/web:window[@index='0' " +
   "or @title='Ticker List']" +
 "/web:document[@index='0']" +
 "/web:table[@index='6']");
web.table(14, "{{tblPath}}").assertCells("MyTableTest",
 web.cells(web.cell(1, 1, "Ticker ",TestOperator.StringExact),
   web.cell(1, 2,"Company ", TestOperator.StringExact),
   web.cell(2, 1, "ORCL*", TestOperator.StringWildCard),
   web.cell(2, 2, "Oracle", TestOperator.StringExact)));
```

web.textArea

Identifies a text area by its recorded ID and path.

Format

The web.textArea method has the following command format(s):

web.textArea(*path*);

web.textArea(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMTextArea object using the specified recorded ID and path.

Example

Clicks on a text area specified by its recorded ID and path and sets the text in the text area.

```
getVariables().set("taPath", "/web:window[@index='0' " +
   "or @title='Input Samples']" +
 "/web:document[@index='0']" +
 "/web:textarea[@name='myTextArea' or @index='0']");
web.textArea(84, "{{taPath}}").click()
web.textArea(85, "{{taPath}}").setText("text area text");
```

web.textBox

Identifies a text box by its recorded ID and path.

Format

The web.textBox method has the following command format(s):

web.textBox(*path*);

web.textBox(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMText object using the specified recorded ID and path.

Example

Clicks on a text box specified by its recorded ID and path and sets the text in the text box.

```
//plain text box
getVariables().set("tbPath", "/web:window[@index='0' " +
   "or @title='Stocks']" +
 "/web:document[@index='0']" +
 "/web:form[@id='loginform' or @name='loginform' " +
   "or @index='0']" +
 "/web:input_text[@id='login' or @name='login' " +
   "or @index='0']");
web.textBox(9,"{{tbPath}}").click();
web.textBox(10, "{{tbPath}}").setText("text box text");
//password box
getVariables().set("tbPath", "/web:window[@index='0' " +
   "or @title='Stocks']" +
 "/web:document[@index='0']" +
 "/web:form[@id='loginform' or @name='loginform' " +
   "or @index='0']" +
 "/web:input_password[@name='password' or @index='0']");
web.textBox(11, "{{tbPath}}").click();
web.textBox(12,
"{{tbPath}}").setPassword(deobfuscate("wKAUCoeFpUbXlwTYN9lp7A=="));
```

web.verifyText

Searches the HTML contents of all documents in all browser windows for the specified text pattern, reporting a warning if the test fails.

This method will never cause a script to fail, regardless of the text matching test error recovery settings. Use web.assertText to make the script fail when the test fails.

Format

The web.verifyText method has the following command format(s):

web.verifyText(*testName*, *pattern*, *sourceType*, *textPresence*, *matchOption*);

Command Parameters

testName
a String specifying the test name.

pattern
a String specifying the pattern of test.

sourceType
a Source enum specifying the location to match, either the source HTML or the HTML excluding markup tags.

textPresence
a TextPresence enum specifying either PassIfPresent or FailIfPresent, depending on if you want the text to be present or not.

matchOption
a MatchOption enum specifying either Exact, RegEx, or Wildcard, depending on match type.

Throws

AbstractScriptException
on any failure when attempting to verify the text.

Example

Adds a Verify only, never fail Text Matching tests for Display Content and Raw HTML respectively.

`myTextMatchingTest1` passes if the text to match string is present in the Display Content and matches exactly.

`myTextMatchingTest2` fails if the text to match string is present in the source HTML and matches the Wildcard.

```
web.verifyText("myTextMatchingTest1", "FMStocks Customer Login",
 Source.DisplayContent, TextPresence.PassIfPresent,
 MatchOption.Exact);
web.verifyText("myTextMatchingTest2", "FMStocks Customer Log*",
 Source.Html, TextPresence.FailIfPresent,
 MatchOption.Wildcard);
```

web.waitForObject

Waits for an object to exist before timing out.

Format

The web.waitForObject method has the following command format(s):

web.waitForObject(*path*, *timeout*);

Command Parameters

path
a String specifying the object path.

timeout
a timeout value to find an object, in milliseconds.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Waits for the object specified by its path. Timeout after 5 seconds.

```
getVariables().set("obPath", "/web:window[@index='1']" +
 "/web:document[@index='1']" +
 "/web:form[@id='loginform' or @name='loginform' or @index='0']" +
 "/web:input_submit[@name='LoginButtonADD']");
web.waitForObject("{{obPath}}", 5000);
```

web.window

Identifies a window by its recorded ID and path.

Format

The web.window method has the following command format(s):

web.window(*path*);

web.window(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMBrowser object using the specified recorded ID and path.

Example

Navigates to and waits for the window specified by its recorded ID and path.

```
web.window(2, "/web:window[@index='0' or @title='about:blank']")
   .navigate("http://testserver2.com/fmstocks");
web.window(4, "/web:window[@index='0' or @title='Stocks']")
   .waitForPage(null);
```

web.xmlDocument

Identifies an xml document by its recorded ID and path.

Format

The web.xmlDocument method has the following command format(s):

web.xmlDocument(*path*);

web.xmlDocument(*recId*, *path*);

Command Parameters

path
a String specifying the object path.

recId
the ID of a previously recorded navigation, used for comparison purposes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

a DOMDocument object using the specified recorded ID and path.

Example

Performs an action on an open xml document specified by its recorded ID and path.

```
getVariables().set("recHtml", web.xmlDocument(1,
 "/web:document[@index='0]'").getHTML());
info("The HTML string is: {{recHtml}}");
getVariables().set("recUrl", web.xmlDocument(2,
 "/web:document[@index='0]'").getRecordedURL());
info("The URL is: {{recUrl}}");
```

Part IV

Utility Modules API Reference

This Part of the OpenScript Programmer's Reference provides a complete listing and reference for the methods in the OpenScript Functional Testing Modules Application Programming Interface (API).

Each chapter in this part contains alphabetical listings and detailed command references for the methods in each class.

This Part contains the following chapters:

16

Browser Utility Module

This chapter provides a complete listing and reference for the methods in the OpenScript BrowserService Class of Browser Module Application Programming Interface (API).

16.1 BrowserService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript BrowserService API.

16.1.1 Alphabetical Command Listing

The following table lists the BrowserService API methods in alphabetical order.

Table 16–1 List of BrowserService Methods

Method	Description
browser.setBrowserType	Sets the browser type.

The following sections provide detailed reference information for each method and enum in the BrowserService Class of Browser Module Application Programming Interface.

browser.setBrowserType

Sets the browser type.

Format

The browser.setBrowserType method has the following command format(s):

browser.setBrowserType(*browserType*);

Command Parameters

browserType
BrowserType enum specifying the supported browser types. For example, BrowserType.InternetExplorer.

Throws

BrowserException
if browser type cannot be changed.

Example

Sets the browser type.

```
//Specify Browser as Mozilla Firefox.
browser.setBrowserType(BrowserType.Firefox);
//Specify Browser as Internet Explorer.
browser.setBrowserType(BrowserType.InternetExplorer);
```

17

DataTable Utility Module

This chapter provides a complete listing and reference for the methods in the OpenScript DataTableService Class of Data Table Module Application Programming Interface (API).

17.1 DataTableService ENUM Reference

The following section provides an alphabetical listing of the enums in the OpenScript DataTableService API.

17.1.1 Alphabetical Enum Listing

The following table lists the DataTableService Enums in alphabetical order.

Table 17–1 List of DataTableService Enums

Enum	Description
ExportMode	Sets the export mode for saving datatables to Excel files.

17.2 DataTableService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript DataTableService API.

17.2.1 Alphabetical Command Listing

The following table lists the DataTableService API methods in alphabetical order.

Table 17–2 List of DataTableService Methods

Method	Description
datatable.addColumn	Adds a column before the column whose index is columnAdded in the specific sheet.
datatable.addSheet	Adds a sheet in the current datatable before given sheet name without using first row as column header.
datatable.changeSheet	Changes the input sheet name to the new sheet name.
datatable.debugDump	Prints the content of current datatable into the Console.
datatable.deleteColumn	Deletes the column whose name is specified by columnName in the sheet whose name is specified by sheetName.
datatable.deleteRow	Deletes a new row before the current row.

Table 17–2 (Cont.) List of DataTableService Methods

Method	Description
datatable.deleteSheet	Deletes a sheet in the current datatable.
datatable.exportSheet	Exports a sheet in the current datatable into a new excel document.
datatable.exportSheets	Exports a sheet in the current excel document into excel document.
datatable.exportToExcel	Exports the current excel document into an excel document whose name is specified by path.
datatable.getColumn	Gets the column name in the specified sheet.
datatable.getColumnCount	Gets column count in the specified sheet.
datatable.getColumnIndex	Gets the column index by column name in the specified sheet.
datatable.getCurrentRow	Gets the current active row number which is 0 based in the current sheet.
datatable.getCurrentSheet	Gets the name of the current sheet.
datatable.getGlobalDatatable	Creates an instance of datatable in top-level script in the chain of parent scripts.
datatable.getParentDatatable	Creates an instance of datatable in parent script, if one exists and declared its own datatable service.
datatable.getRowCount	Gets the total count of the rows in the specific sheet.
datatable.getSheet	Gets the sheet name.
datatable.getSheetCount	Gets the total count of the sheets.
datatable.getValue	Gets the value of the specific cell in the specified sheet.
datatable.importAllSheets	Imports all sheets into the current data table.
datatable.importExcel	Imports a new excel document by path specifying if first row is used as header row.
datatable.importSheet	Imports a sheet into the current data table.
datatable.importSheets	Import sheet list into the current data table.
datatable.insertRow	Inserts a new row before the current row.
datatable.isFirstRowAsColumnHeader	Check to find if a sheet uses the first row as headers for columns.
datatable.setCurrentRow	Sets the specified row number to the active row in the specified sheet.
datatable.setCurrentSheet	Sets the current sheet as indicated by sheet name.
datatable.setNextRow	Sets the next row to the active row in the current sheet.
datatable.setPreviousRow	Sets the previous row to the active row in the current sheet.
datatable.setValue	Sets the value to the specific cell in the specified sheet.
datatable.updateColumn	Updates the name of the column whose name is specified by columnName in the sheet whose name is sheetName.
datatable.useFirstRowAsColumnHeader	Sets the first row of data as columns header.

The following sections provide detailed reference information for each method and enum in the DataTableService Class of Data Table Module Application Programming Interface.

datatable.addColumn

Adds a column before the column whose index is columnAdded in the specific sheet.

Format

The datatable.addColumn method has the following command format(s):

datatable.addColumn(*columnName*);

datatable.addColumn(*columnName*, *columnAdded*);

datatable.addColumn(*sheetName*, *columnName*);

datatable.addColumn(*sheetName*, *columnName*, *columnAdded*);

Command Parameters

columnName
a String specifying the new added column name.

columnAdded
is 0 based column index,the new column will be added beforethis column.

sheetName
a String specifying the sheet name.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Adds a new column after the last column in the specified sheet.

```
datatable.addColumn("Sheet1", "columnName", 1);
```

datatable.addSheet

Adds a sheet in the current datatable before given sheet name without using first row as column header.

First row of new sheet will not be used as column header.

Format

The datatable.addSheet method has the following command format(s):

datatable.addSheet(*sheetName*);

datatable.addSheet(*sheetName*, *firstRowAsColHeader*);

datatable.addSheet(*sheetName*, *aheadSheetName*);

datatable.addSheet(*sheetName*, *aheadSheetName*, *firstRowAsColHeader*);

Command Parameters

sheetName
a String specifying the name of the sheet to be added.

firstRowAsColHeader
a Boolean specifying whether to use the first row as the column name.

aheadSheetName
a String specifying the name of the sheet to add the new sheet before.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Adds a sheet in the current datatable.

```
datatable.addSheet("Sheet1", "Sheet2");
```

datatable.changeSheet

Changes the input sheet name to the new sheet name.

Format

The datatable.changeSheet method has the following command format(s):

datatable.changeSheet(*oldSheetName*, *newSheetName*);

Command Parameters

oldSheetName
a String specifying the sheet name that will be changed.

newSheetName
a String specifying new sheet name.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Changes the input sheet name to the new sheet name.

```
datatable.changeSheet("Sheet1", "mySheet1");
```

datatable.debugDump

Prints the content of current datatable into the Console.

Use this method during script debugging to dump the content of current datatable into Console. Do not use on normal playback, as it slows down execution and increases the size of the Console. After debugging just change value of dumDT to `false`.

Format

The datatable.debugDump method has the following command format(s):

datatable.debugDump(*notes*);

Command Parameters

notes
a String to identify the current datatable dump in the console.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Prints the content of current datatable into the Console.

```
run(){
boolean dumpDT = true;
...
 if(dumpDT)//later in run section
  datatable.debugDump("Test dump");
...
}
```

datatable.deleteColumn

Deletes the column whose name is specified by columnName in the sheet whose name is specified by sheetName.

Format

The datatable.deleteColumn method has the following command format(s):

datatable.deleteColumn(*columnName*);

datatable.deleteColumn(*sheetName*, *columnName*);

Command Parameters

columnName
a String specifying the deleted column name.

sheetName
a String specifying the sheet name in which the column will be deleted.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Deletes the column specified by columnName in the specified sheet.

```
datatable.deleteColumn("Sheet1", "columnName");
```

datatable.deleteRow

Deletes a new row before the current row.

Format

The datatable.deleteRow method has the following command format(s):

datatable.deleteRow(*sheetName*, *rowIndex*);

Command Parameters

sheetName
a String specifying sheet name.

rowIndex
0 based row index.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Deletes a new row before the current row.

```
datatable.deleteRow("Sheet1", 2);
```

datatable.deleteSheet

Deletes a sheet in the current datatable.

Format

The datatable.deleteSheet method has the following command format(s):

datatable.deleteSheet(*sheetName*);

Command Parameters

sheetName
a String specifying the name of the sheet to be deleted.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Deletes a sheet in the current datatable.

```
datatable.deleteSheet("Sheet2");
```

ExportMode

The ExportMode has the following values:

Table 17–3 List of ExportMode Values

Value	Description
CREATE	Create a new Excel file comprised of exported sheets only.
MERGE	Add exported sheets to existing sheets. If the existing spreadsheet has the same name as the exported sheet, then the existing sheet will be overwritten.

datatable.exportSheet

Exports a sheet in the current datatable into a new excel document.

Format

The datatable.exportSheet method has the following command format(s):

datatable.exportSheet(*path*, *sourceSheetName*, *destSheetName*);

datatable.exportSheet(*path*, *sourceSheetName*, *destSheetName*, *exportMode*);

Command Parameters

path
a String specifying the absolute path of the new excel document.

sourceSheetName
a String specifying the name of the sheet to be exported.

destSheetName
a String specifying the new name of the sheet to be exported.

exportMode
an ExportMode enum of ExportMode.CREATE or ExportMode.MERGE. CREATE means create new excel file comprised of exported sheet only. MERGE means add exported sheets to existing sheets. If the existing spreadsheet has the same name as the exported sheet, then the existing sheet will be overwritten.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Exports the current datatable into an excel document.

```
datatable.exportSheet("C:\\OracleATS\\Book1.xls", "Sheet1", "Sheet1",
 ExportMode.CREATE);
```

datatable.exportSheets

Exports a sheet in the current excel document into excel document.

Format

The datatable.exportSheets method has the following command format(s):

datatable.exportSheets(*destPath*, *sheetlist*);

datatable.exportSheets(*destPath*, *sheetlist*, *expMode*);

Command Parameters

destPath
a String specifying the absolute path of the new excel document.

sheetlist
a List specifying the name of the sheet to be exported.

expMode
an ExportMode enum of ExportMode.CREATE or ExportMode.MERGE. CREATE means create a new excel file comprised of exported sheets only. MERGE means add exported sheets to existing sheets. If the existing spreadsheethas the same name as the exported sheet, then the existing sheet will be overwritten.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Exports sheets in the current datatable document into a new excel document.

```
import java.util.List;
import java.util.ArrayList;
//[...]
List<String> sheetList = new ArrayList<String>();
sheetList.add("Sheet1");
sheetList.add("Sheet2");
datatable.exportSheets("C:\\OracleATS\\Book1.xls", sheetList,
 ExportMode.CREATE);
```

datatable.exportToExcel

Exports the current excel document into an excel document whose name is specified by path.

The method does the same job as {@code exportSheets()} for all sheets in the current datatable.

Format

The datatable.exportToExcel method has the following command format(s):

datatable.exportToExcel(*path*);

datatable.exportToExcel(*path*, *expMode*);

Command Parameters

path
a String specifying the absolute path of the new excel document.

expMode
an ExportMode enum of ExportMode.CREATE or ExportMode.MERGE. CREATE means create a new excel file comprised of exported sheets only. MERGE means add exported sheets to existing sheets. If the existing spreadsheet has the same name as the exported sheet, then the existing sheet will be overwritten.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Exports the current datatable into an excel document.

```
datatable.exportToExcel("C:\\OracleATS\\Book1.xls", ExportMode.CREATE);
```

datatable.getColumn

Gets the column name in the specified sheet.

Format

The datatable.getColumn method has the following command format(s):

datatable.getColumn(*sheetIndex*, *columnIndex*);

datatable.getColumn(*sheetName*, *columnIndex*);

Command Parameters

sheetIndex
specifies 0 based sheet index.

columnIndex
specifies 0 based column index.

sheetName
a String specifying sheet name.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the column name.

Example

Gets the column name of column index 0 within the sheet named Sheet1.

```
String getCol = datatable.getColumn("Sheet1", 0);
info("Sheet1, column index 0 = " + getCol);
```

datatable.getColumnCount

Gets column count in the specified sheet.

Format

The datatable.getColumnCount method has the following command format(s):

datatable.getColumnCount(*sheetIndex*);

datatable.getColumnCount(*sheetName*);

Command Parameters

sheetIndex
specifies 0 based sheet index.

sheetName
a String specifyingthe name of sheet.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the column count.

Example

Gets column count in the sheet with the name "Sheet1".

```
int getColCount = datatable.getColumnCount("Sheet1");
info("Sheet1 column count = " + getColCount);
```

datatable.getColumnIndex

Gets the column index by column name in the specified sheet.

Format

The datatable.getColumnIndex method has the following command format(s):

datatable.getColumnIndex(*colName*);

datatable.getColumnIndex(*sheetName*, *colName*);

Command Parameters

colName
a String specifying the name of column.

sheetName
a String specifying the name of sheet.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

zero-based index of column or -1, if there is no such column

Example

Gets column index of the column with the name "Data4" in Sheet1.

```
int getColIndex = datatable.getColumnIndex("Sheet1", "Data4");
info("Sheet1, Data4 column index = " + getColIndex);
```

datatable.getCurrentRow

Gets the current active row number which is 0 based in the current sheet.

Format

The datatable.getCurrentRow method has the following command format(s):

datatable.getCurrentRow( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

row index of current sheet.

Example

Gets the current active row number.

```
int curRow = datatable.getCurrentRow();
info("Current row is: " + curRow);
```

datatable.getCurrentSheet

Gets the name of the current sheet.

Format

The datatable.getCurrentSheet method has the following command format(s):

datatable.getCurrentSheet( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the name of current sheet.

Example

Gets the name of the current sheet.

```
String curSheet = datatable.getCurrentSheet();
info("Current sheet is: " + curSheet);
```

datatable.getGlobalDatatable

Creates an instance of datatable in top-level script in the chain of parent scripts.

Each script declares its own datatable service. Can be used to manipulate top-level datatable from the child script.

Format

The datatable.getGlobalDatatable method has the following command format(s):

datatable.getGlobalDatatable( );

Returns

instance of global datatable, if any,or this datatable if this script runs alone.

Example

Gets the global datatable.

```
DataTableService globalDatatable = datatable.getGlobalDatatable();
globalDatatable.setValue(0, "A", 15);
globalDatatable.save();
```

datatable.getParentDatatable

Creates an instance of datatable in parent script, if one exists and declared its own datatable service.

Can be used to manipulate parent datatable from the child script.

Format

The datatable.getParentDatatable method has the following command format(s):

datatable.getParentDatatable( );

Returns

instance of parent datatable, if any,or `null`, if this script runs alone or parent script doesn't declared datatable service.

Example

Gets the parent datatable.

```
DataTableService parDatatable = datatable.getParentDatatable();
if(parDatatable != null){
   parDatatable.setValue(0, "A", 15);
   parDatatable.save();
 }
```

datatable.getRowCount

Gets the total count of the rows in the specific sheet.

Format

The datatable.getRowCount method has the following command format(s):

datatable.getRowCount( );

datatable.getRowCount(*sheetName*);

Command Parameters

sheetName
a String specifying the name of the sheet.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the total count of the rows.

Example

Gets the total count of the rows in the specific sheet.

```
int rowCount = datatable.getRowCount("Sheet1");
info("Row count is: " + rowCount);
```

datatable.getSheet

Gets the sheet name.

Format

The datatable.getSheet method has the following command format(s):

datatable.getSheet(*index*);

Command Parameters

index
specifies the 0 based sheet index.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the name of the sheet.

Example

Gets the name of the sheet with index value 0.

```
String getSheet= datatable.getSheet(0);
info("Sheet1 name = " + getSheet);
```

datatable.getSheetCount

Gets the total count of the sheets.

Format

The datatable.getSheetCount method has the following command format(s):

datatable.getSheetCount( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the total count of sheets.

Example

Gets the name of the current sheet.

```
int sheetCount = datatable.getSheetCount();
info("Sheet count: " + sheetCount);
```

datatable.getValue

Gets the value of the specific cell in the specified sheet.

Format

The datatable.getValue method has the following command format(s):

datatable.getValue(*row*, *column*);

datatable.getValue(*sheetName*, *row*, *column*);

Command Parameters

row
specifies the 0 based row index.

column
a String specifying the column name.

sheetName
a String specifying the sheet name.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

Object as a cell value.

Example

Gets the value of row 1, column A in the current sheet.

```
Object cellValue1 = datatable.getValue("Sheet1", 0, "A");
info("cell value = " + cellValue1.toString());
```

datatable.importAllSheets

Imports all sheets into the current data table.

If a sheet with the same name as the imported sheet exists in the data table, then behavior depends on overwrite parameter. If overwrite is `true`, then existing sheet will be overwritten. If overwrite is `false`, then the existing sheet will be preserved, but the imported sheet name will be appended with a numerical value from 1 to 50000.

Format

The datatable.importAllSheets method has the following command format(s):

datatable.importAllSheets(*path*, *overwrite*, *firstRowAsColHeader*);

Command Parameters

path
a String specifying the absolute path of the excel document that contains the imported sheets.

overwrite
a Boolean indicating whether to overwrite if same sheet name exists.

firstRowAsColHeader
a Boolean specifying how to treat the first row of imported sheets.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Imports all sheets into the current data table.

```
datatable.importAllSheets("C:\\OracleATS\\Book1.xls", true, false);
```

datatable.importExcel

Imports a new excel document by path specifying if first row is used as header row.

Differs from {@code improtAllSheets} in that existing sheets won't be preserved.

Format

The datatable.importExcel method has the following command format(s):

datatable.importExcel(*path*);

datatable.importExcel(*path*, *firstRowAsColHeader*);

Command Parameters

path
a String specifying the absolute file path of the new document.

firstRowAsColHeader
a Boolean specifying how to treat first row of imported sheets.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Imports a new excel document by path specifying if first row is used as header row.

```
datatable.importExcel("C:\\OracleATS\\Book1.xls", true);
```

datatable.importSheet

Imports a sheet into the current data table.

Existing sheets will be preserved. If the name of the existing sheet matches the name of the imported destination sheet, then name of destination sheet will be appended by a numbered suffix.

Example:

Destination sheet name is Sheet1, but data table already has sheet named Sheet1. After importing existing sheet, Sheet1 will be preserved, and imported sheet will get name Sheet11, unless Sheet11 already exists in data table. After import, the first row of the imported sheet will not be considered as a column header.

Format

The datatable.importSheet method has the following command format(s):

datatable.importSheet(*path*, *sourceSheet*, *destSheet*);

datatable.importSheet(*path*, *sourceSheet*, *destSheet*, *firstRowAsColHeader*);

Command Parameters

path
a String that specifies the absolute path of the Excel document that contains the imported sheet.

sourceSheet
a String specifying the imported sheet name in excel document.

destSheet
a String specifying the new sheet name in the current data table.

firstRowAsColHeader
a boolean specifying how to use first row of the imported sheet.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Import a sheet into the current data table specifying if first row is used as header row.

```
datatable.importSheet("C:\\OracleATS\\Book1.xls", "Sheet1", "Sheet1", true);
```

datatable.importSheets

Import sheet list into the current data table.

If a sheet with the same name as imported sheet exists in the data table, then behavior depends on the overwrite parameter. If overwrite is `true`, then existing sheet will be overwritten. If overwrite is `false`, then existing sheet will be preserved, but the imported sheet name will be appended with a numerical value from 1 to 50000.

Format

The datatable.importSheets method has the following command format(s):

datatable.importSheets(*sourcePath*, *sheetList*, *overwirte*);

datatable.importSheets(*sourcePath*, *sheetList*, *bOverwirte*, *bUsingFirstRow*);

Command Parameters

sourcePath
a String specifying the absolute path of the excel file that contains the imported sheets.

sheetList
a List specifying the sheet(s) that will be imported.

overwirte
a Boolean specifying whether to overwrite if the same sheet name exists.

bOverwirte
a Boolean specifying whether to overwrite if the same sheet name exists.

bUsingFirstRow
a Boolean specifying whether to use the first row as the column name.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Import sheet list into the current data table.

```
import java.util.List;
import java.util.ArrayList;
//[...]
List<String> sheetList = new ArrayList<String>();
sheetList.add("Sheet1");
sheetList.add("Sheet2");
datatable.importSheets("C:\\OracleATS\\Book1.xls", sheetList, true, false);
```

datatable.insertRow

Inserts a new row before the current row.

Format

The datatable.insertRow method has the following command format(s):

datatable.insertRow(*rowIndex*);

datatable.insertRow(*sheetName*, *rowIndex*);

Command Parameters

sheetName
a String specifying sheet name.

rowIndex
0 based row index.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Inserts a new row before row index 0 in Sheet1.

```
datatable.insertRow("Sheet1", 0);
```

datatable.isFirstRowAsColumnHeader

Check to find if a sheet uses the first row as headers for columns.

Format

The datatable.isFirstRowAsColumnHeader method has the following command format(s):

datatable.isFirstRowAsColumnHeader(*sheetName*);

Command Parameters

sheetName
a String specifying the name of a sheet in the data table.

Throws

AbstractScriptException
if Sheet with sheetName does not exist.

Returns

`true`, if the sheet set to use the first row as headers for columns, otherwise `false`.

Example

Checks if the first row of data is used as headers for columns.

```
Boolean isFirstHeader = datatable.isFirstRowAsColumnHeader("Sheet1");
info("First Row is Header = " + isFirstHeader);
```

datatable.setCurrentRow

Sets the specified row number to the active row in the specified sheet.

Format

The datatable.setCurrentRow method has the following command format(s):

datatable.setCurrentRow(*row*);

datatable.setCurrentRow(*sheetName*, *row*);

Command Parameters

row
0 based row index.

sheetName
a String specifying the name of the sheet.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Sets the a row to active in the current sheet.

```
datatable.setCurrentRow("Sheet1", 1);
```

datatable.setCurrentSheet

Sets the current sheet as indicated by sheet name.

Many data table methods are used against the current sheet in the table.

Format

The datatable.setCurrentSheet method has the following command format(s):

datatable.setCurrentSheet(*sheetName*);

Command Parameters

sheetName
a String specifying the name of sheet to set as the current sheet.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Sets the current sheet as indicated by sheet name.

```
datatable.setCurrentSheet("Sheet1");
```

datatable.setNextRow

Sets the next row to the active row in the current sheet.

If the current active row is at the end, set it to the beginning row.

Format

The datatable.setNextRow method has the following command format(s):

datatable.setNextRow( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Sets the next row to the active in the current sheet.

```
datatable.setNextRow();
```

datatable.setPreviousRow

Sets the previous row to the active row in the current sheet.

If the current active row is at the beginning, set it to the end row.

Format

The datatable.setPreviousRow method has the following command format(s):

datatable.setPreviousRow( );

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Sets the previous row to the active row in the current sheet.

```
datatable.setPreviousRow();
```

datatable.setValue

Sets the value to the specific cell in the specified sheet.

If the value begins with an equals sign (=), it will be considered a formula. For example "=sum(a1,a2)".

Format

The datatable.setValue method has the following command format(s):

datatable.setValue(*row*, *column*, *value*);

datatable.setValue(*row*, *column*, *value*);

datatable.setValue(*row*, *column*, *value*);

datatable.setValue(*sheetName*, *row*, *column*, *value*);

datatable.setValue(*sheetName*, *row*, *column*, *value*);

datatable.setValue(*sheetName*, *row*, *column*, *value*);

Command Parameters

row
specifies the 0 based row index.

column
a String specifying the column name.

value
the value to set. There are `setValue` methods for String value, Boolean value, and Double value.

sheetName
a String specifying the sheet name.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Sets the value of row 1, column A in Sheet1 to "=sum(a1,a2)".

```
datatable.setValue("Sheet1", 0, "A", "=sum(a1,a2)");
```

datatable.updateColumn

Updates the name of the column whose name is specified by columnName in the sheet whose name is sheetName.

Format

The datatable.updateColumn method has the following command format(s):

datatable.updateColumn(*col*, *value*);

datatable.updateColumn(*columnName*, *value*);

datatable.updateColumn(*sheetName*, *col*, *value*);

datatable.updateColumn(*sheetIndex*, *col*, *value*);

datatable.updateColumn(*sheetName*, *columnName*, *value*);

Command Parameters

col
0 based column index.

value
a String specifying the new column name.

columnName
a String specifying the column name to be updated..

sheetName
a String specifying sheet name.

sheetIndex
0 based sheet index.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

>Updates the name of the column "A" in Sheet1.

```
datatable.updateColumn("Sheet1", "A", "Data1");
```

datatable.useFirstRowAsColumnHeader

Sets the first row of data as columns header.

This setting can completely change the sheet behavior, if it was not previously set by one of import methods or by the import action in the datatable view.

When the first row is set as column headers, the first row of data in the in datatable sheet is used as a header for columns. The number of rows in the sheet is decremented, as all rows starting from the second row will be moved up. If the first row has any cell values that are not assigned yet, then a default Excel column name will be used for the column [A,B,..,Z,AA,AB,...,ZZ]. For example, before the first column policy is set to use first row as headers, the sheet with 2 rows:

```
col1, col2, col3
```

```
a21, a22, a23
```

was shown in Excel as:

```
headers: A, B, C
```

```
row1: col1, col2, col3
```

```
row2: a21, a22, a33
```

it will be shown the same way in the datatable sheet.

After the first column policy is set to use first row as headers, you will see datatable data in Excel the same as before, but in the datatatable view it shows as:

```
headers: col1, col2, col3
```

```
row1: a21, a22, a23
```

Format

The datatable.useFirstRowAsColumnHeader method has the following command format(s):

datatable.useFirstRowAsColumnHeader(*sheetName*);

Command Parameters

sheetName
is a name for sheet to set the desired behavior

Throws

AbstractScriptException
if the sheet with sheetName doesn't exist.

Example

Checks if the first row of data is used as headers for columns.

```
datatable.useFirstRowAsColumnHeader("Sheet1");
```

18

SharedData Module

This chapter provides a complete listing and reference for the methods in the OpenScript SharedDataService Class of SharedData Module Application Programming Interface (API).

18.1 SharedDataService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript SharedDataService API.

18.1.1 Alphabetical Command Listing

The following table lists the SharedDataService API methods in alphabetical order.

Table 18–1 List of SharedDataService Methods

Method	Description
sharedData.clearMap	Removes all mappings from a hash map.
sharedData.clearQueue	Removes all of the elements from a queue.
sharedData.createMap	Creates a hash map with the name and the life time.
sharedData.createQueue	Creates a queue with the specified name and life time.
sharedData.destroyMap	Removes all mappings and their data listeners from a hash map.
sharedData.destroyQueue	Removes all of the elements and their data listeners from a queue.
sharedData.getFromMap	Returns the value to which the specified key is mapped in a hash map, or blocked until the default timeout if the map contains no mapping for this key.
sharedData.getKeysOfMap	Returns all of the keys contained in a hash map.
sharedData.getLengthOfQueue	Get the length of the queue.
sharedData.offerFirst	Inserts the specified element at the front of the specified queue.
sharedData.offerLast	Inserts the specified element at the end of the specified queue.
sharedData.peekFirst	Retrieves but does not remove the first element of the specified queue, or blocked until the default timeout if the queue is empty.
sharedData.peekLast	Retrieves but does not remove last element of the specified queue, or blocked until the default timeout if the queue is empty.
sharedData.pollFirst	Retrieves and removes the first element of the specified queue, or blocked until the default timeout if the queue is empty.

Table 18–1 (Cont.) List of SharedDataService Methods

Method	Description
sharedData.pollLast	Retrieves and removes the last element of the specific queue, or blocked until the default timeout if the queue is empty.
sharedData.putToMap	Associates the specified value with the specified key in a session hash map.
sharedData.removeFromMap	Removes the mapping for this key from a map if present.
sharedData.setConnectionParameters	Sets the connection parameters for the shared data service.
sharedData.waitFor	Waits for the specific value to be added to the queue until the default timeout.

The following sections provide detailed reference information for each method and enum in the SharedDataService Class of SharedData Module Application Programming Interface.

sharedData.clearMap

Removes all mappings from a hash map.

Format

The sharedData.clearMap method has the following command format(s):

sharedData.clearMap(*name*);

Command Parameters

name
a String specifying the name of the map.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Clears the hash map named "mapA".

```
sharedData.clearMap("mapA");
```

sharedData.clearQueue

Removes all of the elements from a queue.

Format

The sharedData.clearQueue method has the following command format(s):

sharedData.clearQueue(*name*);

Command Parameters

name
a String specifying the name of the queue.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Clears the Queue named "queueA".

```
sharedData.clearQueue("queueA");
```

sharedData.createMap

Creates a hash map with the name and the life time.

If the hash map already exists, update its life time with the new life time value.

Format

The sharedData.createMap method has the following command format(s):

sharedData.createMap(*name*, *lifeTime*);

Command Parameters

name
a String specifying the name of the hash map to be created.

lifeTime
the life time the hash map will live. Its unit is minutes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Creates the hash map named "mapA" with a life time of 10 minutes.

```
sharedData.createMap("mapA", 10);
```

sharedData.createQueue

Creates a queue with the specified name and life time.

If the queue already exists, update its life time with the new life time value.

Format

The sharedData.createQueue method has the following command format(s):

sharedData.createQueue(*name*, *lifeTime*);

Command Parameters

name
a String specifying the name of the queue to be created.

lifeTime
the life time the queue will live. Its unit is minutes.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Creates the Queue named "queueA" with a life time of 10 minutes.

```
sharedData.createQueue("queueA", 10);
```

sharedData.destroyMap

Removes all mappings and their data listeners from a hash map.

Format

The sharedData.destroyMap method has the following command format(s):

sharedData.destroyMap(*name*);

Command Parameters

name
a String specifying the name of the map.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Destroys the hash map named "mapA".

```
sharedData.destroyMap("mapA");
```

sharedData.destroyQueue

Removes all of the elements and their data listeners from a queue.

Format

The sharedData.destroyQueue method has the following command format(s):

sharedData.destroyQueue(*name*);

Command Parameters

name
a String specifying the name of the queue.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Destroys the Queue named "queueA".

```
sharedData.destroyQueue("queueA");
```

sharedData.getFromMap

Returns the value to which the specified key is mapped in a hash map, or blocked until the default timeout if the map contains no mapping for this key.

The default "timeout" is set in the playback settings which has a default value of 20 seconds.

Format

The sharedData.getFromMap method has the following command format(s):

sharedData.getFromMap(*name*, *key*);

sharedData.getFromMap(*name*, *key*, *timeout*);

Command Parameters

name
a String specifying the name of the hash map

key
a String specifying the key with which the specified value is to be associated.

timeout
a Long specifying the maximum time of getting blocked. The unit is milliseconds.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

the value to which the specified key is mapped, or null if the map contains no mapping for this key.

Example

Returns the value of "key1" in the hash map named "mapA". Timeout after 5 seconds.

```
String actualValue = (String) sharedData.getFromMap("mapA", "key1", 5000);
```

sharedData.getKeysOfMap

Returns all of the keys contained in a hash map.

Format

The sharedData.getKeysOfMap method has the following command format(s):

sharedData.getKeysOfMap(*name*);

Command Parameters

name
a String specifying the name of the hash map.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

all of the keys contained in a hash map.

Example

Returns the values of the keys in the hash map named "mapA".

```
String [] lsKey = sharedData.getKeysOfMap("mapA");
for (int i=0; i\<lsKey.length; i++) {
 String key = "key" + i;
 String value = lsKey[i];
 info(key + " is " + value);
}
```

sharedData.getLengthOfQueue

Get the length of the queue.

The "timeout" is set in the playback settings whose default value is 20 seconds.

Format

The sharedData.getLengthOfQueue method has the following command format(s):

sharedData.getLengthOfQueue(*name*);

sharedData.getLengthOfQueue(*name*, *timeout*);

Command Parameters

name
a String specifying the name of the queue.

timeout
a long specifying the maximum time of getting blocked. The unit is milliseconds.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

The length of the queue.

Example

Returns the length of the queue named "queueA". Timeour after 5 seconds.

```
int queueLength = sharedData.getLengthOfQueue("queueA", 5000);
info("Length of queueA is " + queueLength);
```

sharedData.offerFirst

Inserts the specified element at the front of the specified queue.

If the queue does not exist, a new one with the name will be created with a default life time of 20 minutes.

Format

The sharedData.offerFirst method has the following command format(s):

sharedData.offerFirst(*name*, *value*);

Command Parameters

name
a String specifying the name of the queue that the value will be put into.

value
a Serializable value to be put into a queue.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Inserts the different data type elements at the front of "queueA".

```
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);
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);
```

sharedData.offerLast

Inserts the specified element at the end of the specified queue.

If the queue does not exist, a new one with the name will be created with a default life time of 20 minutes.

Format

The sharedData.offerLast method has the following command format(s):

sharedData.offerLast(*name*, *value*);

Command Parameters

name
a String specifying the name of the queue that the value will be put into.

value
a Serializable value to be put into the queue.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Inserts the different data type elements at the end of "queueA".

```
info("parameter type - String");
sharedData.offerLast("queueA", "value");
info("parameter type - list of Strings");
ArrayList<String> listOfStr = new ArrayList<String>();
listOfStr.add(0, "val1");
listOfStr.add(1, "val2");
sharedData.offerLast("queueA", listOfStr);
info("parameter type - boolean");
sharedData.offerLast("queueA", true);
info("parameter type - int");
sharedData.offerLast("queueA", 10);
info("parameter type - double");
sharedData.offerLast("queueA", 10.5);
info("parameter type - long");
sharedData.offerLast("queueA", 100);
```

sharedData.peekFirst

Retrieves but does not remove the first element of the specified queue, or blocked until the default timeout if the queue is empty.

The "timeout" is set in the playback settings which has a default value of 20 seconds.

Format

The sharedData.peekFirst method has the following command format(s):

sharedData.peekFirst(*name*);

sharedData.peekFirst(*name*, *timeout*);

Command Parameters

name
a String specifying the name of the queue.

timeout
a Long specifying the maximum time of getting blocked. The unit is milliseconds.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

null if the queue is empty or the queue does not exist.

Example

Retrieves but does not remove the first element of "queueA". Timeout after 5 seconds.

```
String queueValue = (String) sharedData.peekFirst("queueA", 5000);
```

sharedData.peekLast

Retrieves but does not remove last element of the specified queue, or blocked until the default timeout if the queue is empty.

The "timeout" is set in the playback settings which has a default value of 20 seconds.

Format

The sharedData.peekLast method has the following command format(s):

sharedData.peekLast(*name*);

sharedData.peekLast(*name*, *timeout*);

Command Parameters

name
a String specifying the name of the queue.

timeout
a Long specifying the maximum time of getting blocked. The unit is milliseconds.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

null if the queue is empty or the queue does not exist.

Example

Retrieves but does not remove the last element of "queueA". Timeout after 5 seconds.

```
String queueValue = (String) sharedData.peekLast("queueA", 5000);
```

sharedData.pollFirst

Retrieves and removes the first element of the specified queue, or blocked until the default timeout if the queue is empty.

The "timeout" is set in the playback settings which has a default value of 20 seconds.

Format

The sharedData.pollFirst method has the following command format(s):

sharedData.pollFirst(*name*);

sharedData.pollFirst(*name*, *timeout*);

Command Parameters

name
a String specifying the name of the queue.

timeout
a Long specifying the maximum time of getting blocked. The unit is milliseconds.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

null if the queue is empty or the queue does not exist.

Example

Retrieves and removes the first element of "queueA". Timeout after 5 seconds.

```
String pollValue = (String) sharedData.pollFirst("queueA", 5000);
```

sharedData.pollLast

Retrieves and removes the last element of the specific queue, or blocked until the default timeout if the queue is empty.

The "timeout" is set in the playback settings which has a default value of 20 seconds.

Format

The sharedData.pollLast method has the following command format(s):

sharedData.pollLast(*name*);

sharedData.pollLast(*name*, *timeout*);

Command Parameters

name
a String specifying the name of the queue.

timeout
a Long specifying the maximum time of getting blocked. The unit is milliseconds.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

null if the queue is empty or the queue does not exist.

Example

Retrieves and removes the last element of "queueA". Timeout after 5 seconds.

```
String pollValue = (String) sharedData.pollLast("queueA", 5000);
```

sharedData.putToMap

Associates the specified value with the specified key in a session hash map.

If the map previously contained a mapping for this key, the old value is replaced. If the hash map does not exist, a new one with the name will be created with a default life time of 20 minutes.

Format

The sharedData.putToMap method has the following command format(s):

sharedData.putToMap(*name*, *key*, *value*);

Command Parameters

name
a String specifying the name of the map

key
a String specifying the key with which the specified value is to be associated.

value
a Serializable value to be associated with the specified key.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Associates the value with the key in a session hash map.

```
info("parameter type - String");
sharedData.putToMap("mapA", "key", "value");
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);
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);
```

sharedData.removeFromMap

Removes the mapping for this key from a map if present.

Format

The sharedData.removeFromMap method has the following command format(s):

sharedData.removeFromMap(*name*, *key*);

sharedData.removeFromMap(*name*, *key*, *timeout*);

Command Parameters

name
a String specifying the name of the hash map.

key
a String specifying the key with which the specified value is to be associated.

timeout
a Long specifying the maximum time of getting blocked. The unit is milliseconds.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

previous value associated with the specified key.

Example

Returns the value of "key1" in the hash map named "mapA". Timeout after 5 seconds.

```
String removeValue1 = (String) sharedData.removeFromMap("mapA", "key1", 5000);
```

sharedData.setConnectionParameters

Sets the connection parameters for the shared data service.

The connection parameters can be specified using the Add option on the Script menu.

Format

The sharedData.setConnectionParameters method has the following command format(s):

sharedData.setConnectionParameters(*address*, *userName*, *password*);

Command Parameters

address
a String specifying the address of the Oracle Load Testing server to use for the Shared Data Service.

userName
a String specifying the username to use for authentication.

password
a String specifying the password to use for authentication.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Sets the connection parameters.

```
sharedData.setConnectionParameters("http://testserver2", "username",
   decrypt("vGXUWvDW/F7E6OSYUjRmsQ=="));
```

sharedData.waitFor

Waits for the specific value to be added to the queue until the default timeout.

The "timeout" is set in the playback settings which has a default value of 20 seconds.

Format

The sharedData.waitFor method has the following command format(s):

sharedData.waitFor(*queueName*, *desiredValue*);

sharedData.waitFor(*queueName*, *desiredValue*, *timeout*);

Command Parameters

queueName
a String specifying the name of the queue.

desiredValue
a Serializable value that is expected to be added to the queue.

timeout
a Long specifying the maximum time of waiting. The unit is milliseconds.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Returns

`true` if the value is in the queue, otherwise `false`.

Example

Waits for the specific value to be added to the queue. Timeout after 5 seconds.

```
info("parameter type - String");
Boolean isFound1 = sharedData.waitFor("queueA", "key1", 5000);
 if (!isFound1)
   warn("Can not find item 'Key1' in queueA");
 else
   info("Found item 'key1' in queueA");
info("parameter type - list of Strings");
ArrayList<String> listOfStr = new ArrayList<String>();
listOfStr.add(0, "val1");
listOfStr.add(1, "val2");
Boolean isFound2 = sharedData.waitFor("queueA", listOfStr, 5000);
 if (!isFound2)
   warn("Can not find item 'listOfStr' in queueA");
 else
   info("Found item of queue item 'listOfStr' in queueA");
info("parameter type - boolean");
Boolean isFound3 = sharedData.waitFor("queueA", true, 5000);
 if (!isFound3)
   warn("Can not find item 'true' in queueA");
```

```
 else
   info("Found item 'true' in queueA");
info("parameter type - int");
Boolean isFound4 = sharedData.waitFor("queueA", 10, 5000);
 if (!isFound4)
   warn("Can not find item '10' in queueA");
 else
   info("Found item '10' in queueA");
info("parameter type - double");
Boolean isFound5 = sharedData.waitFor("queueA", 10.5, 5000);
 if (!isFound5)
   warn("Can not find item '10.5' in queueA");
 else
   info("Found item '10.5' in queueA");
info("parameter type - long");
Boolean isFound6 = sharedData.waitFor("queueA", 100, 5000);
 if (!isFound6)
   warn("Can not find item '100' in queueA");
 else
   info("Found item '100' in queueA");
```

19

Utilities Module

This chapter provides a complete listing and reference for the methods in the OpenScript UtilitiesService Class of Utilitis Module Application Programming Interface (API).

19.1 UtilitiesService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript UtilitiesService API.

19.1.1 Alphabetical Command Listing

The following table lists the UtilitiesService API methods in alphabetical order.

Table 19–1 List of UtilitiesService Methods

Method	Description
utilities.getFileService	Gets a file service instance for this Utilities service.
utilities.getSQLService	Gets a SQL service instance for this Utilities service.
utilities.loadCSV	Load the given comma-separated-value file as InputStream into a `Table` object.
utilities.loadXML	Load a specified XML file as InputStream into an XML tree.
utilities.loadXMLContent	Load a specified XML contents into an XML tree.
utilities.parameters	Convenience method to create a list of parameters for SQL a query.
utilities.saveCSV	Saves a table to a CSV file format.

The following sections provide detailed reference information for each method and enum in the UtilitiesService Class of Utilitis Module Application Programming Interface.

utilities.getFileService

Gets a file service instance for this Utilities service. The FileService provides methods for working with the file system in general.

Format

The utilities.getFileService method has the following command format(s):

utilities.getFileService( );

Returns

the File service instance.

Example

Creates a new file and appends the text strings to the 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);
```

utilities.getSQLService

Gets a SQL service instance for this Utilities service. SQLService provides methods for working with SQL databases.

Format

The utilities.getSQLService method has the following command format(s):

utilities.getSQLService( );

Returns

the SQL service instance.

Example

Create a database connection, execute SQL statements and query database, then disconnect the database connecton.

```
//define database
utilities.getSQLService().define("myOracleDB",
 "oracle.jdbc.driver.OracleDriver",
 "jdbc:oracle:thin:@mySystem:1521:mySystem", "username",
 decrypt("ZSL2IzF3WpLh8ydBZYDV3Q=="));
//connect to database
utilities.getSQLService().connect("myOracleDB");
//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("myOracleDB", 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("myOracleDB", query);
//query SQL statement
query = "Select * from Employee";
Table table = utilities.getSQLService().query("myOracleDB", 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("myOracleDB");
```

utilities.loadCSV

Load the given comma-separated-value file as InputStream into a `Table` object.

Format

The utilities.loadCSV method has the following command format(s):

utilities.loadCSV(*filePath*);

utilities.loadCSV(*filePath*);

utilities.loadCSV(*url*);

utilities.loadCSV(*fileInput*);

utilities.loadCSV(*repository*, *pathRelToRepository*);

Command Parameters

filePath
Full path to a file containing CSV-formatted data.

repository
a String specifying the name of repository containing CSV-formatted data file. May contain {{ }} syntax for transforming.

pathRelToRepository
String specifying the path to the CSV-formatted data file within the named repository.May contain {{ }} syntax for transforming. Must not be `null`.Any leading file separator on the path, such as `/` or `\`, is ignored.

url

fileInput
a FileInputStream.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

CSVException
on any exception while loading and reading the CSV file.

Example

Loads a CSV file into a Table object and prints the contents to the Results view.

```
String filePath = "c:\\fmstocks_data.csv";
InputStream in = null;
in = new FileInputStream(new File(filePath));
Table table = utilities.loadCSV(in);
//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);
}
```

utilities.loadXML

Load a specified XML file as InputStream into an XML tree.

Format

The utilities.loadXML method has the following command format(s):

utilities.loadXML(*filePath*);

utilities.loadXML(*filePath*);

utilities.loadXML(*url*);

utilities.loadXML(*fileInput*);

utilities.loadXML(*repository*, *pathRelToRepository*);

Command Parameters

filePath
a String specifying the file path to be loaded. May contain {{ }} syntax for transforming.

repository
a String specifying the name of repository containing XML file to load. May contain {{ }} syntax for transforming.

pathRelToRepository
String specifying the path to the XML file within the named repository.May contain {{ }} syntax for transforming. Must not be `null`.Any leading file separator on the path, such as `/` or `\`, is ignored.

url
a URL object specifying the URL.

fileInput
a FileInputStream.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

FileNotFoundException
if the file is not found.

XMLServiceException
if an error occurs loading the XML.

Returns

the root XML element for the given XML or null if XML file doesn't exist.

Example

Loads an XML file into a XML object and prints the contents to the Results view.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
//[...]
//load XML file by InputStream
String filePath = "c:\\myxmlfile.xml";
InputStream in = null;
in = new FileInputStream(new File(filePath));
XML xml = utilities.loadXML(in);
info(xml.toString());
info("*** getChildren() ***");
XML [] children = xml.getChildren();
String lsChild = "";
for (int i=0; i<children.length; i++) {
 if (i != 0)
   lsChild += ", ";
 lsChild += children[i].getTagName() + " ";
}
if (lsChild.equals(""))
 info("No child present of xml element " + xml.getTagName());
else
 info("List of all child elements of xml element " +
   xml.getTagName() + ": " + lsChild);
for (int i=0; i<children.length; i++) {
 XML child = children[i];
// print tag name
String tagName = child.getTagName();
info("Tag Name: " + tagName);
//print attributes
String [] allAttr = child.getAttributeNames();
if (allAttr != null) {
 for (int j=0; j<allAttr.length; j++) {
   info("Attribute Name: " + allAttr[j]);
   info("Attribute Value: " + child.getAttribute(allAttr[j]));
  }
 }
else
 info("No attribute");
// print values
if (child.getValue() != null) {
   info("Value: " + child.getValue());
 }
// print parent
info("Parent: \n" + child.getParent());
}
```

See Also

oracle.oats.scripting.modules.utilities.api.xml.XML

utilities.loadXMLContent

Load a specified XML contents into an XML tree.

Format

The utilities.loadXMLContent method has the following command format(s):

utilities.loadXMLContent(*contents*);

Command Parameters

contents
The XML contents to be loaded.May contain {{ }} syntax for transforming.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

XMLServiceException
if an error occurs loading the XML.

Example

Loads an XML string into an XML object and prints the contents to the Results view.

```
//load XML file by file name string.
String contents = "<GroceryStore><StoreName>Fourth Coffee</StoreName>" +
  "<Departments><Department Name=\"Breads\">" +
  "<Item ID=\"B2\" Type=\"Muffin\"><Name>Blueberry Muffin</Name>" +
  "<Price>3.99</Price><New/></Item>" +
  "</Department></Departments></GroceryStore>";
XML xml = utilities.loadXMLContent(contents);
info(xml.toString());
```

See Also

oracle.oats.scripting.modules.utilities.api.xml.XML

utilities.parameters

Convenience method to create a list of parameters for SQL a query.

Format

The utilities.parameters method has the following command format(s):

utilities.parameters(*obj*, *param*);

Command Parameters

obj
a List<Object> of parameters for SQL a query.

param
Optional list of parameters for the SQL query List<Object>.

Returns

a new List<String>

Example

Creates a database connection, execute SQL statements and query database with parameters object, then disconnect the database connecton.

```
//define database
utilities.getSQLService().define("myOracleDB",
 "oracle.jdbc.driver.OracleDriver",
 "jdbc:oracle:thin:@mySystem:1521:mySystem", "username",
 decrypt("ZSL2IzF3WpLh8ydBZYDV3Q=="));
//connect to database
utilities.getSQLService().connect("myOracleDB");
//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("myOracleDB", query);
//execute update SQL statement
query = "Update Employee SET LastName = 'Davis'
   WHERE ID = ? and LastName = ?";
info("Query: " + query);
utilities.getSQLService().executeUpdate("myOracleDB", query,
   utilities.parameters(101, "Smith"));
//query SQL statement
query = "Select * from Employee";
Table table = utilities.getSQLService().query("myOracleDB", 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("myOracleDB");
```

utilities.saveCSV

Saves a table to a CSV file format.

Format

The utilities.saveCSV method has the following command format(s):

utilities.saveCSV(*table*, *csvFilename*, *overwrite*);

utilities.saveCSV(*table*, *repository*, *pathRelToRepository*, *overwrite*);

Command Parameters

table
a `Table` object.

csvFilename
a String specifying the full path and name of a file where the `Table` is saved. May contain {{ }} syntax for transforming.

overwrite
a Boolean that specifies whether or not to overwrite an existing file.

repository
a String specifying the name of repository where the `Table` is to be saved. May contain {{ }} syntax for transforming. Must not be `null`.

pathRelToRepository
String specifying the path to the file within the named repository.May contain {{ }} syntax for transforming. Must not be `null`.Any leading file separator on the path, such as / or \, is ignored.

Throws

AbstractScriptException
on any error when saving.

IOException
file input/oputput exception.

Example

Saves a CSV Table object to a new file in overwrite mode

```
String repository = "Default";
String pathRelToRepository = "Data/fmstocks_data.csv";
String newCSVFile = "new_fmstocks_data.csv";
boolean overwrite = true;
Table table = utilities.loadCSV(repository, pathRelToRepository);
utilities.saveCSV(table, repository, pathRelToRepository, overwrite);
```

20

Web Services Module

This chapter provides a complete listing and reference for the methods in the OpenScript WSService Class of WebService Module Application Programming Interface (API).

20.1 WSService ENUM Reference

The following section provides an alphabetical listing of the enums in the OpenScript WSService API.

20.1.1 Alphabetical Enum Listing

The following table lists the WSService Enums in alphabetical order.

Table 20–1 List of WSService Enums

Enum	Description
AttachmentMechanism	Specifies the attachement Transfer type.

20.2 WSService API Reference

The following section provides an alphabetical listing of the methods in the OpenScript WSService API.

20.2.1 Alphabetical Command Listing

The following table lists the WSService API methods in alphabetical order.

Table 20–2 List of WSService Methods

Method	Description
ws.addSecurityAttachments	Add Security Attachment property settings to current URL.
ws.attachment	Convenience method to create a new Attachment.
ws.attachments	Convenience method to create a new Attachment[].
ws.post	Request a web service using an HTTP POST method with requestSoap string, headers, URL encoding, and character sets.
ws.security	Convenience method to create a new WSSecurity object with a specified timeout value.
ws.solveXpath	Extract values from the browser's last retrieved contents using XPath notation.

The following sections provide detailed reference information for each method and enum in the WSService Class of WebService Module Application Programming Interface.

ws.addSecurityAttachments

Add Security Attachment property settings to current URL.

Format

The ws.addSecurityAttachments method has the following command format(s):

ws.addSecurityAttachments(*url*, *security*, *attachments*);

Command Parameters

url
a String specifying the URL to request.

security
a WSSecurity object specifying the security settings for the specified URL.

attachments
a WSAttachments object specifying the attachment settings for the specified URL.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

Example

Add Security Attachments with indicated property settings.

```
//No username token
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 null, null);
//Username, Password Text
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, false, false), null);
//Username, Password digest
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   false, false, false), null);
//Username, Password text, Add Created Header
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, true, false), null);
//Username, Password text, Add Timestamp, valid for 10 seconds
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, false, false, 10), null);
//Username, Password text, Add Nonce
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, false, true), null);
//Username, Password text, Add Created Header,
//Add Timestamp, valid for 10 seconds, Add Nonce
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
```

```
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, true, true, 10), null);
//Username, Password text, Add Created Header,
//Add Timestamp, valid for 10 seconds, Add Nonce
//With attachments
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, true, true, 10),
 ws.attachments(AttachmentMechanism.DEFAULT,
   ws.attachment("C:\\MTOMService.xml", "attachmentPart"),
   ws.attachment("C:\\SecureService.xml","attachmentPart")));
```

ws.attachment

Convenience method to create a new Attachment.

Format

The ws.attachment method has the following command format(s):

ws.attachment(*filename*, *attachmentPart*);

Command Parameters

filename
a String specifying the attachment file name.

attachmentPart
a String specifying the Attachment part.

Returns

a new Attachment

Example

Add Security with indicated attachments.

```
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, true, true, 10),
 ws.attachments(AttachmentMechanism.DEFAULT,
   ws.attachment("C:\\MTOMService.xml", "attachmentPart"),
   ws.attachment("C:\\SecureService.xml","attachmentPart")));
```

AttachmentMechanism

The AttachmentMechanism has the following values:

Table 20–3 List of AttachmentMechanism Values

Value	Description
DEFAULT	Specifies the default transfer type specified by the Content-Type header.
SWA	Specifies Security SOAP Messages with Attachments.
MTOM	Specifies SOAP Message Transmission Optimization Mechanism.
DIME	Specifies Direct Internet Message Encapsulation.

ws.attachments

Convenience method to create a new Attachment[].

Format

The ws.attachments method has the following command format(s):

ws.attachments(*am*, *attachments*);

Command Parameters

am
an AttachmentMechanism enum specifying the attachment transfer type.

attachments
the Attachment objects.

Returns

a new WSAttachments

Example

Add Security with indicated attachments.

```
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, true, true, 10),
 ws.attachments(AttachmentMechanism.DEFAULT,
   ws.attachment("C:\\MTOMService.xml", "attachmentPart"),
   ws.attachment("C:\\SecureService.xml","attachmentPart")));
```

ws.post

Request a web service using an HTTP POST method with requestSoap string, headers, URL encoding, and character sets.

Format

The ws.post method has the following command format(s):

ws.post(*recid*, *url*, *requestSoap*, *headers*, *bEncode*, *reqCharset*, *respCharset*);

Command Parameters

recid
ID of a previously recorded navigation, used for comparison purposes. May be null.

url
URL to request

requestSoap
A string representation of SOAP contents

headers
a Header array specifying the Additional headers to add/remove from the request before submitting it to the server.

bEncode
a boolean specifying the parameter encoding. Set to `true` to URL-encode the parameters before submitting them to the sever.

reqCharset
a String specifying the character set to use if URL-encoding any of the request parameters.

respCharset
a String specifying the character set to use when reading the response contents.

Throws

Exception
if an error occurs Posting to the server.

Example

Specifies a POST navigation with requestSoap string, headers, URL encoding, and character sets.

```
//Example UI added Post
ws.post(7, "http://myserver/EmployeeLookup.asmx",
 "requestSoap string",
 http.headers(http.header("name1", "value1", Header.HeaderAction.Add),
  http.header("name2", "value2", Header.HeaderAction.SetIfNotSet),
  http.header("name3", "value3", Header.HeaderAction.GlobalAdd),
  http.header("name4", "value4", Header.HeaderAction.GlobalSetIfNotSet)),
 true, "ASCII", "ASCII");
//Example WSDL Manager added Post with requestSoap string
ws.post(8,"http://myserver/EmployeeLookup.asmx",
```

```
"<soapenv:Envelope " +
 "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" " +
 "xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\" " +
 "xmlns:soapenv=\"http://schemas.xmlsoap.org/soap/envelope/\" " +
 "xmlns:web=\"http://mysite.com/webservices\">\r\n " +
 "<soapenv:Header/>\r\n " +
 "<soapenv:Body>\r\n " +
  "<web:addEmployee " +
    "soapenv:encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\">\r\n " +
   "<data xsi:type=\"enc:Employee\" " +
     "xmlns:enc=\"http://mysite.com/webservices/encodedTypes\">\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\">0</EmployeeID>\r\n " +
    "<Address xsi:type=\"xsd:string\">string</Address>\r\n " +
    "<City xsi:type=\"xsd:string\">string</City>\r\n " +
    "<State xsi:type=\"xsd:string\">string</State>\r\n " +
    "<Zip xsi:type=\"xsd:string\">string</Zip>\r\n " +
    "<Phone xsi:type=\"xsd:int\">0</Phone>\r\n " +
   "</data>\r\n " +
  "</web:addEmployee>\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","\"http://mysite.com/webservices/addEmployee\"",
  Header.HeaderAction.Modify)),
 true, null, null);
```

ws.security

Convenience method to create a new WSSecurity object with a specified timeout value.

Format

The ws.security method has the following command format(s):

ws.security(*username*, *password*, *plaintextPassword*, *addCreatedHeader*, *addNonce*);

ws.security(*username*, *password*, *plaintextPassword*, *addCreatedHeader*, *addNonce*, *timeToLive*);

Command Parameters

username
a String specifying the username.

password
a String specifying the password.

plaintextPassword
a boolean specifying the password type. `true` for Password Text, `false` for Password Digest.

addCreatedHeader
a boolean specifying if a created header is added.

addNonce
a boolean specifying if Nonce is added.

timeToLive
a value specifying the timeout.

Returns

a new WSSecurity object.

Example

Add Security with indicated property settings.

```
//Username, Password Text
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, false, false), null);
//Username, Password digest
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   false, false, false), null);
//Username, Password text, Add Created Header
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, true, false), null);
//Username, Password text, Add Timestamp, valid for 10 seconds
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, false, false, 10), null);
//Username, Password text, Add Nonce
```

```
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, false, true), null);
//Username, Password text, Add Created Header,
//Add Timestamp, valid for 10 seconds, Add Nonce
ws.addSecurityAttachments("http://myserver/EmployeeLookup.asmx",
 ws.security("username", decrypt("sO7ijipCONXoxAURd+8BQw=="),
   true, true, true, 10), null);
```

ws.solveXpath

Extract values from the browser's last retrieved contents using XPath notation.

The value will be stored to the given variable.

Format

The ws.solveXpath method has the following command format(s):

ws.solveXpath(*varName*, *xpath*, *resultIndex*);

Command Parameters

varName
a String specifying the variable name where the value will be stored. Must not be `null`.

xpath
a String specifying the XPath describing which value to parse from the last retrieved contents. Must not be `null`.

resultIndex
an index value specifying the 0-based index of the specific result to retrieve if the XPath returns multiple results. A `null` value specifies that all results should be returned.

Throws

AbstractScriptException
represents an exception that may be thrown during the execution of a script where the exception should be reported to an end user through the controller.

SolveException
if the XPath cannot be solved.

XMLServiceException
if happens error when parsing the xpath.

Example

Verifies the Xpath set to the script variable "solvedXPath" can vbe solved.

```
http.get(13, "http://example.com/stocks/", null,
 null, true, "ASCII", "ASCII");
ws.solveXpath("solvedXPath", ".//FORM[@name='loginform']/@action", 0);
if (getVariables().get("solvedXPath").equals("default.asp"))
 info ("method \"solveXpath()\" is passed.");
else
 warn ("method \"solveXpath()\" is failed.");
```

Index

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X