Oracle Utilities Testing Accelerator

User's Guide

Release 6.0.0.0.2

F13724-01

December 2018

ORACLE®

Contents

Chapter 8

Appendix A

Appendix B

Appendix C

Appendix D

Appendix E

Appendix F

Preface

Welcome to the Oracle Utilities Testing Accelerator User's Guide.

The guide explains how to use Oracle Utilities Testing Accelerator to automate the business test flows for testing the Oracle Utilities' applications.

This preface focuses on the following:

- Audience
- Prerequisite Knowledge
- Abbreviations
- Supported Oracle Utilities Applications
- Related Documents
- Conventions

Audience

This guide is intended for Automation Developers, and Test Engineers who automate the business test flows for testing the Oracle Utilities' applications.

Prerequisite Knowledge

The metadata driven automation development paradigm of Oracle Utilities Testing Accelerator does not require any programming experience to develop scripts for testing. However, the advanced programming features available in the application require experience with the Java programming language.

Abbreviations

The following abbreviations are used throughout this document:

Term	Expanded Form
BDD	Behavior Driven Development
CCB	Oracle Utilities Customer Care and Billing
C2M	Oracle Utilities Customer To Meter
MDM	Oracle Utilities Meter Data Management
MWM	Oracle Utilities Mobile Workforce Management
UTA	Oracle Utilities Testing Accelerator
WAM	Oracle Utilities Work and Asset Management

Supported Oracle Utilities Applications

The Oracle Utilities' applications supporting this Oracle Utilities Testing Accelerator v6.0.0.0.2 release are as follows:

*   Core

*   Oracle Utilities Mobile Workforce Management/ Oracle Real-Time Scheduler v2.3.0.0

*   Oracle Utilities Customer Care and Billing v2.6.0.0

*   Oracle Utilities Customer Care and Billing v2.6.0.1

*   Oracle Utilities Customer To Meter v2.6.0.1

*   Oracle Utilities Work and Asset Management/ Oracle Utilities Operational Device Management v2.2.0.1

*   Oracle Utilities Meter Data Management / Oracle Utilities Smart Grid Gateway v2.2.0.1

*   Oracle Utilities Meter Data Management / Oracle Utilities Smart Grid Gateway v2.2.0.2

Related Documents

For more information, refer to the following Oracle resources.

Release Notes

- *Oracle Utilities Testing Accelerator Release Notes*

Installation and Administration Guide

- *Oracle Utilities Testing Accelerator Installation and Administration Guide*

User and Reference Guides

- *Oracle Utilities Testing Accelerator Upgrade Guide*

- *Oracle Utilities Testing Accelerator Licensing Information User Manual*

- *Oracle Utilities Testing Accelerator Reference Guide for Core*

- *Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Mobile Workforce Management/ Oracle Real-Time Scheduler v2.3.0.0*

- *Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Customer Care and Billing v2.6.0.0*

- *Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Customer Care and Billing v2.6.0.1*

- *Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Customer To Meter v2.6.0.1*

- *Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Work and Asset Management/ Oracle Utilities Operational Device Management v2.2.0.1*

- *Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Meter Data Management / Oracle Utilities Smart Grid Gateway v2.2.0.1*

- *Oracle Utilities Testing Accelerator Reference Guide for Oracle Utilities Meter Data Management / Oracle Utilities Smart Grid Gateway v2.2.0.2*

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.

Chapter 1

Overview

This chapter introduces the Oracle Utilities Testing Accelerator application and provides an overview of the application architecture and features.

- Introduction

- Terminology

- Application Architecture

- Application Features

Introduction

Oracle Utilities Testing Accelerator comprises test automation accelerators for the automated testing of Oracle Utilities applications. It is a framework based on Java and Selenium for creating the web services and user interface automation scripts.

Oracle Utilities Testing Accelerator enables you to create the automation scripts using keywords or metadata, and without using any programming language. This saves the test automation development effort and avoid programming the scripts manually.

The accelerators contain out-of-the-box delivered test components that can be used to build test flows for the Oracle Utilities applications. You can extend the delivered components or create a new component to build their customized test flows. Utilities' application-specific sample test flows are provided in the respective reference guides. For information about the reference guides included in this release, refer to the Related Documents section in Preface.

Terminology

The different terms used in this document are as follows:

Term	Description
Oracle Utilities Test Accelerator (UTA)	Helps to build and maintain components and flows for automated testing.
Keyword	A pre-defined word used to define a specific step in a test case.
Component	Reusable automated test or part of a test.
	A component is the building block of an automated test flow. Each component is made up of a definition which allows users to define a keyword and associate values and parameters for the keyword.
Flow	An automated test.
	A flow comprises one or more components and/or component sets that are called in a pre-determined sequence.
Databank	Container of test data used by an automated test flow.
	The databank is defined using comma separated values (.csv) in a text file.

All components and flows in Oracle Utilities Testing Accelerator are organized into hierarchy for better manageability. The hierarchy is:

Release > Portfolio > Product > Module

Release	Represents the highest level of hierarchy.
	There is one release per an Oracle Utilities Testing Accelerator version, and it contains one or more portfolios.
Portfolio	Represents a product family consisting of one or more related products.
	A portfolio contains one or more products.

Term	Description
Product	Represents an Oracle Utilities application. For example: CCB
	A product contains one or more modules.
Module	Represents an Oracle Utilities application functional area. For example: Billing in CCB
	A module contains one or more components that are used to automate a specific functional area in an Oracle Utilities application.

For information about these terms, refer to Chapter 2: Oracle Utilities Testing Accelerator Features.

Application Architecture

The following diagram depicts the high-level architecture of Oracle Utilities Testing Accelerator.
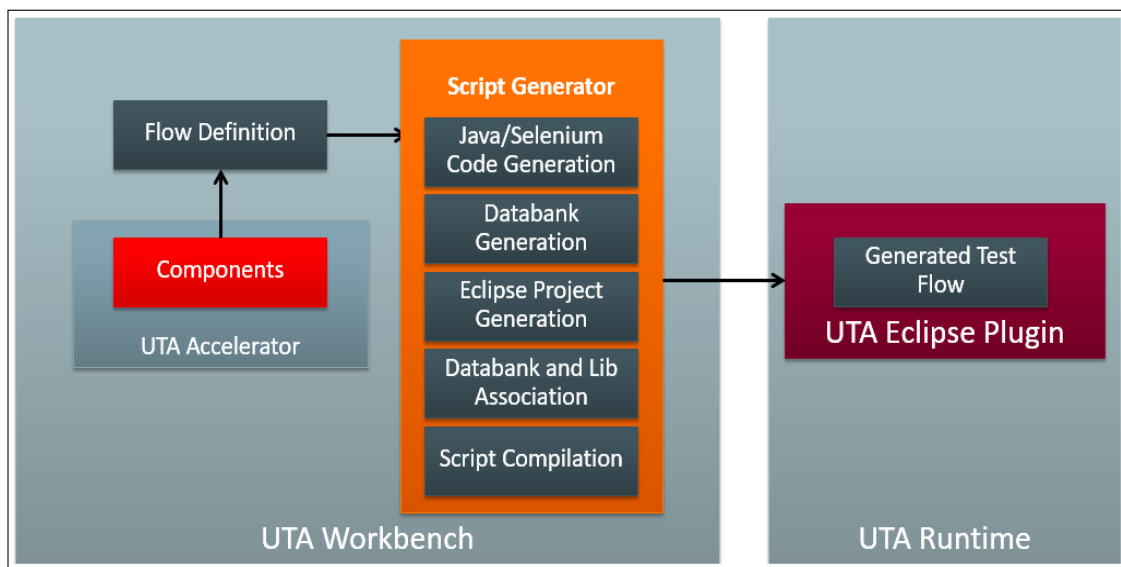
High-Level Architecture Diagram

Components are defined using metadata in Oracle Utilities Testing Accelerator. Using these components a flow can be assembled and generated. Execute the generated script using Eclipse IDE for Java Developers that has the Oracle Utilities Testing Accelerator plugin installed.

For more information about Oracle Utilities Testing Accelerator, refer to the Oracle Utilities Testing Accelerator *Installation and Administration Guide.*

Application Features

The features available in this Oracle Utilities Testing Accelerator release are the dashboard, components, flows, various tools, and administration.

For more information about these features and their significance, refer to Chapter 2: Oracle Utilities Testing Accelerator Features.

Chapter 2

Oracle Utilities Testing Accelerator Features

This chapter describes the features available in this Oracle Utilities Testing Accelerator release:

- Administration

- Components

- Dashboard

- Flows

- Tools

Administration

The **Administration** tab allows the users with Administrator role to do the following:

• Create/edit release, portfolio, product and modules

• Create/manage Oracle Utilities Testing Accelerator application user accounts

• Allow upgrading CM content from one version of an Oracle Utilities application to a later version

  For example: From CCB v2.6.0.0 to CCB v2.6.0.1

For more details, refer to Chapter 4: Oracle Utilities Testing Accelerator Administration.

Components

The **Components** page displays all the available components imported/created in the application. On this page, you can do the following:

• Create a new component

• Define/update the definition of a component

• Submit the component for approval

• Accept/reject the approval based on the state of the component

For more information about components, refer to Chapter 5: Creating Components.

Dashboard

This is the **Home** page of the application and displays the following information:

• Notifications assigned to your role

• Statistics about the number of components/flows in the application

• Total number of custom components vs Total number of approved components

• Total number of flows vs Total number of approved flows

Notifications

The **Dashboard** page displays notifications of interest to the user currently logged in and also some basic analytics on the total number of components and flows in Oracle Utilities Testing Accelerator and a breakdown of those based on their lifecycle state.

On the **Dashboard** page, you can do the following:

• Click the bell icon to navigate directly to the **Notifications** page.

• Click **List All** to display a popup with all the unread notifications applicable to the current user.

• Click the **Refresh** icon to populate the notification area with all the latest notifications generated.
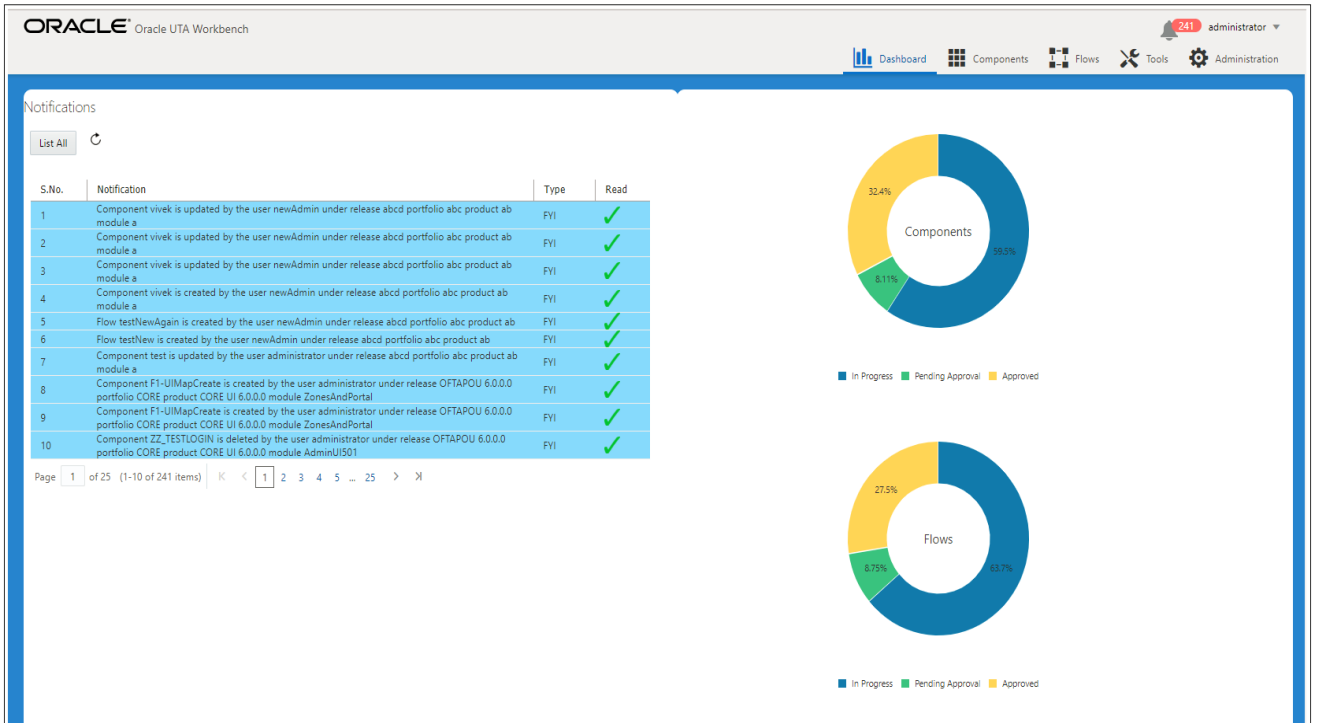
The Dashboard page

Any event of interest in the application triggers a notification that is sent to one or more users. Events could be either of the following:

- Creating/updating any hierarchy related entity (for example: Release/Portfolio/Product/Module)

- Change in lifecycle state of a component/flow (for example: submitting a component for approval/rejection, etc.)

The different types of notifications are as follows:

- **FYI Notifications** - For informational purpose only, and are generated when the following are performed:

    - A component/flow for all users is created.

    - A release/portfolio/product/module for an administrator is created.

    - A user for an administrator is created.

    - A flow/component for approval for a developer is submitted.

  Click an FYI notification for more information about the event and also mark the notification as 'read'. Once an FYI notification is read, it is removed from the notification area.

- **ToDo Notifications** - For a component/flow when submitted for approval by an approver/administrator. They require some action from the user. They are displayed in the **Notification** area for users with Approver/Administrator role.

  A ToDo notification displays detailed information about the respective event. It also allows users to take appropriate action as applicable. (for example: Reject, Revert to Approve, Approve, or Send to in progress (Flow)). Click the **Read** column to mark a ToDo notification as 'read'.

Flows

This page displays all the available flows imported/created in the application. On this page, you can do the following:

- Create a new flow

- Define the flow

- Submit the flow for approval

- Accept/reject the approval based on the state of the flow

You can also generate a flow by providing a plain text Behavior Driven Development scenario. For more details, refer to the Creating Flows section in Chapter 6: Creating Test Flows.

Tools

This feature provides access to various tools that allow you to import/export components and flows in the application. web service components are automatically generated by specifying the WSDL of the web service that the component makes a call to in the Oracle Utilities applications, such as C2M or CCB.

For more details, refer to Chapter 7: Development Accelerator Tools.

Chapter 3

Developing Metadata Driven Web Service Based Test Automation

The Oracle Utilities Testing Accelerator components, component sets, and flows are organized in a tree hierarchy. This hierarchy compartmentalizes these for different Oracle Utilities applications.

This chapter is intended primarily for automation developers and testers. It describes the metadata-driven automation development methodology and the set up of automation development environment.

- Metadata Driven Automation Development Methodology

- Setting Up Automation Development Environment

Metadata Driven Automation Development Methodology

This section describes the metadata-driven automation development methodology that enables a test automation engineer to create automation scripts for an Oracle Utilities application.

An application has to be tested for its base functionality and extensions or customization. For this, you can create granular tests or larger end-to-end business test flows. Irrespective of the test design techniques, these tests can be used for regression testing the application in case of upgrades or customization to ensure that the existing functionality is not broken.

Typically, automation development is a time consuming exercise and teams have challenges in knowing and implementing the industry best practices and automation tools that work best for their product technology stack, helping them be successful in their efforts. Few of such challenges are as follows:

*   Selecting an automation tool

*   Creating the automation framework

*   Identifying the automation development methodology

*   Ensuring the automated tests are updated for new releases

*   Ensuring the coverage levels are up to date

*   Configuration management of automated test programs

The metadata-driven automation development methodology provides solutions to such challenges.
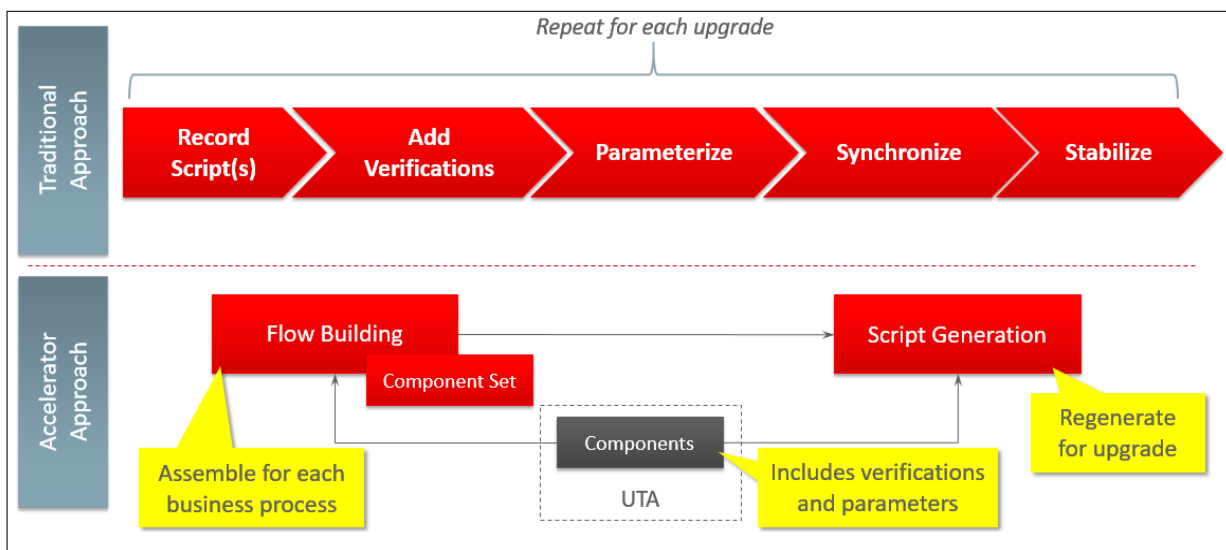
Development Methodology

For the Oracle Utilities applications built on Oracle Utilities Application Framework, web service based automated testing is proven to be more robust, maintainable, and faster to develop and execute. Oracle Utilities Testing Accelerator comprises web services and UI based components that enable creation and execution of test flows.

The following sections provide the test automation development phases in which an automated test flow is created.

*   Planning

*   Design and Development

*   Test Execution

Planning

To plan an automated test flow, identify the business test flow to be automated and the components required for the flow. If necessary, create additional components or extend the delivered components.

For details about how to extend the components, refer to the Copying Components section in Chapter 5: Creating Components.

Design and Development

A flow design explains the order in which the components will be used to interact with each other in the flow. It also defines the test data combinations to use.

To design and develop an automated test flow, follow these steps:

1. Create/extend the required components that are identified in planning phase.

2. Create a test flow in Oracle Utilities Testing Accelerator that maps to the identified business test flow in the application.

   For details about how to create a test flow, refer to the Creating Flows section in Chapter 6: Creating Test Flows.

   For information about delivered sample flows to understand the flow creation, refer to the **Sample Work Flows** chapter in the respective product-specific reference guides. For a list of reference guides available in this release, refer to the Related Documents section in Preface.

3. Drag and drop the required components into the flow.

4. Add the test data for the flow.

   The test data can be modified at the runtime using the standard Oracle Utilities Testing Accelerator databanks. For more details, refer to the Test Data Management section in Chapter 6: Creating Test Flows.

5. Assemble and generate the script for the test flow.

6. Download the test script.

Test Execution

To execute the automated test flow, execute the script in Oracle Utilities Testing Accelerator.

To use another data set to execute the script, change the databanks in the generated scripts project and execute the script. For more details, refer to the Executing Test Flows section in Chapter 6: Creating Test Flows.

The components and test flows developed using this approach are stored and version controlled in the Oracle Utilities Testing Accelerator database. It takes care of the challenges in configuration management of automated tests.

Setting Up Automation Development Environment

The steps involved to set up the development environment for Oracle Utilities Testing Accelerator are as follows:

- Step 1: Setting Up the Oracle Utilities Testing Accelerator Server

- Step 2: Setting Up Workstations for Development/ Testing

- Step 3: Setting Up Application Under Test

Setting Up the Oracle Utilities Testing Accelerator Server

This section explains the steps to be performed to setup the server.

- Installing Oracle Utilities Testing Accelerator

- Installing Oracle Utilities Testing Accelerator Client Runtime

- Creating Users

Installing Oracle Utilities Testing Accelerator

For detailed instructions to install Oracle Utilities Testing Accelerator, refer to the Oracle Utilities Testing Accelerator *Installation and Administration Guide*.

Installing Oracle Utilities Testing Accelerator Client Runtime

Oracle Utilities Testing Accelerator Client Runtime has to be installed on a client workstation.

For installation instructions, refer to the **Installing on Client Admin Workstation** section in Oracle Utilities Testing Accelerator *Installation and Administration Guide*.

> **Note:** The Oracle Utilities Testing Accelerator application need not be installed on the user workstations. Users only need:
>
> - A browser access to it for the component and flow development.
>
> - An installation of Eclipse IDE for Java Developers with Oracle Utilities Testing Accelerator Eclipse plugin to enable the execution of flows.

Creating Users

Create users with Administrator access. A user with Administrator access can create the necessary users with various roles (Developer/Approver/Administrator) so the automation developers can start using the Oracle Utilities Testing Accelerator application.

Setting Up Workstations for Development/ Testing

This section provides the steps to set up the Oracle Utilities Testing Accelerator developer workstations. The tasks include:

- Extracting Oracle Utilities Testing Accelerator Client Runtime

- Installing Oracle Utilities Testing Accelerator Eclipse IDE for Java Developers

- Installing Oracle Utilities Testing Accelerator Eclipse Plugin

Extracting Oracle Utilities Testing Accelerator Client Runtime

The Oracle Utilities Testing Accelerator package downloaded from Oracle Software Delivery Cloud (OSDC) (https://edelivery.oracle.com/) contains UTA_Client.zip file that includes all the Client Runtime artifacts.

After the UTA_Client.zip file is unzipped, a folder structure similar to as shown in the following diagram is created.
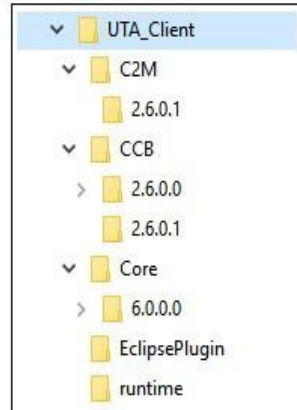
Runtime Folder Structure

Creating Oracle Utilities Testing Accelerator Client Runtime Folder Structure

To create a Oracle Utilities Testing Accelerator client runtime folder structure, do the following:

1. Create/select a folder to use as your runtime folder.

    Note: This folder is referred to as <UTA_CLIENT_WORK_DIR> in the following sections.

2. Copy the contents of the runtime folder from UTA_Client.zip.

    For more details about UTA_Client.zip refer to the Extracting Oracle Utilities Testing Accelerator Client Runtime section.

After copying the contents of the runtime folder, the <UTA_CLIENT_WORK_DIR> should look as follows.
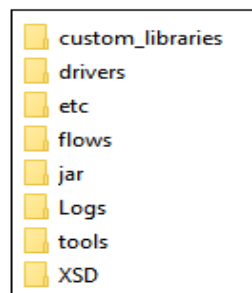
Folder Structure for Generated Scripts

- **drivers**

    The Chrome and Firefox browser drivers are used to invoke the browser during execution of flows that contain Graphical User Interface (GUI) based components.

- **etc**

    The configuration.properties file includes all the properties that the Oracle Utilities Testing Accelerator flows refer to during the flow execution. The log4j.properties file controls the logging output from flows.

    Note: All passwords contained in this file have to be encrypted. The Password Encryption Tool can be used to encrypt plain-text passwords. For more details, refer to the Encrypting Passwords section in Chapter 6: Creating Test Flows.

- **flows**

  The scripts that are generated and downloaded from Oracle Utilities Testing Accelerator should be placed in this folder.

- **jar**

  The 3rd party jar files that are needed to execute the Oracle Utilities Testing Accelerator flows.

- **Logs**

  The runtime generated test execution logs that can be later used for debugging.

- **tools**

  The Password Encryption Tool that you can use to encrypt any passwords. For information about how to encrypt passwords that are stored in the configutaion.properties file, refer to the Encrypting Passwords section in Chapter 6: Creating Test Flows.

- **xsd**

  The run-time generated XSDs required for processing the web services request.

Installing Oracle Utilities Testing Accelerator Eclipse IDE for Java Developers

Ensure Eclipse IDE for Java Developers is installed on each user workstation where automation execution is performed or where component and flow development is intended to be performed.

For certified Eclipse IDE for Java Developers version details, refer to the **System Requirements** section in Oracle Utilities Testing Accelerator *Installation and Administration Guide*.

To install Eclipse IDE for Java Developers, do the following:

1. Download Eclipse IDE for Java Developers from the following location:

    https://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/oxygen3a

2. Extract the downloaded zip file to a folder. For example: C:\UTA_Eclipse

3. Navigate to the unzipped Eclipse folder and double-click the eclipse.exe file to launch the Eclipse IDE.

4. When prompted for workspace, select the runtime folder (as created in the Creating Oracle Utilities Testing Accelerator Client Runtime Folder Structure section) as the workspace folder.

Oracle Utilities Testing Accelerator Eclipse IDE for Java Developers is successfully installed.

Installing Oracle Utilities Testing Accelerator Eclipse Plugin

The Oracle Utilities Testing Accelerator Eclipse Plugin provides a custom Eclipse perspective with necessary information about the Oracle Utilities Testing Accelerator generated script execution (such as components executed, request/response xml content, etc.).

To install the Oracle Utilities Testing Accelerator Eclipse Plugin, do the following:

1. Extract the UTA_Eclipse_Plugin.zip file from the EclipsePlugin folder.

    For instructions to extract the plugin, refer to the Extracting Oracle Utilities Testing Accelerator Client Runtime section.

2. Launch Eclipse.

    For instructions to install Eclipse, refer to the Installing Oracle Utilities Testing Accelerator Eclipse IDE for Java Developers section.

3. Navigate to **Help** menu > **Install New Software**.

4. On the **Install** dialog box, click **Add**.

5. On the **Add Repository** dialog box, click **Local.**

6. Browse to the location where the UTA_Eclipse_Plugin.zip was extracted and click **OK**.

7. Click the **Uncategorized** checkbox.

8. Click **Next**.

9. Accept the **License Agreement** and click **Finish**.

10. When prompted for click **Install Anyway**.

11. Restart Eclipse IDE to use the Oracle Utilities Testing Accelerator Eclipse Plugin.

12. To use the Oracle Utilities Functional Test perspective, navigate to **Window** > **Perspective** > **Open Perspective** > **Other...**.

13. Select Oracle Utilities Functional Test from the list of perspectives.

14. Click **Open**.

The Eclipse with the Oracle Utilities Testing Accelerator Plugin is now setup.

Setting Up Application Under Test

For setup details, refer to the respective Oracle Utilities' application-specific installation guide.

Ensure that Oracle Utilities Testing Accelerator related metadata exists in this application instance. For more details, refer to the **Post-Installation Tasks** section in Oracle Utilities Testing Accelerator *Installation and Administration Guide.*

Chapter 4

Oracle Utilities Testing Accelerator Administration

This chapter introduces the Administration feature in Oracle Utilities Testing Accelerator. It focuses on the following:

- Overview

- Administration Tab

Overview

The Administration feature in Oracle Utilities Testing Accelerator allows the users with Administrator role to do the following:

- Create/edit release, portfolio, product, and modules

- Create/manage Oracle Utilities Testing Accelerator application user accounts

- Allow upgrading CM content from one version of an Oracle Utilities application to a later version. For example: from CCB v2.6.0.0 to CCB 2.6.0.1

Administration Tab

The **Administration** tab in the Oracle Utilities Testing Accelerator application allows users with Administrator role to perform the following actions:

- Managing Releases

- Managing Portfolios

- Managing Products

- Managing Modules

- Managing Users

- User Access Types

The following diagram shows the organization of components and flows as per hierarchy in the Oracle Utilities Testing Accelerator application.
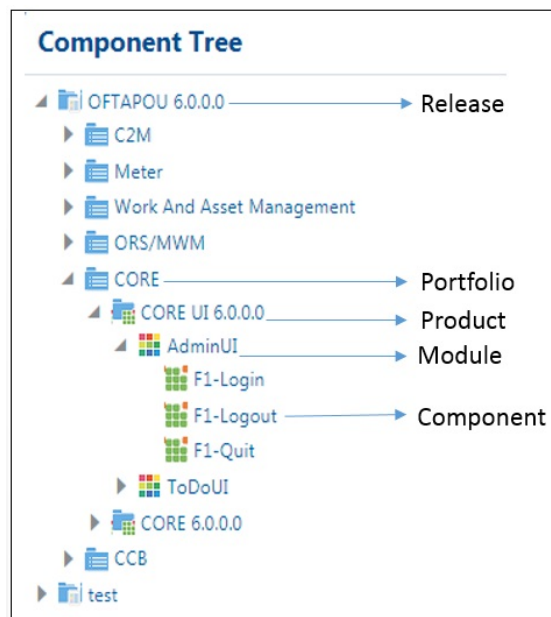
Component Tree

Managing Releases

A release represents the highest level of hierarchy. There is one release per an Oracle Utilities Testing Accelerator version, and it contains one or more portfolios.

Creating a Release

To create a release:

1. On the **Administration** tab, click **Releases** in the left pane.

2. In the **Create Release** window, enter the release name and its description.

3. Click **Save**.

Updating a Release

Note that you can only edit a custom release.

To update an existing release:

1. On the **Components** (or **Flows**) tab, expand the **Component** (or **Flow**) tree.

2. Select and right-click the release to be updated.

3. From the **Context** menu, click **Update Release**.

4. Enter the modified description and click **Update**.

Managing Portfolios

A portfolio represents a product family consisting of one or more related products. It contains one or more products.

Creating a Portfolio

To create a portfolio:

1. On the **Administration** tab, click **Portfolios** in the left pane.

2. In the **Create Portfolio** window, enter the portfolio name and its description.

3. Click **Save**.

Updating a Portfolio

Note that you can only edit a custom portfolio.

To update an existing portfolio:

1. On the **Components** (or **Flows**) tab, expand the **Component** (or **Flow**) tree.

2. Select and right-click the portfolio to be updated. From the **Context** menu, click **Update Portfolio**.

3. Enter the modified description and click **Update**.

Managing Products

A product represents an Oracle Utilities application. A product contains one or more modules.

For example: CCB

Creating a Product

To create a new product:

1. On the **Administration** tab, click **Products** in the left pane.

2. In the **Create Product** window, enter the product name and its description.

3. Click **Save**.

Alternatively, you can create a new product as follows:

1. On the **Components** (or **Flows**) tab, expand the **Component** (or **Flow**) tree.

2. Select and right-click the portfolio under which the product has to be created. From the **Context** menu, click **Create Product**.

3. Enter the new product name and its description.

4. Click **Save**.

Updating a Product

Note that you can only edit a custom product.

To update an existing product:

1. On the **Components** (or **Flows**) tab, expand the **Component** (or **Flow**) tree.

2. Select and right-click the product name to be updated. From the **Context** menu, click **Update Product**.

3. Enter the modified description and click **Update**.

Managing Modules

A module represents an Oracle Utilities application functional area. For example: Billing in CCB

Creating a Module

To create a new module:

1. On the **Administration** tab, click **Modules** in the left pane.

2. In the **Create Module** window, enter the module name and its description.

3. Click **Save**.

Alternatively, you can create a module as follows:

1. On the **Components** (or Flows) tab, expand the **Component** (or Flow) tree.

2. Select and right-click the product under which the module has to be created.

3. From the **Context** menu, click **Create Module**.

4. Enter the new module name and its description.

5. Click **Save**.

Updating a Module

Note that you can only edit a custom module.

To update an existing module:

1. On the **Components** (or Flows) tab, expand the **Component** (or Flow) tree.

2. Select and right-click the module name to be updated.

3. From the **Context** menu, click **Update Module.**

4. Enter the modified description and click **Update**.

Managing Users

Only users with an Administrator role can manage the other users.

To create, search for, upgrade, or delete a user, you need to be on the **Administration** page to perform the task. You can also change the password from this page.

To get to the **Administration** page, do the following:

1. Log into the application.

2. Navigate to the **Administration** tab.

3. On the **Administration** page, details of all users are shown in the right pane.

Creating a User

To create a new user:

1. On the **Administration** page, click **Add** to create a new user.

2. Fill in the following fields and click **Save**.

   • Username

   • Full name

   • Manager name

   • Password (length should be 6-15 characters)

   • E-mail

   • Access type (For information about access types, refer to the User Access Types section.)

   On success, the "User created successfully" message is displayed. If the user name entered already exists, the "User Name already exists" message is displayed.

Tip: Click **Back** to get redirected to the **Search Users** page. Click **Cancel** to refresh all the fields.

Searching for User(s)

To search for a user, do the following:

1. On the **Administration** page, click **Search** to search for a specific user.

2. Enter "%" in the **User Name** field to view all the existing users.

   You can also search based on a particular name or use a wild card search (For example: "J%" will fetch all the users starting with the alphabet "J").

Updating User Details

To update the details of an existing user, do the following:

1. On the **Administration** page, click **Search** and select the record you want to update.

2. Modify the required fields and click the update icon 🖊.

Deleting a User

To delete an existing user, do the following:

1. On the **Administration** page, click **Search** to select the record to be deleted.

2. Click the delete icon 🗑 .

   A window appears asking for confirmation about the deletion.

3. Select **Yes** to delete the user.

Please note that deleting the default "administrator" user is not allowed.

Changing a User's Password

To change the password of an existing user, do the following:

1. On the **Administration** page, click **Search** and select the record for which you want to change the password.

2. Click the change password icon.

3. On the **Update Password** window, enter the new password in both the fields.

4. Click **Submit**.

   On success, the "Password is changed" message is displayed.

User Access Types

The following table lists the privileges provided to users for various types of access.

Access	Developer	Approver	Administrator
Approve/reject a component		✓	✓
Approve/reject a flow		✓	✓
Change Password (self)	✓	✓	✓
Component - Create/update/view/delete	✓	✓	✓
Copy Component	✓	✓	✓
Create Module			✓
Create Portfolio			✓
Create Product			✓
Create Release			✓
Export Component	✓	✓	✓
Export Flow	✓	✓	✓
Flow - Create/update/view/delete	✓	✓	✓
Generate a component from WSDL	✓	✓	✓
Import Component	✓	✓	✓
Import Flow	✓	✓	✓
Update Module	✓	✓	✓
Update Portfolio	✓	✓	✓
Update Release	✓	✓	✓

Access	Developer	Approver	Administrator
User Management • Create/update/view/ delete • Change other user password			✓

Chapter 5

Creating Components

The Oracle Utilities Testing Accelerator components, component sets, and flows are organized in a tree hierarchy. The hierarchy is organized as follows:

Oracle Utilities Testing Accelerator Release > Portfolio> Product > Module > Components

This chapter describes the component hierarchy and also the steps to create different types of components in Oracle Utilities Testing Accelerator.

- Component Structure
- Component Lifecycle
- Component Types
- Creating Web Service Based Components
- Creating GUI Based Components
- Creating REST Web Service Components
- Copying Components

Component Structure

The following figure shows the high-level component structure.
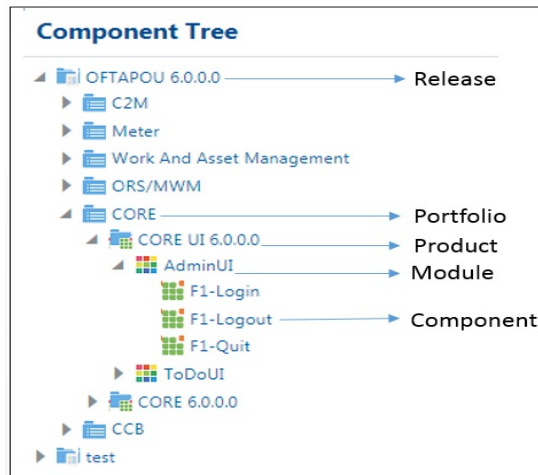
Component Structure

Component Lifecycle

The component lifecycle begins once a component is created in Oracle Utilities Testing Accelerator. It can exist in one of the several possible lifecycle states as shown in the following diagram.
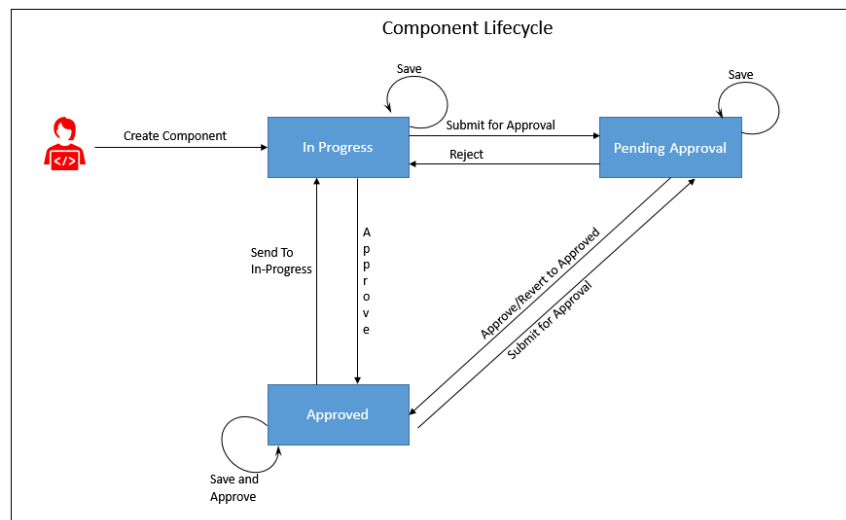
Lifecycle of a Component

The state of a component determines the actions that can be performed on the component. The following table summarizes the component states, and the possible actions and roles that can take the actions.

Component Lifecycle State	Permitted Actions	Role	Resultant State (after action)
In Progress	Submit for Approval	Developer Approver Administrator	Pending Approval
	Approve	Approver Administrator	Approved
	Save	Developer Approver Administrator	In Progress
Pending Approval	Send to In Progress / Reject	Approver Administrator	In Progress
	Approve	Approver Administrator	Approved
	Revert to Approved	Approver Administrator	Approved (Reverts to Previous Approved version of the component)
	Save	Developer Approver Administrator	Pending Approval
Approved	Send to In Progress	Developer Approver Administrator	In Progress
	Submit for Approval	Developer Approver Administrator	Pending Approval
	Approve (save and approve)	Approver Administrator	Approved

Locking/Unlocking Components

A component is/can be locked in the following scenarios:

• To prevent any other users from editing the component until the component definition is complete.

• By default when the component is submitted for approval.

• When moved to the 'In Progress' state, the component gets locked. You can then unlock and edit it as needed.

Click the icon to lock/unlock a component in the Oracle Utilities Testing Accelerator application.

Tip: After a component is moved to 'Approved' status, it gets unlocked automatically.

Component Types

Ensure the component is created under the required hierarchy level.

Oracle Utilities Testing Accelerator supports the following types of components:

- Web Service Based Components

- GUI Based Components

Web Service Based Components

A web service based component represents an Inbound Web Service/Business Object/Business Service in an Oracle Utilities application (such as CCB/C2M/WAM/Meter).

A distinguishing feature of the web service component is that its component type is defined as "WS" and the keywords used in defining it are specific to a web service request.

For information about web service specific keywords, refer to Appendix A: Web Service Component Keywords.

GUI Based Components

A GUI based component typically represents a page/part of the page in an Oracle Utilities application (such as CCB/C2M/WAM/Meter).

A distinguishing feature of the GUI based component is that it's component type is defined as "Web" and the keywords used in defining the component are specific to a web page. For example: Click, Edit, etc.

For information about GUI-specific keywords, refer to Appendix B: GUI Component Keywords.

Creating Web Service Based Components

You can create web service based components in either of the following ways:

- Using the Component Generation Tool feature in Oracle Utilities Testing Accelerator.

    For detailed instructions about the Component Generation Tool, refer to the Component Generation Tool section in Chapter 7: Development Accelerator Tools.

- Create the component manually.

This section focuses on the following:

- Creating a Component

- Creating a Component Definition

- Defining Default Data at Component Level

- Setting Up Operation Name for a Web Service

- Using Runtime Variables in Components

- Using Function Libraries

- Resolving the Repeating Elements in Response XML

- Adding Validations

- Logging and Reporting

- Handling the List Elements

Creating a Component

To create a web service based component manually, follow these steps:

1. Navigate to the component tree where the component has to be created.

2. Right-click the feature (release/product/module) in the component tree.

> **Note**: Create a new feature folder if it is not found in the delivered tree structure.
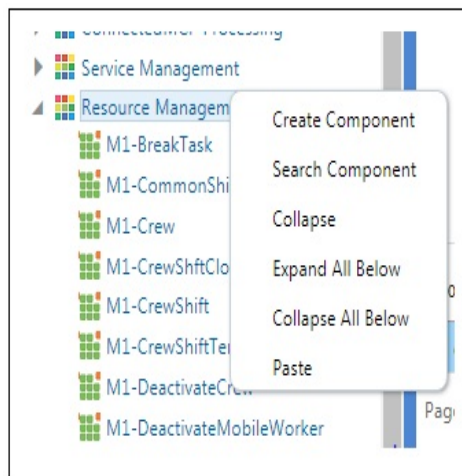
Creating a Component

3. Select **Create Component**.

> **Note**: The component name must be prefixed with 'CM' and the **Tags** field should have a CM tag for every component. The tagging enables porting the custom components to latest Oracle Utilities Testing Accelerator release.

4. Enter the component name in the **Component** field.

> **Note**: For information about extending components, refer to the Copying Components section.

5. Select **Web Service** in the **ComponentType** drop-down list.

6. Enter a description in the **Description** field.

7. Click **Attach Code** to add the metadata. The **Component** window is displayed.

8. Create component definitions.

9. Click **Save & Unlock** to save and create the component.

Following is an example to create the CM-MobileWorker component under the Resource Management feature for the Oracle Real-Time Scheduler v2.3.0.0.

1. Navigate to **UTA 6.0.0.0** > **ORS / MWM Portfolio** > **ORS 2.3.0.0** > **Resource Management**.

2. Right-click the **Resource Management** module.

3. Select **Create Component**.

4. Enter **CM-MobileWorker** in the **Component** field to name the component.

5. Follow steps 5-9 as shown in the procedure mentioned above.

Creating a Component Definition

A component consists of several component definition lines. Each component definition line comprises a keyword, object, display name, attribute values, default data, function name, and output parameters.
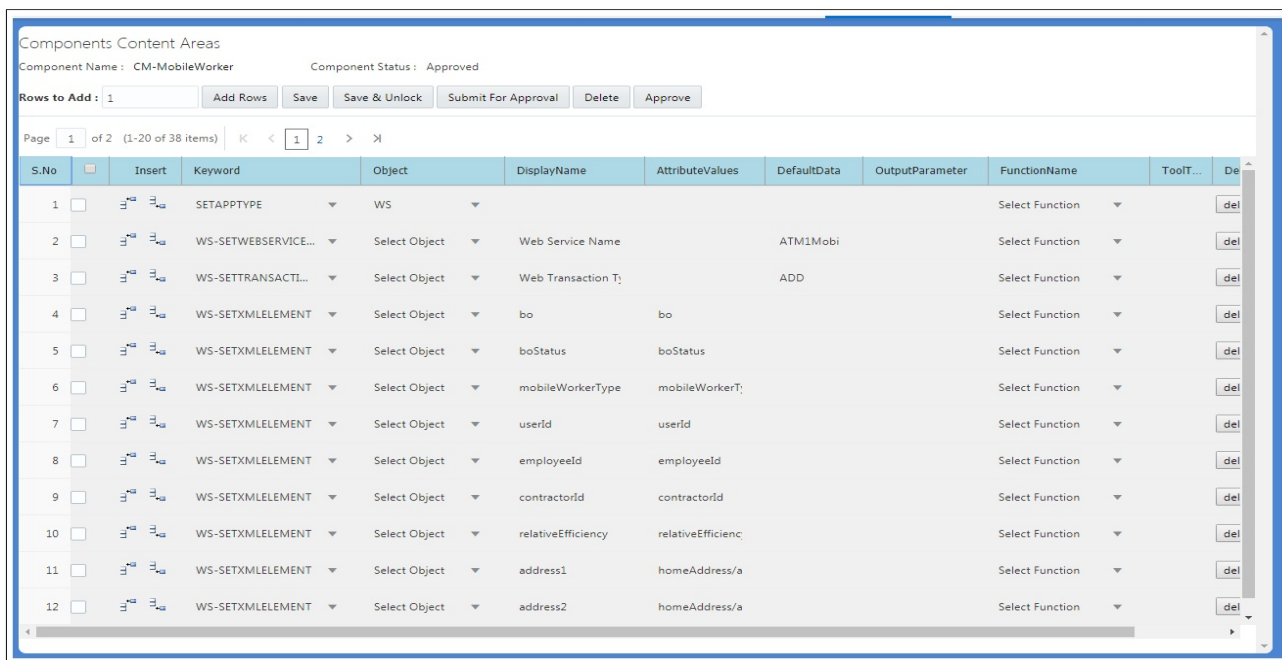
The following list describes each entity in a component definition:

- **Keyword**: The step to be performed. Example: WS-SETVARIABLEFROM RESPONSE, WS-VALIDATE, etc

- **Object**: The Oracle Utilities Testing Accelerator function library name from where the function is called.

- **Display Name**: The component definition.

- **Attribute Values**: The web service XML tag name used as variable to store its value.

- **Default Data**: The default data used in the component definition.

- **Function Name**: The function name called from the library.

- **Output Parameters**: The output in the form of a variable.

    For more options, refer to Appendix D: Generating Re-runnable Test Data.

- **Tooltip**: The data presented as a tool tip during the flow creation.

The following figure shows the **Component** page with the available component definitions.

Component Definition Page

Add the required component definition lines using the **Keyword** drop-down list to define the web services based component.

For a list of keywords used to define the web service based components, refer to Appendix A: Web Service Component Keywords.

The following example shows different component lines created for the CM-MobileWorker component.

1. Select SETAPPTYPE in the **Keyword** drop-down list to define the application type.

2. Select WS in the **Object** drop-down list to denote that it is a web service based component.

3. Select the WS-SETWEBSERVICENAME keyword to define the web service name.

4. Select the WS-SETTRANSACTIONTYPE keyword to define the transaction type of the web service call.

> **Note**: The final script of a component is web service call to create, update, and delete.

5. Select the WS-LOGMESSAGE keyword to log comments in component definition. This helps in debugging the script code for that component.

6. Select the WS-SETXMLELEMENT keyword to set the value into a specific element of request XML.

Consider the CM-MobileWorker component in Oracle Utilities Mobile Workforce Management. This component maps to the MobileWorker business object. It includes elements, such as:

```
<mobileWorkerType />
<contractorId />
```

7. Select the WS-SETXMLLISTELEMENT keyword to set a value into the list element tags. The list element is 'skills'.

> **Note**: The schema of a web service/business object/business service can be complex (the schema has group elements which in turn may have group elements within them).
>
> For instructions about how to handle such scenarios, refer to the Handling the List Elements section.

8. Click **Save**.

Defining Default Data at Component Level

In Oracle Utilities Testing Accelerator the test data is maintained at component level for quick and easy use at the flow level.

In each component definition line the "Default Data" column is available to hold the default data. Using this field, default test data can be populated in the component. While using a component with default data in a flow, the default data can easily be copied from component to flow using the "Move" option available on the **Flow Test Data** window.

Even after the default data is populated in the flow test data, data elements in the test data entry page can still be edited, if required. This helps to build the flow faster for cases where administration and master test data are pre-determined.
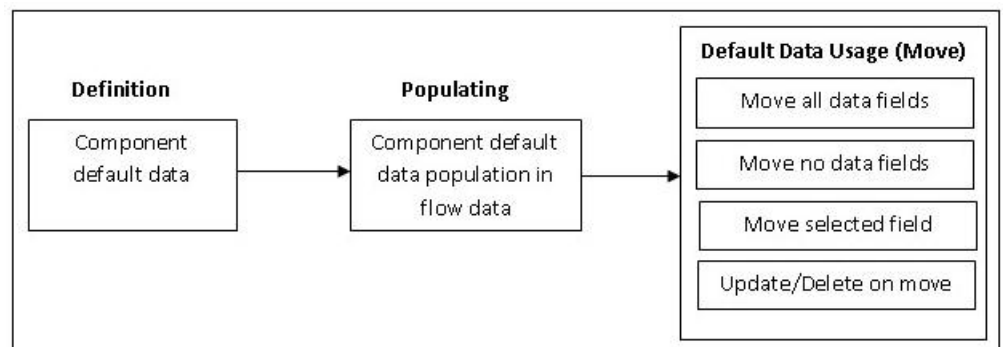
Data Flow

Setting Up Operation Name for a Web Service

An operation name determines the action to be taken while executing a web service request. The value for the WS-SETTRANSACTIONTYPE keyword is specified while adding the test data for the flow. If designed so, the same component can be used to add record, update record, or delete record operations.

For example: To create a new mobile worker, or to update or delete an existing mobile worker, set up the transaction type for appropriate the instance of the component in the flow.

Using Runtime Variables in Components

In some cases, few elements from the response component execution have to be passed as inputs to another component's request XML. To achieve this, store the output of first component in the global variable by using the WS-SETVARIABLEFROMRESPONSE keyword. This keyword requires Xpath of the response element whose value are to be stored. It should be specified in the **Attribute Values** column. The global variable which holds this value in the script is defined in the **Output Parameter** column.

The WS-SETVARIABLEFROMRESPONSE keyword stores the mobileWorkerId obtained after a mobile worker component execution to the global variable gVar_mobileWorkerId1 declared in the **Output Parameter** column.

For information about how a dependent component reads such global variables, refer to the Using Global Variables section.

Using Function Libraries

This section explains how to use the function libraries shipped with this Oracle Utilities Testing Accelerator release and create new help libraries.

Function libraries shipped with Oracle Utilities Testing Accelerator can be accessed in the **Component** window using the FUNCTIONCALL key word and specifying the library name in the **Object** column and the function name in the **Function Name** column. Define the variable name in the **Output Parameters** field to store the return value of the function.

Function parameters can be provided while entering test data for the component in a flow. For more details, refer to the Test Data Management section in Chapter 6: Creating Test Flows.

For a list of libraries and functions available in Oracle Utilities Testing Accelerator, refer to Chapter 8: Function Library Reference.

Resolving the Repeating Elements in Response XML

If the response XML has repeating elements, the value embedded within the repeating elements is retrieved as follows.

```
<ContactDetails>
<Phone> 123-456-7890 </Phone>
<Phone>234-567-8901 </Phone>
<email> joe@oracle.com </email>
</ContactDetails>
```

1. Use the WS-SETVARIABLEFROMRESPONSE keyword to retrieve the response of the web service invocation into the global variable. gVar1 is defined in the **Output Parameter** column.

   The keyword resolves all occurrences of the Phone element and stores all values in the gVar1 variable separated by comma. gVar1 will be set to "123-456-7890,234-567-8901".

2. Use the FUNCTIONCALL keyword to call the setVariableValueUsingListIndex function available in the OUTSPCORE library.

The keyword retrieves the value(s) based on the parameters passed. Parameters passed are global variables storing the values (gVar1 and index).

For more information, refer to the Chapter 8: Function Library Reference.

Adding Validations

The different ways in which you can add validations are as follows:

• Using the FUNCTIONCALL keyword

To validate the response, use the FUNCTIONCALL keyword to validate the content; in particular, the Xpath of response XML.

Select the wSVALIDATELIB function library from the **Object** drop-down list. Select the function to be called from the **Function Name** drop-down list.

For a complete reference of the validation function library, refer to Chapter 8: Function Library Reference.

• Using flow-level validations

Validations can be added before and after the existing flow. The same flow can be reused with different or no validations before (pre-level validations) and after (post-level validations).

For more information about the flow-level validations, refer to the Flow-Level Validations section.

Flow-Level Validations

Please note that this feature is available only for web service based components.

Apart from being able to define validations at the component level, you can also define validations at a flow level as follows:

1. Navigate to the component in the flow.

2. Right-click and select **Edit Test Data** from the context menu.

3. On the **Test Data** page, click **Open Pre Section**.

keyword	Object	OP Var Name	Function Na...	Logical Name	value1	value2	value3	value4	value5	value6

Component Pre Data

4. Specify the validations that should be triggered before the web service request is sent to the Oracle Utilities application.

Enter Component pre TestData

5. On the **Test Data** page, click **Open post Section**.

Component PostData

6. Specify the validations that should be triggered after a response comes back from the Oracle Utilities application.

Logging and Reporting

Oracle Utilities Testing Accelerator provides the following types of logging and reporting:

- **Test execution log file**: The test execution logs are created in the **Logs** folder and separate logs are generated for each flow.

- **Email report in HTML format**: The test execution email provides brief information about the overall test execution. It comprises the following:

    - Test step

    - Test data

    - Result (Pass/Fail)

The following figure is a snippet of the email generated.

Logging and Reporting Result

Handling the List Elements

The list elements of a schema should be defined using the keyword WS-SETXMLLISTELEMENT.

Consider the following partial schema. Note that the node usageDetails has a usagePeriods list element which in turn has another list element serviceQty and other non-list nodes (leaf nodes) (such as startDateTime, standardStartDateTime, endDateTime, etc.,). The list node serviceQty has non-list nodes such as seq, uom, tou, etc.

```xml
<usageDetails>
    <usagePeriods>
        <serviceQty>
            <seq>1</seq>
            <uom>TH</uom>
            <tou>ON</tou>
            <sqi>LOSSADJ</sqi>
            <qty>103.772922</qty>
        </serviceQty>
        <serviceQty>
            <seq>2</seq>
            <uom>TH</uom>
            <tou>SH</tou>
            <sqi>LOSSADJ</sqi>
            <qty>61.976037</qty>
        </serviceQty>
        <serviceQty>
            <seq>3</seq>
            <uom>TH</uom>
            <tou>OFF</tou>
            <sqi>LOSSADJ</sqi>
            <qty>189.281789</qty>
        </serviceQty>
        <startDateTime>2012-12-01T02:00:00</startDateTime>
        <standardStartDateTime>2012-12-01T02:00:00</standardStartDateTime>
        <endDateTime>2013-02-01T02:00:00</endDateTime>
        <standardEndDateTime>2013-02-01T02:00:00</standardEndDateTime>
        <usageRequestType>C1IS</usageRequestType>
    </usagePeriods>
    <usagePeriods>
        <serviceQty>
            <seq>1</seq>
            <uom>TH</uom>
            <tou>ON</tou>
            <sqi>LOSSADJ</sqi>
            <qty>103.772922</qty>
        </serviceQty>
        <serviceQty>
            <seq>2</seq>
            <uom>TH</uom>
            <tou>SH</tou>
            <sqi>LOSSADJ</sqi>
            <qty>61.976037</qty>
        </serviceQty>
        <serviceQty>
            <seq>3</seq>
            <uom>TH</uom>
            <tou/>
            <sqi/>
            <qty>355.030748</qty>
        </serviceQty>
        <serviceQty>
            <seq>4</seq>
            <uom>TH</uom>
            <tou>OFF</tou>
            <sqi>LOSSADJ</sqi>
            <qty>189.281789</qty>
        </serviceQty>
        <startDateTime>2012-12-01T02:00:00</startDateTime>
        <standardStartDateTime>2012-12-01T02:00:00</standardStartDateTime>
        <endDateTime>2013-02-01T02:00:00</endDateTime>
        <standardEndDateTime>2013-02-01T02:00:00</standardEndDateTime>
        <usageRequestType>C1IN</usageRequestType>
    </usagePeriods>
</usageDetails>
```

Sample Partial Schema

To define this schema in the component, consider the non-list nodes and enter a row for each of them, with the keyword as WS-SETXMLLISTELEMENT and Attribute value as the full xpath of the element, making sure to enter the appropriate Display names.

WS-SETXMLLISTELEMENT	▼	Select Object	▼	uom	usageDetails/usagePeriods/serviceQty/uom
WS-SETXMLLISTELEMENT	▼	Select Object	▼	tou	usageDetails/usagePeriods/serviceQty/tou
WS-SETXMLLISTELEMENT	▼	Select Object	▼	sqi	usageDetails/usagePeriods/serviceQty/sqi
WS-SETXMLLISTELEMENT	▼	Select Object	▼	qty	usageDetails/usagePeriods/serviceQty/qty
WS-SETXMLLISTELEMENT	▼	Select Object	▼	startDateTime	usageDetails/usagePeriods/startDateTime
WS-SETXMLLISTELEMENT	▼	Select Object	▼	standardStartDa'	usageDetails/usagePeriods/standardStartDateTime
WS-SETXMLLISTELEMENT	▼	Select Object	▼	endDateTime	usageDetails/usagePeriods/endDateTime
WS-SETXMLLISTELEMENT	▼	Select Object	▼	standardEndDat	usageDetails/usagePeriods/standardEndDateTime
WS-SETXMLLISTELEMENT	▼	Select Object	▼	usageRequestTy	usageDetails/usagePeriods/usageRequestType

Defining Schema

Note: If any of the list nodes repeat (serviceQty occurrs thrice inside usagePeriods, which in turn occurrs twice in usageDetais), do not define the elements multiple times in the component definition. The number of occurrences can be controlled in the test data (as defined in the Entering Test Data section).

Entering Test Data

On the test data page, each of the list nodes (usageDetails, usagePeriods and serviceQty for example) has an **Add** button next to them and are expandable. Expand the list node to view the children of that particular node.

For example: Expand usageDetails to view usagePeriods, and expand usagePeriods to view serviceQty, startDateTime, standardStartDateTime, etc.

Initially only one instance exists for all the list nodes. To add more nodes, click **Add** next to the desired element.

For example: To have two instances of usagePeriods inside usageDetails, click **Add** next to usagePeriods. There will be two usagePeriods nodes inside usageDetails, each of which will have the same content.

To view three serviceQty nodes in the first usagePeriods node and four in the second one:

1. Expand the first usagePeriods and add three serviceQty nodes.

2. Expand the second usagePeriods and add four serviceQty nodes.

The complete structure of the final schema is ready. You can add data to all the leaf nodes.

Add	◢ usageDetails	
Add	◢ usagePeriods	
Add	◢ serviceQty	
	seq	☐ 1
	uom	☐ TH
	tou	☐ ON
	sqi	☐ LOSSADJ
	qty	☐ 103.772922
Add	◢ serviceQty	
	seq	☐ 2
	uom	☐ TH
	tou	☐ SH
	sqi	☐ LOSSADJ
	qty	☐ 61.976037
Add	▶ serviceQty	
	startDateTime	☐ 2012-12-01T02:00:00
	standardStartDateTime	☐ 2012-12-01T02:00:00
	endDateTime	☐ 2013-02-01T02:00:00
	standardEndDateTime	☐ 2013-02-01T02:00:00
	usageRequestType	☐ C1IS
Add	◢ usagePeriods	
Add	▶ serviceQty	
Add	▶ serviceQty	
Add	▶ serviceQty	
Add	▶ serviceQty	

Entering Test Data

Creating GUI Based Components

To manually create a GUI based component, follow these steps:

1. Navigate to the component tree where the component has to be created.

2. Right-click the feature (release/product/module) in the component tree.

    Note: Create a new feature folder if it is not found in the delivered tree structure.

3. Select **Create Component**.

4. Enter the name of the component in the **Component** field.
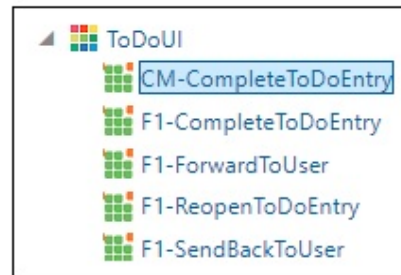
Creating Component

Note: The component name must be prefixed with 'CM' and the **Tags** field should have a CM tag for every component. The tagging enables porting the custom components to latest Oracle Utilities Testing Accelerator release.

For information about extending components, refer to the Copying Components section.

5. Select **User Interface** in the **ComponentType** drop-down list.

6. Enter a description in the **Description** field.

7. Click **Attach Code** to add the metadata. The **Component** window is displayed.

8. Create component definitions.

9. Click **Save & Unlock** to save and create the component.

Following is an example to create the CM-CompleteToDoEntry component under the ToDoUI feature for the Core product:

1. Navigate to **UTA 6.0.0.0** > **CORE 6.0.0.0 Portfolio** > **CORE UI 6.0.0.0** > **ToDoUI**.

2. Right-click the **ToDoUI** module.

3. Select **Create Component**.

4. Enter CM-CompleteToDoEntry in the **Component** field.

5. Follow steps 5 to 9 as mentioned in the procedure above.

Creating a Component Definition for GUI Components

A user interface component consists of several component definition lines. Each component definition line comprises of a keyword, object, display name, attribute values, default data, function name, and output parameters.

The following list describes each entity in a component definition:

- **Keyword**: The step to be performed. Example: WS-SETVARIABLEFROM RESPONSE, WS-VALIDATE, etc

- **Object**: The Oracle Utilities Testing Accelerator function library name from where the function is called.

- **Display Name**: The component definition (mandatory).

- **Attribute Values**: The xpath/ID of the UI element (mandatory).

  For example: If ID is specified, specify the attribute value as id;TD_ENTRY_ID, where TD_ENTRY_ID is the unique ID of the UI element being defined.

  If xpath is specified, it can be provided similar to //li[@id='CI_ADMINMENU_topMenuItem0x18']/span

  > **Note**: If the attribute values contain special characters (such as '$'), the character should be prefixed by the backslash ('\') character.
  >
  > For example: To input the attribute value as id;ZONE_PRM\:0$ZONE_PARM_VAL, specify it as id;ZONE_PRM\:0\$ZONE_PARM_VAL.

- **Default Data**: The default data used in the component definition.

- **Function Name**: The function name called from the library.

- **Output Parameters**: The output in the form of a variable.

  For more options, refer to Appendix D: Generating Re-runnable Test Data.

- **Tooltip**: The data presented as a tool tip during the flow creation.

The following figure shows the **Component** page with the available component definitions.

Components Content Areas

Component Name : F1-CompleteToDoEntry   Component Status : Approved

Rows to Add : 1   [Add Rows] [Save] [Save & Unlock] [Submit For Approval] [Delete] [Approve]

Page 1 of 1 (1-17 of 17 items) |< < [1] > >|

S.No		Insert	Keyword		Object		DisplayName	AttributeValues	DefaultData	OutputParameter	FunctionName
1	☐		SETAPPTYPE	▼	WEB	▼					Select Function
2	☐		FUNCTIONCALL	▼	OuafCoreLib	▼	Navigate to To		To Do Entry		navigatePageThroughMen
3	☐		WAIT	▼	NORMAL	▼					Select Function
4	☐		FUNCTIONCALL	▼	OuafCoreLib	▼	Function Call				switchToWindow
5	☐		WAIT	▼	NORMAL	▼					Select Function
6	☐		SETTEXT	▼	EDIT	▼	Enter Id	id;TD_ENTRY_ID			Select Function
7	☐		CLICK	▼	BUTTON	▼	Click search	id;BU_criteria_tc			Select Function
8	☐		FUNCTIONCALL	▼	OuafCoreLib	▼					windowClose
9	☐		FUNCTIONCALL	▼	OuafCoreLib	▼	tabPage		tabPage		switchToFrame
10	☐		WAIT	▼	NORMAL	▼					Select Function
11	☐		CLICK	▼	BUTTON	▼		id;COMPLETE_S			Select Function
12	☐		SWITCHTO	▼	STARTFRAME	▼	LOG_GRID	LOG_GRID			Select Function

Component Definition Page

Add the required component definition lines using the **Keyword** drop-down list to define the web services based component.

For a list of keywords used to define the GUI based components, refer to Appendix B: GUI Component Keywords.

The following example shows different component lines created for the CM-CompleteToDoEntry component.

1. Select SETAPPTYPE in the **Keyword** drop-down list to define the application type.

2. Select **Web** in the **Object** drop-down list to denote that it is a web services based component.

3. Select the WS-LOGMESSAGE keyword to log comments in component definition. This helps in debugging the script code for that component.

4. Add more component definition lines as needed and select appropriate keywords based on the GUI page that the component represents.

5. Click **Save**.

Creating REST Web Service Components

To create a REST web service component:

1. Navigate to the component tree where the component has to be created.

2. Right-click the feature (release/product/module) in the component tree.

> **Note**: Create a new feature folder if it is not found in the delivered tree structure.

3. Select **Create Component**.

4.   Enter the name of the component in the **Component** field.

   Note: The component name must be prefixed with 'CM' and the Tags field should have a CM tag for every component. The tagging enables porting the custom components to latest Oracle Utilities Testing Accelerator release.

   For information about extending components, refer to the Copying Components section.

5.   Select **REST SERVICE** in the **ComponentType** drop-down list.

6.   Enter a description in the **Description** field.

7.   Click **Attach Code** to add the metadata. The **Component** window is displayed.

8.   Create component definitions.

9.   Click **Save & Unlock** to save and create the component.

Once the component is created, use it in a flow by dragging and dropping the component into the desired location of the **Flow Definition** page. Once the component has been added to the flow, you can enter the test data for the component.

Entering Test Data

To enter the test data:

1.   Navigate to the **Flows** tab.

2.   On the left pane, expand the desired flow from the flow tree.

3.   Right-click the flow and select **Create/Update Flow Structure**.
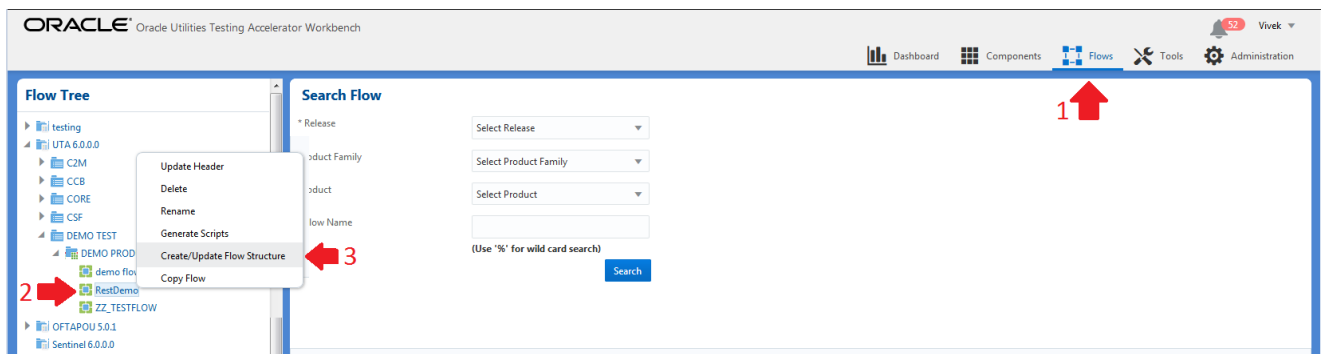
4.   Expand the flow on the right pane.

5.   Right-click a REST component and select **Edit test data**. The **Rest Test Data** window opens with four tabs:

   •   Pre Validations

   •   Test Data (Default tab)

   •   Body

   •   Post Validations
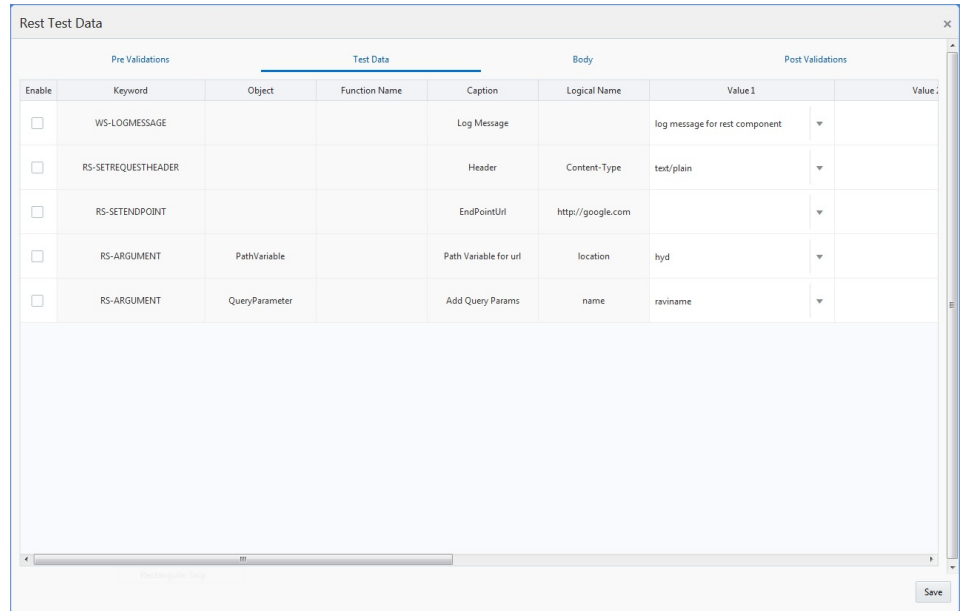
The table below provides description of each of the tabs.

Pre Validations/Post Validations	Test Data	Body
Lists the validations performed pre or post (before/after) the component is executed. For a new component, this section is empty. Add validations by clicking **New Row**. The **Object** column in each added row contains a list of functions used for validation. Specify the parameters for the functions by providing value against the **Value** column. Use the **OP Variable Name** column to store the output of the function, used in test data of the current component or in subsequent components in the flow. **OP Variable Name** should be unique for a flow.	Used to enter the component's test data. Enter plain text as data or use the value of an already defined global variable by choosing the variable name from the drop-down list.	Allows to choose whether the REST service invocation uses form data, raw data, or binary data. • Form data - A table with key-value pairs. These are sent as body with the REST request. • Raw data - Sends request text that cannot be sent in as either form data or binary data. • Binary data - A single flow attachment can be added as body data in REST.

Pre Validations/Post Validations	Test Data	Body
To use an **OP Variable Name** declared in the previous scenarios/ components, click the **Values** drop-down list. You can also provide a custom value by typing in the value into the column.		

After the test data entry is complete, click **Save**.

Copying Components

The components delivered can be customized; however, modifying the existing components is not a good practice.

A component can be extended by making its copy and saving it with a different name prefixed and tagged by CM, and then adding or modifying the metadata or key words as follows:

1. Right-click an existing component and select **Copy Component**.

2. Select and right-click a module.

3. From the context menu, select **Paste Component**.

   If the component name already exists in the module, a prompt is displayed to provide a new name to the component.

4. Click **Save as New Component**.

   The component is copied successfully.

Chapter 6

Creating Test Flows

Test flows are actual business tests executed on the application under test. The flows are assembled in Oracle Utilities Testing Accelerator by using predetermined components and are updated with data to guide the flow execution.

A test flow consists of one or more scenarios, which in turn consist of one or more components.

This chapter describes the steps to create a flow, including:

- Creating Flows

- Creating Scenarios

- Adding the Email Capabilities to Flows

- Support for HTTPS Web Services

- Support for Integration Flows

- Executing Test Flows

- Executing Flows from Command Line

- Encrypting Passwords

Creating Flows

You can create a flow in either of the following ways:

- By drag-and-drop of components into a flow (Creating Flows By Dragging-and-Dropping Components)

- By using the behavior driven development scenario (Creating Flows Using Behavior Driven Development Scenario)

Creating Flows By Dragging-and-Dropping Components

Before creating a flow, identify the components required to create the flow.

> **Note**: The components delivered with Oracle Utilities Testing Accelerator have to be extended or new components have to be created.

To create a flow, follow these steps:

1. Navigate to the product in the flow tree to create the flow.

2. Right-click the product and select **Create Flow**.

3. In the **Create Flow** pane, enter the **Flow Name**, **Flow Type**, **Tags**, and **Description**.

4. Save in either of the following ways:

   - **Save**: Saves the flow and redirects to the **Search Flow** page.

   - **Create Structure**: Creates the flow with a default scenario and redirects to the **Flow Structure** page.

5. Expand the flow tree.

6. Drag and drop the components from the **Approved Components** pane to the **Flow Creation** pane. For information about adding scenarios to a flow, refer to the Creating Scenarios section.

7. The test data needs to be entered at the component level while defining a flow and before the flow is assembled.

   To add data for a component, right-click it and select **Edit Test Data**. Similarly, data can be added for the remaining components.

8. Enter the test data in the **Test Data** page. For more details, refer to the Flow-Level Validations section in Chapter 5: Creating Components.

   If the test data contains the double quotes character (" "), it needs to be escaped with another double quote character. For example: To enter My "Test Data", you need to enter it as My ""Test Data"".

9. Click **Save & Close** to return to the **Flow Creation** page.

Creating Flows Using Behavior Driven Development Scenario

This Oracle Utilities Testing Accelerator feature uses a training dictionary that contains entries against each component and the likely keywords that match the components. For more details, refer to the Creating Behavior Driven Development Training Dictionary section.

Use the Behavior Driven Development (BDD) scenario to create a flow definition, review it, modify it as needed, and then finalize it thereby creating the flow.

The following is a typical BDD scenario used to create a flow that has two scenarios (pre-requisites and sanity testing) and each scenario has the standard BDD notation of Given-When-Then that defines the test case.

Feature: Flow

This flow can be used in an initial installation environment. It installs minimal data to perform the end-to-end sanity test flows.

Scenario Outline: Pre-Requisites

> **Given** user is already authorized
>
> **When** he performs Bundle Import
>
> **Then** user configures installation options
>
> **Then** user changes the display profile

Scenario: Sanity Testing

> **Given** there is 1 coffee left in the machine
>
> **And** I deposited 1$
>
> **When** I press the coffee button
>
> **Then** I should be served a **"coffee"**

To create a flow using the BDD scenario (mentioned above), follow these steps:

1. Navigate to the **Flows** tab.

2. Expand the **Flow** tree (on the left pane) and right-click the product for which the flow needs to be created.

3. Click **Create Flow** in the context menu.

4. Enter the necessary details for the flow (such as flow type, flow name, and tags).

5. Click **Create Structure.**

6. Click **BDD** on the right pane.

7. Enter the BDD scenario in the text editor.

8. Click **Generate Flow-Tree**.

   This creates a flow definition in the **Flow Creation** section. You can see it to the right of the **Generate Flow-Tree** button.

9. Review the flow definition generated.

10. Replace the components that were incorrectly suggested by the application.

11. Right-click the component to be replaced and select the alternate components that are suggested.

12. After confirming that the flow definition matches the requirement, click **Finalize Flow** to create the flow.

Creating Behavior Driven Development Training Dictionary

Oracle Utilities Testing Accelerator uses a Behavior Driven Development (BDD) Training dictionary to suggest components that best match the BDD scenario used to generate a flow. This training dictionary should contain one or more entries for every component in the application, more entries per component increase the accuracy with which the component will be matched with a given BDD scenario.

Few sample entries for a F1-Login component are as follows:

• F1-Login the logged in user

• F1-Login the authorized user

- F1-Login the user currently logged into the application

- F1-Login user login

Based on these entries, Oracle Utilities Testing Accelerator suggests F1-Login as the component when any of the keywords match the given entries. Each time you replace the suggested component with another component, Oracle Utilities Testing Accelerator makes a new entry into the training file and suggests more accurately when those keywords are used.

You can specify the location of the training file in the application.properties file in the folder where Oracle Utilities Testing Accelerator was installed.

Creating Scenarios

To create a scenario, do the following:

1. Navigate to the flow to be modified.

2. Select **Create/Update Flow Structure**.

3. Select and right-click the flow or a scenario inside the flow. You can create a scenario from the **Flow** menu or from the **Scenario** menu.

4. Click **Add Scenario** from the **Flow** menu.

   Alternatively, click **Add Scenario Above/Add Scenario Below** from the **Scenario** menu.

5. Enter the new scenario name.

6. Click **Go**.

Using Global Variables

This section explains the usage of global variables to pass data across components.

The component M1-CrewShift is a dependent component and during runtime needs The IDs of the M1-MobileWorker, M1-Vehicle, and M1 MultiPersonCrew components in addition to the other data.

To add component references to a dependent component (M1-CrewShift), follow these steps:

1. To add the MobileWorker ID, select **gVar_mobileWorkerId1** from the **Value1** drop-down list against the **resourceAllocationList/resourceId** display name.

2. To add VehicleId, select **gVar_vehicleId1** from the **Value2** drop-down list against the **resourceAllocationList/resourceId** display name.

3. To add MultiPersonCrewId, select **gVar_CrewId** in the **Value 1** drop-down list against the **crewId** display name.

Invoking OpenScript scripts

Manual OpenScript scripts can be triggered from within an Oracle Utilities Testing Accelerator component by using the invokeOpenScript function available in the OUTSPCORELIB library.

Create an OpenScript.properties property file under the existing Oracle Application Testing Suite (OATS) client installation. Make sure the property file has the following entries.

Property	Description
parentFolder	Path of the base folder where all OpenScript scripts exist

Property	Description
runScriptBatLocation	Location of the runOpenScript_OUTA.bat batch file.
	This file is present in the installation folder as runOpen-Script_ OUTA.bat. Copy it to the OpenScript folder inside the OATS client installation folder.

In the configuration.properties file located in the Oracle Utilities Testing Accelerator client folder structure, add a new entry to point to the OpenScript.properties file. A sample entry is as below.

```
openScriptPropertiesLocation=<Full path of the folder where the
OpenScript.properties can be found>
```

In the component from where the OpenScript script is invoked, update the component definition to include an additional definition entry. Make sure the entry is the format mentioned below.

```
KEYWORD =FUNCTIONCALL, OBJECT=OUTSPCORELIB,
FUNCTION=invokeOpenScript
```

After the component definition is saved, navigate to the test data page of the component in a flow and provide the test data to the added component definition page.

- *Parameter 1* - Relative path of OpenScript script's jwg file to be invoked, with respective to the path mentioned in parentFolder property from the OpenScrip.properties file.

- *Parameter 2* - Relative path of the app.properties file, with respective to the path mentioned in parentFolder property from OpenScript.properties.

- *Parameter 3* - Either 1 or 0 to indicate whether script execution should stop or proceed based on the success/failure of OpenScript script.

The app.properties file contains all the parameters needed to be passed to the OpenScript script.

For example:

```
accountId=1232343433
propertiesPath=absolute path for playback settings file
```

Flow Lifecycle

The flow lifecycle begins once a flow is created in Oracle Utilities Testing Accelerator. It can exist in one of the several possible lifecycle states as shown in the following diagram.
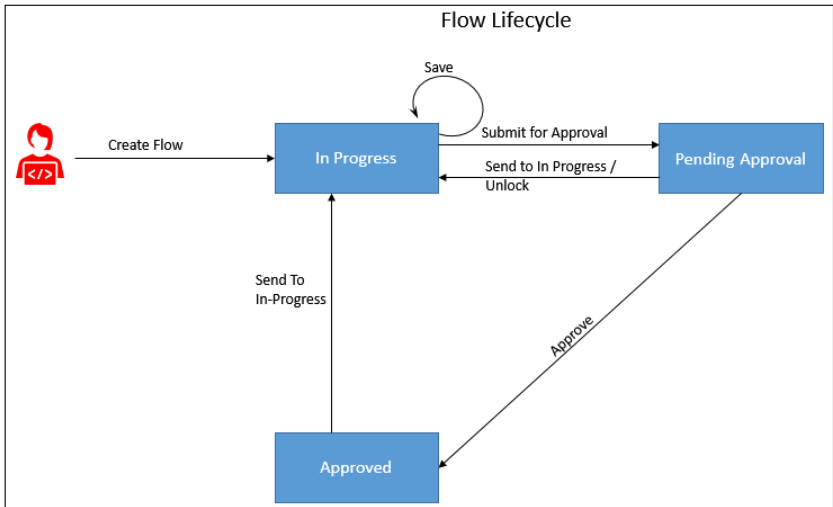
Flow Lifecycle

The state of a flow determines the actions that can be performed on the component. The following table summarizes the component states, and the possible actions and roles that can take the actions.

Flow Lifecycle State	Permitted Actions	Role	Resultant State (after action)
In Progress	Submit for Approval	Developer, Approver, Administrator	Pending Approval
Pending Approval	Send to In Progress	Developer, Approver, Administrator	In Progress
	Unlock	Developer, Approver, Administrator	In Progress
	Approve	Approver, Administrator	Approved
Approved	Send to In Progress	Developer, Approver, Administrator	In Progress

Locking/Unlocking Flows

A flow is/can be locked in the following scenarios:

• To prevent any other users from editing the flow until the flow is complete.

• By default when the flow is submitted for approval.

• If the flow is unlocked while in the 'Pending Approval' state, its state is changed back to 'In Progress'. However, if it is moved to 'In Progress' state from 'Pending Approval' state, it stays locked until the user unlocks it.

Click the ⬤ icon to lock/unlock a flow in the Oracle Utilities Testing Accelerator application.

Note that scripts can be generated only when the flow is in an "Approved" state.

Copying Flows

To copy a flow from one product to another product(s):

1. Login to the application.

2. Navigate to the **Flows** menu.

3. In the left navigation pane, expand the flow to be copied.

4. Right-click the flow to be copied and select **Copy Flow**.

5. Navigate to the product to which the flow needs to be copied.

6. Right-click the product and select **Paste Flow**.

7. In the pop-up window, enter the name for the new flow.

8. Click **Paste flow**.

Reordering Components in a Flow

Note that a flow needs to be "In progress" for components to be re-ordered. You cannot re-order components in a flow that is locked by another user.

To change the sequence of components in a scenario:

1. Log into the application.

2. Navigate to the **Flows** menu.

3. In the left pane, right-click the flow for which components have to be reordered.

4. Select **Create/update Flow Structure**.

5. Click the **Reorder Flow**. A popup appears with the flow in it.

6. Expand the flow.

7. Right-click the component to be moved and select **Move Component.**

8. Move the selected component in any of the following ways:

    • Right-click another component in the flow and choose **Paste Above**.

    • Right-click another component in the flow and choose **Paste Below**.

    • Right-click a scenario in the flow and choose **Paste**. This will move the selected component to the first position in the scenario.

9. After reordering the components, click **Save** to save the modified flow.

The popup closes and the flow tree is refreshed to reflect the correct order of components.

Copying Test Data from One Component to Another in a Flow

To copy the test data from one instance of a component to another instance of the same component within and across the scenario/flow:

1. Log into application and navigate to the **Flows** tab.

2. In the left navigation pane, right-click the flow and select **Create/update Flow Structure**.

3. Expand the flow.

4. Right-click a component from which you want copy the test data and select **Copy Test Data**.

5. Navigate to the component in the flow.

6. Right-click the component where you want to paste the test data and select **Paste**.

Test Data Management

The test data can be mentioned in the flow meta data. The Oracle Utilities Testing Accelerator code generator generates Oracle Utilities Testing Accelerator databanks (CSV files) for each component. Number and names of the columns in the generated databanks are based on the test data provided. The databanks can be updated with new data before test execution.

Sample CSV File

The generated script for the test flow can be executed for multiple sets of data. The data sets have to be provided in the component databank CSV for each of the component. The first data set for a flow will be generated by the script generator using the Oracle Utilities Testing Accelerator data.

Case 1: In the flow, the component is called just once and has repeated list elements (such as location). The following figure shows the CSV generated for the component.

Case 1: CSV for Component

The following figure shows the CSV after adding a second set of data. (Since the data has repeating list elements, the second data set starts two rows after the first with the prefix SET2_).

Case 1: CSV After Adding Second Data Set

Case 2: The component is called more than once in the flow and it does not have repeating list elements. The following figure shows the CSV generated for the component. In the figure, the text enclosed in the curly brackets is the variable.

DBRowSearch_ID	taskId	bo	boStatus
SET1_100000110	{{AssgnmntId}}	M1-Assignment	ENROUTE
SET1_100000111	{{AssgnmntId}}	M1-Assignment	ONSITE
SET1_100000112	{{AssgnmntId}}	M1-Assignment	COMPLETED

Case 2: CSV for Component

Note: The **taskId** column points to the **AssgnmntId** global variable declared within the curly brackets. The value stored in **AssgnmntId** will be set from the execution or the previous component and stored in **AssgnmntId**.

The following figure shows the CSV values after adding the second data set. (since the data has repeating list elements, the second data set starts two rows after the first with the prefix SET2_).

DBRowSearch_ID	taskId	bo	boStatus
SET1_100000110	{{AssgnmntId}}	M1-Assignment	ENROUTE
SET1_100000111	{{AssgnmntId}}	M1-Assignment	ONSITE
SET1_100000112	{{AssgnmntId}}	M1-Assignment	COMPLETED
SET2_100000110	{{AssgnmntId}}	M1-Assignment	ENROUTE
SET2_100000111	{{AssgnmntId}}	M1-Assignment	ONSITE
SET2_100000112	{{AssgnmntId}}	M1-Assignment	COMPLETED

Case 2: CSV after adding Second Data Set

Adding the Email Capabilities to Flows

The test execution report can be sent to users as an email. To add email capabilities in a flow, add the component line mentioned in the following table towards the end in the flow.

Usage Details	Value
Keyword	FUNCTIONCALL
Object	wSCOMMONLIB
Function Name	generateAndSendReport

The email related properties have to be specified in the configuration.properties file located under <UTA-CLIENT>/etc folder, where <UTA-CLIENT> is the location of the folder where the UTA_Client_FolderStructure.zip was extracted.

Update the following values as mentioned to configure the email:

```
#Email Details
gStrSMTP_HOST_NAME=<mention your SMTP server details here>
gStrSMTP_PORT=<mention SMTP port here>
gStrTO_EMAIL_RECIPIENTS=<mention target user/group email id>
```

Support for HTTPS Web Services

While connecting to the edge applications that use the HTTPS protocol, before executing the Oracle Utilities Testing Accelerator scripts, the security certificate should be saved on the system from where the Oracle Utilities Testing Accelerator test cases are being executed. Register the certificate in the Java security certificates repository.

To import the security key store into Java key store:

1. Enter the URL (HTTPS) of the application in the browser (Example: Internet Explorer).

2. Click **Continue to this Website (not recommended)** link on the **Security certificate** page.

3. Click **Certificate error** in the address bar.

4. Click the **View certificates** link on the **Certificate Invalid** pop-up window.

5. On the **Details** tab, click **Copy to File**.

6. Click **Browse** and select the file you want to export. Click **Next**.

7. Review the settings and click **Finish**.

8. Login to the machine where this certificate has to be imported into the Java key store, and open the command prompt.

9. If the Java path is not set in the environment variables, navigate to the Java/jdk/bin directory and execute the following command:

```
keytool -import -alias <Alias Name> -file <path of the file which
we exported in Step 7> -keystore <Java keystore path>
```

10. Enter "changeit" as the **Password**.

11. Click **Yes** to import the certificate. The property file attributes for HTTPS requests are as follows:

```
##Handling Https WSDL - Java key Store
gStrJavaKeyStorePath=C:\\jdk8\\jre\\lib\\security\\
gStrJavaKeyStorePwd=changeit
```

The setup is ready to process the HTTPS requests.

Support for Integration Flows

To test an end-to-end flow, the functional testing typically involves accessing different applications integrated for running the flow. In order to execute the integration tests, create flows that span multiple applications. These flows send/receive information to and from different applications.

To perform complete end-end tests, add the URL as an attribute in the properties file.

configuration.properties

```
# Integration Environments gStrMWMApplication=https\://<server
name>\:<port>/ouaf/webservices/ gStrCCBAppliation=https\://<server
name>\:<port>/ouaf/webservices/ gStrMDMAppliation=https\://<server
name>\:<port>/ouaf/webservices/
```

Following is the example where the Oracle Utilities Customer Care and Billing service AT-C1Premise is called.

Test data:

```
Calling CCB service: gStrCCBAppliation/AT-C1Premise
Calling MWM Service: gStrMWMApplication/AT-M1CrewShift
```

Note that this test data is a combination of environment and application service names.

In cases where the integration environments used for testing have different authentication credentials (userID/password), support is provided for setting the username and password for each environment. To enable this support, in the configuration.properties file, provide the user name/password variable prefixed with the environment URL variable, separated by an underscore.

Following is an example that extends the above integration environments example to use different user names and passwords for each environment:

configuration.properties

```
# Integration Environments gStrMWMApplication=https\://<server
name>\:<port>/ouaf/webservices/
gStrCCBAppliation=https\://<server name>\:<port>/ouaf/webservices/
gStrMDMAppliation=https\://<server name>\:<port>/ouaf/webservices/
```

Setting user name/password for a Oracle Utilities Customer Care and Billing service:

```
gStrCCBAppliation_gStrApplicationUserName=<%CCBUsername%>
gStrCCBAppliation_gStrApplicationUserPassword=<%CCBPassword%>
```

Setting user name/password for a Oracle Utilities Mobile Workforce Management service:

```
gStrMWMApplication_gStrApplicationUserName=<%MWMUsername%>
gStrMWMApplication_gStrApplicationUserPassword=<%MWMPassword%>
```

> **Note**: If the integration flow spans across multiple environments that use common user credentials (user name and password), then the above setting is not required.

The sample configuration file in this case is as follows.

configuration.properties

```
# Integration Environments
gStrMWMApplication=https\://<server name>\:<port>/ouaf/webservices/
gStrCCBApplication=https\://<server name>\:<port>/ouaf/webservices/
gStrMDMApplication=https\://<server name>\:<port>/ouaf/webservices/
gStrApplicationUserName=<%CommonUsername%>
gStrApplicationUserPassword=<%CommonPassword%>
```

Executing Test Flows

This section explains the steps to execute a test flow.

- Generating Oracle Utilities Testing Accelerator Scripts
- Importing the Generated Oracle Utilities Testing Accelerator Script into Eclipse IDE

Generating Oracle Utilities Testing Accelerator Scripts

To generate Oracle Utilities Testing Accelerator scripts for a flow:

1. Login to the Oracle Utilities Testing Accelerator application.

2. On the flow tree page, right-click a flow and select **Generate Scripts**.

   Note that the **Generate Scripts** option is available only for those flows in "Approved" state.

   A zip file containing the generated flow is downloaded by the browser.

The Oracle Utilities Testing Accelerator scripts generated for the test flow have the following structure:

- **Databank** folder - The databanks (text files with comma separated values (.csv)) for the flow that was downloaded. Each component in the flow has a corresponding databank(s) file generated that contains the test data.

- **src** folder - The generated script files for the flow.

Importing the Generated Oracle Utilities Testing Accelerator Script into Eclipse IDE

To import the Oracle Utilities Testing Accelerator scripts into Eclipse IDE:

1. Launch the Eclipse IDE for Java Developers that has the Oracle Utilities Testing Accelerator Eclipse plugin installed.

   For instructions to install Eclipse IDE, refer to the Installing Oracle Utilities Testing Accelerator Client Runtime section in Chapter 3: Developing Metadata Driven Web Service Based Test Automation.

2. In the Eclipse IDE, right-click the **Project Explorer** panel and click **Import**.

3. Click **General** and select **Existing Projects into Workspace**.

4. Click **Next**.

5. Select **Select archive file:** and click **Browse**.

6. Navigate to the downloaded zip file and click **Open**.

7. Click **Finish**.

   You can now view the project in the **Project Explorer** panel.

8. Right-click the masterdrive and click **Run**.

   Important: Before executing GUI component based flows, make sure to download the ChromeDriver v2.40/geckodriver v0.20.1 and copy it into the 'drivers' folder mentioned in the Creating Oracle Utilities Testing Accelerator Client Runtime Folder Structure section in Chapter 3: Creating Test Flows.

   Download the drivers from the following URLs:

While executing the Sentinel script, if a failure occurs at any point, it is possible to resume execution from the point of failure by setting the runFromLastPointOfError flag to "true" in the script.

```java
package com.oracle.tests;

import static org.junit.Assert.assertNotNull;

public class F1_BundleImport extends FunctionalTestScript{

    WSCOMMONLIB wSCOMMONLIB = new WSCOMMONLIB();
    WSVALIDATELIB wSVALIDATELIB = new WSVALIDATELIB();
    OUTSPCORELIB oUTSPCORELIB = new OUTSPCORELIB();
    oUAFLIB oUAFLIB = new oUAFLIB();

    final static boolean runFromLastPointOfError = false;

    @BeforeClass
    public static void preSetup() throws SQLException{
        System.out.println("@BeforeExecution: Initializing Configurations Variables before Script Startup");
        WSCOMMONLIB.initalizeSetUp("F1_BundleImport", System.getProperty("user.dir"),runFromLastPointOfError);
        beginScript("F1_BundleImport");
```

The runFromLastPointOfError Flag

Executing Flows from Command Line

To execute the scripts downloaded from Oracle Utilities Testing Accelerator from the command line:

1. Navigate to the directory where the runtime folder is created.

    For information about the folder, refer to the Creating Oracle Utilities Testing Accelerator Client Runtime Folder Structure section in Chapter 3: Creating Test Flows.

2. Open a Windows command line window at this location.

3. Execute the flow as follows:

    ```
    java -jar utascriptwrapper.jar -f "<directory containing the flow>"
    -t "<Scenario name>"
    ```

4. After the script execution is complete, the Console shows the execution status.

    The detailed status can be found in the log file under the logs directory.

Encrypting Passwords

All the password fields (gStrApplicationUserPassword, gStrApplicationDBPassword, gStrJavaKeyStorePwd) in the configuration.properties file must be encrypted. Before encrypting the configuration.properties file, the keystore should be generated.

You can generate the keystore in either of the following ways:

- From Windows Explorer (Generating Keystore for Encryption from Windows Explorer)

- From Command Line (Generating Keystore for Encryption from Command Prompt)

Use the Password Encryptor Tool to encrypt the configuration.properties file. You can use the Password Encryptor Tool in either of the following ways:

- From Windows Explorer (Using Password Encryptor Tool From Windows Explorer)

- From Command Line (Using PasswordEncryptor Tool From Console/Command Line)

Generating Keystore for Encryption from Windows Explorer

To generate the keystore for encryption from Windows Explorer:

1. Launch **Windows Explorer**.

2. Navigate to the **<UTA-CLIENT-WORK-DIR>\tools** folder.

3. Double-click the **KeystoreGenerator.jar** file.

4. If the keystore and password store do not exist, they will be generated. Else, you will be prompted to confirm to overwrite the existing keystore and password store with the new one.

5. Click **Yes** to overwrite the existing keystore.

   To continue with the existing keystore and password store, click **No**.

If you regenerate a keystore, all the passwords should be re-encrypted using the Password Encryptor tool.

Generating Keystore for Encryption from Command Prompt

To generate the keystore for encryption from the command prompt:

1. Navigate to the <UTA-CLIENT-WORK-DIR>/tools folder.

2. Run the command: java -jar KeystoreGenerator.jar.

3. If the keystore and the password store for the keystore do not already exist, they will be generated. Else, you will be prompted to overwrite the existing keystore.

4. Enter **'y'** to generate a new keystore. Enter **'n'** if you want to continue using the existing keystore.

If you regenerate a keystore, all the passwords should be re-encrypted using the Password Encryptor tool.

Using Password Encryptor Tool From Windows Explorer

Important: Make sure to generate the keystore before encrypting the configuration.properties file. For instructions, refer to the Generating Keystore for Encryption from Windows Explorer section.

To use the password encryption tool from Windows:

1. Launch **Windows Explorer**.

2. Navigate to the **<UTA-CLIENT-WORK-DIR>\tools** folder.

3. Double-click the P**asswordEncryptor.jar** file.

   You will be prompted to enter the password.

4. After entering the password, click **OK**.

   A dialog box with encrypted text appears.

5. Copy this text and enter it as the value of the respective password field in the configuration.properties file.

Using PasswordEncryptor Tool From Console/Command Line

To use the password encryption tool from console/command line:

1. Navigate to **<UTA-CLIENT-WORK-DIR>/tools** directory.

2. Enter the following command:

   ```
   java -jar PasswordEncryptor.jar
   ```

   The tool prompts to enter a password.

3. Type in the password and press **Enter**. A dialog box with encrypted text appears.

4. Copy this text and enter it as the value of the respective password field in the configuration.properties file.

Configuring the Runtime Properties

The **configuration.properties** file is located in the <UTA-CLIENT-WORK-DIR>\etc\ folder. It is used to store the runtime test execution parameters, such as application URL, application access information, email configuration, etc.

To use the email functionality for receiving the test execution report, provide the following values as mentioned in the configuration.properties file.

```
#Email Details
gStrSMTP_HOST_NAME=
gStrSMTP_PORT=
gStrTO_EMAIL_RECIPIENTS=
```

To provide the test environment details, provide the following values:

```
# Application URL pointing to test execution
gStrApplicationURL =
gStrApplicationXAIServerPath=
gStrEnvironmentName=
```

To provide the application user information for login, provide the values for the following keys:

```
gStrApplicationUserName =
gStrApplicationUserPassword =<Encrypted Password>
```

If the test suite has any database-side validations, provide the database details as follows:

```
gStrApplicationDBConnectionString =
gStrApplicationDBUsername =
gStrApplicationDBPassword =<Encrypted Password>
```

The path for the output files generated for reporting is as follows:

```
# Output file details
gStrOutputFilePath =
gStrXSDFiles=
```

For properties related to the integration environment, refer to the Support for Integration Flows section.

Chapter 7

Development Accelerator Tools

This chapter describes the development accelerator tools available in this Oracle Utilities Testing Accelerator release:

- Component Export Tool

- Flow Export Tool

- Component/ Flow Import Tool

- Component Generation Tool

- Password Encryption Tool

Component Export Tool

This tool is used to export one or more components to another environment. Note that only components in "Approved" state can be exported.

To export a component pack:

1. Login to the application.

2. Navigate to the **Tools** tab.

3. Click **Export Components** in the left pane.

4. Select the **Release**, **Portfolio**, **Product**, **Module**, **Component Name**, **Tags** (example: CM) and **Owner Flag** as required.

5. Click **Export**.

   A prompt appears on the screen to open or save the generated zip file "component.zip".

6. Click **Save** to download the zip file.

The component has been exported as a .zip file.

Flow Export Tool

This feature is used to export one or more flows to another environment. Note that only flows in "Approved" state can be exported.

To export a flow:

1. Login to the application.

2. Navigate to the **Tools** tab.

3. Click **Export Flows** in the left pane.

4. Select the **Release**, **Portfolio**, **Product**, **Flow Name**, **Tags** (example: CM) and **Owner Flag** as required.

5. Click **Export**.

   A prompt appears on the screen to open or save the generated zip file "flow.zip".

6. Click **Save** to download the zip file.

The flow has been exported as a .zip file.

Component/ Flow Import Tool

This feature is used to import components and/or flows to another environment.

To import a component/flow:

1. Login to the application.

2. Navigate to the **Tools** tab.

3. Click **Import** in the left pane.

4. Drop the component/flow pack in to **Import** wizard in the right pane.

   When a file is selected/ dropped in the wizard, the file name appears.

5. Click **Save**.

6. If the component/flow already exists in the database, a pop-up is displayed giving a choice to continue or abort the process.

7. When you click **Cancel**, the import component/flow process is not triggered and it goes back to step 3 (you can still import it again).

8. When you click **OK** on the pop-up, the process of importing component/flows begins with progress bar.

The component/flow is imported successfully.

Component Generation Tool

This feature is used to generate components from WSDL.

To generate components:

1. Login to the application.

2. Navigate to the **Tools** tab.

3. Click **Generate Components** on the left pane.

4. Enter the data in the required fields.

5. Specify the number of rows to add and click **Add Rows**.

6. Enter the component name, tags, and description, and provide a WSDL URL.

7. Click **GenerateComp**.

The **WSDL Method** column is an operation in WSDL. The following figure shows the name of operation in WSDL.

```
▼<wsdl:portType name="D2-DetermineEstimatedAndHighLowScalarReadingsPortType">
  ▼<wsdl:documentation>
      D2-DetermineEstimatedAndHighLowScalarReadings version 1: Determine Estimated, Hi-Lo Scalar Readings
    </wsdl:documentation>
  ▼<wsdl:operation name= D2-DetermineEstimatedAndHighLowScalarReadings >
      <wsdl:input message= tns:D2-DetermineEstimatedAndHighLowScalarReadingsRequest"/>
      <wsdl:output message="tns:D2-DetermineEstimatedAndHighLowScalarReadingsResponse"/>
      <wsdl:fault name="fault" message="tns:Fault"/>
    </wsdl:operation>
  </wsdl:portType>
```

Upon successful component generation, a list of generated components and failed components is displayed.

Password Encryption Tool

The Password Encryption Tool is used to encrypt passwords for additional security.

This section provides the following details about the tool:

• Overview

• Executing the Password Encryption Tool

Overview

All the password fields (gStrApplicationUserPassword, gStrApplicationDBPassword) in the configuration.properties file and security.properties file, if any, must be encrypted.

Use the PasswordEncryptor tool to encrypt the password.

Executing the Password Encryption Tool

To encrypt plain text using the Password Encryption Tool, follow these steps:

1. Double-click the **PasswordEncryptor.jar** file. when asked for, enter the password.

2. Click **OK**. A dialog box with the encrypted text is displayed.

3. Copy this text and paste it in the respective password field in the configuration.properties or security.properties file.

Alternatively, you can perform the following steps in the Console:

1. Navigate to <ECLIPSE-WORKDIR>/tools.

2. Enter the following command:

    ```
    java -jar PasswordEncryptor.jar
    ```

3. When prompted, enter the password.

4. Press **Enter**.

5. Copy this text and paste it in the respective password field in the configuration.properties or security.properties file.

Chapter 8

Function Library Reference

This chapter lists the Oracle Utilities Testing Accelerator function libraries and functions available to create components and flows in Oracle Utilities Testing Accelerator Workbench for testing Oracle Utilities Testing Accelerator.

The following function libraries are described:

- CLOUDLIB
- OUTSPCORELIB
- WSCOMMONLIB
- WSVALIDATELIB

CLOUDLIB

This library includes functions to support the Oracle Utilities applications deployed in Oracle Cloud.

login

Logs into the Oracle Utilities Application Cloud environment (uses configuration.properties for URL, user name, password, etc).

```
login

Input Parameters: none
Return Type: void
```

importBundle

Imports a bundle into the Oracle Utilities Application Cloud environment. Note that the bundle has to be attached to the script.

```
importBundle ()

Input Parameters: strFilename - name of bundle file
Return Type: void
```

createTimeZone

Creates a TimeZone in the Oracle Utilities Application Cloud environment using data provided in the component.

```
createTimeZone

Input Parameters:None
Return Type: void
```

updateInstallOptionsWithTimeZone

Adds an existing TimeZone to the installation options in the Oracle Utilities Application Cloud environment using data provided in the component.

```
updateInstallOptionsWithTimeZone

Input Parameters: None
Return Type: void
```

OUTSPCORELIB

This library develops the component code and flows for web services and general applications. It includes functions with date and time processing and string processing capabilities, as well as database and file operations.

This section provides a list of the functions included in the library, along with their usage details.

runBatchFile

Executes an existing batch file.

Example:

```
runBatchFile("C://Test//Test.bat")

Input Parameters: String
Return Type: void
```

killBatchFile

Kills the batch process in execution.

Example:

```
killBatchFile ("cmd.exe")

Input Parameters: String
Return Type: void
```

getCurrentTimeInMilliSeconds

Gets the time in milliseconds.

Example:

```
getCurrentTimeInMilliSeconds ()

Input Parameters: <none>
Return Type: Sting
```

rand

Gets the random number for the given range.

Example:

```
rand(int lo, int hi) ()

Input Parameters: lo, hi
Return Type: int
```

randomStringWithGivenRange

Gets the random string in lower case for the given range.

Example:

```
randomStringWithGivenRange(int lo, int hi)

Input Parameters: lo, hi
Return Type: String
```

Randomstring

Generates the random string based on the parameters passed. 'lo' and 'hi' are the lowest and highest numbers to be used to generate the random string.

Example:

```
randomstring (lowerLim, higherLim)

Input Parameters: int lowerLim , int higherLim
Return Type: String
```

compare2Strings

Compares two strings and returns a boolean result based on the result of comparison.

> **Note**: This function returns "True" if strings provided are same. Else, it returns 'False'.

Example:

```
compare2Strings (String_A, String_B)

Input Parameters: String_A, String_B
Return Type: String
```

randomNumberUsingDateTime

Gets the random string with date and time in it.

Example:

```
randomNumberUsingDateTime()

Input Parameters: <none>
Return Type: String
```

getCurrentDateTimeWithGivenDateFormat

Gets the current date and time in the specified format.

Example:

```
getCurrentDateTimeWithGivenDateFormat(String dFormat)
getCurrentDateTimeWithGivenDateFormat("mm-dd-yyyy:hh.mm.ss")

Input Parameters: dFormat
Return Type: String
```

getDateDiffInSecsWithGivenDateFormat()

Gets the difference in the date.

Example:

```
getDateDiffInSecsWithGivenDateFormat(String dateStart, String
dateStop, String dFormat)
getDateDiffInSecsWithGivenDateFormat("12-13-2014", "12-29-2014",
"mm-dd-yyy")

Input Parameters: String dateStart, String dateStop, String dFormat
Return Type: String
```

getAdjustedTimeWithGivenDateTime

Gets the adjusted time with the given date and time.

Example:

```
getAdjustedTimeWithGivenDateTime(String dateTime, String offset,
String dFormat)
getAdjustedTimeWithGivenDateTime("12-13-2014", "-02:30","mm-dd-
yyyy")

Input Parameters: String dateTime, String offset, String dFormat
Return Type: String
```

getAdjustedTimeWithCurrentDateTime

Returns the date and time after adding the specified offset to the current date and time in the specified date/time format. Date and time are the inputs to this function.

Example:

```
getAdjustedTimeWithCurrentDateTime(String offset, String dFormat)
getAdjustedTimeWithCurrentDateTime("-2.30", "mm-dd-yyyy")

Input Parameters: String dateTime, String offset, String dFormat
Return Type: String
```

getAdjustedTimeWithGivenDateAndTime

Returns the date and time after adding the specified offset to specified date and time in the specified date/time format.

Example:

```
getAdjustedTimeWithGivenDateAndTime(String cuDate,String
cuTime,String offset, String dFormat)
getAdjustedTimeWithGivenDateAndTime("12-13-2014","12:15:00","-
2.30", "mm-dd-yyyy")

Input Parameters: String cuDate, String cuTime, String offset,
String dFormat
Return Type: String
```

addDaysToCurrentDateWithGivenFormat

Adds the number of days to the current date and returns the result in the specified format.

Example:

```
addDaysToCurrentDateWithGivenFormat(String noOfDays, String
dFormat)
addDaysToCurrentDateWithGivenFormat("45", "mm-dd-yyyy")

Input Parameters: String noOfDays, String dFormat
Return Type: String
```

serverDate

Gets the server date.

Example:

```
serverDate()

Input Parameters: <none>
Return Type: String
```

executeSQLQry

Executes SQL and returns the record set.

Example:

```
executeSQLQry(String Query)
executeSQLQry("SELECT * FROM EMP")

Input Parameters: String Query
Return Type: Result Set
```

executeSQLQryWithGivenDBDetails

Executes SQL and returns the record set.

Example:

```
executeSQLQryWithGivenDBDetails(String Query, String
ConnectionString ,String DBUsername,String DBPassword)
executeSQLQryWithGivenDBDetails("SELECT * FROM EMP", CONN_STR,
"system", "system00")

Input Parameters: String Query, String ConnectionString,String
DBUsername,String DBPassword
Return Type: Result Set
Exceptions: Database exception
```

serverTime

Gets the server time.

Example:

```
serverTime()

Input Parameters: <none>
Return Type: String
```

waitForTime

Waits for the specified time.

Example:

```
waitForTime(String strWaitTimeInMinutes)
waitForTime("15")

Input Parameters: String strWaitTimeInMinutes
Return Type: void
```

verifyLastBatchRun

Verifies if the batch is in execution in the last x minutes.

Example:

```
verifyLastBatchRun(String Batch_CD, String strMaXTimeToCheck)
verifyLastBatchRun("1234567890", "90")

Input Parameters: String Batch_CD, String strMaXTimeToCheck
Return Type: String
```

getCurrentOffsetTime

Gets the current offset time.

Example:

```
getCurrentOffsetTime(String cuDate, String cuTime, String
offset,String timeFormat)
getCurrentOffsetTime("12-13-2014", "12:30:00", "+2:30","mm-dd-
yyyy")

Input Parameters: String cuDate, String cuTime, String offset,
String timeFormat
Return Type: String
```

addDaysToAGivenDate

Adds days to the provided date.

Example:

```
addDaysToAGivenDate(String date, String noOfDays)
addDaysToAGivenDate("12-13-2014", "19")

Input Parameters: String date, String noOfDays
Return Type: String
```

randomNumber

Gets the random number.

Example:

```
randomNumber()
```

```
Input Parameters: <none>
Return Type: String
```

createFile

Creates file in the specified path.

Example:

```
createFile(String FilePath)
createFile("C:\Logs.txt")

Input Parameters: String FilePath
Return Type: void
```

getWaitConditionState

Waits for the specified time.

Example:

```
getWaitConditionState(long StartTime, float TimeInMinutes)
getWaitConditionState("12345L", "12.00")

Input Parameters: long StartTime, float TimeInMinutes
Return Type: boolean
```

compare2Files

Compares two files.

Example:

```
compare2Files(String strFileName_A, String strFileName_B)
compare2Files("C:\Logs01", "C:\Logs04")

Input Parameters: strFileName_A, String strFileName_B
Return Type: String
```

copyFile

Copies files from source to destination.

Example:

```
copyFile(String srcFilePath, String destFilePath)
copyFile("C:\temp.txt", "D:\temp.txt")

Input Parameters: strFileName_A, String strFileName_B
Return Type: void
```

deleteFile

Deletes a file.

Example:

```
deleteFile(String filePath)
deleteFile("C:\temp.txt")

Input Parameters: String filePath
Return Type: void
```

executeSQLQryUpdate

Executes SQL for the update query.

Example:

```
executeSQLQryUpdate(String Query, String ConnectionString,String
DBUsername,String DBPassword)
executeSQLQryUpdate("UPDATE EMP SET NAME="Oracle" where
EMPID='123'" CONN_STR, "system", "system00")

Input Parameters: String Query, String ConnectionString,String
DBUsername, String DBPassword
Return Type: String
```

getDistinctObjects

Returns the count of distinct objects in a specific column in a table.

Example:

```
getDistinctObjects(String tableName, String columnName, String
condition, String ConnectionString ,String DBUsername,String
DBPassword)
getDistinctObjects ("EMP", "EMPID" CONN_STR, "system", "system00")

Input Parameters: String tableName, String columnName, String
condition, String ConnectionString,String DBUsername, String
DBPassword
Return Type: String
```

setVariableValueUsingListIndex

Handles the resolving repeating elements in the response XML and retrieves the value(s) based on the parameters passed. The parameters passed are global variable (gVar1) and index value.

Example:

```
setVariableValueUsingListIndex(String listVariableName,String
index)
setVariableValueUsingListIndex("data1,data2,data3", 2)

Input Parameters: String listVariableName: List values separated by
comma
String index: the index number to retrieve value
Return Type: String: Value
```

closeConnections

Closes the database connection opened for database verification.

Example:

```
closeConnection()

Input Parameters: <none>
Return Type: <none>
```

executeSQLQryForSingleRecord

Executes the SQL query for a single record.

Example:

```
executeSQLQrySingleRecord(String Query,String recId)

Input Parameters: Query,recId
Return Type: String
```

appendStrings

Appends strings provided in the parameters.

Example:

```
appendStrings (String strValue1, String strValue2, String
strValue3, String strValue4, String strValue5, String strValue6

Input Parameters: string1, string2, string3, string4, string5,
                      string6
Return Type: String
```

getCurrentMonth

Gets the current month.

Example:

```
getCurrentMonth()

Input Parameters: none
Return Type: String
```

WSCOMMONLIB

This library performs common operations in the Oracle Utilities Testing Accelerator web services testing, such as composing request, sending request, composing email summary, converting it to HTML format, sending an email, and parsing WSDL.

> **Note**: This library does not have any component development functions other than the generateAndSendReport function that provides result reporting and email capabilities to the user. For more details. refer to the Logging and Reporting section in Chapter 5: Creating Components.

This section provides the functions included in the library, along with their usage details.

generateAndSendReport

Generates the HTML test execution report and sends the execution summary via email. The email settings can be specified in the configuration.properties file available in the /etc directory of the execution folder structure.

```
generateAndSendReport ()

Input Parameters: NA
Return Type: NA
```

WSVALIDATELIB

Use the WSVALIDATELIB function library to validate the test components (referred to as verification points) in the components. The library covers validation routines for string and XML elements in the returned response XML.

This section provides a list of functions in the library, along with the usage details.

elementListNotNull

Verifies if all the elements with the specified xpath in response are not null.

Example:

```
elementListNotNull(String xPath) elementNotNull(contact/
mobileNumber)
```

```
Input Parameters: String xPath
Return Type: void
```

elementListNull

Verifies if all the elements in response with the specified xpath are null.

Example:

```
elementListNull(String xPath) elementNotNull(contact/mobileNumber)

Input Parameters: String xPath
Return Type: void
```

validateXpathOccurrenceCount

Verifies if the specified xpath occurs the specified number of times in the response.

Example:

```
validateXpathOccurenceCount (String xpath,String expectedCount)
validateXpathOccurenceCount(contact/mobileNumber,20)

Input Parameters: String xpath,String expectedCount
Return Type: void
```

elementNotNull

Verifies if the specified element in response is null.

Example:

```
elementNotNull(String responseTag)
elementNotNull(mobileNumber)

Input Parameters: String responseTag
Return Type: void
```

elementIsNull

Verifies if the specified element in response is not null.

Example:

```
elementIsNull (String responseTag)
elementIsNull (mobileNumber)

Input Parameters: String responseTag
Return Type: void
```

elementValueEquals

Verifies if the specified element value in response is equal to the provided value.

```
elementValueEquals(String responseTag, String expectedValue)
elementValueEquals(mobileNumber, "1234567890")

Input Parameters: String responseTag, String expectedValue
Return Type: void
```

elementValueNotEquals

Verifies if the specified element value in response is not equal to the provided value.

Example:

```
elementValueNotEquals(String responseTag, String expectedValue)
elementValueNotEquals (mobileNumber, "1234567890")
```

```
Input Parameters: String responseTag, String expectedValue
Return Type: void
```

elementValueGreaterThan

Verifies if the specified element value in response is greater than the provided value.

Example:

```
elementValueGreaterThan(String responseTag, String valueToCompare)
elementValueGreaterThan("count","5")

Input Parameters: String responseTag, String valueToCompare
Return Type: void
```

elementValueGreaterThanEqualTo

Verifies if the specified element value in response is greater than or equal to the provided value.

Example:

```
elementValueGreaterThanEqualTo(String responseTag,String
valueToCompare)
elementValueGreaterThanEqualTo("totalRecords", "50")

Input Parameters: String responseTag, String valueToCompare
Return Type: void
```

elementValueLesserThan

Verifies if the specified element value in response is less than the provided value.

```
elementValueLesserThan(String responseTag,String valueToCompare)
elementValueLesserThan ("counter", "50")

Input Parameters: String responseTag, String valueToCompare
Return Type: void
```

elementValueLesserThanEqualTo

Verifies if the specified element value in response is less than or equal to the provided value.

Example:

```
elementValueLesserThanEqualTo(String responseTag,String
valueToCompare)
elementValueLesserThanEqualTo ("attempts", "10")

Input Parameters: String responseTag, String valueToCompare
Return Type: void
```

elementContains

Verifies if the specified element is available in the response.

Example:

```
elementContains(String responseTag,String valueToBeChecked)
elementContains("batchName", "F1-BILLING)

Input Parameters: String responseTag, String valueToCompare
Return Type: void
```

elementNotContains

Verifies if the specified element is not available in the response.

Example:

```
elementNotContains(String responseTag, String valueToBeChecked)
elementNotContains ("description", "billing")

Input Parameters: String responseTag, String valueToCompare
Return Type: void
```

reponseNotContains

Verifies if the specified value or element is not available in the response.

Example:

```
reponseNotContains(String value)
reponseNotContains("Failed")

Input Parameters: String responseTag, String valueToCompare
Return Type: void
```

responseContains

Verifies if the specified value or element is available in the response.

```
responseContains(String value)
responseContains("Exception")
Input Parameters: String responseTag, String valueToCompare
Return Type: void
```

Appendix A

Web Service Component Keywords

This chapter provides the list of keywords used in a web service based component.

- WS-SETWEBSERVICENAME
- WS-SETXMLELEMENT
- WS-SETXMLLISTELEMENT
- WS-SETVARIABLE
- WS-SETVARIABLEFROMRESPONSE
- WS-SETTRANSACTIONTYPE
- WS-LOGMESSAGE
- WS-CREATEWSREQUEST
- WS-PROCESSWSREQUEST
- WS-STARTPOLLWS
- WS-STOPPOLLWSIF

WS-SETWEBSERVICENAME

Sets the name of the application web service.

Use Case: Defines the web service to which the component's web service request is sent. The web service name is provided in the attribute values column during the component development. This service name is appended with the WebContainerURL to form a complete WSDL URL for processing the request. The WebContainerURL has to be specified in the flow runtime configuration property file.

Usage Details	Value
Keyword	WS-SETWEBSERVICENAME
Display Name	User Defined Display Name
Attribute Values	Web Service Name

WS-SETXMLELEMENT

Sets the element (Xpath) value in the web service request using either a variable or a value.

Use Case: Enables the web service creation request (XML) with the element values populated by setting each value for the defined element.

Usage Details	Value
Keyword	WS-SETXMLELEMENT
Display Name	User Defined Display Name
Attribute Values	Xpath of the element

WS-SETXMLLISTELEMENT

Sets the repeating list element (Xpath) value in the web service request using either a variable or a value.

Use Case: Enables the web service creation request (XML) with repeating list element values populated by setting each value set for the defined element list. The values are provided from the test data.

Usage Details	Value
Keyword	WS-SETXMLLISTELEMENT
Display Name	User Defined Display Name
Attribute Values	Xpath of the element

WS-SETVARIABLE

Sets a value to a global variable.

Use Case: Used for setting a value to a global variable used across the flow for validations or for setting XML elements. The values are provided from the test data.

Usage Details	Value
Keyword	WS-SETVARIABLE
Display Name	User Defined Display Name
Output Parameters	Variable Name

WS-SETVARIABLEFROMRESPONSE

Used to retrieve the XML element value from the response and stores it in a global variable for further processing.

Use Case: Enables use of a response value, such as ID from a component, as an input to a request in another component.

Usage Details	Value
Keyword	WS-SETVARIABLEFROMRESPONSE
Display Name	User Defined Display Name
Attribute Values	Xpath of the element in response
Output Parameters	Variable Name

WS-SETTRANSACTIONTYPE

Sets a value for the transaction type.

Use Case: Used to set a value to a transaction type variable used in the request XML to pass a request for specific operations, such as ADD, UPDATE, READ, DELETE, etc. The transaction type is provided from the test data.

Usage Details	Value
Keyword	WS-SETTRANSACTIONTYPE
Display Name	User Defined Display Name

WS-LOGMESSAGE

Used to set custom log messages in the execution results report.

Use Case: Provides the necessary extensibility to provide custom log messages for the generated results report, such as to identify the start and completion of a transaction, etc.

Usage Details	Value
Keyword	WS-LOGMESSAGE

Usage Details	Value
Display Name	User Defined Value
Attribute Values	Message

WS-CREATEWSREQUEST

Creates a web service request XML and stores it in the "WSDLXML" global variable.

Use Case: Enables the manipulation of the web service XML request generated before submitting it to the application for processing, giving greater flexibility in development.

Usage Details	Value
Keyword	WS-CREATEWSREQUEST
Display Name	User Defined Display Name
Attribute Values	Web Service Name

WS-PROCESSWSREQUEST

Sends the web services request and receives the response from the application for the specified WSDL URL.

Use Case: Posts the generated XML request from WS-CREATEWSREQUEST to the application and processes the response. This keyword performs the core process of the web services based request-response model.

Usage Details	Value
Keyword	WS-PROCESSWSREQUEST
Display Name	User Defined Display Name
Attribute Values	Web Service Name

WS-STARTPOLLWS

Starts the polling of the web services request and receives the response from the application for the specified WSDL URL. It takes two parameters, the first is for the total time for which polling should occur and the second is the interval between polls.

Use Case: Provides a means to run a loop to keep polling a web service for a specified time measure or till a condition is met (specified in WS-STOPPOLLWSIF).

Usage Details	Value
Keyword	WS-STARTPOLLWS
Display Name	User Defined Display Name
Attribute Values	User Defined Display Name

WS-STOPPOLLWSIF

Indicates the end of the polling specified by WS-STARTPOLLWS.

Use Case: The condition to stop the poll can be specified here. The attribute takes the xpath of the element against which the condition is to be compared. The condition is specified while entering the test data. If the test data is just a string, say <val>, then polling would stop when element value is <val>.

For example, if a web service needs to be polled unless the element BatchJobId is "ED", the attribute value should be set as the xpath of BatchJobId and the test data should be entered as "ED".

Similarly, if polling needs to continue as long as a certain value is returned, a "!" should be prefixed to the value of test data. If we want to continue polling as long as the BatchJobId is "PD", test data should be "!PD" (the symbol ! indicates "not equals"). Similar conditions can be set for greater than, less than, greater than equal to and less than equal to, by prefixing the test data with ">", "<", ">=" and "<=" respectively.

Usage Details	Value
Keyword	WS- STOPPOLLWSIF
Display Name	User Defined Display Name
Attribute Values	Xpath of element

Appendix B

GUI Component Keywords

This chapter provides the list of keywords used in a GUI based component.

- APPROVE
- CANCEL
- CHECK
- CLICK
- CLOSE
- GET_ATTRIBUTE_VALUE
- GET_ATTRIBUTE_ID
- LAUNCH
- MAXIMIZE
- MINIMIZE
- POPUP
- PRESSTABKEY
- SELECT
- SETTEXT
- SWITCHTO
- UNCHECK
- UNSELECT
- UI-STARTBROWSER
- UI-ENDBROWSER
- WAIT

APPROVE

Approves the specified object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format.

You can replace xpath with ID or name.

Usage Details	Value
Keyword	APPROVE
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	ALERT - Closes the Alert box, gets its text, and sets the value to 'true'.

CANCEL

Cancels the specified object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format.

You can replace xpath with ID or name.

Usage Details	Value
Keyword	CANCEL
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	ALERT

CHECK

Checks a checkbox object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format.

You can replace xpath with ID or name.

Usage Details	Value
Keyword	CHECK
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	CHECKBOX

CLICK

Cancel the specified object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format.

You can replace xpath with ID or name.

Usage Details	Value
Keyword	CLICK
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	IMAGE LINK TAB EDIT (edit is for a text box) BUTTON ELEMENT

CLOSE

Closes the specified window object. It is used to close a single browser window.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format.

You can replace xpath with ID or name.

Usage Details	Value
Keyword	CLOSE
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	CLOSE

GET_ATTRIBUTE_VALUE

Retrieves the "value" attribute of the specified object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format.

You can replace xpath with ID or name.

Usage Details	Value
Keyword	GET_ATTRIBUTE_VALUE
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	No objects required

GET_ATTRIBUTE_ID

Retrieves the "ID" attribute of the specified object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format.

You can replace xpath with ID or name.

Usage Details	Value
Keyword	GET_ATTRIBUTE_ID
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	No objects required

LAUNCH

Launches the specified browser object.

Use Case: The attribute value takes the browser URL.

Usage Details	Value
Keyword	LAUNCH
Display Name	User Defined Display Name
Attribute Values	Browser URL
Objects Valid	BROWSER

MAXIMIZE

Maximizes the specified window object.

Usage Details	Value
Keyword	MAXIMIZE
Display Name	User Defined Display Name
Attribute Values	None
Objects Valid	WINDOW

MINIMIZE

Minimizes the specified window object.

Usage Details	Value
Keyword	MINIMIZE
Display Name	User Defined Display Name
Attribute Values	None

Usage Details	Value
Objects Valid	WINDOW

POPUP

Handles the pop-up of the window.

Usage Details	Value
Keyword	POPUP
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	STARTPOPUP - To go to the popup ENDPOPUP - To come out from the popup

PRESSTABKEY

Performs a tab key press on the specified object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format.

You can replace xpath with ID or name.

Usage Details	Value
Keyword	PRESSTABKEY
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	EDIT

SELECT

Performs a 'select' action on the specified object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format. You can replace xpath with ID or name. The test data should be given a value from the list of values present in the select list.

Usage Details	Value
Keyword	SELECT
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	LISTBOX RADIOBUTTON LIST

SETTEXT

Closes the specified window object. It can be used to close a single browser window.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format. You can replace xpath with ID or name. Provide the actual text for that object in the test data.

Usage Details	Value
Keyword	SETTEXT
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	EDIT (edit is for a text box) TEXTAREA PASSWORD DATE

SWITCHTO

Used to switch between different frames.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format. You can replace xpath with ID or name.

Usage Details	Value
Keyword	SWITCHTO
Display Name	User Defined Display Name
Attribute Values	Xpath of the element for STARTFRAME None for ENDFRAME
Objects Valid	STARTFRAME - Switches to the specified frame ENDFRAME - Navigates back to the parent frame

UNCHECK

Uncheck the specified checkbox object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format. You can replace xpath with ID or name.

Usage Details	Value
Keyword	UNCHECK
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	CHECKBOX

UNSELECT

Unselects the specified object.

Use Case: The attribute value takes xpath of an element in the "xpath; actual xpath of the element" format. You can replace xpath with ID or name. The test data should be given a value from the list of values in the list selected previously.

Usage Details	Value
Keyword	UNSELECT
Display Name	User Defined Display Name
Attribute Values	Xpath of the element
Objects Valid	LISTBOX

UI-STARTBROWSER

Starts the browser.

Usage Details	Value
Keyword	UI-STARTBROWSER
Display Name	User Defined Display Name
Attribute Values	None
Objects Valid	No objects required

UI-ENDBROWSER

Closes the browser.

Usage Details	Value
Keyword	UI-ENDBROSER
Display Name	User Defined Display Name
Attribute Values	None
Objects Valid	No objects required

WAIT

Waits for the specified object. It is used to communicate to wait for a certain amount of time before throwing an exception that it cannot find the element on the page.

Use Case: The attribute value takes the xpath of the element in the "xpath; actual xpath of the element" format. xpath can be replaced with ID or name.

In the test data, provide the wait time(number) for LIST, LISTBOX, TEXTAREA, LINK, and BUTTON objects. For EDIT, IMAGE, WINDOW, and NORMAL objects, no test data is required.

Usage Details	Value
Keyword	WAIT
Display Name	User Defined Display Name
Attribute Values	Xpath of the element EDIT, IMAGE, WINDOW, NORMAL - None
Objects Valid	LIST LISTBOX TEXTAREA LINK BUTTON EDIT IMAGE WINDOW NORMAL - Can be used for any element

Appendix C

Setting Up Inbound Web Services

The Oracle Utilities application-specific components are developed using the web services method, and these components need the Inbound Web Services to be defined in the application.

This chapter includes the following sections:

- Creating Inbound Web Services
- Importing Inbound Web Services
- Searching Inbound Web Services

Creating Inbound Web Services

To create an Inbound Web Service, follow these steps:

1. Login to the Oracle Utilities application.

2. Navigate to **Admin** > **Integration** > **Inbound Web Service** > **Add**.

3. On the **Inbound Web Service** page, enter the **Inbound Web Service Name**.

4. Enter the **Description** and the **Detailed Description**.

5. Select the appropriate **trace,debug.active,post error** option from the drop down.

6. Select the **Annotation**.

7. Enter the **Operation Name**.

8. Select the **Schema Type**, **Schema Name**, and **Transaction Type**.

9. Click **Save**.

Importing Inbound Web Services

To import an Inbound Web Service into the Oracle Utilities application, follow these steps:

> **Note**: Ensure the exported Inbound Web Services are available in the local machine.

1. Login to the Oracle Utilities application.

2. Click **Admin** > **Implementation Tools** > **Bundle Import** > **Add**.

3. On the **Bundle Import** page, enter the reference and detailed description.

4. Copy paste the bundle details from the Inbound Web Services bundle.

5. Click **Apply bundle**. The "Imported Successfully" message appears in the **Message text** column.

Searching Inbound Web Services

To search an Inbound Web Service in an Oracle Utilities application, follow these steps:

1. Login to the Oracle Utilities application.

2. Navigate to **Admin** > **Integration** > **Inbound Web Service** > **Search**.

3. On the **Inbound Web Service Search** page, enter the name of the required web service in the **Name** field.

4. Enter the description in the **Description** field.

5. Click **Refresh**.

The web service, if found, is retrieved and displayed.

Appendix D

Generating Re-runnable Test Data

To run a flow multiple times, some fields might need unique values for each execution. Instead of changing the value in the databank, we can enable re-runnable test data so that the test data is generated randomly every time the flow is executed.

This chapter describes the options available on how the random data generated can be configured.

Requirement	Test Data Structure	Example Test Data	Generated String
A specified number of random lower case characters need to be appended to the given test data.	\<int\>?data	4?van 3?appl 6?AC 2? ?	vancara applxtg ACkdbvdl nd ufdbn
A specified number of random upper case characters need to be appended to the given test data.	\<int\>U?data	4U?van 3u?appl 6U?AC 2U? U?	vanCARA applXTG ACKDBVDL ND UFDBN
A specified number of random lower case characters need to be prefixed to the given test data.	\<int\>B?data	4B?van 3b?appl 6B?AC 2B? B?	caravan xtgappl kdbvdlAC nd ufdbn
A specified number of random upper case characters need to be prefixed to the given test data.	\<int\>BU?data	4BU?van 3bu?appl 6Bu?AC 2BU? BU?	CARAvan XTGappl KDBVDLAC ND UFDBN

Appendix E

Connecting to Multiple Databases

While building the integration flows that involve using components from various product packs, there might be a need to access databases of individual products as part of verification process. Users can connect to different databases by specifying the database connection properties prefixed with the application prefix. This is similar to the application prefix that is used for connecting to different application URLs.

Following is a sample configuration.properties file. It shows the entries for two environments.

```
# ************************************************
# Test Environment Details - UAT1
# ************************************************

# Application URL pointing to test execution
gStrUAT1_gStrApplicationURL = http://myuat1environment:8001/ouaf
gStrUAT1_gStrApplicationXAIServerPath=/webservices/
gStrUAT1_gStrEnvironmentName=UAT1

# Application Database Details
gStrUAT1_gStrApplicationDBConnectionString
=jdbc\:oracle\:thin\:@myuatserver1\:1521\:UATSID1
gStrUAT1_gStrApplicationDBUsername=mydbuser1
gStrUAT1_gStrApplicationDBPassword==<Entrypted Password>


# ************************************************
# Test Environment Details - UAT2
# ************************************************

# Application URL pointing to test execution
gStrUAT2_gStrApplicationURL = http:// myuat1environment2:8001/ouaf
gStrUAT2_gStrApplicationXAIServerPath=/webservices/
gStrUAT2_gStrEnvironmentName=UAT2

# Application Database Details
gStrUAT2_gStrApplicationDBConnectionString =jdbc\:oracle\:thin\:@
myuatserver2\:1521\: UATSID2
gStrUAT2_gStrApplicationDBUsername= mydbuser2
gStrUAT2_gStrApplicationDBPassword= <Entrypted Password>

##Handling Https  WSDL - Java key Store
gStrJavaKeyStorePath=
gStrJavaKeyStorePwd=<Entrypted Password>

#Email Details
gStrSMTP_HOST_NAME==<SMTP server address>
gStrSMTP_PORT=25
```

```
gStrTO_EMAIL_RECIPIENTS=
# Application user name
gStrApplicationUserName=<Your OUAF Application Username>
# Application user password
gStrApplicationUserPassword=<Encrypted Password>

# Output file details
gStrOutputFilePath=/<UTA-CLIENT-WORK-DIR>//Logs/
gStrXSDFiles=/<UTA-CLIENT-WORK-DIR>//XSD/
```

Appendix F

Configuring Authentication for Web Service Requests

Based on the version of the Oracle Utilities application (and the Oracle Utilities Application Framework), the web service requests are expected to include additional information apart from the user credentials. In order to support this, two new properties have been introduced in the configuration.properties file using which users can specify the authentication used by the environment.

For the latest versions of Oracle Utilities applications, a timestamp is expected in the web service requests. For these environments, specify the header type as TIMESTAMP, the other property gStrTimeToLive specifies the validity of the request in seconds.

```
#Header Type
gStrApplicationHeaderType=TIMESTAMP
#Timestamp interval
gStrTimeToLive=120
```

In cases where the configuration.properties contains details of more than one environment, prefix the header property with the application string.

```
#Header Type
gStrUAT_gStrApplicationHeaderType=TIMESTAMP
#Timestamp interval
gStrUAT_gStrTimeToLive=120
```

For the older versions of Oracle Utilities applications, only the user credentials are expected. So specify the header as USERTOKEN.

```
#Header Type
gStrApplicationHeaderType=USERTOKEN
```

In cases where there is a mix of environments that use the new header type and old header type in the same configuration.properties file, specify the properties for individual environments as follows.

```
#Header Type
gStrUAT_gStrApplicationHeaderType=TIMESTAMP
#Timestamp interval
gStrUAT_gStrTimeToLive=120

#Header Type
gStrINT_gStrApplicationHeaderType=USERTOKEN
```

Note: The user credentials are sent as digest by default. To send them as plain text, set the property mentioned needs to 'true'.

```
gStrSendPasswordAsText = true
```