Oracle Utilities Testing Accelerator

User's Guide for Cloud

Release 20B

F33208-01

July 2020

ORACLE®

Oracle Utilities Testing Accelerator User's Guide for Cloud, Release 20B

F33208-01

Contents

Chapter 5

Chapter 6

Chapter 7

Chapter 8

Chapter 9

Chapter 10

Appendix A

Appendix B

Appendix C

Appendix D

Appendix E

Preface

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

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
- Related Documents
- Conventions

Audience

This guide is intended for Automation Developers, and Test Automation 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 in-depth programming experience to develop scripts for testing. However, good understanding and working knowledge of Oracle Utilities Application Framework and its metadata based objects along with in-depth functional understanding of the application being tested, is required. The advanced programming features available in the application require experience with the programming concepts and groovy scripting language.

Abbreviations

The following abbreviations are used throughout this document:

Term	Expanded Form
UTA	Oracle Utilities Testing Accelerator
CCS	Oracle Utilities Customer Cloud Service

Related Documents

For more information, refer to the following Oracle resources.

Release Notes
- *Oracle Utilities Testing Accelerator Release Notes*

User and Reference Guides
- *Oracle Utilities Testing Accelerator Reference Guide for Core*

- *Oracle Utilities Testing Accelerator in Oracle Utilities Customer Cloud Service Enablement Guide*

Conventions

The following text conventions are used in this document:

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

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

- Supported Oracle Utilities Applications

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.
Flow Test Data	A test data set specific for a given flow.
Databank	Container of test data used by an automated test flow.
	The databank is defined using comma separated values (.csv) in a text file. (generated and used internally by UTA)

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.

Term	Description
Portfolio	Represents a product family consisting of one or more related products.
	A portfolio contains one or more products.
Product	Represents an Oracle Utilities application.
	For example: CCS
	A product contains one or more modules.
Module	Represents an Oracle Utilities application functional area. For example: Billing in CCS
	A module contains one or more components that are used to automate a specific functional area in an Oracle Utilities application.
Flow Module	Represents a group of flows typically a group of related flows meant to test a functional area.

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.
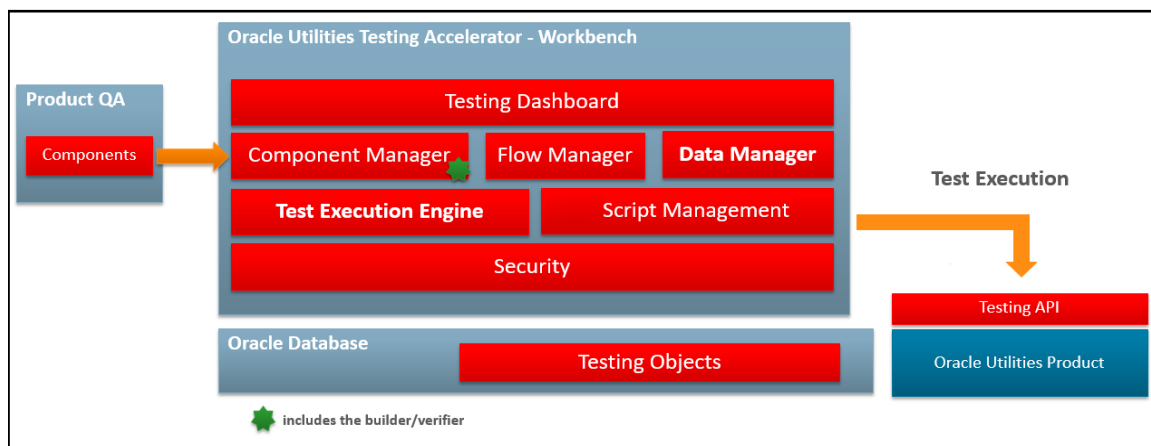
Components are defined using metadata in Oracle Utilities Testing Accelerator. Using these components a flow can be assembled and executed.

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.

Supported Oracle Utilities Applications

This table lists the Oracle Utilities' applications supporting this Oracle Utilities Testing Accelerator v20B release.

Product	Version
Oracle Utilities Customer Cloud Service	20B
Oracle Utilities Meter Solution Cloud Service	20B
Oracle Utilities Work and Asset Cloud Service	20B
Oracle Utilities Operational Device Cloud Service	20B

Chapter 2

Oracle Utilities Testing Accelerator Features

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

- Administration

- Components

- Dashboard

- Flows

- Flow Sets

- Tools

Administration

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

• Create/edit release, portfolio, product and modules

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 6: 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 custom components/flows in the application

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

• Total number of custom flows vs total number of approved custom 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.

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 structure

- Submit the flow for approval

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

For more details, refer to the Creating Flows section in Chapter 5: Creating Test Flows.

Flow Sets

This page displays all the available flow sets imported/created in the application. You can do the following:

- Create a new flow set

- Define/manage a flow set

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 URL end point of the web service that the component makes a call to in the Oracle Utilities applications, such as Oracle Utilities Customer Care and Billing or Oracle Utilities Customer To Meter.

For more details, refer to Chapter 8: Development Accelerator Tools.

Chapter 3

Developing Metadata Driven Web Service Based Test Automation

The Oracle Utilities Testing Accelerator components, 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
- Configuring the 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 test automation flows 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:

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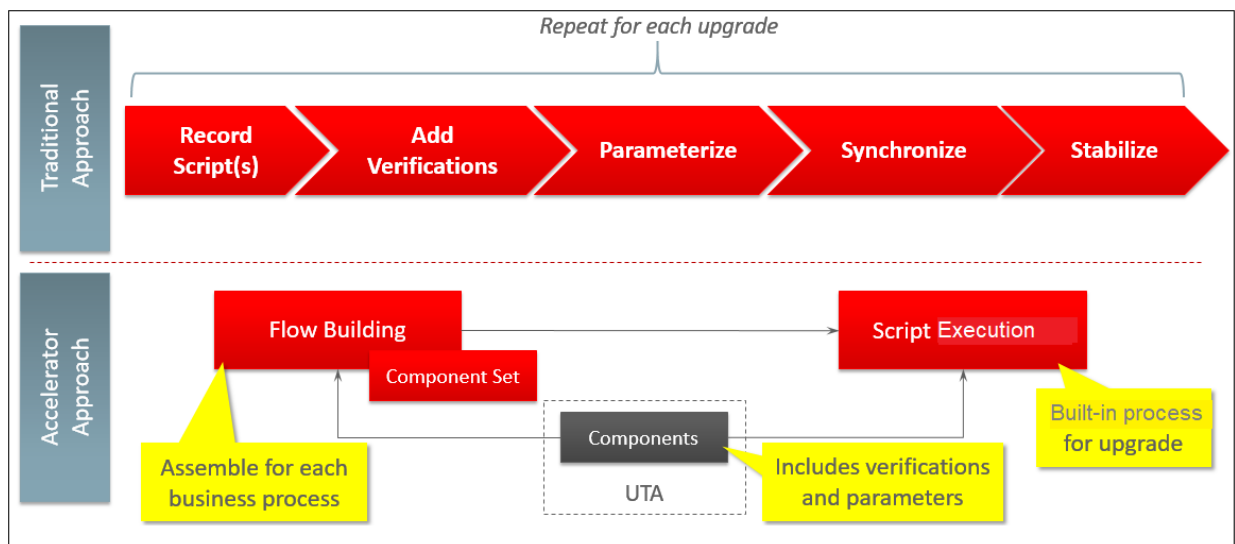
For the Oracle Utilities applications built on Oracle Utilities Application Framework, web service based automated testing is proven to be more robust, maintainable, and faster to develop and execute. Oracle Utilities Testing Accelerator comprises web services and UI based components that enable creation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

For details about how to extend the components, refer to the Copying Components section in Chapter 6: 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 5: 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 5: 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 5: 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.

Configuring the Automation Development Environment

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

*   Step 1: Setting Up Flow and User Configuration Sets

*   Step 2: Setting Up Application under Test

Setting Up Flow and User Configuration Sets

Before a flow can be executed, appropriate flow and user configuration sets have be to created. These hold the user credentials for authentication of the user to access the Oracle Utilities SaaS service being tested.

Setting Up Application under Test

For the test flows to be able to communicate with the Oracle Utilities SaaS service, corresponding Inbound Web Services should exist in the Oracle Utilities SaaS service. Each of the Utilities SaaS service content (components/flows) pack comes with an ImportBundles flow in the Pre-Requisites module under the corresponding flow tree structure. This flow containing all the requisite Inbound Web Services should to be executed to setup the application under test.

For more details about the flows and components, refer to the corresponding Oracle Utilities SaaS Service Oracle Utilities Testing Accelerator component reference 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

- Upgrade CM content (flows) from one version of an Oracle Utilities application to a later version.

  For example: From Oracle Utilities Customer Cloud Service 19B to Oracle Utilities Customer Cloud Service 19C

- Add custom function libraries to Oracle Utilities Testing Accelerator

- Purging old flow execution logs/results

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

- Custom Content Upgrade

- Purging Flow Execution Data

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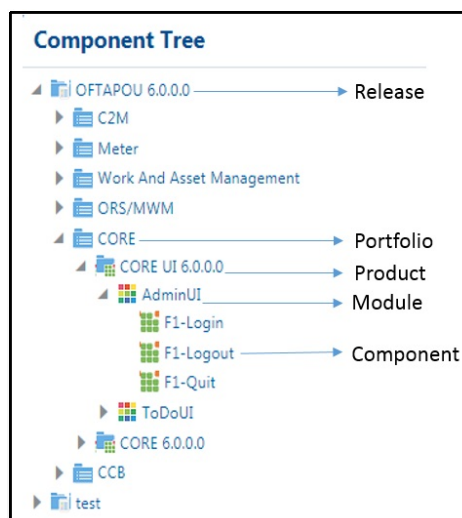
Managing Releases

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**.

Deleting a Release

Note that you can only delete an empty release.

To delete an existing release:

1. On the **Components** tab, expand the **Component** tree.

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

3. From the **Context** menu, click **Delete Release** (context menu option only appears if the release is empty).

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.

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

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

Deleting a Portfolio

Note that you can only delete an empty portfolio.

To delete an existing portfolio:

1. On the **Components** tab, expand the **Component** tree.

2. Select and right-click the portfolio to be deleted.

3. From the **Context** menu, click **Delete Portfolio** (context menu option only appears if the Portfolio is empty).

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.

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**.

Deleting a Product

Note that you can only delete an empty product.

To delete an existing product:

1. On the **Components** tab, expand the **Component** tree.

2. Select and right-click the product name to be deleted.

3. From the **Context** menu, click **Delete Product** (context menu option only appears if the product is empty).

Managing Modules

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

> **Note**: Modules created through the Administration section only apply to the component tree hierarchy. Flow modules should be created and managed through the flow hierarchy tree structure.

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.

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**.

Deleting a Module

Note that you can only delete an empty module.

To delete an existing module:

1. On the **Components** tab, expand the **Component** tree.

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

3. From the **Context** menu, click **Delete Module** (context menu option only appears if the module is empty).

Custom Content Upgrade

Oracle Utilities SaaS service's/application's version specific Oracle Utilities Testing Accelerator test components are released with each of the Oracle Utilities SaaS major version updates, such as 20B, 20C, etc. The custom content upgrade process allows custom flows to be automatically upgraded to the latest component pack version that correspond to the latest version of Oracle Utilities SaaS service/application.

Example: Flows may have been built using components from Oracle Utilities Customer Cloud Service 20A version. When a new version say Oracle Utilities Customer Cloud Service 20B is released, a corresponding set of components for 20B are also released as part of UTA. Using the

CM Content Upgrade option in the administration, the flows can be automatically upgraded to use the components from the latest 20B version instead of components from 20A version.

This ensures that the flows are using the components that correspond to the latest release of Oracle Utilities SaaS service.

Executing the CM Content Upgrade Process

To upgrade an existing set of flows:

1. Select a **Release Name**.

2. Select the **Product Family** under which the flows exist.

3. From the **From Product** field select the product version under which the flows exit.

4. From the **To Product** field select the product version to which the flows should be upgraded.

5. If only a subset of flows have to be upgraded, provide a "Tag" corresponding to these flows (the tag that has been specified in the flows header). "%" can also be used.

6. If the destination product version already has a set of flows, these can either be overwritten during the upgrade or skipped from being upgraded. It applies only to the flows in the destination product that have the same name as the flows from the source product. Select either "Override" or "Skip" based on the requirement.

7. Click **Upgrade**.

The upgrade process should run to completion with appropriate messages displayed.

> Note:
>
> - For a flow to be picked up by the upgrade process, the flow header should have a tag specified.
>
> - If a custom component has been created and used in the flows being upgraded, the upgrade process checks for the custom component name to start with "CM". If the name doesn't start with CM, the upgrade process copies the custom component across and prefixed "CM" to the component name. All references to this component in flows will be updated accordingly to remain intact. Ensure that the flow works fine. But, if the name starts with "CM", the upgrade process simply copies the custom component across from the source to the destination product.
>
> - The test data defined in the flows in the source product will remain intact in the destination product flows.

Purging Flow Execution Data

When the flow execution logs and flow execution history entries accumulate, it impacts the performance/usability. An administrator can decide to purge some of the existing flow execution data for maintenance purposes. The flow execution can be purged by specifying the cut-off date for purging entries; the data older than the specified date will be purged.

- **Flow Execution Logs** - Allows purging of all the flow execution log files that meet the specified criteria.

- **Flow Execution History** - Allows purging of flow execution history that helps in keeping the Flow Execution History page more manageable.

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 ComponentsCreating REST Web Service Components
- Copying Components

Component Structure

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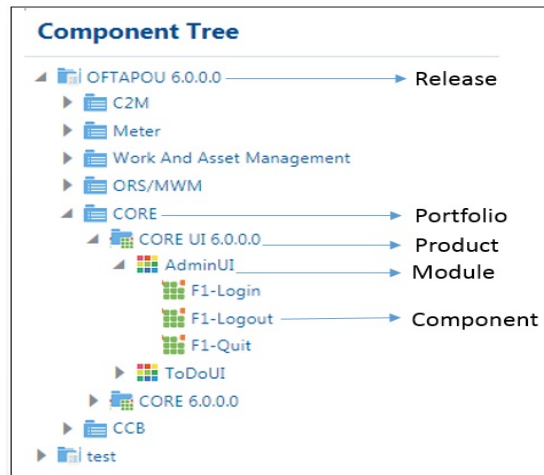
Component Lifecycle

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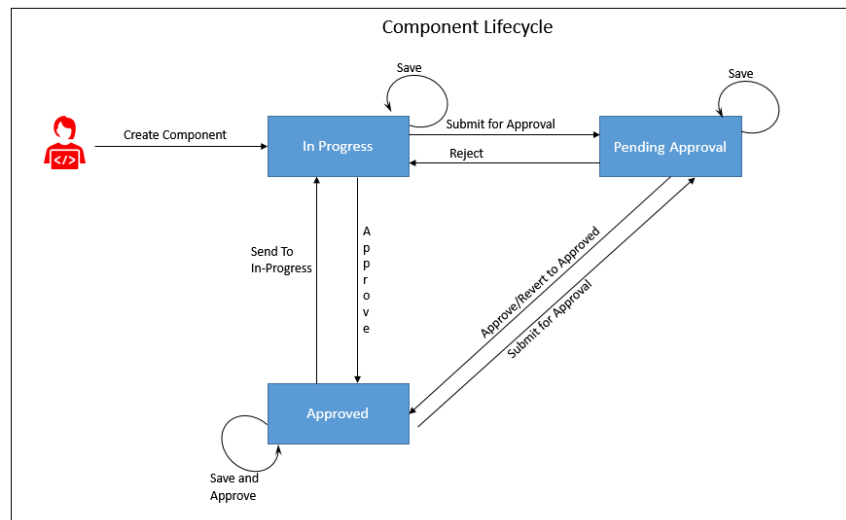
The state of a component determines the actions that can be performed on the component. The following table summarizes the component states, and the possible actions and roles that can take the actions.

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 ComponentsREST Web Service Components

Web Service Based Components

A web service based component represents an Inbound Web Service/Business Object/Business Service in Oracle Utilities Customer Cloud Service.

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.

REST Web Service Components

A REST web service component represents a REST interface in Oracle Utilities Customer Cloud Service application.

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

For information about REST-specific keywords, refer to Appendix C: REST 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 8: 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.
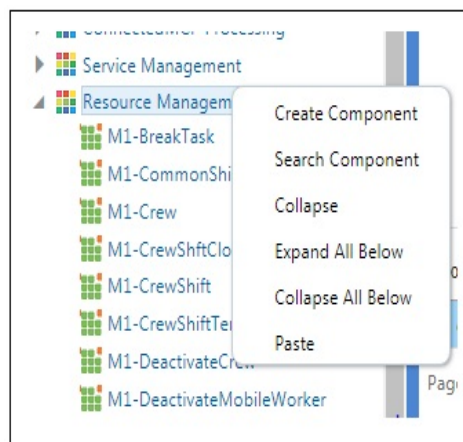
Creating a Component

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.

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**: Defines the action to be performed by the component line. Example: WS-SETVARIABLEFROM RESPONSE, WS-VALIDATE, etc

- **Object**: The name of the function library whose functions may be used for defining a component.

- **Display Name**: Description of the component line that is made visible to the user while entering test data against the component line in a flow.

- **Attribute Values**: The xpath of the component's element as defined in the Oracle Utilities Enterprise application.

- **Default Data**: The default data that may be used while providing test data for a component in a flow.

- **Function Name**: The name of the function that is used as a plugin to perform actions such as generating randomized test data or performing validation on web service response values.

- **Output Parameters**: If a function returns an output, the output can be stored in a variable which is defined against the Output Parameters field. This variable can be used across components in a flow to pass test data from one component to another.

- **Tooltip**: The information 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 component 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 allow for the web service name to be set for this component in a flow.

4. Select the WS-SETTRANSACTIONTYPE keyword to allow for the transaction type of the web service call to be set for this component, in a flow.

   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 step as part of a flow execution. This helps in better understanding of the Flow execution results in which the component is used.

6. Select the WS-SETXMLELEMENT keyword to allow test data to be set against 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 allow multiple sets of test data to be set against the list element of request XML. 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.

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 "Set Default Data" 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.
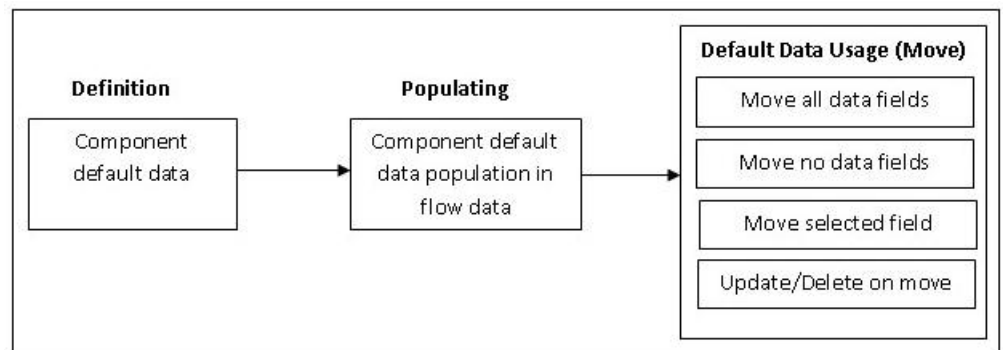
Data Flow

Setting Up Operation Name for a Web Service

An operation name determines the action to be taken while executing a web service request. This is dictated by the operation name of the web service in Oracle Utilities Application Framework based applications. 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 FUNCTIONCALL keyword along with the library rSVALIDATELIB and the function getElementValue. This function requires Xpath of the response element whose value is 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. For information about how a dependent component reads such global variables, refer to the Using Global Variables section.

file: prefix - csv file

Any attribute value containing a csv filename as value should be prefixed with "file:" to allow Oracle Utilities Testing Accelerator to process it correctly. For example: If a component contains an attribute name inputFile for which "InputData.csv" is the value, ensure to prefix the filename with "file:". The value of "inputFile" should be "file:InputData.csv".

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 5: Creating Test Flows.

For a list of libraries and functions available in Oracle Utilities Testing Accelerator, refer to Chapter 9: 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 9: Function Library Reference.

Adding Validations

The different ways in which you can add validations are:

• 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 9: 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

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 **Pre Validations** or **Post Validations** to specify validations that need to be performed either before sending the request or after the response is received from a Utilities application.
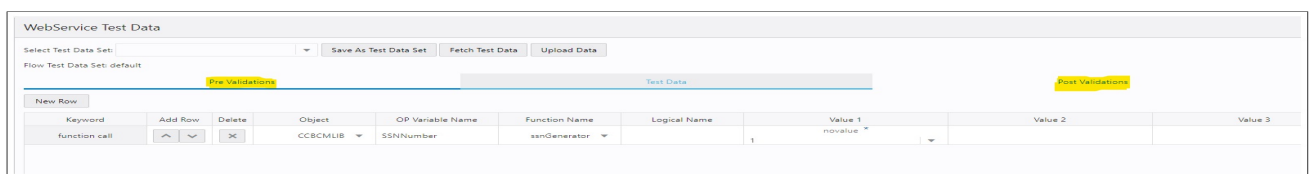
Note: In addition to adding validations in the pre-validations section, function calls can be made to generate (randomization) test data and stored in variables. These variables can then be used to set test data against component elements.

The post validation section can be used to add functions that retrieve and store any values from the response that can be used further down the flow, as test data in other components.

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 flowand can be viewed using the "View Logs" button on the execution results page in the workbench.

- **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 Email report is generated and sent to the email ids defined in the flow or user configuration sets. If no email ids are provided in the configuration, then the email is not sent. This applies to both flows and flow set executions.

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.

```
<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 usageDetials, 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 REST Web Service Components

To create REST web service based component:

1. Login to the application.

2. Navigate to the **Components** menu.

3. In the left pane, navigate to the module where the new component needs to be added.

4. Right-click the component and select **Create Component**.

5. On the **Create Component** page, select the component type as REST SERVICE.

6. Fill in the required fields and click Save.

7. Click **Attach Code** to save the component and edit it.

This section focuses on the following:

*   Creating a REST Service Component Definition

*   Entering Test Data for a REST Component

Creating a REST Service 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.

    For example: RS-SETREQUESTHEADER, RS-SETENDPOING, RS-PROCESSREQUEST, 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 E: 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.

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

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

The following example shows different component lines that can be created

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

2. Select **RS** 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. Select **RS-SETREQUESTHEADER** keyword to specify any headers that need to be passed to the REST end point.

5. Select **RS-SETMETHOD** keyword to specify whether the REST end point needs to be invoked using a GET/POST call.

6. Select **RS-PROCESSRESTREQUEST** keyword to specify processing of the response from the REST end point.

7. Add more component definition lines as needed and select appropriate keywords based on the REST web service that the component represents.

8. Click **Save**.

Entering Test Data for a REST Component

To enter test data for a REST component:

1. Navigate to the **Flows** menu.

2. On the left pane, right-click the flow and select **Create/Update Flow Structure**.

3. On the **Flow Definition** page, right-click the REST component and select **Edit Test Data**.

Rest Test Data

Select Test Data Set:

[▼] [Save As Test Data Set]

| Pre Validations | | | Test Data | | Body | | Post Validations |

Enable	Keyword	Object	Function Name	Caption	Logical Name	Value 1	Value 2
☐	WS-LOGMESSAGE			Log Message		testDataSet3 ▼	
☐	RS-SETREQUESTHEADER			Header	Content-Type	▼	
☐	RS-SETENDPOINT			EndPointUrl	http://google.com	▼	
☐	RS-ARGUMENT	PathVariable		Path Variable for url	location	▼	
☐	RS-ARGUMENT	QueryParameter		Add Query Params	name	testDataSet3 ▼	
☐	RS-ARGUMENT	PathVariable		Path Variable	cost	▼	

4. Add any pre-validation and post-validation functions by specifying the library and function details in the **Pre Validations** and **Post Validations** tabs.

Rest Test Data

Select Test Data Set:

[▼] [Save As Test Data Set]

| Pre Validations | | | Test Data | | Body | | Post Validations |

[New Row]

Keyword	Add Row	Delete	Object	OP Variable Name	Function Name	Logical Name	Value 1	
FUNCTIONCALL	∧ ∨	✕	oUCCBLIB ▼	newOpVar	getElementValue... ▼	Root1	sSubElement	▼
FUNCTIONCALL	∧ ∨	✕	oUCCBLIB ▼	uiuiupoi	getElementValue... ▼	Root	sSubElement	▼

The REST request body can be any of the following:

• Form Data - Key pair values

• RAW Data - Raw text that would be sent out as body

• Binary - Attach a file that contains the request that would be sent as request to REST end point

Rest Test Data

Select Test Data Set:

[▼] [Save As Test Data Set]

| Pre Validations | Test Data | Body | Post Validations |

● Form Data   ○ Raw   ○ Binary

[New Row]

	Value
	value
	value

Form Data

RAW Data

Binary

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
- Creating Flow Modules
- Adding the Email Capabilities to Flows
- Support for Integration Flows
- Executing Test FlowsEncrypting Passwords

Creating Flows

A flow simulates a business process that needs to be tested. Flows may be synonymous with test cases or test scenarios based on how test automation strategy is developed. Each flow may have one or more test scenarios. You can create a flow by dragging and dropping components into a default scenario under the flow.

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 may have to be extended or new components have to be created.

To create a flow:

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.

    For information about adding scenarios to a flow, refer to the Creating Scenarios section.

5. Expand the flow tree. The flow contains a default scenario with the same name as the flow name.

6. In the sequence defined by the business scenario being tested, drag and drop the components from the **Approved Components** pane to the **Flow Creation** pane.
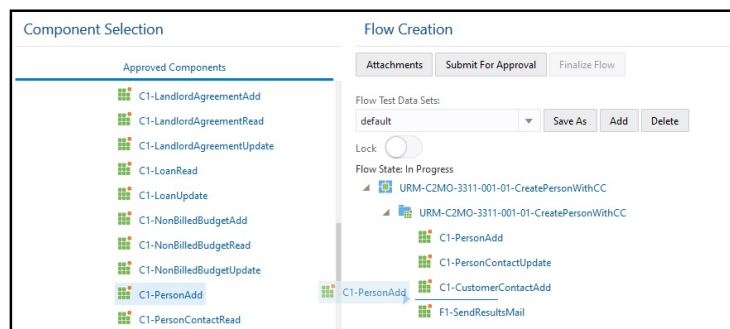
> **Note**: Flow definition can be modified(components added or removed) only if the flow is in "In Progress" state.

The test data needs to be entered at the component step level while defining a flow.

Adding Test Data in a Flow

To add data to a component in a flow:

1. In the flow tree structure, right-click the component and select **Edit Test Data**.

2. Enter the test data in the **Test Data** page. The **WebService Test Data** page has 3 sections.

    a. **Pre Validations**

Click **New Row** to add new rows. The **Pre Validations** section is used to add functions in the components that may be specific to the flow being developed.

Example: A function to generate random test data such as name, social security number or mobile number, for creating a person using C1-PersonAdd component.

The OP Variable Name holds the name of the custom variable into which the function output gets stored. This variable can be used as test data in **Test Data** section. The function inputs have to be specified against the Value1, Value2 through to Value 6 columns based on the number of input parameters of the function.

b. **Test Data**

The data corresponding to each of the elements in the component can be specified in the Test Data GUI.

- The test data pertaining to a component line can be specified against the Value 1 column. The keyword defines what the component lines does and the **Caption/Name** columns specify the details of the component line in context of the keyword.

- If the element is a repeatable group or list element, the "ADD" button under the "Add" column is used to add multiple repetition of the list elements.

- To select variables as inputs to the test data, the drop-down button under the column, select "Value 1" for each row and appropriate variable.

- The default data column holds a set of default test data that can be used in the component. The default data can be copied across to the test data field, "Value 1" by clicking **Set Default Data**.

- Validations/functions defined in the component can be enabled or disabled by selecting the **Enable** checkbox in the first column. The checkbox only appears for rows to which this feature is applicable.

c. **Post Validations**

The **Post Validations** section is used to add verification functions post the base validations. Each of the component comes with a base set of validations and these can be disabled or enabled in the **Test Data** GUI. And, if more of these verifications are added based on the flow specific requirements or if a specific set of values have to be retrieved from the response of the component execution, use the **Post Validations** section.

The post validations section allows users to add any number of functions to the component in a flow. These will be specific to the component's instance in that flow. These will not apply to the component when used in other flows.
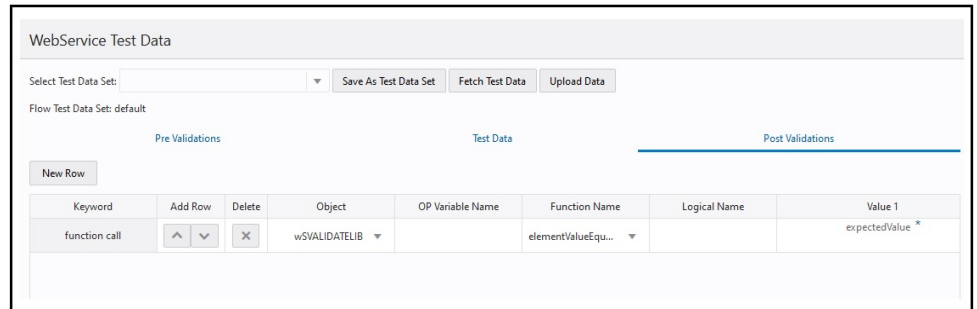
Only function calls can be added in the **Post Validations** section. For more details, refer to the **Flow-Level Validations** section in Chapter 5: Creating Components.

3. To add data for a component in a flow, in the flow tree structure, right-click the component and select **Edit Test Data.** For more details, refer to the **Flow-Level Validations** section in Chapter 5: Creating Components.

   Note: If the test data includes the double quotes character (" "), it needs to be escaped with another double quote character. Example: To enter My "Test Data", enter it as My ""Test Data"".

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

Annotating Components in a Flow

Annotations can be added for each component step to describe the purpose of each of the steps in an Oracle Utility Testing Accelerator test flow. This helps in understanding the functional aspects of the flow just by looking at the flow tree structure.

To add an annotation right click a component step in the flow definition. Select **Update Component Description** and enter the description. The description replaces the default display of the component name in the flow step.
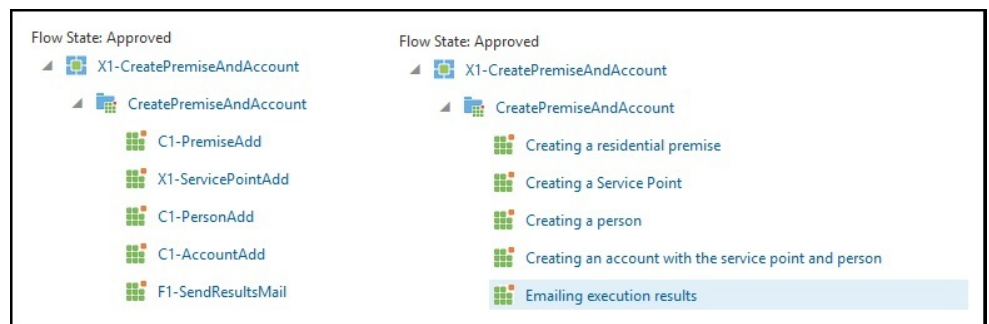
This figure shows a flow without and with annotations.

Creating Scenarios

To create a scenario:

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**.

Creating Flow Modules

Related flows can be grouped into a flow module. By default, each product has a "Default" module under which all flows are created unless they are explicitly created under a named module.

To create a flow module:

1. Navigate to **Flow** menu > product under which the flow module should be created.

2. Right-click the product and click **Create Flow Module** to create a new flow module.

To create a flow under a flow module:

1. Navigate to **Flow** menu > product and flow module under which the new flow should be created.

2. Right-click the flow module and click **Create Flow** to create a new flow under the selected flow module.

To move an existing flow to a flow module:

1. Navigate to **Flow** menu > flow that should be moved to a flow module.

2. Right-click the flow and click **Move to Flow Module**.

3. Select the target flow module.

4. Click **Move**.

Using Global Variables

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

In a simple test flow, add a new person in Oracle Utilities Customer Care and Billing and add a customer contact for that person. The C1-CustomerContactAdd component is a dependent component and during runtime needs the ID of the person, created using C1-PersonAdd component within the same flow.

To add component references to a dependent component (C1-CustomerContactAdd):

1. In the **Edit Test Data** GUI of the C1-CustomerContactAdd component, find the component line that requires the personId as input.

2. Against the personId row, click the "Value 1" drop-down list. This lists all variables that are exposed by the preceding components; in this case, the personId exposed by the C1-PersonAdd component.

3. Select the personId variable from the drop down list and set it as test data against this element.
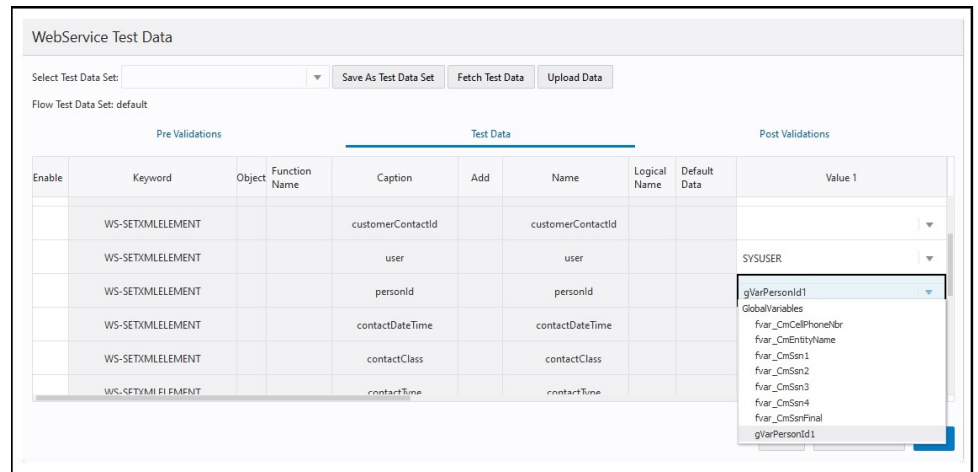
Each of the base components expose one or more global variables that hold the output of the component during execution. These global variables can be used to set the output of one component as the input of another component.

These global variables are automatically suffixed with their occurrence number. If the C1-PersonAdd component is used twice in the flow, there will be two variables (gVarPersonId1, gVarPersonId2) one for each occurrence of the component, suffixed with it's occurrence number. Custom global variables can be defined and exposed by the components through the **Pre-Validations** and **Post Validations** sections. These variables are automatically prefixed with "fvar_", to differentiate them from the component's base global variables.

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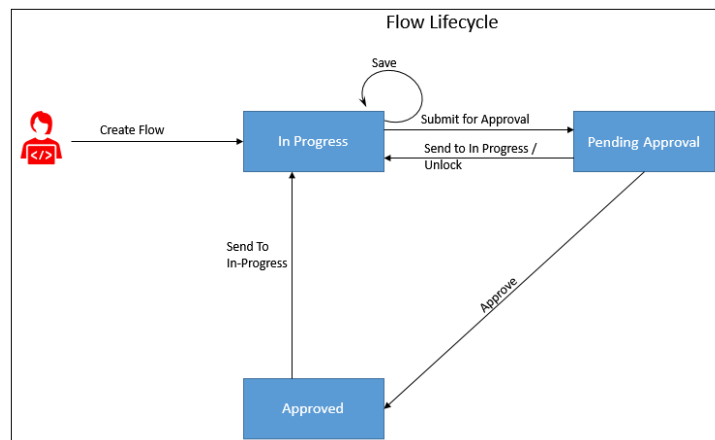
The state of a flow determines the actions that can be performed on the component. The following table summarizes the component states, and the possible actions and roles that can take the actions.

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. Reorder the components in any of the following ways:

    • By drag-and-drop method

    • Moving the components to a desired location using menu

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

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

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

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

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

8. 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**.

Fetching Component Test Data from an Utilities Application

Instead of manually entering the test data for a component, you can fetch the test data from a Utilities application (such as Customer Care and Billing, Meter Data Management, etc.). Select the User, Flow configuration and provide the web service name and the transaction/operation name to access the WSDL and then provide values against required fields that are mandatory for the specified operation. Oracle Utilities Testing Accelerator calls the WSDL with provided details and fetches the response from web service and populates in the test data of the component.

To fetch the test data:

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

2. Select and right-click the flow and then click **Create/Update Flow Structure**.

3. On the **Flow Definition** page, navigate to the component. Right-click and select **Edit Test Data**.

4. On the **Edit Test Data** page, click **Fetch Test Data**.

5. On the **Fetch Test Data** page, enter in the web service name from which the test data has to be retrieved, operation (typically READ operation) to invoke and necessary credentials and any required info (for example: to retrieve data related to ToDoRole).

```
<wsdl:operation name="ATF1ToDoRole_READ">
    <wsdl:input message="tns:ATF1ToDoRole_READRequest"/>
    <wsdl:output message="tns:ATF1ToDoRole_READResponse"/>
    <wsdl:fault name="fault" message="tns:Fault"/>
</wsdl:operation>
```

6. Enter the WSDL name, operation name, username, and password. Then, click **Populate Form** to populate the form with all fields that the web service supports.

Alternatively, use the URL and user credentials from the Flow/User Configuration properties file. Click **Use Configuration Properties** and select the appropriate flow/user configuration from the respective drop-down menus.

> **Note**: While creating an integration flow (a flow where components may send requests to more than one environment) prefix the URLs with keywords that can be used while specifying the WSDL to connect to.
>
> Example: If a flow should connect to an Oracle Utilities Meter Data Management instance apart from the Oracle Utilities Customer Cloud Service instance, specify the three properties mentioned below either in the flow or user configuration properties.
>
> ```
> MDM=<MDM url>
> MDM_gStrApplicationUserName=johnDoe
> MDM_gStrApplicationUserPassword=enc(pj0TFjXMczsoyzmQ8GuXPt2PSyd
> O7VCbR2jhxtkUH06Fuz+zmChpGSCr241KggFC6FwgMg==)
> ```
>
> To fetch the test data for an Oracle Utilities Meter Data Management component:
>
> 1.Select the flow/user configuration file from the drop-down menu.
>
> 2.Enter the WSDL URL as shown below.

Web Service Name *	MDM/D1AddMeter
Web Service Operation Name *	Read

7. Provide the necessary key information to retrieve data (for example: in this case the ToDoRole name) and then click **Fetch Test Data**.

Use Configuration Properties	Flow Configuration ▼	User Configuration ▼
Web Service Name *		
Web Service Operation Name *		
Application Username *		
Application Password *		

8. After the data is retrieved from the target application, review/validate it. Click **Save and Close**.

Bulk Replacing Component Test Data in Multiple Flows

The **Replace Test Data** feature allows to replace/edit value of one or more elements of a component in multiple flows, at once. If the component is used in multiple flows, select all or specific flows in which the test data needs to be changed for the component. This feature allows an easy way to change an existing test data value in several flows to a new value to reflect change in test data setup.

- Access the option to replace component test data. Navigate to the **Component** menu and right-click the component whose test data needs to be edited/replaced. Click **Find Component Usage**.

- In the **Find Component Usage** interface, select the flows under which the component test data needs to be replaced. Select the checkbox next to the flow name(s) and click **Replace Test Data**.

- Click **Add Row** to add a row to choose the element of the component whose specified existing test data value needs to be replaced with a new value. To replace the test data of multiple elements of the component, add multiple rows that specify the xpath of the element whose test data value needs to be replaced.

- Set an existing element value to blank or enter test data for component element whose current test data value does not exist. Use #EMPTY as the value in appropriate field (**Existing Value**/**New Value**).

- Specify a particular occurrence of an element in a group element. Indicate the index of the element in the group. To replace the zip code of second address group element, specify similar to /user/address[2]/zipCode and specify the **Existing Value** and **New Value**.

- Use wildcard "%" in the **Existing Value** field to indicate replacing of any existing value that matches the pattern. Example: To replace a field value that contain anything that starts with a "Building" to "Apartment 123" specify the **Existing Value** as "Building%" and **New Value** as "Apartment 123".

Flow Subroutines

A flow subroutine is a flow that can be included/used in other flows. It improves reuse of a flow. For example: Many test cases expect a 'V' setup to be available before being able to verify some business test cases. In this case, create a flow for 'V' setup and all other test case flows can reuse this 'V' setup flow as a subroutine in their respective flows. Specify any variables/parameters that the subroutine expects from the parent flow and also expose any variables/parameters that are created in the subroutine. Right-click **Edit Test Data** on the flow subroutine component in the flow.

> **Note**: The default test data set of the subroutine is used when the subroutine is executed as part of the parent flow.

- For a given flow test data set pertaining to the flow calling the subroutines, the test data set of the subroutine can be selected in the subroutine's test data GUI. Right-click the subroutine and select **Edit Test Data**.

- Only the variables defined in the default test data set of a subroutine flow can be used as input or output of the subroutine. This is to ensure standardized API for the subroutine.

Adding Subroutines to a Flow

To add an existing flow as a subroutine in a flow:

1. Right-click the scenario/component in the flow.

2. Select **Add SubRoutine**.

3. Specify the **Release**, **Product Family**, and **Product** to filter the flows.

4. From the **Flows** drop-down list (search-able), select the flow to be included.

5. Click **Add** to add it to the current flow as a subroutine.

> **Note**: A flow cannot be added to itself as a subroutine. Ensure that nested subroutines do not create a cyclic dependency.

Defining Input-Output Parameters of a Subroutine

To define input and output parameters for a subroutine:

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

2. Right-click the flow name in the product and navigate to module > flow tree structure in the left pane. Select Define Subroutine Interface.

3. Specify the parameters the subroutine expects from the calling flow and the parameters the subroutine exposes to the calling flow.

   Example: If the subroutine creates an Account, it expects a personId value to be provided for it to create an Account. After an account is created, it returns the accountId. The subroutine should be defined with one input variable "personId" and another output variable "accountId".

4. Add additional input/output variable.

   a. Click **Add IN/OUT Variable**.

   b. Enter the name and parameter type.

   c. Click **Save**.

This figure shows a subroutine interface definition for a flow that creates both a person and account and exposes personId and accountId as outputs, so they can be used by the calling flow.

5. After a subroutine is added to a flow calling a subroutine, map the input or output variable(s) of the subroutine.

   a. Right-click the subroutine in the flow tree structure of the calling flow and select **Edit Test Data**.

   b. Map the input/output variable of the subroutine to a variable in the calling flow.

      Example: The subroutine might be exposing accountId as the variable. To use the exposed variable in the calling flow, create a new variable in the calling flow using **Create**

New Variable. Map the output accountId variable from the subroutine flow to the newly created variable in the calling flow. This new variable can be used in the test data GUI of any component that succeeds the subroutine in the calling flow.
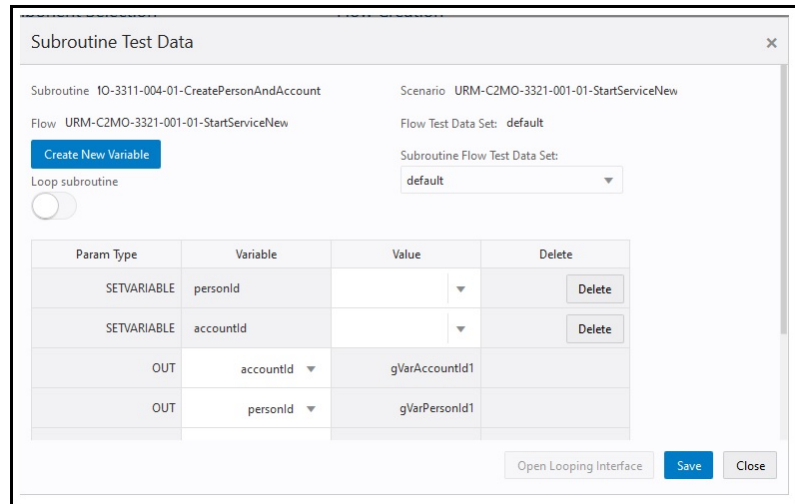
This figure shows the **Edit Test Data** screen for a subroutine that outputs a personId and accountId. New variables, personId and accountId are created and mapped to the outputs of the subroutine, which are gVarAccountId1 and gVarPersonId1.

Executing Subroutine in a Loop

A subroutine in a loop can be executed either a fixed number of times or until an exit condition is satisfied. Example: If the subroutine creates a meter read, the user can loop the subroutine 24 times to create a meter read for every hour of a particular day.

> **Note**: This feature only works with simple subroutines and not intended for nested subroutines. The flow re-run feature will not work as intended if the flow has a loop defined.

Specify the loop criteria and other details in the subroutine **Test Data** page. Click **Loop subroutine** to enable **Open Looping Interface** to provide the criteria for executing the subroutine in a loop.

Open Looping Interface provides the following options:

• **Maximum Number of Iterations**: Represents the maximum number of iterations that the subroutine will be executed for, irrespective of the exit criteria specified. This is useful in scenarios where the subroutine can wait but not execute indefinitely (either due to wrong test data/unexpected application behavior). Use this option to execute the subroutine a fixed number of times.

Example: If the subroutine creates a person entity in the application, specify the value 10 to execute the subroutine for 10 times resulting in creation of 10 person entities in Oracle Utilities Customer Cloud Service.

• **Incrementor Type**: Indicates if the loop incrementor would be a number or a date-time, user can choose date as incrementor in case the subroutine creates meter reads for a meter and user wants to execute the subroutine to create meter reads in a certain date range.

• **Initial Number/Initial Date-Time**: Based on the **Incrementor Type** selected, specify the starting number or the starting date-time to be used.

• **Increment Value**: Based on the **Incrementor Type** selected, specify by how much the initial number would be incremented by (either a number or in days, hours, minutes and seconds).

- **Exit Condition**: The exit condition controls when the subroutine loop would end before the Maximum Number of Iterations. Specify the exit condition as follows:

    - **Variable**: Can be either based on a value of the incrementor variable or any OUT variable of the subroutine.

    - **Condition**: Specifies if the value of variable should be less than, greater than, equal to etc. of the value that is specified for the Exit Condition.

    - **Value**: The value that the variable is compared with using the condition specified above, to check if the loop needs to be terminated or continued.

        Example: Assuming that the Incrementor Type was number and Initial value was 1 and Increment Value was 1 then the below values for the Exit Condition means that the subroutine is looped until the Incrementor value is equal to 5.

        Variable: "Incrementor" Condition: "equals to" Value: "5"

        Note: To specify the exit condition value when using a date, the date format to be specified is the same as the initial date format.

Component Test Data Sets

The component level test data sets allow to create test data sets specific to the component. These can be thought of as master test data sets for a component.

Example: For a C1-PremiseAdd component in Oracle Utilities Customer Cloud Service, the component level test data sets can be residential premise test data set and commercial premise test data set. Every time the C1-PremiseAdd component is used in a flow, instead of filling up the test data manually, the appropriate component test data set can be selected which automatically populates the test data from the component test data set into the component's test data GUI in the flow. This reduces a lot of work while providing test data in a flow.

Component test data sets save current test data of a component with a given name, which can later be retrieved and auto-populated it into another instance of the component either in the same flow or another flow.

Creating Reference Test Data for a Component

Save the current test data of a component for future use by saving it as a component test data set. After saving the test data set, the component can be populated with the test data contained in a **Test Data Set**. On the **Edit Test Data** page, select **Test Data Set** from the drop-down menu.

To create a test data set:

1. Login to Oracle Utilities Testing Accelerator.

2. On the **Flows** menu, navigate to the flow in which the component is used. Right-click and select **Create/Update Flow Structure** to open the **Flow Definition** page.

3. Navigate to the component for which the test data set needs to be created. Right-click the component and click **Edit Test Data**.

4. Click **Save As Test Data Set** to save the test data of the component. Specify the name of the test data set and click Save. Then, click **OK** to return to the **Edit Test Data** page.

    Note: If a test data set with the same name already exists, the application asks for confirmation to overwrite the test data.
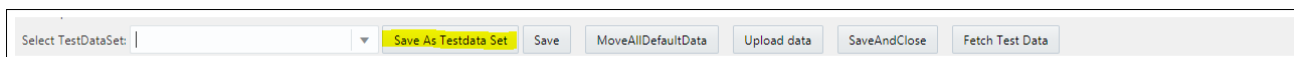
Loading Test Data from a Component Test Data Set

To populate the test data from a given component test data set:

1. Login to Oracle Utilities Testing Accelerator.

2. On the **Flows** menu, navigate to the flow in which the component is used.

3. Right-click the flow and select **Create/Update Flow Structure**.

4. On the **Flow Definition** page, navigate to the component for which the test data set needs to be created.

5. Right-click the component and then click **Edit Test Data**.

6. Select the test data set from the drop-down menu. The test data gets populated into the component.
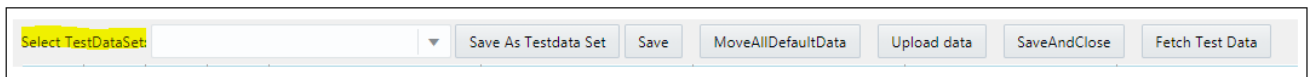
Flow Test Data Sets

Flow Test Data Sets allow users to create and manage multiple test data sets for the same flow. These test data sets can be used for selective or iterative execution of the flow. This feature is aimed at creating multiple sets of test data per flow and swap between these test data sets before executing a flow.

The Flow Test Data sets store the data specified against all the components within the flow, as a single data set. Users can copy the data set to create a new test data set and update it to reflect any changes. This feature has been provided to enhance reusability where test cases which do not differ in the flow structure, but only in the test data that is used, can be automated without having to recreate a test automation flow.

For more information, see Iterative Flow Execution.

Creating Flow Test Data Sets

To create a flow test data set:

1. Login to Oracle Utilities Testing Accelerator.

2. On the **Flows** menu, navigate to the flow for which the test data set should be created.

3. Right-click and select **Create/Update Flow Structure**.

4. On the **Flow Definition** page, click **Add** under **Flow Test Data Sets**.

5. Specify the data set name and click **Add**.

6. Click **Save As** to save the test data of the flow.

7. Specify the name of the test data set and click **OK**.

> **Note**: If a test data set with the same name already exists, the application asks for confirmation to overwrite the test data.

8. If the flow definition includes a subroutine, select the test data set for the subroutine. Right-click the subroutine and select **Edit Test Data**. Select the test data set from the **Subroutine Flow Test Data Set** drop-down list.

9. To edit or add test data against a flow test data set, the corresponding flow test data set has to be selected on the flow structure definition pane.

10. Navigate to the **Edit Test Data** page of each component in the flow and update/add the test data.

This figure shows flow test data sets option for the selected flow.

Loading Test Data from Flow Test Data Sets

To populate the test data from a given test data set:

1. Login to Oracle Utilities Testing Accelerator.

2. On the **Flows** menu, navigate to the flow in which the component is used.

3. Right-click the flow and select **Create/Update Flow Structure**.

4. On the **Flow Definition** page, select the test data set to be populated in the flow from the **Flow Test Data Sets** drop-down list.

   To edit or add test data against a flow test data set, the corresponding flow test data set has to be selected on the flow structure definition pane. Navigate to the **Edit Test Data** page of each component in the flow and update/add the test data.

5. If the flow definition includes a subroutine, select the test data set for the subroutine. Right-click the subroutine and select **Edit Test Data**. Select the test data set from the **Subroutine Flow Test Data Set** drop-down list.

Adding the Email Capabilities to Flows

The email capability allows the flow execution summary report to be e-mailed to the specified email accounts. Specify the email account IDs to which the report needs to be sent in the flow configuration set or the user configuration set. The email capability for a flow or flow set execution is auto enabled if the user configuration set or flow configuration set has the email property defined.

> **Note**: The generateAndSendReport function in WSCOMMONLIB is redundant and need not be included in the flow as the last step.

Support for Integration Flows

Support for integration flows allows users to create a single test flow that can interact with multiple applications. Individual components in the flow can be configured to post web service requests to different application URLs. The integration flow support includes support for creation, management and execution of hybrid test flows, where Oracle Utilities Testing Accelerator that is within Oracle Utilities Enterprise SaaS cloud service can have a test flow that interacts with that cloud service and one or more Oracle Utilities enterprise applications that may be at customer's premise/data center.

By default, Oracle Utilities Testing Accelerator automatically constructs the web service end point URL based on the web service name provided in the flow test data. This default URL executes test flows against the Oracle Utilities Enterprise SaaS cloud service that it is part of. To configure a component in a flow to post a request to a different application outside of the Oracle Utilities Enterprise SaaS cloud, the following configuration needs to be added to either the flow configuration set or the user configuration set.

Define the environment configuration properties pertaining to the on-premise Oracle Utilities Enterprise application. Prefix them with a custom keyword. This keyword is used in the component's test data in a flow to specify the application configuration context to a component for execution.

Example: Oracle Utilities Customer Care and Billing is the on-premise application and the custom keyword chosen is "CCB". In the flow configuration set or user configuration set, use the "Add Property" option and add the following properties:

Property Name	Property Value
CCB	<CCB url>
CCB_gStrApplicationUserName	<username>
CCB_gStrApplicationUserPassword	<encryptedpassword>

CCB property holds the external web service end point URL prefixed with "@ext_pub@", up to but not including the web service name. If the web service end point URL for the WSDL of person object in CCB is https:myccbserver.mycompany.com/webservices/ATC1PersonAdd?WSDL, the CCB property should hold the value - "@ext_pub@myccbserver.mycompany.com/webservices/".

The properties CCB_gStrApplicationUserName and CCB_gStrApplicationUserPassword hold the user name and password respectively for authenticating the user posting the web service request to Oracle Utilities Customer Care and Billing.

The figure below shows a sample setup of environment configuration for integration flows.

The integration flow in Oracle Utilities Testing Accelerator may contain a mix of components pertaining to the cloud service that it is part of and components pertaining to on-premise Oracle Utilities Enterprise applications. To get a component in flow to post a request to the on-premise Oracle Utilities Enterprise applications, the web service name in the component step's test data should be prefixed with the keyword.

In the example where Oracle Utilities Customer Care and Billing is the on-premise application and "CCB" is the keyword, the web service name for the C1-PersonAdd component's test data in the integration flow should be specified as "CCB/ATC1PersonAdd". This ensures that the C1-PersonAdd component posts the request to Oracle Utilities Customer Care and Billing whose configuration has been specified in the flow configuration set or user configuration set using the keyword prefix "CCB".

The figure below shows a sample usage of prefix keyword in component step's test data, to specify the application's context.

During the integration flow execution, the flow configuration set and user configuration set that have the required environment properties should be selected.

More than one such configuration can be set so that a test flow can interact with multiple applications. Each application can have its own custom keyword which is used while specifying the web service name in the component step's test data in a flow.

Executing Test Flows

This section focuses on executing a test flow.

- Executing Test Flows Using a Browser
- Iterative Flow Execution
- Stopping Flow Execution on Validation Failure
- Viewing Flow Execution Details
- Viewing Flow Execution Summary Report
- Debugging Flows using Conversational Test Data Management

Executing Test Flows Using a Browser

To execute a test flow using a browser:

1. Login to Oracle Utilities Testing Accelerator.

2. On the **Flows** menu, select the product to which the flow belongs. Right-click the test flow and select **Execute Flow**.

    Note: The test flow can be executed only if it is in either "Pending Approval" or "Approved" state.

3. Select the **Flow Configuration** and **User Configuration** to be used to execute the test flow.

4. Click **Execute** to start the test flow execution.

    Note: For more details about flow configuration and user configuration, refer to the Runtime Configuration for Flow Execution section.

5. On the **Flow Execution** page, the execution details are displayed.

    The tree shows each of the scenarios and components of the flow. Select a component in the tree to display the corresponding request and response details. Click **View Logs** to view the logs of the execution.

Iterative Flow Execution

To execute a test flow using a browser:

1. Login to Oracle Utilities Testing Accelerator.

2. On the **Flows** menu, select the product to which the flow belongs.

3. Right-click the test flow and select **Execute Flow**.

    Note: The test flow can be executed only if it is either in "Pending Approval" or "Approved" state.

4. Select the Flow Configuration and User Configuration used to execute the test flow.

5. Select **Iterative** as the **Flow Execution Type**.

6. In the **Number of Iterations** field, specify the number of iterations to execute the flow.

7. If the flow has more than one flow test data set, specify more than one flow test data set to be used during the iterative execution. Select the checkbox next to the flow test data set name.

    Note: Based on the number of iterations and flow test data sets specified, application will use test data sets for each of the iterations. For example: If number of iterations is specified as 10 and two flow test data sets are selected, the application executes the flow with first data set for first iteration and second data set for second iteration, and switch back to first data set for 3rd iteration and so on. At the end of 10 flow iterations, there would be total of 5 executions of the flows with first data set and 5 executions of the flow with second data set.

8. Click **Execute** to start the test flow execution.

    Note: For more details about flow configuration and user configuration, see Runtime Configuration for Flow Execution.

The execution details are displayed on the **Flow Execution** page.

The tree shows each of the scenarios and components of the flow. Select a component in the tree to display the corresponding request and response details. Click **View Logs** to view the execution logs.

Stopping Flow Execution on Validation Failure

By default, the flow execution continues until the last component in the flow even if there is a validation failure for a component in the flow. This behavior can be changed to make the flow execution stop when a validation fails by setting the property "continueExecutionOnFailure" in the user or flow configuration to "false".

Viewing Flow Execution Details

To view the execution details of a flow:

1. Login to Oracle Utilities Testing Accelerator.

2. Navigate to the **Flow** menu.

3. On the left pane, navigate and right-click the flow to view the execution summary. Click **View Execution History**.

4. Click the flow execution entries to view the respective details of that execution.

The **Flow Execution Status** page displays the details about the flow execution, including logs and request and response for each of the component.

Viewing Flow Execution Summary Report

To view the execution summary of a flow:

1. Login to Oracle Utilities Testing Accelerator.

2. Navigate to the **Flow** menu.

3. On the left pane, navigate and right-click the flow to view the execution summary.

4. Click **View Execution History**.

5. Click any of the flow execution entries to view the respective details.

6. On the **Flow Execution Status** page, click **Summary**.

7. On the **Flow Execution Summary Report** page, click **Summary**. The summary of the flow execution is displayed, including total scenarios passed/failed, percentage of pass/fail, etc. You can also drill down individual scenarios and check more details.

The flow execution summary can be sent via email. Specify the email address in the **Summary Report** page and click **Email**.

Conversational Test Data Management

As an alternative to the **Edit Test Data** GUI, test data can also be provided in XML format through the conversational **Test Data Entry** page. Before accessing this GUI, the component in the flow has to be executed at least once, by providing the web service name and the transaction type. In case a component of a flow fails during execution, user can use the **Edit Request** feature to access the **Conversational Test Data Management** page.

> **Note**: This feature is only supported for web service components.

The **Edit Request** feature allows to:

a. Open the failed request of a component.

b. Make changes to the test data and resend the request to edge application (without executing the flow multiple times).

c. Observe the response for the modified request.

d. Save the modified request as test data for the flow's component step.

Repeat the above steps until the test data for the successful request is identified.

The **Edit Request** feature can be accessed from the flow execution interface (including from flow execution history, flow set execution/history, iterative execution/history). User can invoke the **Edit Request** user interface. Right-click a component from the flow execution status tree and selecting the **Edit Request** option.
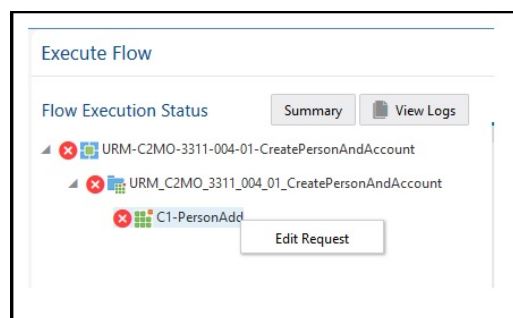
After selecting the **Edit Request** option from the right-click menu, a new window is displayed and prompts to select the configuration sets to be used. Modify the request by either changing test data value of one or more elements, adding/deleting new elements in the request and click **Send**

to send the request to the edge application and observe the response. Continue to make modifications until the desired response is received from the edge application.

The user has below options to manage request content better:

- Repopulate all schema elements. Click **Refresh Schema**. This helps in resetting fields that are required to make a successful request that were not present/left out in the previous/original request.

- Click **Settings** to allow the user to control some header level information that is sent as part of the request, including username token, timestamp and whether the request should include any schema elements that do not have any test data filled in.

- Choose to save the test data of the request to a flow test data set. Click Save to select the flow test data set to which the test data needs to be saved to in the dialog box.

  > **Note**: While saving the test data in the request XML in to the component step's test data, only static test data values defined in the component are replaced. The variables defined in the component's test data are not replaced with the test data in the XML request.

Runtime Configuration for Flow Execution

A test flow is executed using flow configuration sets and user flow configuration sets that contain the required properties, such as URL of the environment against which the flow to be executed, username/password to be used, etc.

The *flow configuration set* includes configuration applicable for a particular environment. It is expected that the flow level configuration sets do not contain any user specific properties, thereby allowing many users to use a single flow configuration set.

The *user configuration sets*, on the other hand, are specific to each user and typically contain user-specific properties, such as the username/password used to connect to an environment.

It is expected that customers setup some generic flow configuration sets with common runtime properties and users have their personal user configuration sets that contain their credentials. While executing a test flow/flow set, specify a flow configuration set and a user configuration set in combination to get generic runtime properties and user specific properties to be used for test flow/flow set execution.

> **Note**: For Oracle Utilities Testing Accelerator that is available as part of Oracle Utilities Enterprise SaaS services, the corresponding environment's URL is predefined within UTA. Users need not provide the web service URL for the same. However, user's will need to provide the user name and password for authentication.

This section focuses on managing the flow and user configuration sets:

- Creating a Flow Configuration Set

- Creating a User Configuration Set

- Editing a Flow Configuration Set

- Editing a User Configuration Set

- Copying a Flow Configuration Set

- Copying a User Configuration Set

Creating a Flow Configuration Set

1. Login to the application.

2. Navigate to the **Tools** menu.

3. On the left pane, click **Flow Configuration Sets**.

4. On the **Manage Flow Configuration Sets** page, click **Create**.

5. Specify the name of the flow configuration set being created and then click **Create**.

6. If the flow configuration set is created successfully, a message appears confirming that the operation was successful and redirects to the **Manage Flow Configuration Sets** page.

7. Search for the configuration set created and click **Edit** to create flow level configuration properties.

8. Each of the property is a key-value pair. By default some of the property names are listed on the **Edit** page. You can either enter a value for the existing property or choose to create a new property by clicking **Add Property**.

Important! It is recommended that sensitive information (such as passwords) be encrypted. Click **Encrypt** to encrypt the corresponding row of the property.

Creating a User Configuration Set

1. Login to the application.

2. Navigate to the **Tools** menu.

3. On the left pane, click **User Configuration Sets**.

4. On the **Manage User Configuration Sets** page, click **Create**.

5. Specify the name of the user configuration set being created and then click **Create**.

6. If the user configuration set is created successfully, a message appears confirming that the operation was successful and redirects to the **Manage User Configuration Sets** page.

7. Search for the configuration set created and click **Edit** to create user level configuration properties.

8. Each of the property is a key-value pair. By default some of the property names are listed on the **Edit** page. You can either enter a value for the existing property or choose to create a new property by clicking **Add Property**.

Important! It is recommended that sensitive information (such as passwords) be encrypted. Click **Encrypt** to encrypt the corresponding row of the property.

Editing a Flow Configuration Set

1. Login to the application.

2. Navigate to the **Tools** menu.

3. On the left pane, click **Flow Configuration Sets**.

4. On the **Manage Flow Configuration Sets** page, search for the flow configuration set to be edited, and then click **Edit**.

5. On the **Update Flow Configuration Set** page, click **Add Property** to either enter a value for the existing property or choose to create a new property.

Important! It is recommended that sensitive information (such as passwords) be encrypted. Click **Encrypt** to encrypt the corresponding row of the property.

Editing a User Configuration Set

1. Login to the application.

2. Navigate to the **Tools** menu.

3. On the left pane, click **User Configuration Sets**.

4. On the **Manage User Configuration Sets** page, search for the user configuration set to be edited, and then click **Edit**.

5. On the **Update User Configuration Sets** page, click **Add Property** to either enter a value for the existing property or choose to create a new property.

Important! It is recommended that sensitive information (such as passwords) be encrypted. Click **Encrypt** to encrypt the corresponding row of the property.

Copying a Flow Configuration Set

1. Login to the application.

2. Navigate to the **Tools** menu.

3. On the left pane, click **Flow Configuration Sets**.

4. On the **Manage Flow Configuration Sets** page, search for the flow configuration set to be copied, and then click **Copy**.

5. Enter the new configuration set name and click **Confirm** to create a copy.

Copying a User Configuration Set

1. Login to the application.

2. Navigate to the **Tools** menu.

3. On the left pane, click **User Configuration Sets**.

4. On the **Manage User Configuration Sets** page, search for the user configuration set to be copied, and then click **Copy**.

5. Enter the new configuration set name and click **Confirm** to create a copy.

Chapter 7

Creating Test Flow Sets

Flow sets offer a level of abstraction above the flows that allows more flexibility in managing flows. Several related flows can be grouped into a flow set and can be executed in sequence. Multiple flow sets can be executed in parallel, whereas flows in a flow set will be executed in the specified sequence. Unlike the flow subroutines, the flows in a flow set do not have a direct dependency on each other i.e., the test data/outputs cannot be passed from one-flow to another, within Oracle Utilities Testing Accelerator.

This chapter focuses on flow sets including:

- Creating Flow Sets
- Adding Flows to a Flow Set
- Deleting Flows from a Flow Set
- Executing Flow Sets
- Aborting Flow Set Execution
- Exporting Flow Sets
- Viewing Flow Set Execution History
- Viewing Flow Set Execution Summary Report

Creating Flow Sets

To create a flow set:

1. Login to the application.

2. Navigate to the **Flow Sets** menu.

3. On the left pane, click **Create Flow Set**.

4. Provide the **Flow Set Name** and **Description**, and click Save to save the flow set.

5. Navigate to the **Manage Flow Set** menu to add flows to the flow set.

Adding Flows to a Flow Set

To add flows to a flow set:

1. Login to the application.

2. Navigate to the **Flow Sets** menu.

3. On the left pane, click **Manage Flow Set**. The list of available flow sets is displayed.

4. Select the flow set name to which the flow(s) needs to be added.

5. Click **Add Flows**. You can search for one or more flows using the wildcard "%" to search for flows matching a name.

    For example: Search for all flows that contain the text "person" in their name by searching for string "%person%".

6. Click **Save** to save the flow set.

Deleting Flows from a Flow Set

To delete flows from a flow set:

1. Login to the application.

2. Navigate to the **Flow Sets** menu.

3. In the left pane, click **Manage Flow Set**. The list of available flow sets is displayed.

4. Select the flow set from which the flow(s) needs to be deleted.

5. Delete one or more flows from the flows displayed. Select the checkbox for each of the flow to be deleted and then click **Delete**.

6. Click **Save** to save the flow set.

Executing Flow Sets

To execute a flow set:

1. Login to the application.

2. Navigate to the **Flow Sets** menu.

3. In the left pane, click **Manage Flow Set**. The list of available flow sets is displayed.

4. Select the flow set to be executed and click **Execute**.

5. Select the flow configuration and user configuration to be used to execute the flow set.

6. Click **Confirm**.

> **Note**: For more details about on flow configuration and user configuration, refer to the Runtime Configuration for Flow Execution (For Execution Using Browser) section.

7. Once the flow set execution starts, click each of the flows to view more details about the execution.

Aborting Flow Set Execution

The Abort feature allows the active flow execution to complete and stops all subsequent flows in the flowset from execution.

To abort a flow set execution:

1. Login to the application.

2. Navigate to the **Flow Sets** menu.

3. On the left pane, click **Manage Flow Set**. A list of available flow sets are displayed.

4. Click **View History** on the flow set name whose execution needs to be aborted.

5. Select the currently executing instance of the flow set and click **Abort**.

Exporting Flow Sets

To export a flow set:

1. Login to the application.

2. Navigate to the Flow Sets menu.

3. On the left pane, click Manage Flow Set. A list of available flow sets are displayed.

4. Click Export against the Flow Set to be exported.

Viewing Flow Set Execution History

To view the execution history of a flow set:

1. Login to the application.

2. Navigate to the **Flow Sets** menu.

3. On the left pane, click **Manage Flow Set**. A list of available flow sets are displayed.

4. Click **View History** of a flow set to display all previous executions. You can view details such as the flow set execution status, date and time of the run, the user who triggered the execution, etc.

5. Click any of the previous executions to view flow-level details of that particular execution.

   You can drill-down even further by clicking a flow name and view the details of the flow execution, including overall status, request and response details for each of the component and even view the log file details of a particular component execution.

Viewing Flow Set Execution Summary Report

To view the execution summary of a flow set:

1. Login to the application.

2. Navigate to the **Flow Sets** menu.

3. On the left pane, click **Manage Flow Set**. A list of available flow sets are displayed.

4. Click **View History** of a flow set to display all previous executions. You can view details such as the flow set execution status, date and time of the run, the user who triggered the execution, etc.

5. Click any of the previous executions to view flow-level details of that particular execution.

6. Click **Summary** to display a summary of the flow set execution, including total flows passed/failed, percentage of pass/fail, etc. You can also drill down individual flows to view the respective details.

7. Email the flow set execution summary. Specify the email address on the **Summary Report** page and click **Email**.

Chapter 8

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

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 webservice name, operation name.

7. Click **Generate Component(s)** and select the **Flow/User configuration**. The application URL and user credentials will be taken from the specified configuration file.
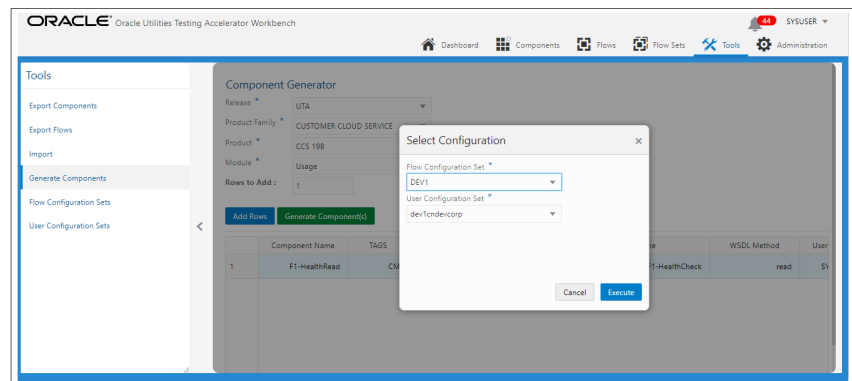
   Note: When attempting to generate components from more than one application at the same time, prefix the URLs with keywords in the configuration files that can be used while specifying the WSDL to connect to.

   Example: If an Oracle Utilities Meter Data Management component should be generated along with an Oracle Utilities Customer Cloud Service component, specify the three properties mentioned below either in flow or user configuration properties.

```
opdev=<MDM url>
opdev_gStrApplicationUserName=johnDoe
opdev_gStrApplicationUserPassword=enc(pj0TFjXMczsoyzmQ8GuXPt2PS
ydO7VCbR2jhxtkUH06Fuz+zmChpGSCr241KggFC6FwgMg==)
```

To generate a Oracle Utilities Meter Data Management component, enter the webservice name prefixed with "opdev/".
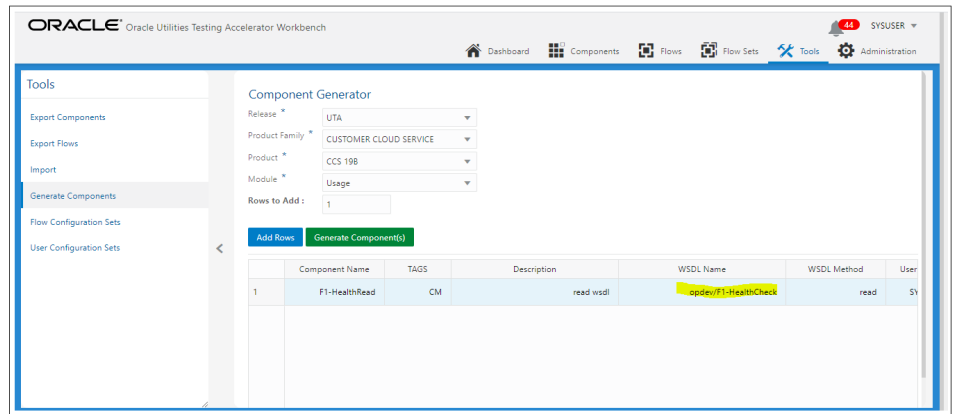
8. Upon successful component generation, a list of generated components and failed components is displayed.

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

Chapter 9

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.

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
```

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
```

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
```

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
```

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 6: 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
```

Chapter 10

Custom Libraries

This chapter focuses on creating custom libraries that include custom validation functions used for component validation.

> **Note**: Only Groovy script language custom libraries are supported. Due to security constraints, only a few Groovy packages are allowed in custom libraries.

- Creating/Updating Custom Libraries
- Using Custom Library Functions

Creating/Updating Custom Libraries

Make sure to have Administrator privileges to manage custom libraries.

To create a custom library:

1. Login to Oracle Utilities Testing Accelerator as an Administrator.

2. Navigate to the **Administration** tab and click **Libraries** on the left pane.

3. Enter the name of the new custom library in the **Library Name** field. Click **Create Library**.

4. From the **Library Type** drop-down list, select the library type being created. It is used only for web services based requests, UI or REST requests, or all type of requests.

5. Click **Create**.

   > **Note**: This step only creates a definition of the library, the actual code supporting/implementing the definition is expected to have been already developed using an IDE like Eclipse or Groovy consoles. See the example at the end of this section for more details.

6. Once a library is created, add the function definitions using "+Add". The function definition should specify the function name (Function), number of input parameters of the function (Parameter Count), definition of parameters (as @param1 and @param2), comments and description.

7. On the **Create/Update Library** page, click "+" in the **Library Functions** section. Add functions exposed by the custom library and other details, such as parameters of the functions.

8. Add separate rows for each exposed function in the custom library.

   • **Function**: Function name in the custom library

   • **Parameter Count**: Total number of parameters for the function. A function can have a maximum of 6 parameters.

   • **Parameters**: Name of the parameter(s). If more than one parameter exists, separate them by a comma and should be named as @param1, @param2 ..@param6.

   • **Parameter Comments**: Description about parameters, helps to show more information about the parameters. If the function has more than one parameter, descriptions should be separated by a comma.

   • **Description**: Function description

9. Click **Save**.

10. Click **Upload Library File** to upload the Groovy script containing actual implementation of the functions included.

11. Specify the package name of the custom library.

12. Select the file to be uploaded. Click **Upload**.

13. Click **Confirm** to overwrite any existing library with an updated version or a new library being uploaded.

In case of any compilation issues in the custom library an error message is displayed. Rectify the errors and re-upload.

Example: **Creating a Groovy Library**

To create a function to generate a random social security number as test data to create a person, create a *.groovy* file with the function definition. The library name is "UTATEST" and the function name is "generateSSN". It takes an input prefix and returns a random set of digits prefixed with the input value. Create a UTATEST.groovy file with the function's implementation.

The UTATEST.groovy contents are as follows:

```
package uta.oracle;

import java.util.ArrayList;

import java.util.List;

import com.oracle.utilities.core.plugin.FunctionalTestScript;

import com.oracle.utilities.core.lib.WSCOMMONLIB;

import java.util.logging.Logger;

import com.oracle.utilities.core.lib.OUTSPCORELIB;

public class UTATEST {

private static final Logger _logger =
        Logger.getLogger(UTATEST.class.getName());

public String generateSSN(String prefix) throws Exception{

        Random random = new Random();

        int x = random.nextInt(900) + 100;

        int y = random.nextInt(90) + 10;

        int z=random.nextInt(9000) + 1000;

        String zz = x+"-"+y+"-"+z;

        return prefix+zz;

    }}
```

Below is the function definition in Oracle Utilities Testing Accelerator.

Click **Upload Library File** to upload the .groovy file. It can be plugged into any custom component or the pre-validations and post-validations section of flow test data definition.

Using Custom Library Functions

After successfully uploading the custom library into Oracle Utilities Testing Accelerator use any of the exposed custom library functions in any of their components/flows. It is similar to how the built-in libraries are provided with Oracle Utilities Testing Accelerator.

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 name.

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 name. 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

REST Component Keywords

This chapter provides the following REST component keywords:

- RS-SETREQUESTHEADER

- RS-SETENDPOINT

- RS-ARGUMENT

- RS-SETMETHOD

- RS-PROCESSRESTREQUEST

RS-SETREQUESTHEADER

Sets the header in the defined REST request.

Use case: The attribute value takes the name of the request header.

Usage Details	Value
Keyword	RS-SETREQUESTHEADER
Display Name	User Defined Display Name
Attribute Values	User Defined Header Name
Objects Valid	No objects required

RS-SETENDPOINT

Sets the endpoint for the REST request.

Use Case: Defines the static part of the application's REST endpoint.

Usage Details	Value
Keyword	RS-SETENDPOINT
Display Name	User Defined Display Name
Attribute Values	User Defined End Point
Objects Valid	No objects required

RS-ARGUMENT

Sets the query parameter or the path parameter for the REST request.

Use Case: Used for setting the query parameter and path variable in the REST request. The values are provided from the test data.

Usage Details	Value
Keyword	RS-ARGUMENT
Display Name	User Defined Display Name
Attribute Values	User Defined Query Parameter Name for QueryParameter None for PathVariable
Objects Valid	QueryParameter - Appends the query parameter name in the component definition and value given in the test data to the REST end point. PathVariable - Appends the user defined value in test data to the REST end point.

RS-SETMETHOD

Sets the method type for the REST request.

Use Case: Used to set the REST request method type.

Usage Details	Value
Keyword	RS-SETMETHOD
Display Name	User Defined Display Name
Attribute Values	None
Objects Valid	GET - Creates a GET method to hit the REST end point.
	POST - Creates a POST method to hit the REST end point.

RS-PROCESSRESTREQUEST

Sends the REST request and receives the response from the application for the specified REST.

Use Case: Used to send the REST request using the methods and data provided using the above keywords.

Usage Details	Value
Keyword	RS-PROCESSRESTREQUEST
Display Name	User Defined Display Name
Attribute Values	None
Objects Valid	No objects required

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
A specified number of random numbers need to prefixed to the given test data	\<int\> d?data	d?ABCD 2d?ABCD	ABCD32940 ABCD43
A specified number of random numbers need to suffixed to the given test data	\<int\> bd?data	bd?ABCD 4bd?ABCD	32940ABCD 1534ABCD

Appendix E

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