

Oracle Utilities Testing Accelerator

User's Guide

Release 6.0.0.3.0

F35956-01

June 2021

Copyright © 2019, 2021 Oracle and/or its affiliates. All rights reserved.

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

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

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

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

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

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

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience	ii
Prerequisite Knowledge.....	ii
Abbreviations	ii
Related Documents	ii
Updates to the Documentation.....	iii
Documentation Accessibility	iii
Conventions.....	iii

Chapter 1

Overview	1-1
Introduction.....	1-2
Terminology	1-2
Application Architecture	1-3
Application Features	1-4
Supported Oracle Utilities Applications.....	1-4

Chapter 2

Oracle Utilities Testing Accelerator Features	2-1
Administration	2-2
Components	2-2
Dashboard	2-2
Notifications.....	2-2
Flows.....	2-3
Flow Sets	2-3
Tools	2-3

Chapter 3

Developing Metadata Driven Web Service Based Test Automation	3-1
Metadata Driven Automation Development Methodology.....	3-2
Planning	3-3
Design and Development	3-3
Test Run	3-3
Setting Up Automation Development Environment	3-3
Setting Up Oracle Utilities Testing Accelerator Server	3-4
Setting Up Workstations for Development/ Testing.....	3-4
Setting Up Application under Test.....	3-7

Chapter 4

Oracle Utilities Testing Accelerator Administration.....	4-1
Overview	4-2
Administration Tab	4-2
Managing Products	4-2

Managing Modules	4-3
Managing Users	4-4
User Access Types	4-6
Purging Flow Run Data.....	4-6
Purging Notification Data.....	4-7
Custom Content Upgrade.....	4-7

Chapter 5

Creating Test Flows	5-1
Creating Flow Modules.....	5-2
Creating Flows	5-2
Creating Flows By Dragging-and-Dropping Components.....	5-2
Adding Test Data in a Flow	5-3
Annotating Components in a Flow	5-5
Using Global Variables.....	5-5
Flow Lifecycle.....	5-6
Locking/Unlocking Flows.....	5-7
Copying Flows.....	5-7
Reordering Components in a Flow	5-7
Copying Test Data from One Component to Another in a Flow	5-8
Fetching Component Test Data from an Utilities Application	5-8
Unit Testing a Component in a Flow	5-9
Bulk Replacing Component Test Data in Multiple Flows.....	5-10
Flow Subroutines	5-10
Running Subroutine in a Loop.....	5-13
Conditional Bypass of Components in a Flow Run	5-14
Component Test Data Sets.....	5-15
Creating Reference Test Data for a Component	5-15
Loading Test Data from a Component Test Data Set	5-16
Deleting Component Test Data Sets	5-16
Loading Test Data from a .csv File	5-16
Flow Test Data Sets.....	5-17
Adding Email Capabilities to Flows	5-18
Support for HTTPS Web Services.....	5-19
Support for Integration Flows.....	5-19
Running Test Flows	5-22
Running Test Flows Using a Browser	5-22
Iterative Flow Run	5-23
Stopping Flow Run on Validation Failure.....	5-23
Stopping Flow Run Manually.....	5-23
Viewing Flow Run Details.....	5-23
Viewing Flow Run Summary Report	5-24
Conversational Test Data Management.....	5-24
Generating Oracle Utilities Testing Accelerator Scripts	5-25
Importing the Generated Oracle Utilities Testing Accelerator Script into Eclipse IDE.....	5-26
Running Flows from Command Line	5-27
Encrypting Passwords.....	5-27
Generating Keystore for Encryption from Windows Explorer	5-27
Generating Keystore for Encryption from Command Prompt.....	5-28
Using Password Encryptor Tool From Windows Explorer.....	5-28
Using PasswordEncryptor Tool From Console/Command Line.....	5-28
Configuring the Runtime Properties (For Run Using Eclipse).....	5-29
Runtime Configuration for Flow Run (For Run Using Browser).....	5-30

Chapter 6

Creating Components	6-1
Component Structure.....	6-2
Component Lifecycle.....	6-2

Locking/Unlocking Components.....	6-3
Component Types	6-4
Web Service Based Components.....	6-4
GUI Based Components.....	6-4
REST Web Service Components.....	6-4
Creating Web Service Based Components	6-4
Creating a Component	6-5
Creating a Component Definition.....	6-6
Defining Default Data at Component Level	6-8
Setting Up Operation Name for a Web Service.....	6-8
Using Runtime Variables in Components.....	6-8
file: prefix - csv file.....	6-9
Using Function Libraries.....	6-9
Resolving the Repeating Elements in Response XML.....	6-9
Adding Validations.....	6-9
Logging and Reporting.....	6-10
Handling the List Elements.....	6-10
Creating GUI Based Components.....	6-14
Creating a Component Definition for GUI Components.....	6-14
Creating REST Web Service Components	6-16
Creating a REST Service Component Definition	6-16
Entering Test Data for a REST Component.....	6-17
Copying Components	6-19

Chapter 7

Creating Test Flow Sets	7-1
Creating Flow Sets	7-2
Adding Flows to a Flow Set.....	7-2
Deleting Flows from a Flow Set.....	7-2
Running Flow Sets.....	7-2
Stopping Flow Set Run	7-3
Exporting Flow Sets.....	7-3
Viewing Flow Set Run History	7-3
Viewing Flow Set Execution Summary Report	7-3

Chapter 8

Development Accelerator Tools	8-1
Component Export Tool	8-2
Flow Export Tool.....	8-2
Component/ Flow Import Tool.....	8-2
Component Generation Tool.....	8-3
Password Encryption Tool	8-4
Overview	8-4
Running the Password Encryption Tool.....	8-4
Component Definition Validation Tool	8-5

Chapter 9

Function Library Reference	9-1
OUTSPCORELIB	9-2
WSVALIDATELIB	9-7
RESPONSEUTILIB.....	9-11
COREDATETIMELIB	9-32
COREDATAGENLIB	9-35
COREVALIDATEVARIABLELIB.....	9-35
COREVERIFYCONDITIONVARIABLELIB.....	9-41
CORESTOREVALUES	9-46
COREFILEOPS.....	9-46
CORESTRINGOPS.....	9-47

CORENUMBEROPS	9-47
COREUTAOPS	9-50
Chapter 10	
Custom Libraries	10-1
Creating/Updating Custom Libraries.....	10-2
Exporting/Importing Custom Libraries	10-4
Using Custom Library Functions	10-4
Appendix A	
Web Service Component Keywords.....	A-1
WS-SETWEBSERVICENAME.....	A-2
WS-SETXMLELEMENT	A-2
WS-SETXMLLISTELEMEN.....	A-2
WS-SETVARIABLE	A-3
WS-SETVARIABLEFROMRESPONSE.....	A-3
WS-SETTRANSACTIONTYPE	A-3
WS-LOGMESSAGE.....	A-3
WS-CREATEWSREQUEST	A-4
WS-PROCESSWSREQUEST	A-4
WS-STARTPOLLSIF	A-4
WS-STOPPOLLWSIF.....	A-5
Appendix B	
GUI Component Keywords	B-1
APPROVE	B-2
CANCEL	B-2
CHECK.....	B-2
CLICK.....	B-3
CLOSE.....	B-3
GET_ATTRIBUTE_VALUE	B-3
GET_ATTRIBUTE_ID	B-4
LAUNCH	B-4
MAXIMIZE	B-4
MINIMIZE	B-4
POPUP.....	B-5
PRESSTABKEY	B-5
SELECT.....	B-5
SETTEXT	B-6
SWITCHTO.....	B-6
UNCHECK.....	B-6
UNSELECT.....	B-7
UI-STARTBROWSER.....	B-7
UI-ENDBROWSER.....	B-7
WAIT.....	B-7
Appendix C	
REST Component Keywords.....	C-1
RS-SETREQUESTHEADER	C-2
RS-SETENDPOINT.....	C-2
RS-ARGUMENT.....	C-2
RS-SETMETHOD.....	C-3
RS-PROCESSRESTREQUEST	C-3
Appendix D	
Setting Up Inbound Web Services.....	D-1
Creating Inbound Web Services.....	D-2
Importing Inbound Web Services.....	D-2
Searching Inbound Web Services.....	D-2

Appendix E

Generating Re-runnable Test Data	E-1
--	-----

Appendix F

Connecting to Multiple Databases	F-1
--	-----

Appendix G

Configuring Authentication for Web Service Requests.....	G-1
--	-----

Appendix H

OUTA REST Services.....	H-1
Prerequisites.....	H-2
Flow Run.....	H-2
Flow Set Run	H-4
Flow Run Analytics	H-5
Flow Set Run Analytics.....	H-6
Flow Run Summary.....	H-6
Flow Set Run Summary	H-7

Preface

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

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

This preface focuses on the following:

- [Audience](#)
- [Prerequisite Knowledge](#)
- [Abbreviations](#)
- [Related Documents](#)
- [Updates to the Documentation](#)
- [Documentation Accessibility](#)
- [Conventions](#)

Audience

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

Prerequisite Knowledge

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

Abbreviations

The following terms are used in this document:

Term	Expanded Form
CCB	Oracle Utilities Customer Care and Billing
C2M	Oracle Utilities Customer To Meter
MDM	Oracle Utilities Meter Data Management
MWM	Oracle Utilities Mobile Workforce Management
WAM	Oracle Utilities Work and Asset Management
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*

Installation and Administration Guide

- *Oracle Utilities Testing Accelerator Installation and Administration Guide*

User and Reference Guides

- *Oracle Utilities Testing Accelerator Upgrade Guide*
- *Oracle Utilities Testing Accelerator Security Guide*
- *Oracle Utilities Testing Accelerator Licensing Information User Manual*

Additional Documentation

The following resources are available on [My Oracle Support](#).

- *Practice exercises for Oracle Utilities Testing Accelerator (Doc ID 2726629.1)*
- *Flow Subroutines and Test Data Sets (Doc ID 2632033.1)*
- *Building Custom Components And Functions for Oracle Utilities Application Framework Based Products (Doc ID 2662058.1)*

- *Test Strategy Best Practices Guidance for Oracle Utilities Application Framework Based Products (Doc ID 2659556.1)*

Updates to the Documentation

Documentation updates are posted on the [Oracle Technology Network](#) page as they become available.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the [Oracle's Accessibility Program](#) website.

Access to Oracle Support

Oracle customers have access to electronic support through [My Oracle Support](#). If you are hearing impaired, visit the [Oracle Accessibility Learning and Support](#) website for more information.

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.
<i>italic</i>	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 new custom components to build customized test flows. 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:

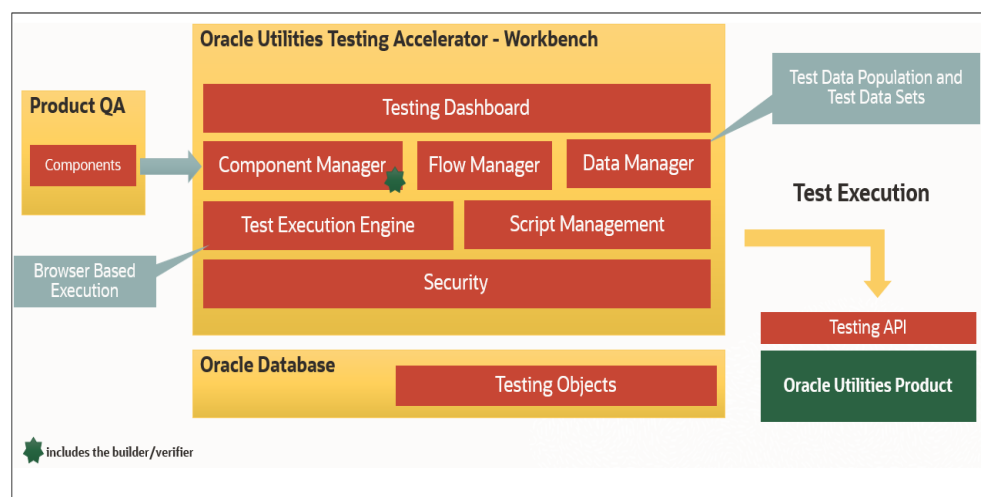
Term	Description
Oracle Utilities Test Accelerator (UTA)	Helps to build and maintain components and flows for automated testing.
Keyword	A pre-defined word used to define a specific step in a test case.
Component	Reusable automated test or part of a test. A component is the building block of an automated 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.
All components and flows in Oracle Utilities Testing Accelerator are organized into hierarchy for better manageability. The hierarchy is: Release > Portfolio > Product > Module	
Release	Represents the highest level of hierarchy. There is one release per an Oracle Utilities Testing Accelerator version, and it contains one or more portfolios.
Portfolio	Represents a product family consisting of one or more related products. A portfolio contains one or more products.
Product	Represents an Oracle Utilities application. For example: CCB A product contains one or more modules.

Term	Description
Module	<p>Represents an Oracle Utilities application functional area. For example: Billing in CCB</p> <p>A module contains one or more components that are used to automate a specific functional area in an Oracle Utilities application.</p> <p>A module in the flow tree hierarchy can be used for logical grouping of custom flows for easier access.</p> <p>Note: The module in the hierarchy of flow and component tree structure is different.</p>
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.



Oracle Utilities Testing Accelerator's workbench can be accessed using a web browser, such as Mozilla Firefox or Google Chrome. The workbench allows users to create and manage components and flows. Additionally, flow runs and their corresponding history can be managed from the workbench.

There are various modules within the workbench:

- The **Component Manager** supports auto generation creation, update and delete of components.
- The **Flow Manager** provides features to create and manage test flows in Oracle Utilities Testing Accelerator.
- The **Data Manager** helps to manage various test data sets, use conversational test data entry and fetch test data for easier entry and management of test data used within a flow.

Security module in Oracle Utilities Testing Accelerator workbench makes sure that only authorized users have access to the workbench. The module also provides necessary support to add authentication to outbound requests that are used for testing an application. Additionally, the module controls the access to the flow runs using the Oracle Utilities Testing Accelerator REST APIs.

All the components and flows are defined using metadata as Testing Objects. The metadata and the flow run history gets stored in the database for unified, concurrent access by various users of Oracle Utilities Testing Accelerator.

Oracle Utilities Testing Accelerator comes with several predefined components provided by the corresponding product's Quality Assurance teams.

All the web service based test flow runs use Testing APIs on the Oracle Utilities Enterprise products. These APIs are web service end points on the Enterprise applications and are delivered along with Oracle Utilities Testing Accelerator.

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

Application Features

The features available in this Oracle Utilities Testing Accelerator release are the dashboard, components, flows, flow sets, 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

Oracle Utilities Testing Accelerator v6.0.0.3 release supports the respective versions that each of the below listed Oracle Utilities product currently supports:

- Oracle Real-Time Scheduler
- Oracle Utilities Mobile Workforce Management.
- Oracle Utilities Customer Care and Billing
- Oracle Utilities Customer To Meter
- Oracle Utilities Work and Asset Management
- Oracle Utilities Operational Device Management
- Oracle Utilities Meter Data Management
- Oracle Utilities Smart Grid Gateway

Refer to the *Certification Matrix for Oracle Utilities Products (Document ID 1454143.1)* on [My Oracle Support](#) to determine if support for newer versions of the listed products have been added.

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
- Create/manage Oracle Utilities Testing Accelerator application user accounts
- Upgrade CM (custom) content from one version of an Oracle Utilities application's Test Accelerator pack to a later version

For example: From Oracle Utilities Customer Care and Billing v2.6.0.0 to Oracle Utilities Customer Care and Billing v2.6.0.1

- Purging old flow run log and history
- Create and manage custom function libraries

For more details, refer to [Chapter 4: Oracle Utilities Testing Accelerator Administration](#).

Components

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

- Create a new component
- Define/update the definition of a component
- Submit the component for approval
- Accept/reject the approval based on the state of the component

For more information about components, refer to [Chapter 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
- Total number of flows run and their break down into passed vs failed
- Total number of flow sets run and their break down into passed vs failed

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 **Get All Notifications** to display all unread notifications applicable to the current user.

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 a 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/FYA 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. (example: Reject, Revert to Approve, Approve, or Send to in progress (Flow)). Click the **Read** column to mark a ToDo notification as 'read'.

Flows

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

- Create a new flow
- Define the flow
- Submit the flow for approval
- Accept/reject the approval based on the state of the flow

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:

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

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

- [Metadata Driven Automation Development Methodology](#)
- [Setting Up Automation Development Environment](#)

Metadata Driven Automation Development Methodology

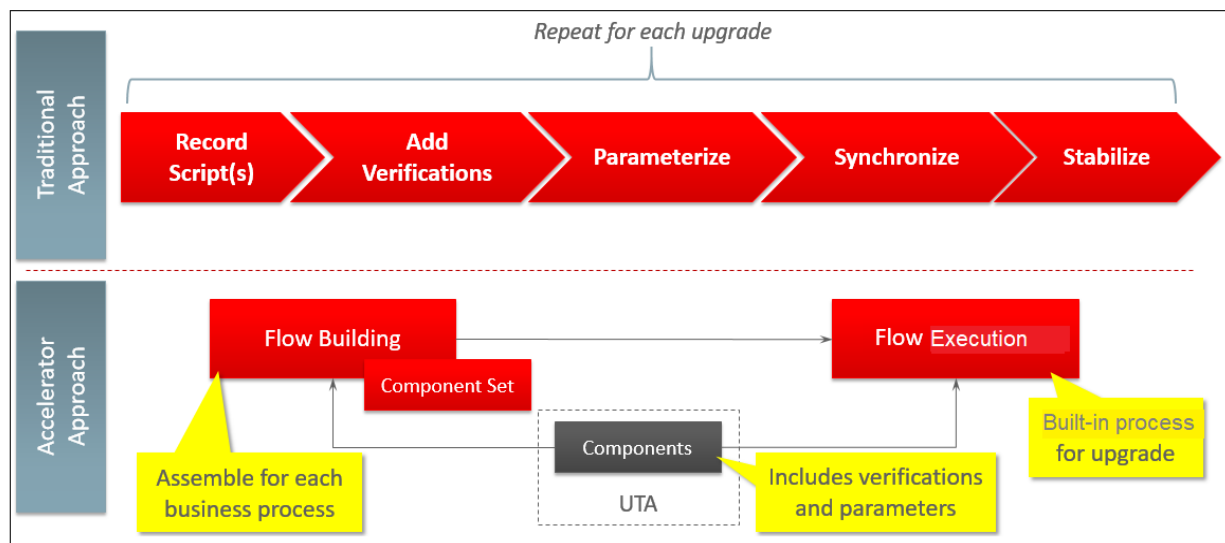
This section describes the metadata-driven automation development methodology that enables a test automation engineer to create automation scripts for an Oracle Utilities application.

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

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

- 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 running 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 Run](#)

Planning

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

For details about how to extend the components, refer to the [Copying Components](#) section in [Chapter 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 [Chapter 5: Creating Test Flows](#).

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

Test Run

To run the automated test flow, execute the script in Oracle Utilities Testing Accelerator workbench. Alternately, the scripts can be generated and run using the Eclipse IDE.

To use another data set to run the script in Eclipse IDE, change the databanks in the generated scripts project and execute the script. For more details, refer to the [Running Test Flows](#) section in [Chapter 5: Creating Test Flows](#).

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

Setting Up Automation Development Environment

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

- Step 1: [Setting Up Oracle Utilities Testing Accelerator Server](#)
- Step 2: [Setting Up Workstations for Development/ Testing](#)
- Step 3: [Setting Up Application under Test](#)

Setting Up Oracle Utilities Testing Accelerator Server

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

- [Installing Oracle Utilities Testing Accelerator](#)
- [Installing Oracle Utilities Testing Accelerator Client Runtime](#)
- [Creating Users](#)

Installing Oracle Utilities Testing Accelerator

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

Installing Oracle Utilities Testing Accelerator Client Runtime

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

Note: Oracle Utilities Testing Accelerator Client Runtime is required only for GUI based flow run. For web service based flow run, it is recommended to use Oracle Utilities Testing Accelerator work bench (the client runtime need not be installed for running web services based test flows in OUTA).

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

Note: The Oracle Utilities Testing Accelerator application need not be installed on the user workstations. Users only need:

- A browser access to it for the component and flow development.
- An installation of Eclipse IDE for Java Developers with Oracle Utilities Testing Accelerator Eclipse plugin to enable the run of flows.

Creating Users

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

Setting Up Workstations for Development/ Testing

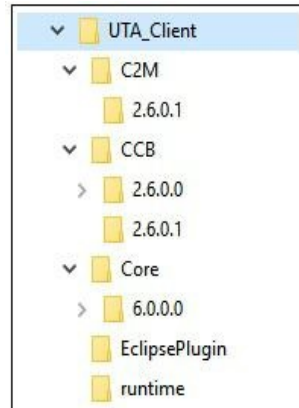
This section provides the steps to set up the Oracle Utilities Testing Accelerator developer workstations. The tasks include:

- [Extracting Oracle Utilities Testing Accelerator Client Runtime](#)
- [Installing Oracle Utilities Testing Accelerator Eclipse IDE for Java Developers](#)
- [Installing Oracle Utilities Testing Accelerator Eclipse Plugin](#)

Extracting Oracle Utilities Testing Accelerator Client Runtime

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

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



Creating Oracle Utilities Testing Accelerator Client Runtime Folder Structure

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

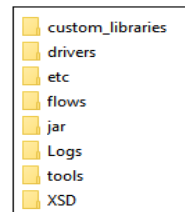
1. Create/select a folder to use as your runtime folder.

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

2. Copy the contents of the runtime folder from UTA_Client.zip zip into <UTA_CLIENT_WORK_DIR>.

For more details about UTA_Client.zip refer to the [Extracting Oracle Utilities Testing Accelerator Client Runtime](#) section.

After copying the contents of the runtime folder, the <UTA_CLIENT_WORK_DIR> should look as follows.



- **drivers**

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

- **etc**

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

Note: All passwords contained in this file have to be encrypted. The Password Encryption Tool can be used to encrypt plain-text passwords. For more details, refer to the [Encrypting Passwords](#) section in [Chapter 5: Creating Test Flows](#).

- **flows**

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

- **jar**
The 3rd party jar files that are needed to execute the Oracle Utilities Testing Accelerator flows.
- **Logs**
The runtime generated test run logs that can be later used for debugging.
- **tools**
The Password Encryption Tool that you can use to encrypt any passwords. For information about how to encrypt passwords that are stored in the configuration.properties file, refer to the [Encrypting Passwords](#) section in [Chapter 5: Creating Test Flows](#).
- **xsd**
The run-time generated XSDs required for processing the web services request.

Installing Oracle Utilities Testing Accelerator Eclipse IDE for Java Developers

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

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

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

1. Download Eclipse IDE for Java Developers from the following location:
<https://www.eclipse.org/downloads/packages/release/2019-09/r/eclipse-ide-java-developers>
2. Extract the downloaded zip file to a folder. For example: C:\UTA_Eclipse
3. Navigate to the unzipped Eclipse folder and double-click the eclipse.exe file to launch the Eclipse IDE.
4. When prompted for workspace, select the <UTA_CLIENT_WORK_DIR>/flows folder (as created in the [Creating Oracle Utilities Testing Accelerator Client Runtime Folder Structure](#) section) as the workspace folder.

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

Installing Oracle Utilities Testing Accelerator Eclipse Plugin

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

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

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

For instructions to extract the plugin, refer to the [Extracting Oracle Utilities Testing Accelerator Client Runtime](#) section.

2. Launch Eclipse.

For instructions to install Eclipse, refer to the [Installing Oracle Utilities Testing Accelerator Eclipse IDE for Java Developers](#) section.

3. Navigate to **Help** menu > **Install New Software**.
4. In the **Install** dialog box, click **Add**.

5. Navigate to **Location**. Enter the following URL and click **Add**. The URL is added as one of the update sites to Eclipse.

<http://download.eclipse.org/nebula/releases/latest>

6. Again, in the **Install** dialog box, click **Add**.
7. On the **Add Repository** dialog box, click **Local**.
8. Browse to the location where the UTA_Eclipse_Plugin.zip was extracted and click **OK**.
9. Click the **OUTA Eclipse Plugin** checkbox.
10. Click **Next**.
11. Accept the **License Agreement** and click **Finish**.
12. When prompted for click **Install Anyway**.
13. Restart Eclipse IDE to use the Oracle Utilities Testing Accelerator Eclipse Plugin.
14. To use the Oracle Utilities Functional Test perspective, navigate to **Window > Perspective > Open Perspective > Other...**
15. Select Oracle Utilities Functional Test from the list of perspectives.
16. Click **Open**.

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

Setting Up Application under Test

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

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

Chapter 4

Oracle Utilities Testing Accelerator Administration

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

- [Overview](#)
- [Administration Tab](#)

Overview

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

- Create/edit release, portfolio, product, and modules
- Create/manage Oracle Utilities Testing Accelerator application user accounts
- Upgrade CM content from one version of an Oracle Utilities application to a later version.

For example: From Oracle Utilities Customer Care and Billing V2.6.0.0 to Oracle Utilities Customer Care and Billing V2.6.0.1

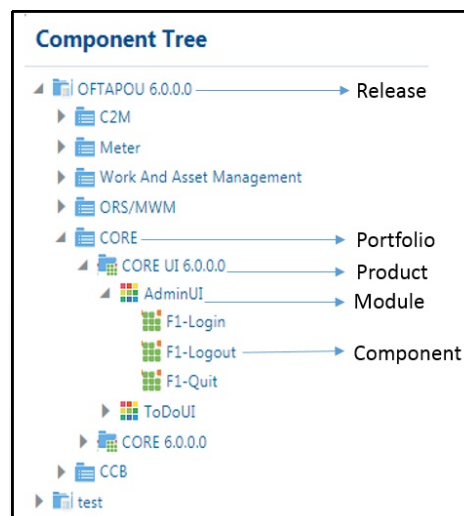
- Purging old flow run logs/results
- Create and manage custom function libraries

Administration Tab

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

- [Managing Products](#)
- [Managing Modules](#)
- [Managing Users](#)
- [User Access Types](#)
- [Purging Flow Run Data](#)
- [Purging Notification Data](#)
- [Custom Content Upgrade](#)

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



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

Though this is an admin function, a product can be deleted via the component or flow tree structure only. Only an administrator can exercise the delete option.

It is always a best practice to export all the custom flows and components from a product hierarchy before deleting the product as a whole. Deleting a product removes all the flows and components under the product hierarchy permanently. Appropriate caution should be exercised while using this feature.

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**

Note: If a product (example: Oracle Utilities Customer Care and Billing V2.7.0.3.0) includes flows that use components from another product (example: Oracle Utilities Meter Data Management V2.3.0.1.0), to delete the Oracle Utilities Meter Data Management product, the flows in the first product (Oracle Utilities Customer Care and Billing V2.7.0.3.0) that use the components from the second product should be deleted first.

Managing Modules

A module represents an Oracle Utilities application functional area for the components. Example: Billing in CCB

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).

Managing Users

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

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

To get to the **Administration** page:

1. Log into the application.
2. Navigate to the **Administration** tab.
3. On the **Administration** page, details of all users are shown in the right pane.

Creating a User

To create a new user:

1. On the **Administration** page, click **Add** to create a new user.
2. Fill in the following fields and click **Save**.

- Username
- Full name
- Manager name
- Password (length should be 6-15 characters)
- E-mail
- Access type (For information about access types, refer to the [User Access Types](#) section.)

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

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

Searching for User(s)


To search for a user:

1. On the **Administration** page, click **Search** to search for a specific user.
2. Enter “%” in the **User Name** field to view all the existing users.

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

Updating User Details

To update the details of an existing user:

1. On the **Administration** page, click **Search** and select the record you want to update.
2. Modify the required fields and click the update icon .

Deleting a User

To delete an existing user:

1. On the **Administration** page, click **Search** to select the record to be deleted.
2. Click the delete icon .


A window appears asking for confirmation about the deletion.

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

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

Changing a User's Password

To change the password of an existing user:

1. On the **Administration** page, click **Search** and select the record for which you want to change the password.
2. Click the change password icon .
3. On the **Update Password** window, enter the new password in both the fields.
4. Click **Submit**.

On success, the “Password is changed” message is displayed.

User Access Types

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

Access	Developer	Approver	Administrator
Approve/reject a component		✓	✓
Approve/reject a flow		✓	✓
Change Password (self)	✓	✓	✓
Component - Create/update/view/delete	✓	✓	✓
Copy Component	✓	✓	✓
Create Module			✓
Create Portfolio			✓
Create Product			✓
Create Release			✓
Export Component	✓	✓	✓
Export Flow	✓	✓	✓
Flow - Create/update/view/delete	✓	✓	✓
Generate a component from WSDL	✓	✓	✓
Import Component	✓	✓	✓
Import Flow	✓	✓	✓
Update Module	✓	✓	✓
Update Portfolio	✓	✓	✓
Update Release	✓	✓	✓
User Management			✓
<ul style="list-style-type: none"> • Create/update/view/delete • Change other user password 			

Purging Flow Run Data

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

- **Flow Run Logs** - Allows purging of all the flow run log files that meet the specified criteria.
- **Flow Run History** - Allows purging of flow run history that helps in keeping the Flow Run History page more manageable.

Purging Notification Data

An administrator can decide to purge some of the existing notifications for maintenance purposes. The notifications can be purged by specifying the cut-off date for purging entries; the data older than the specified date will be purged.

- **Notifications** - Allows purging of all the notifications that meet the specified criteria.

Custom Content Upgrade

Version specific Oracle Utilities Testing Accelerator test components are released with each of the Oracle Utilities Enterprise major version updates, such as Oracle Utilities Customer Care and Billing V2.7.0.1.0, V2.7.0.3.0, etc. The custom content upgrade process allows custom flows to be automatically upgraded to the latest component pack version that corresponds to the latest version of Oracle Utilities Enterprise application.

Example: Flows may have been built using components from Oracle Utilities Customer Care and Billing V2.6.0.1.0. When a new version (Oracle Utilities Customer Care and Billing V2.6.0.1.0) is released, a corresponding set of components for V2.7.0.3.0 are also released as part of Oracle Utilities Testing Accelerator. Using the CM Content Upgrade option in the administration, the flows can be automatically upgraded to use the components from the latest V2.7.0.3.0 instead of components from V2.6.0.1.0.

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

The CM Content Upgrade process checks to see if there are any structural changes in each of the components between old and newer versions of the product pack. If any changes are found, the flows using the updated components are automatically highlighted, so you can review (updated test data if required) and clear the highlight marker for each flow. If required, the highlight marker can be cleared at once for all the flows directly at the module or product level.

Running the CM Content Upgrade Process

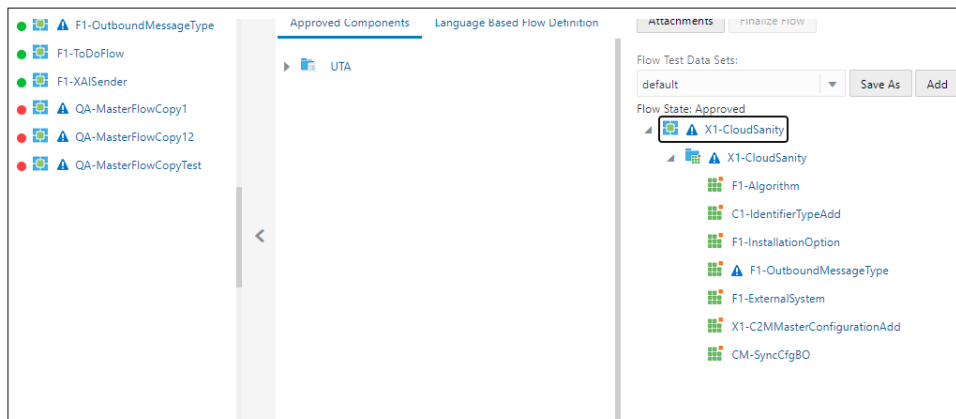
To upgrade an existing set of flows:

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 exist.
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.

When you upgrade the custom test flows to a newer version of a product pack using the CM Content Upgrade feature in Oracle Utilities Testing Accelerator, flows using components that have been updated between the older and the newer version will automatically be highlighted with a marker. This ensures that you have clear visibility into the impact of changes in the application being tested on the automated test flows.

The following figure shows the flows marked with the highlight marker as part of the CM Content Upgrade process.



Clearing the Highlight Markers

The CM Content Upgrade process checks to see if there are any changes in the component between the current/older and a newer version of the product pack and highlights a Flow with a marker, if any component used in the flow has changed in its structure. The feature also highlights the component in the flow which caused the flow to be highlighted. This enables you to quickly identify and update the test data in the flows that may have been impacted because of the upgrade, without having to run the flows first. Navigate to each of the highlighted flows, review it, update test data if necessary.

After updating the test data, clear the highlight marker. Right-click the flow and select **Clear Highlight** to clear the highlight marker. Alternately, the highlight marker can be cleared at the module or product level. Right-click the module/product in the flow tree in the leftmost frame and select **Clear Highlighted Flows**. Clearing the highlight marker at the product or the module level clears the marker for all the flows under the corresponding module/product.

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 prefixes “CM” to the component name. All references to this component in flows will be updated accordingly so that the flow remains intact. 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.

Running the CM Content Upgrade Process for Integration Flows

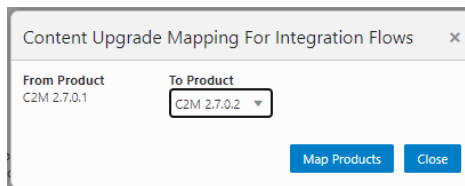
Integration flows are developed using components from two or more Oracle Utilities Testing Accelerator product packs belonging to different Oracle Utilities Enterprise applications.

Triggering the CM Content Upgrade process is the same for both the integration flows and non-integration flows. To upgrade an existing set of flows, follow the steps in the [Running the CM Content Upgrade Process](#) section.

After step 7, during the initiation phase, the CM Content Upgrade process checks to see if any of the flows being upgraded use components from two or more product packs. If it finds such a

flow/flows, it determines them as integration flows. The CM Content Upgrade process will then prompt to select the “from” and “to” product pack versions for each of the source product from which components have been used in the flow.

The following figure shows the mapping option for upgrading integration flows.



After selecting the appropriate “from” and “to” product versions, click **Map Products**. The CM Upgrade process upgrades the flows by mapping the components appropriately between various product packs.

The flows being upgraded will still be created under the **To Product** specified in the main screen of the CM Content Upgrade process (before step 7 of the process). The upgrade mapping for integration flows only defines the component mapping to be done for integration flows.

Chapter 5

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 run.

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](#)
- [Adding Email Capabilities to Flows](#)
- [Support for HTTPS Web Services](#)
- [Support for Integration Flows](#)
- [Running Test Flows](#)
- [Running Flows from Command Line](#)
- [Encrypting Passwords](#)

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**.

Creating Flows

You can create a flow:

- By drag-and-drop of components into a flow ([Creating Flows By Dragging-and-Dropping Components](#))
- By including another flow into the flow being created ([Using Global Variables](#))

Creating Flows By Dragging-and-Dropping Components

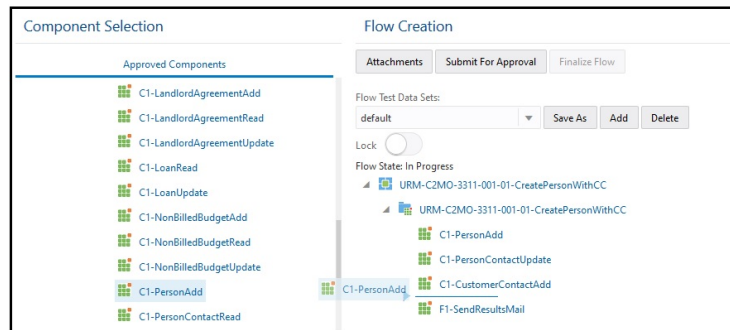
Before creating a flow, identify the components required to create the flow.

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.
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 scenario or components within the flow

structure. The component moved will be added below the scenario/component to which it was moved.



Note: Flow definition can be modified (components added or removed) only if the flow is in “In Progress” state.

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

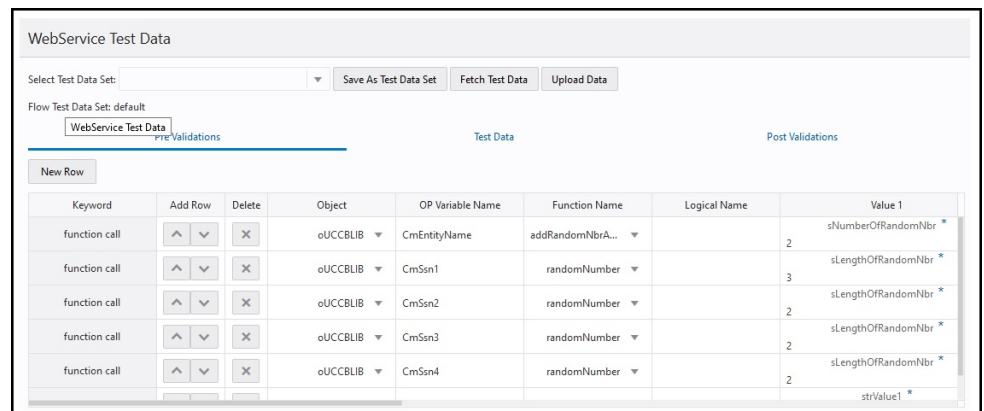
Adding Test Data in a Flow

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 library in which the function exists can be selected in the column “Object”: Based on the library selected the function can be selected from the “Function Name” drop-down list. If the function outputs a value, provide the custom global variable name in the **OP Variable Name** column into which the function output is stored. This variable can be used as test data in the **Test Data** section. The function inputs can be specified against the Value1, Value2 through to Value 6 columns based on the number of input parameters of the function needs. The variable names defined in the pre-validations and post-validations sections will be automatically prefixed with “fvar” and presented in the test data column’s drop down (value 1 through value 6 columns), so they can clearly be distinguished from the global variables defined in the component definition.

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 global 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**.
- Enable or disable the validations/functions defined in the component by appropriately switching the **Enable** checkbox in the first column. If the checkbox is not selected, during the course of the test run, the function/validation will not be triggered. The checkbox only appears for rows to which this feature is applicable.

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””.

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 by not selecting the checkbox corresponding to the validation line in the test data UI for the component. And, if any new or more of these verifications are to be added based on the flow specific requirements or if a specific set of values have to be retrieved from the response of the component run, the **Post Validations** section can be used.

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. Specification of functions in the **Post Validations** section follows the same pattern as the one specified in the **Pre-validations** section.

The screenshot shows the 'WebService Test Data' interface. It includes a 'Select Test Data Set:' dropdown, 'Save As Test Data Set', 'Fetch Test Data', and 'Upload Data' buttons. Below, there are tabs for 'Pre Validations', 'Test Data', and 'Post Validations'. A 'New Row' button is present. The main table has the following structure:

Keyword	Add Row	Delete	Object	OP Variable Name	Function Name	Logical Name	Value 1
function call	▲ ▼	✕	w\$VALIDATELIB ▼		elementValueEqu... ▼		expectedValue *

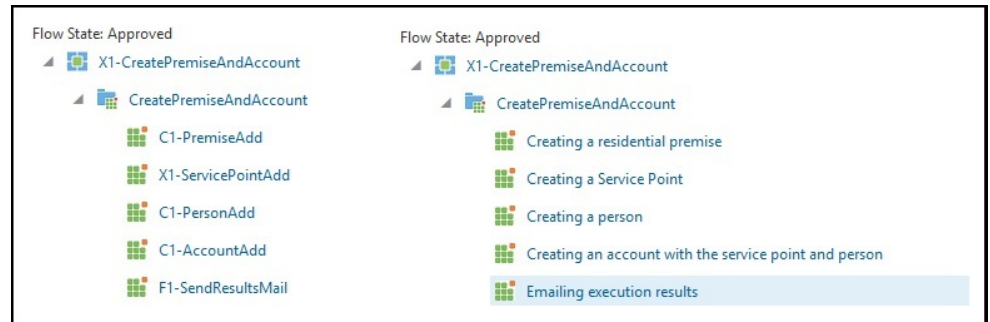
Only function calls can be added in the **Post Validations** section. For more details, refer to the **Flow-Level Validations** section in [Chapter 5: Creating Components](#).

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. The annotation can be removed or updated through the same process. Clearing the component description in the flow removes the annotation and displays the component name. The following figure shows a flow without and with annotations.



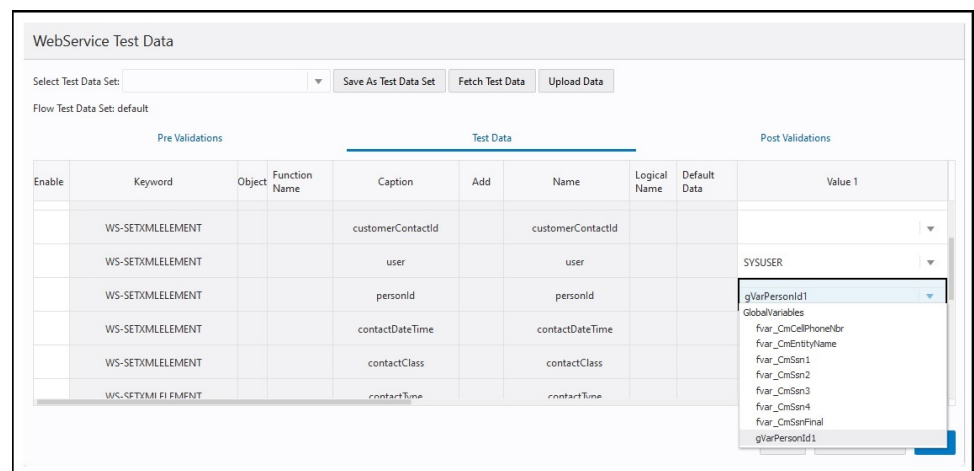
Using Global Variables

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.

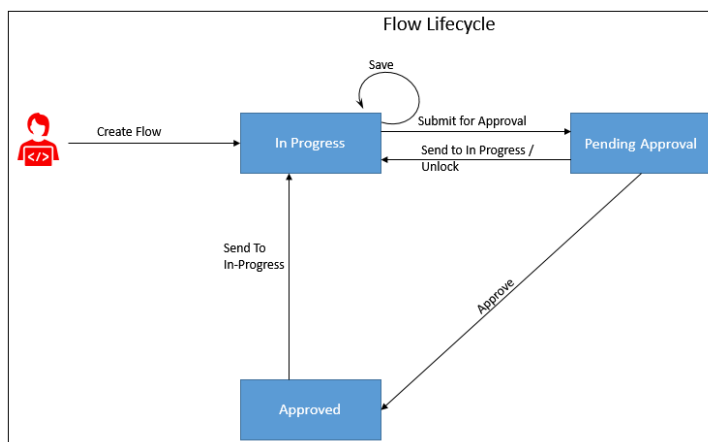


Each of the base components expose one or more global variables that hold the output of the component during execution. These global variables can be used to set the output of one component as the input of another component.

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 its 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.

Note: Use the **Search Component** field on the top of the approved component tree to find the components you need. The available components are listed similar to a type ahead search, with the product and module names under which the component is available. Select the appropriate component from the prompted results and the corresponding component is highlighted in the approved component tree.

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. Login to the application.
2. Navigate to the **Flows** menu.
3. In the left pane, right-click the flow for which components have to be reordered.

Note: Use the **Search Component** field on the top of the approved component tree to find the components you need. The available components are listed similar to a type ahead search, with the product and module names under which the component is available. Select the appropriate component from the prompted results and the corresponding component is highlighted in the approved component tree.

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. Login to 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.). Provide the required WSDL name, operation name (typically read operation), the user credentials to access the WSDL and 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
07VCbR2jhxtkUH06Fuz+zmChpGSCr241KggFC6FwgMg==)
```

To fetch the test data for an Oracle Utilities Meter Data Management component:

- a. Select the flow/user configuration file from the drop-down menu.
- b. 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 click **Fetch Test Data**.

Use Configuration Properties	Flow Configuration ▼	User Configuration ▼
Web Service Name *	<input type="text"/>	<input type="text"/>
Web Service Operation Name *	<input type="text"/>	<input type="text"/>
Application Username *	<input type="text"/>	<input type="text"/>
Application Password *	<input type="text"/>	<input type="text"/>

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

Unit Testing a Component in a Flow

As part of the flow development, test data needs to be provided for a component in a flow. After the test data is added, a component may have to be unit tested to make sure that the provided test data gets the flow working as expected.

To unit test a component that is part of a flow:

1. Navigate to the **Flows** tab.
2. Select and right-click the flow. 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, provide the web service name in the component's test data. Also, provide the operation name/transaction type in the appropriate component's test data field. Fill up all the test data for the component as necessary.
5. Save the test data and click **Close**.

6. On the **Flow Definition** page, right-click the component and select the test component.
This will open up the conversational test data entry screen for the component.
7. Select the flow and user configuration needed to test the component. Click **OK**.
8. Click **Send** to post the request to the application being tested.
9. After receiving the response, validate it (for errors) to see if the test data provided is appropriate. Else, adjust the test data and click **Send** to send a new request to the application being tested.
10. Repeat step 9 till the expected response is obtained.
11. Once the appropriate test data is set and the response is as expected, click **Save** to save the updated test data into the component's test data in the flow.

Note: Clicking **Save** will only replace any static values provided in the component's test data for a given element. If the **Test Data** field for a component line contains a global variable, the variable in the field will not be replaced by the static data in the request being saved.

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 subroutines 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, 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. Make sure not to nest the subroutines to include other subroutines.

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**.

Subroutine Interface

Release: UTA Portfolio: CUSTOMER CLOUD SERVICE

Product: CCS 20A Flow: URM-C2MO-3311-004-01-CreatePersonAn

Variable Type	VariableName	Delete
OUT ▼	gVarAccountId1 ▼	Delete
OUT ▼	gVarPersonId1 ▼	Delete
OUT ▼	fvar_CmEntityName ▼	Delete

Subroutine Interface

Add IN/OUT Variable Save Close

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.

Subroutine Test Data

Subroutine: 10-3311-004-01-CreatePersonAndAccount Scenario: URM-C2MO-3321-001-01-StartServiceNew

Flow: URM-C2MO-3321-001-01-StartServiceNew Flow Test Data Set: default

Create New Variable Subroutine Flow Test Data Set: default

Loop subroutine:

Param Type	Variable	Value	Delete
SETVARIABLE	personId		Delete
SETVARIABLE	accountId		Delete
OUT	accountId ▼	gVarAccountId1	
OUT	personId ▼	gVarPersonId1	

Open Looping Interface Save Close

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.

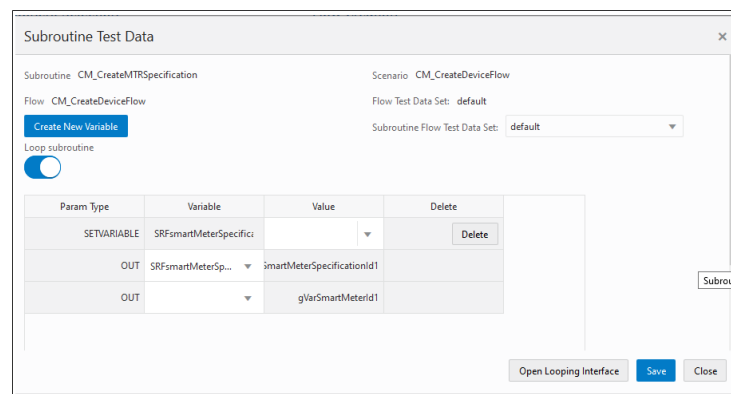
Running Subroutine in a Loop

To achieve the capability to loop one or more components within a flow, create the component(s) as a subroutine. A subroutine in a loop can be run 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 (from the point of failure) feature will not work if the flow has a loop defined.

To define subroutine looping, add the sub-routine flow to the parent flow. To specify the loop criteria and other details for the subroutine, open the **Test Data** page of the subroutine within the parent flow. Enable the **Loop** subroutine switch and click **Open Looping Interface** to provide the criteria for executing the subroutine in a loop.

The figure below shows the **Loop** subroutine switch and **Open Looping Interface**.



Open Looping Interface provides the following options:

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

Example: If the subroutine creates a person entity in the application, specify the value 10 to run 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 run 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).

The following figure shows a subroutine looping interface with the incremator type selected as number.

- **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 incremator 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 Incrementer 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 Incrementer value is equal to 5.

Variable: “Incrementer” Condition: “equals to” Value: “5”

The incremator is a global variable that can be used for setting incrementing test data in the flow.

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. To use the incremator date as an input to a test data field, the date format may be converted to suite the test data needs, using the delivered functions in the COREDATE/TIMELIB or if necessary, a custom function may be created.

Conditional Bypass of Components in a Flow Run

This feature supports finer control of a flow run. It can be specified whether a component has to be skipped or run as part of the flow run, based on the custom conditions in the component's test data in the flow. The feature can be used to selectively run or skip one or more components based on the outcome of the previous component step or based on the Flow Test Data set that is used, as part of the flow run.

To bypass/skip the running of a component within a flow:

1. Login to the application and navigate to the **Flows** tab.
2. On the left navigation pane, right-click the flow name and select **Create/Update Flow Structure**.
3. On the right pane, expand the flow structure.
4. Right-click a component and select **Edit Test Data**.

5. In the component's **Test Data** section, click the **Pre-Validations** tab.
6. Add the “skipStep” function from the “CoreUTAOps” library. Set the test data to “true” so that the component may be skipped during the flow run.
7. If the component should not be skipped during the flow run, set the test data for the function to “false”.

Multiple test data sets can be used to set different test data for the function making sure that the specific components in which the function exists may be skipped or run based on the input to the function.

Skipping more than one component in a flow

Additionally, the function can take in global variable which holds the values true or false as input. If certain components in a flow have to be skipped during a run, then set a global variable in the **Pre Validations** section of the first component. The global variable can be used as an input to the skipStep function in various components. Changing the value of the global variable (using flow level test data sets) will make sure that the defined set of components are either skipped or run as part of the flow run.

Skipping components based on outcome of a component step

In cases where certain component/components have to be skipped based on the outcome of a component step result in a flow, the “CoreVerifyConditionVariableLib” library can be used. The functions in the library can be used to validate the response for a component request, much like the functions in the WSValidateLib that are used to validate a response. But, the functions in the “CoreVerifyConditionVariableLib” library output either a false or true value, but do not fail or pass the test.

The output of the functions in “CoreVerifyConditionVariableLib” can be stored into a global variable and then the global variable may be used as input to the “skipStep” function. This allows conditional bypass/running of a component based on the outcome of another component.

Note: If a certain set of components have to be skipped as part of the flow run using this feature, add the “skipStep” function in the **Pre Validations** section of each of the components.

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 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.

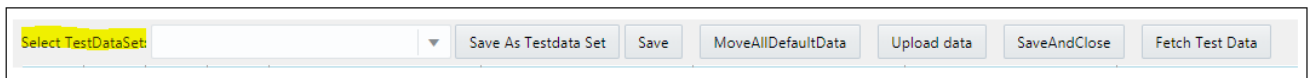
Note: If a test data set with the same name already exists, the application asks for confirmation to overwrite the test data.



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.



Deleting Component Test Data Sets

To delete one or more component test data sets for a given component:

1. Login to Oracle Utilities Testing Accelerator.
2. On the **Components** menu, navigate to the component for which the test data set needs to be deleted.
3. Right-click the component and select **Delete Test Data Sets**.
4. Select the test data set from the popup window. Click **Yes** to delete the selected component test data sets. The test data set/sets should be deleted.

Loading Test Data from a .csv File

Populate the test data of a component using the component's generated databank and filling in the data in the databank. Note that this upload feature is not supported for components with nested groups.

Note:

- The upload will fail if the databank is not in the same format as the generated databank for the component. The upload capability is limited to test data that does not have multilists/groups. It is recommended to use the component test data sets

and flow test data sets in place of CSVs, as the features are designed to provide better reusability of test data.

- This feature will be deprecated in a future release.

To upload test data using the .csv file:

1. Login to Oracle Utilities Testing Accelerator.
2. Navigate to the **Flows** menu and select 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 test data set needs to be created. Right-click the component and click **Edit Test Data**.
5. Click **Load Data**. From the window displayed, drag-and-drop the .csv file to upload.
6. Click **Save** to start loading the test data.

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 run of the flow. This feature is aimed at creating multiple sets of test data per flow and swap between these test data sets before running 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 Run](#).

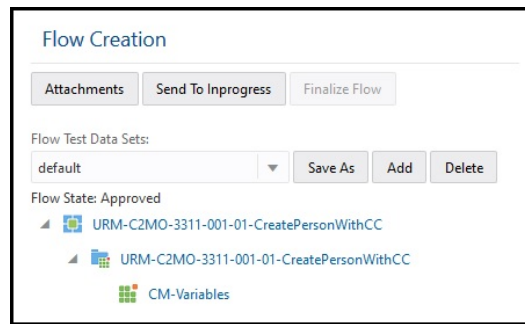
Creating Flow Test Data Sets

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.

- 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:

- Login to Oracle Utilities Testing Accelerator.
- On the **Flows** menu, navigate to the flow in which the component is used.
- Right-click the flow and select **Create/Update Flow Structure**.
- 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.

- 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 Email Capabilities to Flows

For Eclipse based test run

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

Usage Details	Value
Keyword	FUNCTIONCALL
Object	wSCOMMONLIB
Function Name	generateAndSendReport

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

Update the following values as mentioned to configure the email:

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

For Browser based test run

The email capability allows the flow run 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 run 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 for browser based (work bench) flow run.

Support for HTTPS Web Services

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

To import the security key store into Java key store:

1. Enter the URL (HTTPS) of the application in the browser (Example: Internet Explorer).
2. Click **Continue to this Website (not recommended)** link on the **Security certificate** page.
3. Click **Certificate error** in the address bar.
4. Click the **View certificates** link on the **Certificate Invalid** pop-up window.
5. On the **Details** tab, click **Copy to File**.
6. Click **Browse** and select the file you want to export. Click **Next**.
7. Review the settings and click **Finish**.
8. Login to the machine where this certificate has to be imported into the Java key store, and open the command prompt.
9. If the Java path is not set in the environment variables, navigate to the Java/jdk/bin directory and execute the following command:

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

10. Enter “changeit” as the **Password**.
11. Click **Yes** to import the certificate. The property file attributes for HTTPS requests are as follows:

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

The setup is ready to process the HTTPS requests.

Support for Integration Flows

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

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

configuration.properties

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

```
name>\:<port>/ouaf/webservices/ gStrMDMApplication=https\://<server
name>\:<port>/ouaf/webservices/
```

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

Test data:

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

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

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

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

configuration.properties

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

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

```
gStrCCBApplication_gStrApplicationUserName=<%CCBUsername%>
gStrCCBApplication_gStrApplicationUserPassword=<%CCBPassword%>
```

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

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

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

The sample configuration file in this case is as follows.

configuration.properties

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

Configuring Integration Flows for UTA Workbench Based Run

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 and the application URL provided in the flow/user configuration sets. To configure a component in a flow to post a request to a different application as part of an integration flow, 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 second Oracle Utilities Enterprise application with which integration testing should be done. Example: Integration testing between Oracle Utilities Customer Care and Billing and Oracle Utilities Meter Data Management.

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 running it.

Example: Assume that Oracle Utilities Customer Care and Billing is the application to be used with Oracle Utilities Meter Data Management for integration testing 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.

Encrypt	Property Name	Property Value
<input checked="" type="checkbox"/>	CCB_gStrApplicationUserPassword	*****
<input type="checkbox"/>	CCB_gStrApplicationUserName	username
<input type="checkbox"/>	CCB	@ext_pub@myccbserver.mycompany.com/webservices/

The integration flow in Oracle Utilities Testing Accelerator may contain a mix of components to two or more 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 application with which integration testing needs to be performed for Oracle Utilities Meter Data Management 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.

Pre Validations		Test Data		Post Validations
Name	Logical Name	Default Data	WebService Test Data	
Log Message		Start		▼
Web Service Name		ATC1Person	CCB/ATC1Person	▼
Web Transaction Type		C1PersonAdd	C1PersonAdd	▼

During the integration flow run, 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.

Running Test Flows

This section focuses on executing a test flow.

- [Running Test Flows Using a Browser](#)
- [Iterative Flow Run](#)
- [Stopping Flow Run on Validation Failure](#)
- [Stopping Flow Run Manually](#)
- [Viewing Flow Run Details](#)
- [Viewing Flow Run Summary Report](#)
- [Conversational Test Data Management](#)
- [Generating Oracle Utilities Testing Accelerator Scripts](#)
- [Importing the Generated Oracle Utilities Testing Accelerator Script into Eclipse IDE](#)

Running Test Flows Using a Browser

To run 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 run only if it is in either “Pending Approval” or “Approved” state.

3. Select the **Flow Configuration** and **User Configuration** to be used to run the test flow.
4. Click **Run** to start the test flow run.

Note: For more details about flow configuration and user configuration, refer to the [Runtime Configuration for Flow Run \(For Run Using Browser\)](#) section.

5. On the **Flow Run** page, the run 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 run.

Iterative Flow Run

To run 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 **Run 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 run the test flow.
5. Select **Iterative** as the **Flow Run Type**.
6. In the **Number of Iterations** field, specify the number of iterations to run 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 run. 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 runs 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 runs of the flows with first data set and 5 run of the flow with second data set.

8. Click **Run** to start the test flow run.

Note: For more details about flow configuration and user configuration, see [Runtime Configuration for Flow Run \(For Run Using Browser\)](#).

The run details are displayed on the **Flow Run** 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 run logs.

Stopping Flow Run on Validation Failure

By default, the flow run 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 run stop when a validation fails by setting the property “continueExecutionOnFailure” in the user or flow configuration to “false”.

Stopping Flow Run Manually

When a flow starts running, the **Stop Flow** button is enabled in the **Flow Run** page.

To stop a running flow, on the **Flow Run** page showing the current running state of a flow, click **Stop Flow**. The flow run will be stopped.

Note that for flows including subroutines, the parent flow that calls the subroutines should be stopped to stop the flow run. Individual subroutine flow runs cannot be stopped. A stopped flow cannot be resumed.

Viewing Flow Run Details

To view the run 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 run summary. Click **View Run History**.
4. Click the flow run entries to view the respective details of that run.

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

Viewing Flow Run Summary Report

To view the run 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 run summary.
4. Click **View Run History**.
5. Click any of the flow run entries to view the respective details.
6. On the **Flow Run Status** page, click **Summary**.
7. On the **Flow Run Summary Report** page, click **Summary**. The summary of the flow run 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 run 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's test data needs to be populated by providing the web service name and the transaction type.

To navigate to the **Conversational Test Data** GUI, right-click the component in the flow tree structure (in flow development screen) and select **Edit Request**.

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

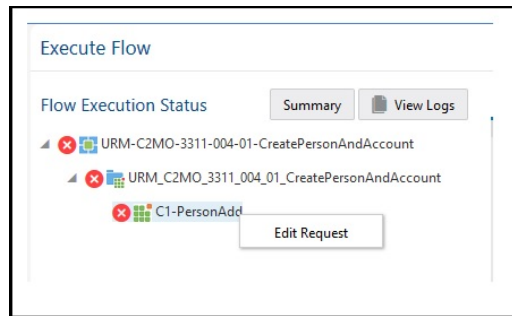
The **Edit Request** feature allows to:

1. Open the failed request of a component.
2. Make changes to the test data and resend the request to edge application (without running the flow multiple times).
3. Observe the response for the modified request.
4. 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 structure screen/flow run interface (including from flow run history, flow set run/history, iterative run/history). User can invoke the

Edit Request user interface. Right-click a component from the flow run 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 application being tested 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.

Generating Oracle Utilities Testing Accelerator Scripts

Note that this section is applicable only for flows containing GUI based components. It is recommended to use the Oracle Utilities Testing Accelerator browser based flow execution feature in all other cases.

To generate Oracle Utilities Testing Accelerator scripts for a flow:

1. Login to the Oracle Utilities Testing Accelerator application.
2. On the flow tree page, right-click a flow and select **Generate Scripts**.

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

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

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

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

Importing the Generated Oracle Utilities Testing Accelerator Script into Eclipse IDE

Note that this section is applicable only for flows containing GUI based components. It is recommended to use the Oracle Utilities Testing Accelerator browser based flow run feature in all other cases.

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

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

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

2. In the Eclipse IDE, right-click the **Project Explorer** panel and click **Import**.
3. Click **General** and select **Existing Projects into Workspace**.
4. Click **Next**.
5. Select **Select archive file:** and click **Browse**.
6. Navigate to the downloaded zip file and click **Open**.
7. Click **Finish**.

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

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

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

Download the drivers from the following URLs:

- **Geckodriver:** <https://github.com/mozilla/geckodriver/releases/download/v0.20.1/geckodriver-v0.20.1-win64.zip>
- **ChromeDriver:** http://chromedriver.storage.googleapis.com/2.40/chromedriver_win32.zip

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

```
package com.oracle.tests;

import static org.junit.Assert.assertNotNull;

public class F1_BundleImport extends FunctionalTestScript{

    WSCOMMONLIB wSCOMMONLIB = new WSCOMMONLIB();
    WSVALIDATELIB wSVALIDATELIB = new WSVALIDATELIB();
    OUTSPCORELIB outSPCORELIB = new OUTSPCORELIB();
    OUAFLIB oUAFLIB = new OUAFLIB();

    final static boolean runFromLastPointOfError = false;

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

Running Flows from Command Line

Note that this section is applicable only for flows containing GUI based components. It is recommended to use the Oracle Utilities Testing Accelerator browser based flow run feature in all other cases.

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

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

For information about the folder, refer to the [Creating Oracle Utilities Testing Accelerator Client Runtime Folder Structure](#) section.

2. Open a Windows command line window at this location.
3. Run the flow as follows:

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

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

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

Encrypting Passwords

Note that this section is applicable only for flows containing GUI based components. It is recommended to use the Oracle Utilities Testing Accelerator browser based flow run feature in all other cases.

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

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

- From Windows Explorer ([Generating Keystore for Encryption from Windows Explorer](#))
- From Command Line ([Generating Keystore for Encryption from Command Prompt](#))

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

- From Windows Explorer ([Using Password Encryptor Tool From Windows Explorer](#))
- From Command Line ([Using PasswordEncryptor Tool From Console/Command Line](#))

Generating Keystore for Encryption from Windows Explorer

Note that this section is applicable only for flows containing GUI based components. It is recommended to use the Oracle Utilities Testing Accelerator browser based flow run feature in all other cases.

To generate the keystore for encryption from Windows Explorer:

1. Launch **Windows Explorer**.
2. Navigate to the <UTA-CLIENT-WORK-DIR>\tools folder.
3. Double-click the **KeystoreGenerator.jar** file.
4. If the keystore and password store do not exist, they will be generated. Else, you will be prompted to confirm to overwrite the existing keystore and password store with the new one.

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

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

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

Generating Keystore for Encryption from Command Prompt

Note that this section is applicable only for flows containing GUI based components. It is recommended to use the Oracle Utilities Testing Accelerator browser based flow run feature in all other cases.

To generate the keystore for encryption from the command prompt:

1. Navigate to the <UTA-CLIENT-WORK-DIR>/tools folder.
2. Run the command: java -jar KeystoreGenerator.jar.
3. If the keystore and the password store for the keystore do not already exist, they will be generated. Else, you will be prompted to overwrite the existing keystore.
4. Enter 'y' to generate a new keystore. Enter 'n' if you want to continue using the existing keystore.

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

Using Password Encryptor Tool From Windows Explorer

Note that this section is applicable only for flows containing GUI based components. It is recommended to use the Oracle Utilities Testing Accelerator browser based flow run feature in all other cases.

Important! Make sure to generate the keystore before encrypting the configuration.properties file. For instructions, refer to the [Generating Keystore for Encryption from Windows Explorer](#) section.

To use the password encryption tool from Windows:

1. Launch **Windows Explorer**.
2. Navigate to the <UTA-CLIENT-WORK-DIR>\tools folder.
3. Double-click the **PasswordEncryptor.jar** file.

You will be prompted to enter the password.

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

A dialog box with encrypted text appears.

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

Using PasswordEncryptor Tool From Console/Command Line

Note that this section is applicable only for flows containing GUI based components. It is recommended to use the Oracle Utilities Testing Accelerator browser based flow run feature in all other cases.

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

1. Navigate to **<UTA-CLIENT-WORK-DIR>/tools** directory.
2. Enter the following command:

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

The tool prompts to enter a password.

3. Type in the password and press **Enter**. A dialog box with encrypted text appears.
4. Copy this text and enter it as the value of the respective password field in the configuration.properties file.

Configuring the Runtime Properties (For Run Using Eclipse)

Note that this section is applicable only for flows containing GUI based components. It is recommended to use the Oracle Utilities Testing Accelerator browser based flow run feature in all other cases.

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

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

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

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

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

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

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

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

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

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

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

For properties related to the integration environment, refer to the [Support for Integration Flows](#) section.

Runtime Configuration for Flow Run (For Run Using Browser)

A test flow is run 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 run, 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 running 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 run.

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

Chapter 6

Creating Components

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

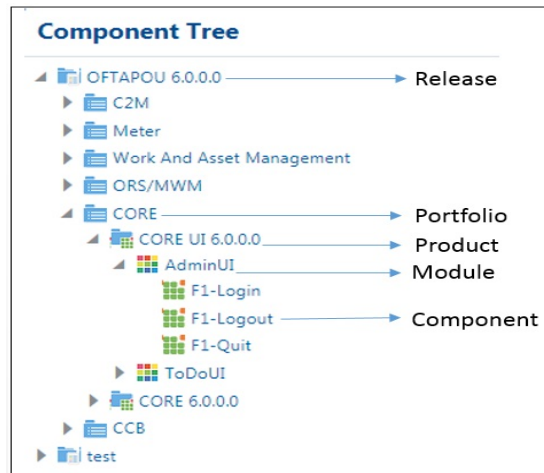
Oracle Utilities Testing Accelerator Release > Portfolio > Product > Module > Components

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

- [Component Structure](#)
- [Component Lifecycle](#)
- [Component Types](#)
- [Creating Web Service Based Components](#)
- [Creating GUI Based Components](#)
- [Creating REST Web Service Components](#)
- [Copying Components](#)

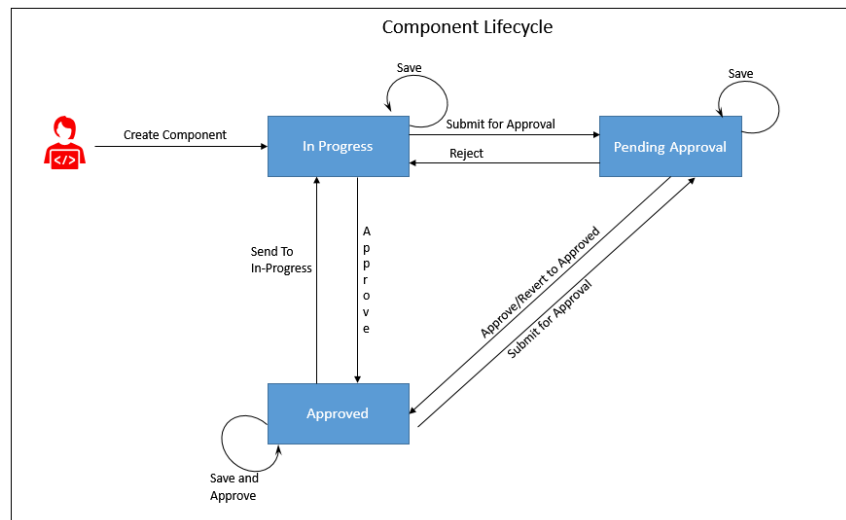
Component Structure

The following figure shows the high-level component structure.



Component 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 Components](#)
- [GUI Based Components](#)
- [REST Web Service Components](#)

Web Service Based Components

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

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

For information about web service specific keywords, refer to [Appendix A: Web Service Component Keywords](#).

GUI Based Components

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

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

For information about GUI-specific keywords, refer to [Appendix B: GUI Component Keywords](#).

REST Web Service Components

A REST web service component represents a REST interface in an Oracle Utilities application (such as CCB/C2M/WAM/Meter).

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](#)

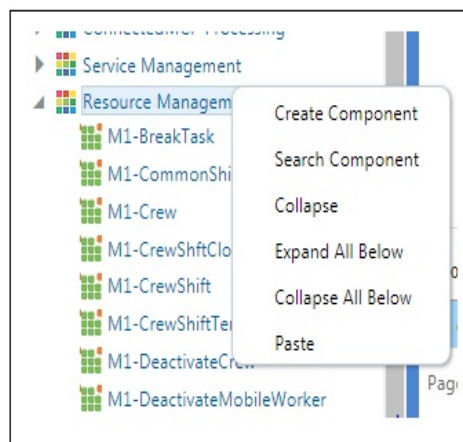
- [Using Runtime Variables in Components](#)
- [Using Function Libraries](#)
- [Resolving the Repeating Elements in Response XML](#)
- [Adding Validations](#)
- [Logging and Reporting](#)
- [Handling the List Elements](#)

Creating a Component

To create a web service based component manually:

1. Navigate to the component tree where the component has to be created.
2. Right-click the feature (release/product/module) in the component tree.

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



3. Select **Create Component**.

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

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

Note: For information about extending components, refer to the [Copying Components](#) section.

5. Select **Web Service** in the **ComponentType** drop-down list.
6. Enter a description in the **Description** field.
7. Click **Attach Code** to add the metadata. The **Component** window is displayed.
8. Create component definitions.
9. Click **Save & Unlock** to save and create the component.

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

1. Navigate to **UTA 6.0.0.0 > ORS / MWM Portfolio > ORS 2.3.0.0 > Resource Management**.
2. Right-click the **Resource Management** module.

3. Select **Create Component**.
4. Enter **CM-MobileWorker** in the **Component** field to name the component.
5. 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.

S.No	Insert	Keyword	Object	DisplayName	AttributeValues	DefaultData	OutputParameter	FunctionName	ToolT...	De
1		SETAPPTYPE	WS					Select Function		del
2		WS-SETWEBSERVICE...	Select Object	Web Service Name		ATM1Mobi		Select Function		del
3		WS-SETTRANSACTION...	Select Object	Web Transaction T		ADD		Select Function		del
4		WS-SETXMLELEMENT	Select Object	bo	bo			Select Function		del
5		WS-SETXMLELEMENT	Select Object	boStatus	boStatus			Select Function		del
6		WS-SETXMLELEMENT	Select Object	mobileWorkerType	mobileWorkerT			Select Function		del
7		WS-SETXMLELEMENT	Select Object	userId	userId			Select Function		del
8		WS-SETXMLELEMENT	Select Object	employeeId	employeeId			Select Function		del
9		WS-SETXMLELEMENT	Select Object	contractorId	contractorId			Select Function		del
10		WS-SETXMLELEMENT	Select Object	relativeEfficiency	relativeEfficienc			Select Function		del
11		WS-SETXMLELEMENT	Select Object	address1	homeAddress/a			Select Function		del
12		WS-SETXMLELEMENT	Select Object	address2	homeAddress/a			Select Function		del

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 run. This helps in better understanding of the Flow run 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).

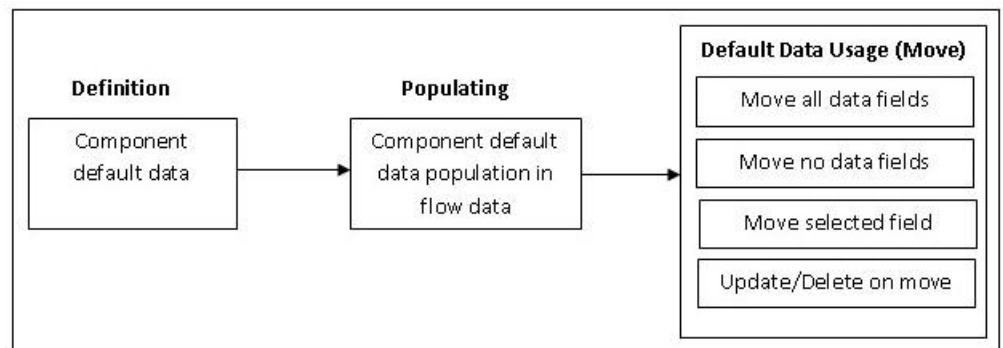
For instructions about how to handle such scenarios, refer to the [Handling the List Elements](#) section.

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.



Setting Up Operation Name for a Web Service

An operation name determines the action to be taken while running 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 run have to be passed as inputs to another component’s request XML. To achieve this, store the output of first component in the global variable by using the WS-SETVARIABLEFROMRESPONSE keyword. This keyword requires Xpath of the response element whose value 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.

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

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

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 [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).

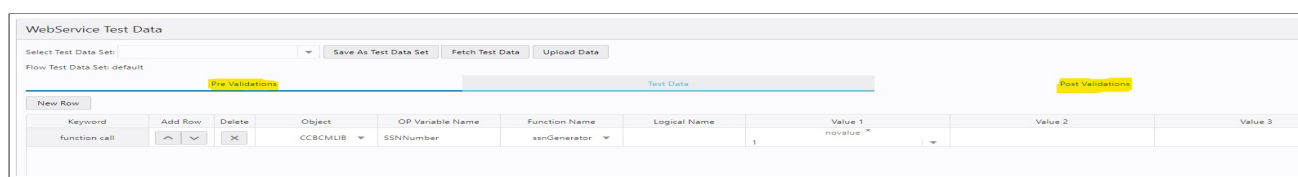
For more information about the flow-level validations, refer to the [Flow-Level Validations](#) section.

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 run log file:** The test run logs are created in the **Logs** folder and separate logs are generated for each flow.
- **Email report in HTML format:** The test run email provides brief information about the overall test run. It comprises the following:
 - Test step
 - Test data
 - Result (Pass/Fail)

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>

```

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

Note: If any of the list nodes repeat (serviceQty occurs thrice inside usagePeriods, which in turn occurs twice in usageDetails), 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 [Providing Test Data](#) section).

Providing 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.

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.

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

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

1. Expand the first usagePeriods and add three serviceQty nodes.
2. Expand the second usagePeriods and add four serviceQty nodes.

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

<input type="button" value="Add"/>	▲ usageDetails	
<input type="button" value="Add"/>	▲ usagePeriods	
<input type="button" value="Add"/>	▲ serviceQty	
	seq	<input type="text" value="1"/>
	uom	<input type="text" value="TH"/>
	tou	<input type="text" value="ON"/>
	sqj	<input type="text" value="LOSSADJ"/>
	qty	<input type="text" value="103.772922"/>
<input type="button" value="Add"/>	▲ serviceQty	
	seq	<input type="text" value="2"/>
	uom	<input type="text" value="TH"/>
	tou	<input type="text" value="SH"/>
	sqj	<input type="text" value="LOSSADJ"/>
	qty	<input type="text" value="61.976037"/>
<input type="button" value="Add"/>	▶ serviceQty	
	startDateTime	<input type="text" value="2012-12-01T02:00:00"/>
	standardStartDateTime	<input type="text" value="2012-12-01T02:00:00"/>
	endDateTime	<input type="text" value="2013-02-01T02:00:00"/>
	standardEndDateTime	<input type="text" value="2013-02-01T02:00:00"/>
	usageRequestType	<input type="text" value="C11S"/>
<input type="button" value="Add"/>	▲ usagePeriods	
<input type="button" value="Add"/>	▶ serviceQty	
<input type="button" value="Add"/>	▶ serviceQty	
<input type="button" value="Add"/>	▶ serviceQty	
<input type="button" value="Add"/>	▶ serviceQty	

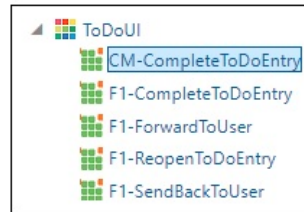
Creating GUI Based Components

To manually create a GUI based component:

1. Navigate to the component tree where the component has to be created.
2. Right-click the feature (release/product/module) in the component tree.

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

3. Select **Create Component**.
4. Enter the name of the component in the **Component** field.



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

For information about extending components, refer to the [Copying Components](#) section.

5. Select **User Interface** in the **ComponentType** drop-down list.
6. Enter a description in the **Description** field.
7. Click **Attach Code** to add the metadata. The **Component** window is displayed.
8. Create component definitions.
9. Click **Save & Unlock** to save and create the component.

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

1. Navigate to **UTA 6.0.0.0 > CORE 6.0.0.0 Portfolio > CORE UI 6.0.0.0 > ToDoUI**.
2. Right-click the **ToDoUI** module.
3. Select **Create Component**.
4. Enter CM-CompleteToDoEntry in the **Component** field.
5. Follow steps 5 to 9 as mentioned in the procedure above.

Creating a Component Definition for GUI Components

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

The following list describes each entity in a component definition:

- **Keyword:** The step to be performed.

Example: WS-SETVARIABLEFROM RESPONSE, WS-VALIDATE, etc

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

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

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

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

Note: If the attribute values contain special characters (such as '\$'), the character should be prefixed by the backslash ('\') character.

For example: To input the attribute value as id;ZONE_PRM\0\$ZONE_PARM_VAL, specify it as id;ZONE_PRM\0\\$ZONE_PARM_VAL.

- **Default Data:** The default data used in the component definition.
- **Function Name:** The function name called from the library.
- **Output Parameters:** The output in the form of a variable.

For more options, refer to [Appendix 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.

Components Content Areas

Component Name : F1-CompleteToDoEntry Component Status : Approved

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

Page 1 of 1 (1-17 of 17 items) < 1 >

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

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

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

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

1. Select SETAPPTYPE in the **Keyword** drop-down list to define the application type.
2. Select **Web** in the **Object** drop-down list to denote that it is a web services based component.
3. Select the WS-LOGMESSAGE keyword to log comments in component definition. This helps in debugging the script code for that component.
4. Add more component definition lines as needed and select appropriate keywords based on the GUI page that the component represents.
5. Click **Save**.

Creating REST Web Service Components

To create 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.

Components Content Area
 Component Name : MyRestComponent2 Component Status : In Progress

Rows to Add : 1 Add Rows Save Save & Unlock Submit For Approval Revert To Approved Delete Approve Lock

Page 1 of 1 (1-9 of 9 items) < > 1 > >

S.No	Insert	Keyword	Object	Display Name	Attribute Values	Default Data	Output Paramet...	Function Name
1	<input type="checkbox"/>	SETAPPTYPE	RS					Select Function
2	<input type="checkbox"/>	WS-LOGMESSAGE	Select Object	Log Message				Select Function
3	<input type="checkbox"/>	RS-SETREQUESTHEADER	Select Object	Header	Add HeaderNar			Select Function
4	<input type="checkbox"/>	RS-SETENDPOINT	Select Object	EndPointUrl	Add EndPoint U			Select Function
5	<input type="checkbox"/>	RS-ARGUMENT	PathVariable	Path Variable fo				Select Function
6	<input type="checkbox"/>	RS-ARGUMENT	QueryParameter	Add Query Para	Add QueryParai			Select Function
7	<input type="checkbox"/>	RS-SETMETHOD	GET	Get Method				Select Function
8	<input type="checkbox"/>	RS-PROCESSRESTREQUEST	Select Object	Process rest rec				Select Function
9	<input checked="" type="checkbox"/>	FUNCTIONCALL	wSCOMMONLIB	Send out repor				generateAndSendReport

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**.

Enable	Keyword	Object	Function Name	Caption	Logical Name	Value 1	Value 2
<input checked="" type="checkbox"/>	WS-LOGMESSAGE			Log Message		testDataSet3	
<input type="checkbox"/>	RS-SETREQUESTHEADER			Header	Content-Type		
<input type="checkbox"/>	RS-SETENDPOINT			EndPointUrl	http://google.com		
<input type="checkbox"/>	RS-ARGUMENT	PathVariable		Path Variable for url	location		
<input type="checkbox"/>	RS-ARGUMENT	QueryParameter		Add Query Params	name	testDataSet3	
<input type="checkbox"/>	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.

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

Form Data
 Raw
 Binary

Value
value
value

The screenshot shows the 'Rest Test Data' configuration window. At the top, there is a 'Select Test Data Set:' dropdown menu and a 'Save As Test Data Set' button. Below this, there are four tabs: 'Pre Validations', 'Test Data', 'Body', and 'Post Validations'. The 'Body' tab is currently selected and highlighted with a blue underline. Under the 'Body' tab, there are three radio buttons: 'Form Data', 'Raw', and 'Binary'. The 'Raw' radio button is selected. Below the radio buttons, there is a table with one row and one column, containing the number '1'.

The screenshot shows the 'Rest Test Data' configuration window. At the top, there is a 'Select Test Data Set:' dropdown menu and a 'Save As Test Data Set' button. Below this, there are four tabs: 'Pre Validations', 'Test Data', 'Body', and 'Post Validations'. The 'Body' tab is currently selected and highlighted with a blue underline. Under the 'Body' tab, there are three radio buttons: 'Form Data', 'Raw', and 'Binary'. The 'Binary' radio button is selected. Below the radio buttons, there is a 'Select a file attachment' dropdown menu with the text 'Select a file'.

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 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 run in sequence. This chapter focuses on flow sets including:

- [Creating Flow Sets](#)
- [Adding Flows to a Flow Set](#)
- [Deleting Flows from a Flow Set](#)
- [Running Flow Sets](#)
- [Stopping Flow Set Run](#)
- [Exporting Flow Sets](#)
- [Viewing Flow Set Run 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. In the test data set column's drop down, select the flow test data set used to run the flow.
7. 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 click **Delete**.
6. Click **Save** to save the flow set.

Running 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 run and click **Run**.
5. Select the flow configuration and user configuration to be used to run the flow set.

6. Click **Confirm**.

Note: For more details about on flow configuration and user configuration, refer to the [Runtime Configuration for Flow Run \(For Run Using Browser\)](#) section.

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

Stopping Flow Set Run

The Stop feature allows the active flow run to complete and stops all subsequent flows in the flow set from running.

To stop a flow set run:

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 name of the flow set whose run should be stopped.
5. Select the currently running instance of the flow set and click **Stop**.

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 Run History

To view the run 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 runs. You can view details such as the flow set run status, date and time of the run, the user who triggered the run, etc.
5. Click any of the previous runs to view flow-level details of that particular run.

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

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 runs. You can view details such as the flow set run status, date and time of the run, the user who triggered the run, etc.
5. Click any of the previous runs to view flow-level details of that particular run.
6. Click **Summary** to display a summary of the flow set run, 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 run 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](#)
- [Password Encryption Tool](#)
- [Component Definition Validation 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.

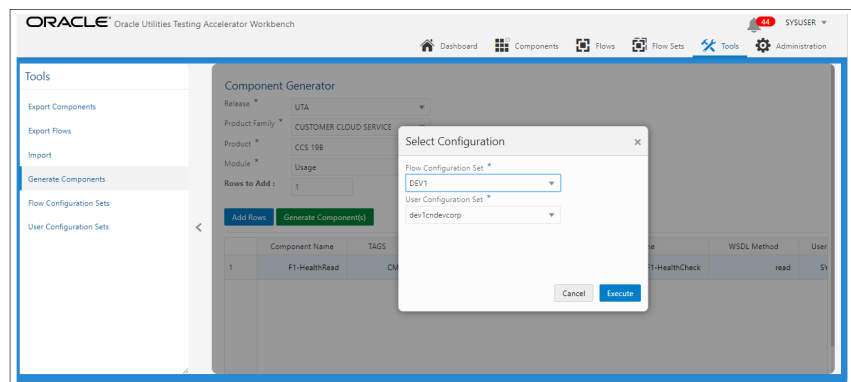
The component/flow is imported successfully.

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.

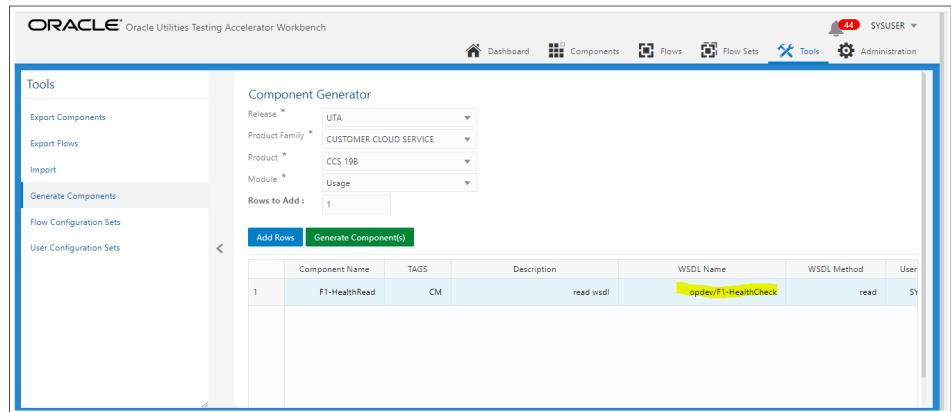


Note: When attempting to generate components from more than one application at the same time, prefix the URLs with keywords in the configuration files that can be used while specifying the WSDL to connect to.

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
yd07VCbR2jhxtkUH06Fuz+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.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<wsdl:portType name="D2-DetermineEstimatedAndHighLowScalarReadingsPortType">
  <wsdl:documentation>
    D2-DetermineEstimatedAndHighLowScalarReadings version 1: Determine Estimated, Hi-Lo Scalar Readings
  </wsdl:documentation>
  <wsdl:operation name="D2-DetermineEstimatedAndHighLowScalarReadings">
    <wsdl:input message="tns:D2-DetermineEstimatedAndHighLowScalarReadingsRequest"/>
    <wsdl:output message="tns:D2-DetermineEstimatedAndHighLowScalarReadingsResponse"/>
    <wsdl:fault name="fault" message="tns:Fault"/>
  </wsdl:operation>
</wsdl:portType>
```

Password Encryption Tool

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

This section provides the following details about the tool:

- [Overview](#)
- [Running the Password Encryption Tool](#)

Overview

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

Use the PasswordEncryptor tool to encrypt the password.

Running the Password Encryption Tool

To encrypt plain text using the Password Encryption Tool:

1. Double-click the **PasswordEncryptor.jar** file. when asked for, enter the password.
2. Click **OK**. A dialog box with the encrypted text is displayed.
3. Copy this text and paste it in the respective password field in the configuration.properties or security.properties file.

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

1. Navigate to <ECLIPSE-WORKDIR>/tools.
2. Enter the following command:

```
java -jar PasswordEncryptor.jar
```
3. When prompted, enter the password.
4. Press **Enter**.
5. Copy this text and paste it in the respective password field in the configuration.properties or security.properties file.

Component Definition Validation Tool

Use the Component Definition Validation Tool to assess the impact of the custom components while upgrading to a newer version of the product. The impact can be assessed either at a product level (report changes to all components of the given product in the newer version) or at a component level.

Note: This tool is intended to be used with only web service based components.

To generate a report at the product level:

1. Login to the application.
2. Navigate to the **Components** menu.
3. On the left pane, select and right-click the product.
4. Select **Validate Components**.
5. Enter the URL of the Utilities application against which the currently selected product components should be validated.

```
https://<server-name>:<port>/ouaf/webservices/
```

6. Click **Validate Components** to generate the report.
7. After generating the report, the tool prompts to save the HTML report to your local system.

Open the report in a browser to assess the impact.

Note: In the report, the **Newly Added** column lists all the new attributes added to the web service and the **Missing** column lists all the existing attributes that are no longer supported by the web service.

To generate a report at the component level:

1. Login to the application.
2. Navigate to the **Components** menu.
3. On the left pane, select and right-click the component to be validated.
4. Select **Validate Components**.
5. Enter the URL of the Utilities application against which the currently selected product components should be validated.

```
https://<server-name>:<port>/ouaf/webservices/
```

6. Click **Validate Components** to generate the report.

- After generating the report, the tool prompts to save the HTML report to your local system.

Open the report in a browser to assess the impact.

Component Validation Report							
Name: administrator							
Date: 2018_11_26_16.40.37							
Validated Components							
S No	Component Name	Release Name	Portfolio Name	Product Name	Module Name	Newly Added	Missing
1	Test1_1542691919528	UTA 6.0.0.0	Meter	MDM 2.2.0.1	Data Model221		1. output/exceptions/exceptionsList/sequence 2. output/exceptions/exceptionsList/messageCategory
2	Test1_1542691919528	UTA 6.0.0.0	Meter	MDM 2.2.0.1	Data Model		1. output/exceptions/exceptionsList/sequence 2. output/exceptions/exceptionsList/messageCategory

Note: In the report, the **Newly Added** column lists all the new attributes added to the web service and the **Missing** column lists all the existing attributes that are no longer supported by the web service.

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:

- [OUTSPCORELIB](#)
- [WSVALIDATELIB](#)
- [RESPONSEUTILLIB](#)
- [COREDATETIMELIB](#)
- [COREDATAGENLIB](#)
- [COREVALIDATEVARIABLELIB](#)
- [CORESTOREVALUES](#)
- [COREFILEOPS](#)
- [CORESTRINGOPS](#)
- [CORENUMBEROPS](#)
- [COREUTAOPS](#)

Note that all input parameters and output parameters of functions are of type String. Type conversions are handled inside the functions.

The input parameters for the functions need to be specified against the logical, Value1, Value2, Value3, Value4, Value5 and Value6 columns, depending on the function definitions.

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.

setVariable

Stores/sets the value provided in the test data Value1 column into a global variable specified in the OP Variable Name column, so it can be used across the flow.

Example:

```
setVariable (String valueToBeStored)
```

Input Parameters:

valueToBeStored - value to be set/stored into a variable

Return Type: String

getCurrentTimeInMilliseconds

Gets the time in milliseconds.

Example:

```
getCurrentTimeInMilliseconds ()
```

Input Parameters: <none>

Return Type: Sting

Randomstring

Generates a random string of random size and stores it into a global variable specified in the OP Variable Name column.

Example:

```
randomstring ()
```

Input Parameters: <none>

Return Type: String

compare2Strings

Compares two strings and returns a boolean result based on the result of comparison which gets stores it into a global variable specified in the OP Variable Name column.

Note: This function returns “True” if strings provided are same. Else, it returns ‘False’.

Example:

```
compare2Strings (String_stringA, String_stringB)
```

stringA - The first String to be compared

stringB - The second String to be compared to stringA

Return Type: String

randomNumberUsingDateTime

Generates a random string with date and time in it and stores it into a global variable specified in the OP Variable Name column.

Example:

```
randomNumberUsingDateTime ()
```


Input Parameters: <none>
Return Type: String

getCurrentDateTimeWithGivenDateFormat

Gets the current date and time in the specified format and stores it into a global variable specified in the OP Variable Name column.

Example:

```
getCurrentDateTimeWithGivenDateFormat(String dFormat)
dFormat- Java date formats are supported

dFormat - The format of the date output by the function
Return Type: String
```

getDateDiffInSecsWithGivenDateFormat()

Takes a start date and end data as and the corresponding data format as input parameters and calculates the difference between the dates in seconds and stores it into a global variable specified in the OP Variable Name column.

Example:

```
getDateDiffInSecsWithGivenDateFormat("12-13-2014", "12-29-2014",
"mm-dd-yyy")

Input Parameters:
dateStart - The start date for calculating the difference
dateEnd - The end date for calculating the difference
dFormat - the format of the date in which start and end dates have
been specified.

Return Type: String
```

getAdjustedTimeWithGivenDateTime

Calculates a date based on the specified date and an offset(adds or subtracts to the specified date) along with the dateformat, gets the adjusted time and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
getAdjustedTimeWithGivenDateTime(String dateTime, String offset,
String dFormat)
```

Example:

```
getAdjustedTimeWithGivenDateTime("12-13-2014", "-02:30", "mm-dd-
YYYY")

Input Parameters:
dateTime - The datetime to which the offset needs to be added.
offset - The offset of time in hh:mi format that needs to be added
to the given datetime.
dFormat - The format of the dateTime input parameter

Return Type: String
```

getAdjustedTimeWithCurrentDateTime

Calculates the date and time after adding the specified offset to the current date and time in the specified date/time format and stores it into a global variable specified in the OP Variable Name column.

Example:

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

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

getAdjustedTimeWithGivenDateAndTime

Calculates the date and time after adding the specified offset to specified date and time in the specified date/time format and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
getAdjustedTimeWithCurrentDateTime(String offset, String dFormat)
```

Example:

```
getAdjustedTimeWithCurrentDateTime("-2.30", "mm-dd-yyyy")
```

Input Parameters:

Offset - The offset to be added to the current date time, specified in hh:mi format

dFormat - The format in which the function should output the datetime

Return Type: String

getAdjustedTimeWithGivenDateAndTime

Calculates the date and time after adding the specified offset to specified date and time in the specified date/time format and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
getAdjustedTimeWithGivenDateAndTime(String cuDate,String
cuTime,String offset, String dFormat)
```

Example:

```
getAdjustedTimeWithGivenDateAndTime("12-13-2014","12:15:00","-
2.30", "mm-dd-yyyy")
```

Input Parameters:

cuDate - The date provided as input

cuTime - Time provided as input

offset - Offset to be added to the date time

dFormat - The format of the date time that the function needs to output in

Return Type: String

addDaysToCurrentDateWithGivenFormat

Calculates the date and time after adding the specified number of days to current date and time in the specified date/time format and stores it into a global variable specified in the OP Variable Name column.

Example:

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

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

waitForTime

Pauses the flow run for the time duration specified, in minutes.

Usage:

```
waitForTime(String strWaitTimeInMinutes)
```

Example:

```
waitForTime("15")
```

Input Parameters:
String strWaitTimeInMinutes - The duration to pause a flow run, specified in minutes.
Return Type: void

addDaysToAGivenDate

Adds days to the specified date and stores the output into a global variable specified in the OP Variable Name column.

Usage:

```
addDaysToAGivenDate(String date, String noOfDays)
```

Example:

```
addDaysToAGivenDate("12-13-2014", "19")
```

Input Parameters:
Date - The date to which the number of days have to be added
noOfDays - number of days to be added to the provided date

Return Type: String

randomNumber

Adds days to the specified date and stores the output into a global variable specified in the OP Variable Name column.

Example:

```
randomNumber()  
Input Parameters: <none>  
Return Type: String
```

setVariableValueUsingListIndex

Handles the retrieval of individual values from repeating elements in a comma separated list. The function retrieves the value based on the list index passed. It retrieves the value from the list which matches the specified index and stores the output into a global variable specified in the OP Variable Name column. The parameters passed are global variable (gVar1)/comma separated list and index value. This function is designed to be used in conjunction with other functions that allow for retrieving repeating elements from the response XML.

Usage:

```
setVariableValueUsingListIndex(String listVariableName, String  
index)
```

Example:

```
setVariableValueUsingListIndex("data1,data2,data3", 2) - Retrieves
data2
```

Input Parameters:

```
listVariableName - List of values separated by comma
index - Index number of the value in the list that needs to be
retrieved
```

Return Type: String: Value

appendStrings

Concatenates strings provided in the parameters and stores the output into a global variable specified in the OP Variable Name column. The default input values to this function are 6 parameters. To concatenate less than 6 strings, provide #EMPTY against the parameters which do not hold any string.

Usage:

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

Input Parameters:

```
strValue1 - The base string
strValue2 - The string to be appended to strValue1
strValue3 - The string to be appended to strValue1+strValue2
strValue4 - The string to be appended to
strValue1+strValue2+strValue3
strValue5 - The string to be appended to
strValue1+strValue2+strValue3+strValue4 strValue6 - The string to
be appended to
strValue1+strValue2+strValue3+strValue4+strValue5
```

Return Type: String

getCurrentMonth

Retrieves the current month and stores the output into a global variable specified in the OP Variable Name column.

Example:

```
getCurrentMonth()
```

Input Parameters: none

Return Type: String

readAttachmentAsString

Retrieves the content of the attachment file whose name is specified as the input parameter and stores the content into a global variable specified in the OP Variable Name column.

Example:

```
readAttachmentAsString(String Filename)
```

Input Parameters: Name of the attachment file that needs to be read

Return Type: String

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. The function automatically fails or passes a flow run subject to the satisfaction of the specified condition.

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. The xpath to be verified needs to be provided in the Logical Name column in the pre/post validations sections. If the value is null, this function validation will fail the flow run.

Example:

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

Input Parameters:

xpath - xpath of the element whose value needs to be checked.

Return Type: void

elementListNull

Verifies if all the elements in response with the specified xpath are null. The xpath to be verified needs to be provided in the Logical Name column in the pre/post validations sections. If the value is NOT null, this function validation will fail the flow run.

Example:

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

Input Parameters:

xpath- xpath of the element whose value needs to be checked.

Return Type: void

validateXPathOccurrenceCount

Verifies if the specified xpath occurs the specified number of times in the response. The xpath to be verified needs to be provided in the Logical Name column in the pre/post validations sections. The function counts the number of occurrences of the xpath and will fail the flow run, if the count in the response doesn't match the specified number. The expected number of occurrences should be specified in the Value1 column.

Usage:

```
validateXPathOccurrenceCount (String xpath, String expectedCount)
```

Example:

```
validateXPathOccurrenceCount (contact/mobileNumber, 20)
```

Input Parameters:

xpath - xpath of the element whose occurrence count needs to be checked.

expectedCount - the expected count of occurrences of the element

Return Type: void

elementNotNull

Verifies if the specified element in response is null. The xpath to be verified needs to be provided in the Logical Name column in the pre/post validations sections. This function fails the flow run if the specified element is found to be null in the response.

Usage:

```
elementNotNull(String xpath)
```

Example:

```
elementNotNull(mobileNumber)
```

Input Parameters:

xpath - xpath of the element whose value needs to be checked.

Return Type: void

elementIsNull

Verifies if the specified element in response is not null.

Usage:

```
elementIsNull (String xpath)
```

Example:

```
elementIsNull (mobileNumber)
```

Input Parameters:

Xpath - xpath of the element whose value needs to be checked.

Return Type: void

elementValueEquals

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

Usage:

```
elementValueEquals(String xpath, String expectedValue)
```

Example:

```
elementValueEquals(mobileNumber, "1234567890")
```

Input Parameters:

Xpath - xpath of the element whose value needs to be checked.

expectedValue - the expected value to be compared to for validation

Return Type: void

elementValueNotEquals

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

Usage:

```
elementValueNotEquals(String xpath, String expectedValue)
```

Example:

```
elementValueNotEquals(mobileNumber, "1234567890")
```

Input Parameters:

Xpath - xpath of the element whose value needs to be checked.

expectedValue - the expected value to be compared to for validation

Return Type: void

elementValueGreaterThan

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

Usage:

```
elementValueGreaterThan(String xpath, String valueToCompare)
```

Example:

```
elementValueGreaterThan("count", "5")
```

Input Parameters:

Xpath - xpath of the element whose value needs to be checked.
valueToCompare - the expected value to be compared to for validation

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:

Xpath - xpath of the element whose value needs to be checked.
valueToCompare - the expected value to be compared to for validation

Return Type: void

elementValueLesserThan

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

Usage:

```
elementValueLesserThan(String xpath, String valueToCompare)
```

Example:

```
elementValueLesserThan ("counter", "50")
```

Input Parameters:

Xpath - xpath of the element whose value needs to be checked.
valueToCompare - the expected value to be compared to for validation

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)
```

Usage:

```
elementValueLesserThanEqualTo ("attempts", "10")
```

Input Parameters:

Xpath - xpath of the element whose value needs to be checked.
valueToCompare - the expected value to be compared to for validation

Return Type: void

elementContains

Verifies if the specified element is available in the response.

Usage:

```
elementContains(String xpath,String valueToBeChecked)
```

Example:

```
elementContains("batchName", "F1-BILLING")
```

Input Parameters:

Xpath - xpath of the element whose value needs to be checked.
valueToBeChecked - the expected value to be compared to for validation

Return Type: void

elementNotContains

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

Usage:

```
elementNotContains(String xpath,String valueToBeChecked)
```

Example:

```
elementNotContains("batchName", "F1-BILLING")
```

Input Parameters:

Xpath - xpath of the element whose value needs to be checked.
valueToBeChecked - the expected value to be compared to for validation

Return Type: void

reponseNotContains

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

Usage:

```
reponseNotContains(String value)
```

Example:

```
reponseNotContains("Failed")
```

Input Parameters:

value - the value to be compared to for validation
Return Type: void

responseContains

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

Usage:

```
responseContains(String value)
```

Example:

```
responseContains("Exception")
```

Input Parameters:

```
value - the value to be compared to for validation
Return Type: void
```

RESPONSEUTILIB

Use the RESPONSEUTILIB function library to retrieve/extract specific values from the response XML for a component request, generated as part of the flow run. The library covers data retrieval routines for in the returned response XML.

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

setVariableFromResponseList

Takes the xpath of an element as input and outputs the value of the xpath in the response into a global variable specified against the OP Variable Name column. If there are multiple occurrences of the element, the corresponding values will be returned as a comma separated list and stored in to the variable specified against the OP Variable Name column.

Usage:

```
setVariableFromResponseList(String xpath)
setVariableFromResponseList (contact/mobileNumber)
```

Input Parameters:

```
String xpath
Return Type: String
```

setVariableFromResponseListWithFilter

This function helps to retrieve specific values from a reoccurring list in the response, based on certain condition that can be applied on other elements in the same list. It takes the xpath of an element whose value needs to be retrieved, the xpath of an element whose value needs to be compared and a filter condition for comparison operation as inputs and outputs the value/values of the xpath in the response that corresponds to the filter condition into a global variable specified against the OP Variable Name column. If there are multiple occurrences of the element satisfying the provided condition, then the corresponding values will be returned as a comma separated list and stored in to the output variable.

The first parameter holds the xpath of the element whose value has to be retrieved.

The second parameter holds the xpath of the element within the repeated list, whose value needs to be compared for a specific condition.

The third parameter holds the comparison condition.

Example:

```
setVariableFromResponseListWithFilter(String xpathToRetrieve,
String xpathToCompare, String Comparison)
```

Input Parameters:

```
String XPath, String XPath, String comparisonCondition
Return Type: String
```

Example:

Consider the following XML as the response XML for a component in a flow:

```
<env:Envelope xmlns:env=\"http://schemas.xmlsoap.org/soap/
envelope/\">
<env:Header/>
<env:Body>
<ouaf:ATF1BatchSubmission_READ xmlns:ouaf=\"http://
ouaf.oracle.com/webservices/cm/ATF1BatchSubmission/\">
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:batchControl>K1-SCLTB</ouaf:batchControl>
<ouaf:submissionMethod>FLOT</ouaf:submissionMethod>
<ouaf:batchNumber>1</ouaf:batchNumber>
<ouaf:batchRerunNumber>0</ouaf:batchRerunNumber>
<ouaf:user>SYSUSER</ouaf:user>
<ouaf:submissionUser>SYSUSER</ouaf:submissionUser>
<ouaf:language>ENG</ouaf:language>
<ouaf:batchStartDateTime>2020-07-02T22:40:31+08:00</
ouaf:batchStartDateTime>
<ouaf:threadCount>0</ouaf:threadCount>
<ouaf:batchThreadNumber>0</ouaf:batchThreadNumber>
<ouaf:maximumCommitRecords>0</
ouaf:maximumCommitRecords>
<ouaf:maximumTimeoutMinutes>0</
ouaf:maximumTimeoutMinutes>
<ouaf:isTracingProgramStart>>false</
ouaf:isTracingProgramStart>
<ouaf:isTracingProgramEnd>>false</
ouaf:isTracingProgramEnd>
<ouaf:isTracingSQL>>false</ouaf:isTracingSQL>
<ouaf:isTracingStandardOut>>false</
ouaf:isTracingStandardOut>
<ouaf:batchJobStatus>ST</ouaf:batchJobStatus>
<ouaf:version>3</ouaf:version>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:sequence>10</ouaf:sequence>
<ouaf:batchParameterName>targetSchema</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049026</ouaf:batchJobId>
<ouaf:sequence>20</ouaf:sequence>
<ouaf:batchParameterName>targetSchema2</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
```

```

ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
</ouaf:ATF1BatchSubmission_READ>
</env:Body>
</env:Envelope>

```

To retrieve all the batchJobId values, where the batchParameterValue is "K1-STAGING" the function call would be as follows:

```

setVariableFromResponseListWithFilter("batchJobExtraParameter/
batchJobId", "batchJobExtraParameter/batchParameterValue", "==K1-
STAGING")

```

The "==" operator in the 3rd input parameter to the function performs an "is equal to" comparison with the specified value.

The following are the supported operators for comparison operation:

- "==" compares to check if the value in the response is equal to the value in the condition
- "!=" compares to check if the value in the response is not equal to the value in the condition
- ">" compares to check if the value in the response is greater than the value in the condition (works only with numerical values)
- ">=" compares to check if the value in the response is greater than or equal to the value in the condition (works only with numerical values)
- "<" compares to check if the value in the response is less than the value in the condition (works only with numerical values)
- "<=" compares to check returns true if the value in the response is less than or equal to the value in the condition (works only with numerical values)
- "*" compares to check for any character
 - If the condition starts with "*", it will evaluate the condition as satisfied if the value in response ends with the value in the condition.

Example: *close will be evaluated as valid for values like enclose.

- If the condition ends with "*", it will evaluate the condition as satisfied if the value in response starts with the value in the condition.

Example: pen* will be evaluated as valid for values like pending.

- If the condition begins and ends with "*", it will evaluate the condition as satisfied if the value in response contains the value in the condition.

Example: *en* will return true for values like pending.

- If only "*" is provided without any value, the function checks for the existence of the element and will evaluate the condition as valid if the element exists irrespective of the value.
- If "!" is present, it will check for the non-existence of the element and will consider the condition as valid if the element is not present in the list.

setVariableFromResponseListWithFilters

This is an extension of the `setVariableFromResponseListWithFilter` function that helps to retrieve specific values from a reoccurring list in the response, based on a condition that can be applied on other elements in the same list, but, instead of a single condition that was allowed in the `setVariableFromResponseListWithFilter` function, this function allows users to provide two conditions.

This function takes the xpath of an element whose value needs to be retrieved, the xpath of the elements whose values need to be compared and two filter conditions corresponding to those xpath elements for comparison operation as inputs. The function outputs the value/ values of the xpath in the response that corresponds to the filter conditions into a global variable specified against the OP Variable Name column. If there are multiple occurrences of the element satisfying the provided conditions, then the corresponding values will be returned as a comma separated list and stored in to the output variable.

The first parameter holds the xpath of the element whose value has to be retrieved.

The second parameter holds the xpath of the element within the repeated list, whose value needs to be compared as the first condition.

The third parameter holds the comparison condition corresponding to the second parameter. The fourth parameter holds the xpath of the element within the repeated list, whose value needs to be compared as the second condition.

The fifth parameter holds the comparison condition corresponding to the fourth parameter (the second condition to be evaluated).

Example:

```
setVariableFromResponseListWithFilter(String xpathToRetrieve,
String xpathToCompareFirstCondition, String ComparisonForFirst
Condition, String xpathToCompareForSecondCondition, String
ComparisonForSecondCondition)
```

Input Parameters:

```
String xpath, String xpath, String comparisonCondition, String
xpath, String comparisonCondition
Return Type: String
```

See the [setVariableFromResponseListWithFilter](#) function for details on valid comparison conditions and their usage.

getValueFromListWithIndex

This is a supporting util function for the [setVariableFromResponseListWithFilter](#) and [setVariableFromResponseListWithFilters](#) functions, which takes the a comma separated list of values and returns the value in the list corresponding to the index provided as the second input to the function.

Example:

```
getValueFromListWithIndex(String commaSeperatedList, String
indexNumber)
```

Input Parameters:

```
String CommaSeperatedList , String InderNumber
Return Type: String
```

getGroupsInIntervalFromResponse

If an XML response to a component request has multiple occurrences of a group of list elements that should be retrieved and used in later components in the same flow, this function can be used to retrieve the groups of XML elements and their values.

The function retrieves a set of the XML group elements under the provided parent element's xpath. The occurrences of reoccurring groups to be retrieved for the given xpath can specified using the startIndex and endIndex parameters. The output of the function is a group of list elements in XML format that can be stored in to a global variable.

The other functions in this library can be used to extract specific values from the output variable containing the repeating list groups.

Example:

```
getGroupsInIntervalFromResponse (String xpath, String startIndex,
String endIndex)
```

Input Parameters:

```
xpath - xpath of the parent element of the list whose elements &
values need to be retrieved.
startIndex - starting occurrence number of list that needs to be
retrieved.
endIndex - Ending occurrence number of the list that needs to be
retrieved.
```

Output Parameters:

```
String - The XML string holding all the elements & values of
repeating group of elements between the start and end indexes
specified.
```

Example:

Consider the following XML as the response XML for a component in a flow:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/
envelope/">
<env:Header/>
<env:Body>
<ouaf:ATF1BatchSubmission_READ xmlns:ouaf="http://
ouaf.oracle.com/webservices/cm/ATF1BatchSubmission/">
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:batchControl>K1-SCLTB</ouaf:batchControl>
<ouaf:submissionMethod>FLOT</ouaf:submissionMethod>
<ouaf:batchNumber>1</ouaf:batchNumber>
<ouaf:batchRerunNumber>0</ouaf:batchRerunNumber>
<ouaf:user>SYSUSER</ouaf:user>
<ouaf:submissionUser>SYSUSER</ouaf:submissionUser>
<ouaf:language>ENG</ouaf:language>
<ouaf:batchStartDateTime>2020-07-02T22:40:31+08:00</
ouaf:batchStartDateTime>
<ouaf:threadCount>0</ouaf:threadCount>
<ouaf:batchThreadNumber>0</ouaf:batchThreadNumber>
<ouaf:maximumCommitRecords>0</
ouaf:maximumCommitRecords>
```

```

<ouaf:maximumTimeoutMinutes>0</
ouaf:maximumTimeoutMinutes>
<ouaf:isTracingProgramStart>>false</
ouaf:isTracingProgramStart>
<ouaf:isTracingProgramEnd>>false</
ouaf:isTracingProgramEnd>
<ouaf:isTracingSQL>>false</ouaf:isTracingSQL>
<ouaf:isTracingStandardOut>>false</
ouaf:isTracingStandardOut>
<ouaf:batchJobStatus>ST</ouaf:batchJobStatus>
<ouaf:version>3</ouaf:version>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:sequence>10</ouaf:sequence>
<ouaf:batchParameterName>targetSchema</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049026</ouaf:batchJobId>
<ouaf:sequence>20</ouaf:sequence>
<ouaf:batchParameterName>targetSchema2</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
</ouaf:ATF1BatchSubmission_READ>
</env:Body>
</env:Envelope>

```

To retrieve the group of elements under the `batchJobExtraParameter` for the second and third occurrence of the group, call the function using the following input parameters:

```

xpath: batchJobExtraParameter startIndex: 2
endIndex: 3

```

Output XML String:

```

<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049026</ouaf:batchJobId>

```

```

<ouaf:sequence>20</ouaf:sequence>
<ouaf:batchParameterName>targetSchema2</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>

```

getFirstGroupFromResponse

If an XML response to a component request has multiple occurrences of a group of list elements to be retrieved and used in later components in the same flow, this function can be used to retrieve the groups of XML elements and their values.

The function retrieves a first set of the XML group elements under the provided parent element's xpath. The output of the function is a group of list elements in XML format that can be stored in to a global variable.

Use the other functions in this library to extract specific values from the output variable containing the XML string.

Example:

```
getFirstGroupFromResponse (String xpath)
```

Input Parameters:

xpath - xpath of the parent element of the list whose first occurrence of elements & values need to be retrieved.

Output Parameters:

String - The XML string holding all the elements & values of the first occurrence of the repeating group of elements.

Example:

Consider the following XML as the response XML for a component in a flow:

```

<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/
envelope/">
<env:Header/>
<env:Body>
<ouaf:ATF1BatchSubmission_READ xmlns:ouaf="http://
ouaf.oracle.com/webservices/cm/ATF1BatchSubmission">
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:batchControl>K1-SCLTB</ouaf:batchControl>
<ouaf:submissionMethod>FLOT</ouaf:submissionMethod>
<ouaf:batchNumber>1</ouaf:batchNumber>
<ouaf:batchRerunNumber>0</ouaf:batchRerunNumber>
<ouaf:user>SYSUSER</ouaf:user>
<ouaf:submissionUser>SYSUSER</ouaf:submissionUser>
<ouaf:language>ENG</ouaf:language>
<ouaf:batchStartDateTime>2020-07-02T22:40:31+08:00</

```

```

ouaf:batchStartDateTime>
<ouaf:threadCount>0</ouaf:threadCount>
<ouaf:batchThreadNumber>0</ouaf:batchThreadNumber>
<ouaf:maximumCommitRecords>0</
ouaf:maximumCommitRecords>
<ouaf:maximumTimeoutMinutes>0</
ouaf:maximumTimeoutMinutes>
<ouaf:isTracingProgramStart>>false</
ouaf:isTracingProgramStart>
<ouaf:isTracingProgramEnd>>false</
ouaf:isTracingProgramEnd>
<ouaf:isTracingSQL>>false</ouaf:isTracingSQL>
<ouaf:isTracingStandardOut>>false</
ouaf:isTracingStandardOut>
<ouaf:batchJobStatus>ST</ouaf:batchJobStatus>
<ouaf:version>3</ouaf:version>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:sequence>10</ouaf:sequence>
<ouaf:batchParameterName>targetSchema</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049026</ouaf:batchJobId>
<ouaf:sequence>20</ouaf:sequence>
<ouaf:batchParameterName>targetSchema2</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
</ouaf:ATF1BatchSubmission_READ>
</env:Body>
</env:Envelope>

```

To retrieve the first group of elements under the batchJobExtraParameter, call the function using the following input parameters:

```
xpath: batchJobExtraParameter
```


Output XML String:

```
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:sequence>10</ouaf:sequence>
<ouaf:batchParameterName>targetSchema</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
```

getFirstNGroupsFromResponse

If an XML response to a component request has multiple occurrences of a group of list elements to be retrieved and used in later components in the same flow, use this function to retrieve the groups of XML elements and their values.

The function retrieves a set of the XML group elements under the provided parent element's xpath. The occurrences of reoccurring groups to be retrieved for the given xpath can be specified using the numberOfOccurrences parameter. The output of the function is a group of list elements in XML format that can be stored in to a global variable.

Use the other functions in this library to extract specific values from the output variable containing the repeating list groups.

Example:

```
getFirstNGroupsFromResponse (String xpath, String
numberOfOccurrences)
```

Input Parameters:

xpath - xpath of the parent element of the list whose elements and values need to be retrieved.
 NumberOfOccurrences - the occurrences of the group of list elements starting from 1, whose elements and values have to be retrieved.

Output Parameters:

String - The XML string holding all the elements and values of repeating group of elements for the specified number of occurrences.

Example:

Consider the following XML as the response XML for a component in a flow:

```
<env:Envelope xmlns:env=\"http://schemas.xmlsoap.org/soap/
envelope/\">>
<env:Header/>
<env:Body>
<ouaf:ATF1BatchSubmission_READ xmlns:ouaf=\"http://
ouaf.oracle.com/webservices/cm/ATF1BatchSubmission\">
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:batchControl>K1-SCLTB</ouaf:batchControl>
<ouaf:submissionMethod>FLOT</ouaf:submissionMethod>
<ouaf:batchNumber>1</ouaf:batchNumber>
<ouaf:batchRerunNumber>0</ouaf:batchRerunNumber>
<ouaf:user>SYSUSER</ouaf:user>
<ouaf:submissionUser>SYSUSER</ouaf:submissionUser>
<ouaf:language>ENG</ouaf:language>
<ouaf:batchStartDateTime>2020-07-02T22:40:31+08:00</
```

```

ouaf:batchStartDateTime>
<ouaf:threadCount>0</ouaf:threadCount>
<ouaf:batchThreadNumber>0</ouaf:batchThreadNumber>
<ouaf:maximumCommitRecords>0</
ouaf:maximumCommitRecords>
<ouaf:maximumTimeoutMinutes>0</
ouaf:maximumTimeoutMinutes>
<ouaf:isTracingProgramStart>>false</
ouaf:isTracingProgramStart>
<ouaf:isTracingProgramEnd>>false</
ouaf:isTracingProgramEnd>
<ouaf:isTracingSQL>>false</ouaf:isTracingSQL>
<ouaf:isTracingStandardOut>>false</
ouaf:isTracingStandardOut>
<ouaf:batchJobStatus>ST</ouaf:batchJobStatus>
<ouaf:version>3</ouaf:version>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:sequence>10</ouaf:sequence>
<ouaf:batchParameterName>targetSchema</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049026</ouaf:batchJobId>
<ouaf:sequence>20</ouaf:sequence>
<ouaf:batchParameterName>targetSchema2</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
</ouaf:ATF1BatchSubmission_READ>
</env:Body>
</env:Envelope>

```

To retrieve the group of elements under the batchJobExtraParameter for first 2 occurrences of the group, call the function using the following input parameters:

```
xpath: batchJobExtraParameter numberOfOccurrences: 2
```

Output XML String:

```
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:sequence>10</ouaf:sequence>
<ouaf:batchParameterName>targetSchema</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049026</ouaf:batchJobId>
<ouaf:sequence>20</ouaf:sequence>
<ouaf:batchParameterName>targetSchema2</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
```

getAllGroupsFromResponse

If an XML response to a component request has multiple occurrences of a group of list elements to be retrieved and used in later components in the same flow, use this function to retrieve the groups of XML elements and their values.

The function retrieves all sets (occurrences) of the XML group elements under the provided parent element's xpath. The output of the function is a group of list elements in XML format that can be stored in to a global variable.

Use the other functions in this library to extract specific values from the output variable containing the repeating list groups.

Example:

```
getAllGroupsFromResponse (String xpath)
```

Input Parameters:

```
xpath - xpath of the parent element of the list whose elements and
values need to be retrieved
```

Output Parameters:

```
String - The XML string holding all the elements and values of
repeating group of under the parent element
```

Example:

Consider the following XML as the response XML for a component in a flow:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/
envelope/">
<env:Header/>
<env:Body>
<ouaf:ATF1BatchSubmission_READ xmlns:ouaf="http://
ouaf.oracle.com/webservices/cm/ATF1BatchSubmission">
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:batchControl>K1-SCLTB</ouaf:batchControl>
<ouaf:submissionMethod>FLOT</ouaf:submissionMethod>
<ouaf:batchNumber>1</ouaf:batchNumber>
<ouaf:batchRerunNumber>0</ouaf:batchRerunNumber>
```

```

<ouaf:user>SYSUSER</ouaf:user>
<ouaf:submissionUser>SYSUSER</ouaf:submissionUser>
<ouaf:language>ENG</ouaf:language>
<ouaf:batchStartDateTime>2020-07-02T22:40:31+08:00</
ouaf:batchStartDateTime>
<ouaf:threadCount>0</ouaf:threadCount>
<ouaf:batchThreadNumber>0</ouaf:batchThreadNumber>
<ouaf:maximumCommitRecords>0</
ouaf:maximumCommitRecords>
<ouaf:maximumTimeoutMinutes>0</
ouaf:maximumTimeoutMinutes>
<ouaf:isTracingProgramStart>>false</
ouaf:isTracingProgramStart>
<ouaf:isTracingProgramEnd>>false</
ouaf:isTracingProgramEnd>
<ouaf:isTracingSQL>>false</ouaf:isTracingSQL>
<ouaf:isTracingStandardOut>>false</
ouaf:isTracingStandardOut>
<ouaf:batchJobStatus>ST</ouaf:batchJobStatus>
<ouaf:version>3</ouaf:version>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:sequence>10</ouaf:sequence>
<ouaf:batchParameterName>targetSchema</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049026</ouaf:batchJobId>
<ouaf:sequence>20</ouaf:sequence>
<ouaf:batchParameterName>targetSchema2</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
</ouaf:ATF1BatchSubmission_READ>
</env:Body>
</env:Envelope>

```

To retrieve the group of elements under the `batchJobExtraParameter` for all the occurrences of the group, call the function using the following input parameters:

```
xpath: batchJobExtraParameter
```

Output XML String:

```
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:sequence>10</ouaf:sequence>
<ouaf:batchParameterName>targetSchema</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049026</ouaf:batchJobId>
<ouaf:sequence>20</ouaf:sequence>
<ouaf:batchParameterName>targetSchema2</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
```

getLastGroupFromResponse

If an XML response to a component request has multiple occurrences of a group of list elements to be retrieved and used in later components in the same flow, use this function to retrieve the groups of XML elements and their values.

The function retrieves the last set (occurrence) of the XML group elements under the provided parent element's xpath. The output of the function is a group of list elements in XML format that can be stored in to a global variable.

Use the other functions in this library to extract specific values from the output variable containing the repeating list groups.

Example:

```
getLastGroupFromResponse (String xpath)
```

Input Parameters:

xpath - xpath of the parent element of the list whose elements and values need to be retrieved.

Output Parameters:

String - The XML string holding the elements and values of the last occurrence of the repeating group of elements under the parent element.

Example:

Consider the following XML as the response XML for a component in a flow:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <ouaf:ATF1BatchSubmission_READ xmlns:ouaf="http://ouaf.oracle.com/webservices/cm/ATF1BatchSubmission">
      <ouaf:batchJobId>82503583049025</ouaf:batchJobId>
      <ouaf:batchControl>K1-SCLTB</ouaf:batchControl>
      <ouaf:submissionMethod>F1OT</ouaf:submissionMethod>
      <ouaf:batchNumber>1</ouaf:batchNumber>
      <ouaf:batchRerunNumber>0</ouaf:batchRerunNumber>
      <ouaf:user>SYSUSER</ouaf:user>
      <ouaf:submissionUser>SYSUSER</ouaf:submissionUser>
      <ouaf:language>ENG</ouaf:language>
      <ouaf:batchStartDateTime>2020-07-02T22:40:31+08:00</ouaf:batchStartDateTime>
      <ouaf:threadCount>0</ouaf:threadCount>
      <ouaf:batchThreadNumber>0</ouaf:batchThreadNumber>
      <ouaf:maximumCommitRecords>0</ouaf:maximumCommitRecords>
      <ouaf:maximumTimeoutMinutes>0</ouaf:maximumTimeoutMinutes>
      <ouaf:isTracingProgramStart>false</ouaf:isTracingProgramStart>
      <ouaf:isTracingProgramEnd>false</ouaf:isTracingProgramEnd>
      <ouaf:isTracingSQL>false</ouaf:isTracingSQL>
      <ouaf:isTracingStandardOut>false</ouaf:isTracingStandardOut>
      <ouaf:batchJobStatus>ST</ouaf:batchJobStatus>
      <ouaf:version>3</ouaf:version>
      <ouaf:batchJobExtraParameter>
        <ouaf:batchJobId>82503583049025</ouaf:batchJobId>
        <ouaf:sequence>10</ouaf:sequence>
        <ouaf:batchParameterName>targetSchema</ouaf:batchParameterName>
        <ouaf:batchParameterValue>K1-STAGING</ouaf:batchParameterValue>
        <ouaf:version>1</ouaf:version>
      </ouaf:batchJobExtraParameter>
      <ouaf:batchJobExtraParameter>
        <ouaf:batchJobId>82503583049026</ouaf:batchJobId>
        <ouaf:sequence>20</ouaf:sequence>
        <ouaf:batchParameterName>targetSchema2</ouaf:batchParameterName>
        <ouaf:batchParameterValue>K1-STAGING</ouaf:batchParameterValue>
      </ouaf:batchJobExtraParameter>
    </ouaf:ATF1BatchSubmission_READ>
  </env:Body>
</env:Envelope>
```

```

<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
</ouaf:ATF1BatchSubmission_READ>
</env:Body>
</env:Envelope>

```

To retrieve the group of elements under the `batchJobExtraParameter` for the last occurrence of the group, use this function to call the following input parameters:

```
xpath: batchJobExtraParameter
```

Output XML String:

```

<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>

```

getLastNGroupsFromResponse

If an XML response to a component request has multiple occurrences of a group of list elements to be retrieved and used in later components in the same flow, use this function to retrieve the groups of XML elements and their values.

The function retrieves a set of the XML group elements under the provided parent element's `xpath`. The occurrences (starting from the end of reoccurrence group) of reoccurring groups to be retrieved for the given `xpath` can be specified using the `numberOfLastNOccurrences` parameter. The output of the function is a group of list elements in XML format that can be stored in to a global variable.

The other functions in this library can be used to extract specific values from the output variable containing the repeating list groups.

Example:

```
getLastNGroupsFromResponse (String xpath, String
numberOfLastNOccurrences)
```

Input Parameters:

xpath - xpath of the parent element of the list whose elements and values need to be retrieved.

NumberOfLastNOccurrences - occurrences of the group of list elements starting from the end/total, whose elements and values have to be retrieved.

Output Parameters:

String - The XML string holding all the elements and values of repeating group of elements for the specified number of occurrences from last.

Example:

Consider the following XML as the response XML for a component in a flow:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header/>
  <env:Body>
    <ouaf:ATF1BatchSubmission_READ xmlns:ouaf="http://ouaf.oracle.com/webservices/cm/ATF1BatchSubmission/">
      <ouaf:batchJobId>82503583049025</ouaf:batchJobId>
      <ouaf:batchControl>K1-SCLTB</ouaf:batchControl>
      <ouaf:submissionMethod>FLOT</ouaf:submissionMethod>
      <ouaf:batchNumber>1</ouaf:batchNumber>
      <ouaf:batchRerunNumber>0</ouaf:batchRerunNumber>
      <ouaf:user>SYSUSER</ouaf:user>
      <ouaf:submissionUser>SYSUSER</ouaf:submissionUser>
      <ouaf:language>ENG</ouaf:language>
      <ouaf:batchStartDateTime>2020-07-02T22:40:31+08:00</ouaf:batchStartDateTime>
      <ouaf:threadCount>0</ouaf:threadCount>
      <ouaf:batchThreadNumber>0</ouaf:batchThreadNumber>
      <ouaf:maximumCommitRecords>0</ouaf:maximumCommitRecords>
      <ouaf:maximumTimeoutMinutes>0</ouaf:maximumTimeoutMinutes>
      <ouaf:isTracingProgramStart>false</ouaf:isTracingProgramStart>
      <ouaf:isTracingProgramEnd>false</ouaf:isTracingProgramEnd>
      <ouaf:isTracingSQL>false</ouaf:isTracingSQL>
      <ouaf:isTracingStandardOut>false</ouaf:isTracingStandardOut>
      <ouaf:batchJobStatus>ST</ouaf:batchJobStatus>
      <ouaf:version>3</ouaf:version>
      <ouaf:batchJobExtraParameter>
        <ouaf:batchJobId>82503583049025</ouaf:batchJobId>
        <ouaf:sequence>10</ouaf:sequence>
        <ouaf:batchParameterName>targetSchema</ouaf:batchParameterName>
        <ouaf:batchParameterValue>K1-STAGING</ouaf:batchParameterValue>
        <ouaf:version>1</ouaf:version>
      </ouaf:batchJobExtraParameter>
      <ouaf:batchJobExtraParameter>
        <ouaf:batchJobId>82503583049026</ouaf:batchJobId>
        <ouaf:sequence>20</ouaf:sequence>
        <ouaf:batchParameterName>targetSchema2</ouaf:batchParameterName>
      </ouaf:batchJobExtraParameter>
    </ouaf:ATF1BatchSubmission_READ>
  </env:Body>
</env:Envelope>
```



```

ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
</ouaf:ATF1BatchSubmission_READ>
</env:Body>
</env:Envelope>

```

To retrieve the group of elements under the `batchJobExtraParameter` for last 2 occurrences of the group, call the function using the following input parameters:

```
xpath: batchJobExtraParameter numberOfOccurrences: 2
```

Output XML String:

```

<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>

```

getGroupsFromResponseWithFilter

If an XML response to a component request has multiple occurrences of a group of list elements to be retrieved based on some conditions to be applied on the elements in the group, to be used in later components in the same flow, use this function to retrieve the groups of XML elements and their values.

The function retrieves a set of the XML group elements under the provided parent element's `xpath` subject to a specified condition. The condition based on which the reoccurring groups are

to be retrieved for the given xpath can be specified using the corresponding xpath and the conditional parameters. The output of the function is a group of list elements in XML format that can be stored in to a global variable.

The other functions in this library can be used to extract specific values from the output variable containing the repeating list groups.

Example:

```
getGroupsFromResponseWithFilter (String xpath, String
elementXPathForCondition, String condition)
```

Input Parameters:

```
xpath - xpath of the parent element of the list whose elements and
values need to be retrieved.
elementXPathForCondition - xpath of the element within the list
whose value needs to be validated against the specified condition,
for match.
condition - The condition to be applied on the element specified
against elementXPathForCondition parameter. For the supported list
of conditions, refer to the list of conditional statements
specified under the function "setVariableFromReponseListWithFilter"
```

Output Parameters:

```
String - The XML string holding the elements and values of
repeating group of elements between whose element matches the
conditional statement specified.
```

Example:

Consider the following XML as the response XML for a component in a flow:

```
<env:Envelope xmlns:env=\"http://schemas.xmlsoap.org/soap/
envelope/\">
<env:Header/>
<env:Body>
<ouaf:ATF1BatchSubmission_READ xmlns:ouaf=\"http://
ouaf.oracle.com/webservices/cm/ATF1BatchSubmission\">
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:batchControl>K1-SCLTB</ouaf:batchControl>
<ouaf:submissionMethod>FLOT</ouaf:submissionMethod>
<ouaf:batchNumber>1</ouaf:batchNumber>
<ouaf:batchRerunNumber>0</ouaf:batchRerunNumber>
<ouaf:user>SYSUSER</ouaf:user>
<ouaf:submissionUser>SYSUSER</ouaf:submissionUser>
<ouaf:language>ENG</ouaf:language>
<ouaf:batchStartDateTime>2020-07-02T22:40:31+08:00</
ouaf:batchStartDateTime>
<ouaf:threadCount>0</ouaf:threadCount>
<ouaf:batchThreadNumber>0</ouaf:batchThreadNumber>
<ouaf:maximumCommitRecords>0</
ouaf:maximumCommitRecords>
<ouaf:maximumTimeoutMinutes>0</
ouaf:maximumTimeoutMinutes>
<ouaf:isTracingProgramStart>>false</
ouaf:isTracingProgramStart>
<ouaf:isTracingProgramEnd>>false</
ouaf:isTracingProgramEnd>
<ouaf:isTracingSQL>>false</ouaf:isTracingSQL>
<ouaf:isTracingStandardOut>>false</
```

```

ouaf:isTracingStandardOut>
<ouaf:batchJobStatus>ST</ouaf:batchJobStatus>
<ouaf:version>3</ouaf:version>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049025</ouaf:batchJobId>
<ouaf:sequence>10</ouaf:sequence>
<ouaf:batchParameterName>targetSchema</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049026</ouaf:batchJobId>
<ouaf:sequence>20</ouaf:sequence>
<ouaf:batchParameterName>targetSchema2</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1-STAGING</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
</ouaf:ATF1BatchSubmission_READ>
</env:Body>
</env:Envelope>

```

To retrieve the group of elements under the batchJobExtraParameter parent element, whose sequence is greater than 20, call the function using the following input parameters:

```

xpath: batchJobExtraParameter elementXPathForCondition: sequence
condition: >20

```

Output XML String:

```

<ouaf:batchJobExtraParameter>
<ouaf:batchJobId>82503583049027</ouaf:batchJobId>
<ouaf:sequence>30</ouaf:sequence>
<ouaf:batchParameterName>table</
ouaf:batchParameterName>
<ouaf:batchParameterValue>CI_NT_UP</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>
<ouaf:batchJobExtraParameter>

```

```

<ouaf:batchJobId>82503583049028</ouaf:batchJobId>
<ouaf:sequence>40</ouaf:sequence>
<ouaf:batchParameterName>action</
ouaf:batchParameterName>
<ouaf:batchParameterValue>K1BT</
ouaf:batchParameterValue>
<ouaf:version>1</ouaf:version>
</ouaf:batchJobExtraParameter>

```

getResponseGroupSize

If an XML response to a component request has multiple occurrences of a group of list elements whose group count needs to be determined, use this function to retrieve the number of occurrences of the groups of XML elements. The output of the function is the number of occurrences of the group that can be stored in to a global variable.

Example:

```
getResponseGroupSize (String xpath)
```

Input Parameters:

xpath - xpath of the parent element of the group whose occurrence count needs to be determined.

Output Parameters:

String - count of the number of occurrences of the specified group

getGroupsInIntervalFromGroupVariable

This function is same as the [getGroupsInIntervalFromResponse](#) function except that it is applied on variables instead of the response XML of the component in a flow. The variable that is used as input to this function needs to hold the XML as String. It retrieves a set of the XML group elements under the provided parent element's xpath. The occurrences of reoccurring groups to be retrieved for the given xpath can specified using the startIndex and endIndex parameters. The output of the function is a group of list elements in XML format that can be stored in to a global variable.

The other functions in this library can be used to extract specific values from the output variable containing the repeating list groups.

Note: This function can be applied to retrieve a subgroup after a group has been retrieved from the response using other functions.

```
getGroupsInIntervalFromGroupVariable (String groupVariable, String
xpath, String startIndex, String endIndex)
```

Input Parameters:

groupVariable - variable holding the group elements in the form of an XML

xpath - xpath of the parent element of the list whose elements and values need to be retrieved.

startIndex - starting occurrence number of list that needs to be retrieved

endIndex - Ending occurrence number of the list that needs to be retrieved

Output Parameters:

String - The XML string holding all the elements & values of repeating group of elements between the start and end indexes specified.

Refer to [getGroupsInIntervalFromResponse](#) function for a detailed example.

getGroupsFromGroupVariableWithFilter

This function is same as the [getGroupsFromResponseWithFilter](#) function, except that this function is applied on variables instead of the response XML of the component in a flow. The variable that is used as input to this function needs to hold the XML as String. It retrieves a set of the XML group elements under the provided parent element's xpath subject to a specified condition. The condition based on which the reoccurring groups are to be retrieved for the given xpath can specified using the corresponding xpath and the conditional parameters.

The output of the function is a group of list elements in XML format that can be stored in to a global variable.

The other functions in this library can be used to extract specific values from the output variable containing the repeating list groups.

Note: This function can be applied to retrieve a subgroup after a group has been retrieved from the response using other functions.

```
getGroupsFromGroupVariableWithFilter (String groupVariable, String
xpath, String conditionElementXPath, String condition)
```

Input Parameters:

groupVariable - variable holding the XML as a string
 xpath - xpath of the parent element of the list whose elements & values need to be retrieved.
 conditionElementXPath - xpath of the element within the list whose value needs to be validated against the specified condition, for match.
 condition - The condition to be applied on the element specified against elementXPathForCondition parameter. For the supported list of conditions, refer to the list of conditional statements specified under the function "setVariableFromReponseListWithFilter"

Output Parameters:

String - The XML string holding the elements and values of repeating group of elements between whose element matches the conditional statement specified.

Refer to [getGroupsFromResponseWithFilter](#) function for a detailed example.

getListFromGroupVariable

This function retrieves the value/values against the specified xpath, within an XML string stored in a variable. It is provided as a support function to the above functions that help retrieve groups from a response/variable.

If the xpath specified as input parameter has multiple occurrences in the XML in the group variable, then the values are returned as a comma separated list. The function outputs the value or list of values under the specified xpath.

The [getValueFromListWithIndex](#) function can be used to pick up individual values from the list.

```
getListFromGroupVariable (String groupVariable, String xpath)
```

Input Parameters:

groupVariable - The variable holding the XML String from which value needs to be retrieved.
 xpath - xpath of the element whose value needs to be retrieved.

Output Parameters:

String - The value of the specified element or the comma separated values of the specified element, if the element is repeating in the XML.

COREDATETIMELIB

Use the COREDATETIMELIB function library to calculate date time operations to be used as test data inputs in a component of a flow. The library also has date time conversion functions to format the date time.

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

getCurrentDatettimeWithGivenDateFormat

Gets the current date and time in the specified format and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
getCurrentDateTImeWithGivenDateFormat(String dFormat) dFormat- Java
date formats are supported.
getCurrentDateTImeWithGivenDateFormat("mm-dd-yyyy:hh.mm.ss")
```

Input Parameters:

Date Format String
 Return Type: String

getCurrentTZDatettimeWithGivenDateFormat

Gets the current date and time in the specified time zone and format and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
getCurrentTZDateTImeWithGivenDateFormat(String dFormat, String
timeZone)

dFormat- Java date formats are supported

getCurrentDateTImeWithGivenDateFormat("mm-dd-yyyy:hh.mm.ss")
```

getFormattedDateWithGivenDate

Converts the date time input provided in to the specified format and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
getFormattedDateWithGivenDate(String sourceDatettime, String
sourceDateFromat, String outputDateFormat)

dFormat- Java date formats are supported

getFormattedDateWithGivenDate ("02/01/2020:23:00:00", "dd/mm/
YYYY:HH24:mi:ss", "mm-dd-yyyy:hh.mi.ss")
```

getDateDiffInSecsWithGivenDateFormat

Takes a start date and end data and the corresponding data format as input parameters and calculates the difference between the dates in seconds and stores it into a global variable specified in the OP Variable Name column.

Example:

```
getDateDiffInSecsWithGivenDateFormat (String dateStart, String
dateEnd, String dFormat)

getDateDiffInSecsWithGivenDateFormat ("12-13-2014", "12-29-2014",
"mm-dd-yyy")
```

Input Parameters:

```
String dateStart, String dateEnd, String dFormat
Return Type: String
```

getAdjustedTimeWithGivenDateTime

Calculates a date based on the specified date and an offset (adds or subtracts to the specified date) along with the dateformat, gets the adjusted time and stores it into a global variable specified in the OP Variable Name column.

Note: Supports the offset in hours:minutes:seconds = hh:mm:ss.

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

Calculates the date and time after adding the specified offset to the current date and time in the specified date/time format and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
getAdjustedTimeWithCurrentDateTime (String offset, String dFormat)

getAdjustedTimeWithCurrentDateTime ("-2.30", "mm-dd-yyyy")
```

Input Parameters:

```
String offset, String dFormat
Return Type: String
```

addDaysToCurrentDateWithGivenFormat

Calculates the date and time after adding the specified number of days to current date and time in the specified date/time format and stores it into a global variable specified in the OP Variable Name column.

Usage:

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

Input Parameters:

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

addDaysToCurrentTZDateWithGivenFormat

Calculates the date and time after adding the specified number of days to current date time of the specified time zone in the specified date/time format and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
addDaysToCurrentTZDateWithGivenFormat(String noOfDays, String
dFormat,String timeZone)

addDaysToCurrentDateWithGivenFormat("45", "mm-dd-yyyy", "PST")
```

Input Parameters:

```
String noOfDays, String dFormat, String timezone
Return Type: String
```

getCurrentTimeinMilliseconds

Gets the current date time in milliseconds and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
getCurrentDateTimeinMilliseconds()
```

Input Parameters: None

```
Return Type: String
```

getEpochInGivenDateTimeFormat

Supports the conversion of the date time value from epoch format, which is the format used for incrementer in the subroutine looping interface, when user selects the incrementer type as date time.

This function takes two parameters, the first is the name of the variable that has the timestamp in epoch (example: incrementer in sub-routine loop), and the second is a valid date-time output format. It returns a string with the date-time in the format specified and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
getEpochInGivenDateTimeFormat(String epochDateTime, String
outputDateFormat)
```

Input Parameters:

```
epochDateTime - The epoch that needs to be converted
outputDateFormat - The format of the date to be output by the
function.
Return Type: String
```


COREDATAGENLIB

This library contains functions that help with the generation of random numbers and strings which can be used as test data inputs for a flow.

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

randomStringWithGivenRange

Generates a random string within the specified number of characters, either in upper case or lower case based on the input parameters and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
randomStringWithGivenRange(String minCharacters, String  
maxCharacters, String isUpperCase)
```

Input Parameters:

minCharacters- The minimum number of characters(min size) in the random string. Only numbers are allowed.
maxCharacters- The maximum number of characters(max size) in the random string. Only numbers are allowed.
isUpperCase - Set to "true" if the random string needs to be in UpperCase, else set to "false". Only Boolean values are allowed.

randomString

Generates a random string of random length and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
randomString ()
```

Input Parameters: None

ReturnType - String

randomNumber

Generates a random number of specified size and stores it into a global variable specified in the OP Variable Name column.

Usage:

```
randomNumber(String length)
```

Input Parameters:

length - The length of the random number to be generated. Only numbers are allowed.

COREVALIDATEVARIABLELIB

Use this function library to validate the values/elements stored in variables (referred to as verification points) in a flow. The library covers validation routines for string and XML elements in the variables. It includes all the functions of the [WSVALIDATELIB](#), except that the functions in this library work on the values/elements stored in variables instead of a response to component request in flow run.

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 the XML string stored in a variable are not null. The xpath to be verified needs to be provided in the value column in the pre/post validations sections. If the value is null, this function validation will fail the flow run.

Usage:

```
elementListNotNull(String variableName, String xpath)
elementNotNull(gVarVariable,contact/mobileNumber)
```

Input Parameters:

variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.
 xpath - xpath of the element to be validated. To be provided in the Value column of test data. Return Type: void

elementListNull

Verifies if all the elements with the specified xpath are null in the XML string stored in a variable. The xpath to be verified needs to be provided in the value column in the pre/post validations sections. If the value is NOT null, this function validation will fail the flow run.

Usage:

```
elementListNull(String variableName, String xpath)
elementNotNull(gVarVariable,contact/mobileNumber)
```

Input Parameters:

variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value1 column.
 xpath - xpath of the element to be validated. To be provided in the Value column of test data.
 Return Type: void

validateXPathOccurrenceCount

Verifies if the specified xpath occurs the specified number of times in the XML string stored in a variable. The xpath to be verified needs to be provided in the value column in the pre/post validations sections. The function counts the number of occurrences of the xpath and will fail the flow run, if the count in the response doesn't match the specified number.

Usage:

```
validateXPathOccurenceCount (String xpath,String VariableName,
String expectedCount) validateXPathOccurenceCount(contact/
mobileNumber,20)
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the Value column of test data.
 variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.
 expectedOccurrenceCount - The expected number of occurrences needs to be specified in the Value column.

Return Type: void

elementNotNull

Verifies if the specified element in the XML string stored in a variable is null. This function fails the flow run if the specified element is found to be null in the response.

Usage:

```
elementNotNull(String xpath, String VariableName)
elementNotNull(mobileNumber)
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections
variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.

Return Type: void

elementIsNull

Verifies if the specified element in the XML string stored in a variable is not null.

Usage:

```
elementIsNull(String xpath, String variableName)
elementIsNull(mobileNumber)
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections
variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.

Return Type: void

elementValueEquals

Verifies if the specified element value in the XML string stored in a variable is equal to the provided value. The functions fails the flow if the value does not match.

Usage:

```
elementValueEquals(String xpath,String variableName, String
expectedValue)
elementValueEquals(mobileNumber,gVarVariable, "1234567890")
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections.
variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.
expectedValue - The value to be compared to for validation.

Return Type: void

elementValueNotEquals

Verifies if the specified element value in the XML string stored in a variable is not equal to the provided value. The functions fails the flow if the value matches.

Usage:

```
elementValueNotEquals(String xpath,String variableName, String
expectedValue)
elementValueNotEquals(mobileNumber,gVarVariable, "1234567890")
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections.
variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.
expectedValue - The value to be compared to for validation

Return Type: void

elementValueGreaterThan

Verifies if the specified element value in the XML string stored in a variable is greater than the provided value. This function fails the flow if the value is not greater than the specified value.

Usage:

```
elementValueGreaterThan(String xpath, String variableName String
valueToCompare)
elementValueGreaterThan("count",gVarVariable,"5")
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections.
variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.
valueToCompare - The value to be compared to for validation. To be provided in the Value column.

Return Type: void

elementValueGreaterThanEqualTo

Verifies if the specified element value in the XML string stored in a variable is greater than or equal the provided value. The function fails the flow if the value is not greater than or equal to the specified value.

Usage:

```
elementValueGreaterThanEqualTo(String xpath, String variableName
String valueToCompare)
elementValueGreaterThanEqualTo("count",gVarVariable,"5")
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections
variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.
valueToCompare - The value to be compared to for validation. To be provided in the Value column.

Return Type: void

elementValueLesserThan

Verifies if the specified element value in the XML string stored in a variable is less than the provided value. The function fails the flow if the value is not less than the specified value.

Usage:

```
elementValueLesserThan(String xpath, String variableName String
valueToCompare)
elementValueLesserThan("count",gVarVariable,"5")
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections
variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.
valueToCompare - The value to be compared to for validation. To be provided in the Value column

Return Type: void

elementValueLesserThanEqualTo

Verifies if the specified element value in the XML string stored in a variable is lesser than or equal to the provided value. The function fails the flow if the value is not greater than or equal to the specified value.

Usage:

```
elementValueLesserThanEqualTo(String xpath, String variableName
String valueToCompare)
elementValueLesserThanEqualTo("count",gVarVariable,"5")
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections
variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.
valueToCompare - The value to be compared to for validation. To be provided in the value column

Return Type: void

elementContains

Verifies if the specified element contains the specified value in the XML string stored in a variable. The function fails the flow if the element does not contain the specified value.

Usage:

```
elementContains(String xpath,String variableName, String
valueToCompare) elementContains("batchName",gVarVariable, "F1-
BILLING)
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections
variableName - Name of the variable that holds the XML string to be

validated. The variable name needs to be provided in the Value column.
 valueToCompare - The value to be compared to for validation. To be provided in the Value column

Return Type: void

elementNotContains

Verifies if the specified element does not contain the specified value in the XML string stored in a variable. The function fails the flow run if the element contains the specified value.

Usage:

```
elementContains(String xpath,String variableName, String
valueToCompare) elementContains("batchName",gVarVariable, "F1-
BILLING)
```

Input Parameters:

xpath - xpath of the element to be validated. To be provided in the value column of test data in the pre/post validations sections
 variableName - Name of the variable that holds the XML string to be validated. The variable name needs to be provided in the Value column.
 valueToCompare - The value to be compared to for validation. To be provided in the Value column

variableNotContains

Verifies if the specified value or element is not available in the XML string held in the variable. The function fails the flow run if the element contains the specified value.

Usage:

```
reponseNotContains(String variableName,String valueToCompare)
reponseNotContains(gVarVariable,"Failed")
```

Input Parameters:

variableName - the name of the variable that holds the XML string.
 valueToCompare - value to compare to

Return Type: void

variableContains

Verifies if the specified value or element is available in the XML string held in a variable. The function fails the flow run if the element does not contain the specified value.

Usage:

```
responseContains(String variableName,String value)
responseContains(gVarVariable,"Exception")
```

Input Parameters:

variableName - the name of the variable that holds the XML string.
 valueToCompare - value to compare to

Return Type: void

CORESTOREVALUES

Use this function library to store values into a variable.

setVariable

Stores the specified input value into a global variable specified in the OP Variable Name column.

Usage:

```
setVariable(String valueToStore)
elementNotNull("32425683235")
```

Input Parameters:

valueToStore - value to be stored in a global variable

Return Type: String

COREFILEOPS

Use this function library to read files that are stored in flow attachments.

readAttachmentAsString

Reads the content of the attachment whose filename is provided as input parameter and stores the content as string into a global variable specified in the OP Variable Name column. The encoding of the file needs to be specified as a second input parameter.

Usage:

```
readAttachmentAsString(String fileName, String fileEncoding)
readAttachmentAsString("testFile.txt", "UTF-8")
```

Input Parameters:

fileName - name of file in flow attachments
fileEncoding - encoding of the file being read

Return Type: String

CORESTRINGOPS

Use this function library perform String operations such as append, etc.

appendStrings

Appends the inputs strings specified in the value 1 to value 6 columns in that sequence and stores the output into a global variable specified in the OP Variable Name column.

The function takes 6 parameters as inputs by default. If less than 6 strings have to concatenated, then provide #EMPTY in the value columns where test data need not be specified.

Usage:

```
appendStrings(String strValue1, String strValue2, String strValue3,
String strValue4, String strValue5, String strValue6)
appendStrings("string1", "string2", "string3", "string4",
"string5", "string6",)
```

Input Parameters:

strValue1 - The base string

```

strValue2 - The string to be appended to strValue1
strValue3 - The string to be appended to strValue1+strValue2
strValue4 - The string to be appended to
strValue1+strValue2+strValue3
strValue5 - The string to be appended to
strValue1+strValue2+strValue3+strValue4
strValue6 - The string to be appended to
strValue1+strValue2+strValue3+strValue4+strValue5

```

Return Type: String

CORENUMBEROPS

Use this function library perform numeric operations such as addition, subtraction etc. This section provides a list of functions in the library, along with the usage details.

getSumOfTwoNumbers

Gets the sum of two input numbers specified in the value 1 and value 2 columns and stores the output into a global variable specified in the OP Variable Name column.

Usage:

```

getSumOfTwoNumbers(String number1, String number2)
getSumOfTwoNumbers ("3", "5")

```

Input Parameters:

```

number1 - The first number
number2 - The second number to be added to number1

```

Return Type: String

getDiffOfTwoNumbers

Gets the difference of two input numbers specified in the value 1 and value 2 columns and stores the output into a global variable specified in the OP Variable Name column.

Usage:

```

getDiffOfTwoNumbers(String number1, String number2)
getDiffOfTwoNumbers ("3", "5")

```

Input Parameters:

```

number1 - The first number
number2 - The second number to be subtracted from number1

```

Return Type: String

getProductOfTwoNumbers

Gets the product of two input numbers specified in the value 1 and value 2 columns and stores the output into a global variable specified in the OP Variable Name column.

Usage:

```

getProductOfTwoNumbers(String number1, String number2)
getProductOfTwoNumbers ("3", "5")

```

Input Parameters:

```

number1 - The first number

```

number2 - The second number to be multiplied with number1

Return Type: String

getModOfTwoNumbers

Gets the modulus of two input numbers specified in the value 1 and value 2 columns and stores the output into a global variable specified in the OP Variable Name column.

Usage:

```
getModulusOfTwoNumbers(String number1, String number2)
getModulusOfTwoNumbers ("3", "5")
```

Input Parameters:

number1 - The first number
number2 - The second number to be used as a divisor for the first number

Return Type: String

getDivisionOfTwoNumbers

Gets the quotient of the division of input numbers specified in the value 1 and value 2 columns and stores the content as byte array into a global variable specified in the OP Variable Name column.

Usage:

```
getDivisionOfTwoNumbers(String number1, String number2)
getDivisionOfTwoNumbers ("3", "5")
```

Input Parameters:

number1 - The first number
number2 - The second number to be used as a divisor for the first number

Return Type: String

getMaxOfTwoNumbers

Gets the maximum value among two input numbers specified in the value 1 and value 2 columns and stores the output into a global variable specified in the OP Variable Name column.

Usage:

```
getMaxOfTwoNumbers(String number1, String number2)
getMaxOfTwoNumbers ("3", "5")
```

Input Parameters:

number1 - The first number
number2 - The second number to be compared to the first to determine the maximum of two numbers.

Return Type: String

getMinOfTwoNumbers

Gets the minimum value among two input numbers specified in the value 1 and value 2 columns and stores the output into a global variable specified in the OP Variable Name column.

Usage:

```
getMinOfTwoNumbers(String number1, String number2)
getMinOfTwoNumbers ("3", "5")
```

Input Parameters:

```
number1 - The first number
number2 - The second number to be compared to the first to
determine the minimum of two numbers.
```

Return Type: String

getAbsoluteOfNumber

Gets the absolute value of the number specified in the value 1 column and stores the output into a global variable specified in the OP Variable Name column.

Usage:

```
getAbsoluteOfNumber(String number1)
getAbsoluteOfNumber ("3.754")
```

Input Parameters:

```
number1 - The number whose absolute value needs to be determined.
```

Return Type: String

COREUTAOPS

Use this function library to perform operations specific to flow run in Oracle Utilities Test Accelerator such as pausing a flow run for a specified time, conditional constructs to exit polling of a specific component IWS.

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

waitForTime

Pauses the flow run for the specified number of minutes. The flow run is resumed after the completion of the wait time.

Usage:

```
waitForTime(String timeInMinutes)
waitForTime ("3")
```

Input Parameters:

```
timeInMinutes - Number of minutes for which the flow run needs to
be paused.
```

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](#)
- [Exporting/Importing Custom Libraries](#)
- [Using Custom Library Functions](#)

Creating/Updating Custom Libraries

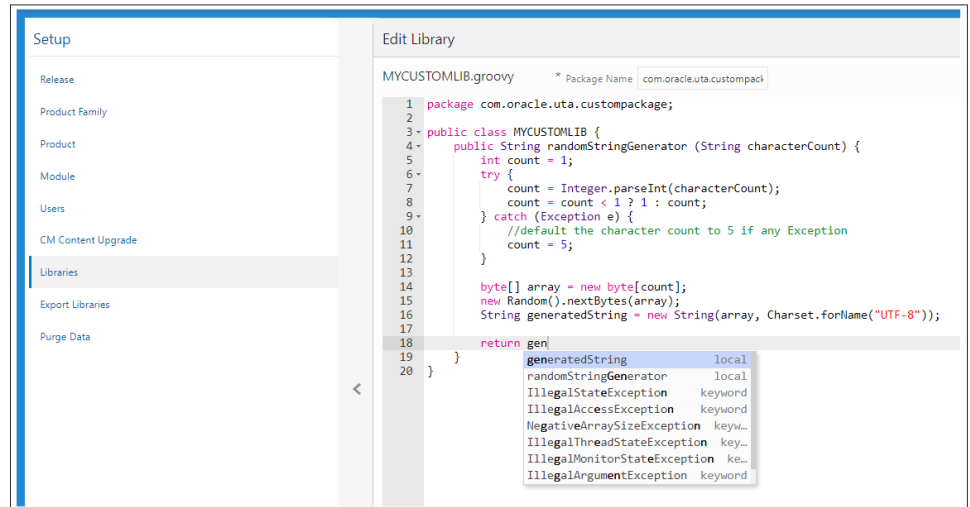
Make sure to have Administrator privileges to manage custom libraries.

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 **Open Editor** to develop or upload a Groovy script containing actual implementation of the functions included.
11. Specify the package name of the custom library.
12. Enter the code or paste it from an external source. Click **Compile**. This triggers the compilation of the custom library Groovy code and displays errors if any. Rectify the code and click **Compile** again to verify the changes.
13. Click **Save** to overwrite any existing library with an updated version or a new library being created.



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.

Library Functions						
Function	Parameter Count	Parameters	Parameter Comments	Description	Delete	
generateSSN	1	@param1	The prefix of the social securit	Random SSN generator	[Delete]	

Click **Open Editor** to create the implementation of the groovy library. It can be plugged into any custom component or the pre-validations and post-validations section of flow test data definition.

Exporting/Importing Custom Libraries

Make sure to have Administrator privileges to manage custom libraries.

To export a single 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 **Search**.
4. After the library functions are displayed in the GUI, click **Export**. This generates an export file of the custom function library currently being displayed.

To export multiple custom libraries:

1. Login to Oracle Utilities Testing Accelerator as an Administrator.
2. Navigate to the **Administration** tab and click **Export Libraries** on the left pane.
3. Search for the library based on names. Select two or more custom function libraries to be exported.
4. Click **Export**. The selected custom function libraries should be exported into an archive file that can be downloaded.

To import custom libraries:

1. Login to Oracle Utilities Testing Accelerator as an Administrator.
2. Navigate to the **Tools** tab and click **Import** on the left pane.
3. Select the archive file that has the library definitions and click **Import**.

The library should be imported and available for use in the components/flows.

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-SETXMLLEMENT

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-SETXMLLEMENT
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 run 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

GUI Component Keywords

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

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

APPROVE

Approves the specified object.

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

You can replace xpath with ID or name.

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

CANCEL

Cancels the specified object.

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

You can replace xpath with ID or name.

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

CHECK

Checks a checkbox object.

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

You can replace xpath with ID or name.

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

CLICK

Cancel the specified object.

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

You can replace xpath with ID or name.

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

CLOSE

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

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

You can replace xpath with ID or name.

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

GET_ATTRIBUTE_VALUE

Retrieves the “value” attribute of the specified object.

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

You can replace xpath with ID or name.

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

GET_ATTRIBUTE_ID

Retrieves the “ID” attribute of the specified object.

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

You can replace xpath with ID or name.

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

LAUNCH

Launches the specified browser object.

Use Case: The attribute value takes the browser URL.

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

MAXIMIZE

Maximizes the specified window object.

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

MINIMIZE

Minimizes the specified window object.

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

Usage Details	Value
Objects Valid	WINDOW

POPUP

Handles the pop-up of the window.

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

PRESSTABKEY

Performs a tab key press on the specified object.

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

You can replace xpath with ID or name.

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

SELECT

Performs a ‘select’ action on the specified object.

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

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

SETTEXT

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

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

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

SWITCHTO

Used to switch between different frames.

Use Case: The attribute value takes xpath of an element in the “xpath; actual xpath of the element” format. You can replace xpath with ID or name.

Usage Details	Value
Keyword	SWITCHTO
Display Name	User Defined Display Name
Attribute Values	Xpath of the element for STARTFRAME None for ENDFRAME
Objects Valid	STARTFRAME - Switches to the specified frame ENDFRAME - Navigates back to the parent frame

UNCHECK

Uncheck the specified checkbox object.

Use Case: The attribute value takes xpath of an element in the “xpath; actual xpath of the element” format. You can replace xpath with ID or name.

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

UNSELECT

Unselects the specified object.

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

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

UI-STARTBROWSER

Starts the browser.

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

UI-ENDBROWSER

Closes the browser.

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

WAIT

Waits for the specified object. It is used to communicate to wait for a certain amount of time before throwing an exception that it cannot find the element on the page.

Use Case: The attribute value takes the xpath of the element in the “xpath; actual xpath of the element” format. xpath can be replaced with ID or name.

In the test data, provide the wait time(number) for LIST, LISTBOX, TEXTAREA, LINK, and BUTTON objects. For EDIT, IMAGE, WINDOW, and NORMAL objects, no test data is required.

Usage Details	Value
Keyword	WAIT
Display Name	User Defined Display Name
Attribute Values	Xpath of the element EDIT, IMAGE, WINDOW, NORMAL - None
Objects Valid	LIST LISTBOX TEXTAREA LINK BUTTON EDIT IMAGE WINDOW NORMAL - Can be used for any element

Appendix C

REST Component Keywords

This chapter provides the following REST component keywords:

- [RS-SETREQUESTHEADER](#)
- [RS-SETENDPOINT](#)
- [RS-ARGUMENT](#)
- [RS-SETPMETHOD](#)
- [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 D

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 E

Generating Re-runnable Test Data

To run a flow multiple times, some fields might need unique values for each run. 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 run.

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

Appendix F

Connecting to Multiple Databases

While building the integration flows that involve using components from various product packs, there might be a need to access databases of individual products as part of verification process. Users can connect to different databases by specifying the database connection properties prefixed with the application prefix. This is similar to the application prefix that is used for connecting to different application URLs.

Following is a sample configuration.properties file. It shows the entries for two environments.

Note: Users are expected to write custom functions for connecting/accessing databases.

```
# *****
# Test Environment Details - UAT1
# *****

# Application URL pointing to test run
gStrUAT1_gStrApplicationURL = http://myuat1environment:8001/ouaf
gStrUAT1_gStrApplicationXAIServerPath=/webservices/
gStrUAT1_gStrEnvironmentName=UAT1

# Application Database Details
gStrUAT1_gStrApplicationDBConnectionString
=jdbc\:oracle\:thin\:@myuatserver1\:1521\ :UATSID1
gStrUAT1_gStrApplicationDBUsername=mydbuser1
gStrUAT1_gStrApplicationDBPassword==<Entrypted Password>

# *****
# Test Environment Details - UAT2
# *****

# Application URL pointing to test run
gStrUAT2_gStrApplicationURL = http:// myuat1environment2:8001/ouaf
gStrUAT2_gStrApplicationXAIServerPath=/webservices/
gStrUAT2_gStrEnvironmentName=UAT2

# Application Database Details
gStrUAT2_gStrApplicationDBConnectionString =jdbc\:oracle\:thin\:@
myuatserver2\:1521\ : UATSID2
gStrUAT2_gStrApplicationDBUsername= mydbuser2
gStrUAT2_gStrApplicationDBPassword= <Entrypted Password>

##Handling Https WSDL - Java key Store
gStrJavaKeyStorePath=
gStrJavaKeyStorePwd=<Entrypted Password>

#Email Details
```

```
gStrSMTP_HOST_NAME==<SMTP server address>
gStrSMTP_PORT=25
gStrTO_EMAIL_RECIPIENTS=

# Application user name
gStrApplicationUserName=<Your OUAF Application Username>

# Application user password
gStrApplicationUserPassword=<Encrypted Password>

# Output file details
gStrOutputFilePath=/<UTA-CLIENT-WORK-DIR>//Logs/
gStrXSDFiles=/<UTA-CLIENT-WORK-DIR>//XSD/
```

Appendix G

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
```

Appendix H

OUTA REST Services

Oracle Utilities Testing Accelerator provides REST services that can be used to integrate with any compatible third party application, such as test management applications, Jenkins, etc.

This chapter focuses on the Oracle Utilities Testing Accelerator REST services to run flows and retrieve flow run analytics. It describes the following services:

- [Prerequisites](#)
- [Flow Run](#)
- [Flow Set Run](#)
- [Flow Run Analytics](#)
- [Flow Set Run Analytics](#)
- [Flow Run Summary](#)
- [Flow Set Run Summary](#)

Prerequisites

All the REST services require an authentication token as one of the parameters. Use the following command to obtain an authentication token.

Endpoint

/token/generate-token

Curl command

```
curl --insecure -X POST https://<hostname>:<port>/token/generate-token -H 'content-type: application/json' -d '{ "username": "<username>", "password": "<password>" }
```

Note: the option `--insecure` is only needed if the application uses a self signed certificate.

Parameters

< username>: Valid OUTA application user's username

< password>: OUTA application user's password

<hostname>: OUTA Application host name

<port>: OUTA application port

Below is a sample response from the call.

Response

```
{
  "token":
  "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJsb2NhbGhvc3QiLCJzY29wZXMiOiJST0xFX0FkbWl1aXN0cmF0b3liLCJpYXQiOiJlMjMjAwMzQ1NjMsImV4cCI6MTYyMDA1MjU2M30u2SrkZPslj6Ba2S8zqJm8YakjaFdZfhIhfuXcEsSE-Q"
}
```

Flow Run

The Flow Run service provides an endpoint allowing a flow to run by passing the relevant flow details.

Endpoint

>/rest/run/flow

Curl command for Flow Run

```
curl -i -X POST 'https://<hostname>:<port>/rest/run/flow' -H 'authorization: Bearer <access_token>' -H 'cache-control: no-cache' -H 'content-type: application/json' -d '{
  "executionType": "flow",
  "release" : "<release>",
  "portfolio" : "<portfolio>",
  "product" : "<product>",
  "flow" : "<flow>",
  "configuration": "<configuration>",
  "identity": "<user configuration>",
  "flowtestdataset": "<flowtestdataset>"
}'
```

Note: <flowtestdataset> parameter is optional, if user does not provide the parameter then the default test data set will be used for run.

Parameters

<hostname>: OUTA application host name

<port>: OUTA application port

<access token>: Authentication token from the step in pre-requisites above

<release>: The name of the release to which the flow belongs to

<portfolio>: The name of the portfolio/product family to which the flow belongs to

<product>: The name of the product to which the flow belongs to

<flow>: Name of the flow to be run

<configuration>: Configuration to be used for running the flow

<user configuration>: The user configuration to be used for run the flow

<flowtestdataset>: The flow test data set that needs to be used during the flow run

Example

Flow Run

```
curl -X POST -k https://my.server.com:8080/rest/execute/flow -H 'authorization: bearer 62593bbd-b057-4f38-8a3e-5915d4ab559f' -H 'content-type: application/json' -d '{"executionType": "flow", "release": "UTA", "portfolio": "CUSTOMER CARE AND BILLING", "product": "CCB 2.7.0.1", "flow": "CreateBusinessWithCC", "configuration": "myCCBConfig", "identity": "myCCBConfig", "flowtestdataset": "CCBTestDataSet" }'
```

Once a flow run is triggered, the service responds with a response similar as follows.

```
{"flowExecutionId":1810,"trackingCode":" CreateBusinessWithCC_100000065_2020_01_29_04.00.24.604_26843725-5fe4-4c38-a481-b7e775ddcff0"}
```

Querying Status of Flow Run

Query the status of the flow run as follows.

Endpoint

```
>/rest/execute/flowstatus
```

Curl for Flow Run Status

```
curl -i -k -X GET -k 'https://<hostname>:<port>/rest/execute/flowstatus?flowExecutionId=<flowExecutionId>' -H 'authorization: Bearer <access_token>'
```

Parameters

<hostname> : Oracle Utilities Testing Accelerator host name

<port> : Oracle Utilities Testing Accelerator port

<access token> : Authentication token (See [Prerequisites](#) for more information.)

<flowExecutionId >: The flow run ID received when the flow run was triggered.

Example

```
curl -i -k -X GET 'https://my.server.com:8080/rest/execute/flowstatus?flowExecutionId=1810' -H 'authorization: bearer 62593bbd-b057-4f38-8a3e-5915d4ab559f'
```

The service responds to the flow run status query with a response.

Response

```
{ "product": "CCB 2.7.0.1", "portfolio": "CUSTOMER CARE AND BILLING", "release": "UTA", "flowName": "CreateBusinessWithCC", "status": "Running" }
```

The status element in the response contains the current flow run status.

Flow Set Run

The Flow Set Run service provides an endpoint allowing a flow set to be run by passing relevant flow set details.

Endpoint

```
>/rest/execute/flow
```

Curl command for Flow Set Run

```
curl -i -X POST 'https://<hostname>:<port>/rest/execute/flow' -H 'authorization: Bearer <access_token>' -H 'cache-control: no-cache' -H 'content-type: application/json' -d '{
    "executionType": "flowSet",
    "flowSet": "<flowSet>",
    "configuration": "<configuration>",
    "identity": "<user configuration>"
}'
```

Parameters

<hostname>	: OUTA Application host name
<port>	: OUTA application port
<access token>	: Authentication token from the step in pre-requisites above
<flowSet>	: Name of the flow set to be executed
<configuration>	: Configuration to be used for executing the flow
<user configuration>	: The user configuration to be used for executing the flow

Example**Flow Set Run**

```
curl -X POST -k https://my.server.com:8080/rest/execute/flow -H 'authorization: bearer 62593bbd-b057-4f38-8a3e-5915d4ab559f' -H 'content-type: application/json' -d '{"executionType": "flowSet", "flowSet": "MyBillingTestSuite", "configuration": "myCCBConfig", "identity": "myCCBConfig" }'
```

Once a flow set run is triggered, the service responds with a response as follows.

```
[{"flowSetExecutionId": "502", "flowSetName": "MyBillingTestSuite", "status": "Running"}]
```

Querying Status of Flow Set Run

User can query the status of flow set run.

Curl for Flow Set Run Status

```
curl -i -X GET -k 'https://<hostname>:<port>/flowset/status?flowSet=<flowSet>&flowSetExecutionId=<flowSetExecutionId>' -H 'authorization: Bearer <access_token>'
```

Parameters

<hostname>	: OUTA Application host name
------------	------------------------------

<port>: OUTA application port

<access token> : Authentication token from the step in pre-requisites above

<flowSetExecutionId>: The flow set run ID that was received when the flow set run was triggered

Example

```
curl -i -k -X GET 'https://my.server.com:8080/rest/execute/flowset/status?flowSet=MyBillingTestSuite &flowSetExecutionId=502' -H 'authorization: bearer 62593bbd-b057-4f38-8a3e-5915d4ab559f'
```

The service responds to the flow run status query with a response as follows.

Response

```
[{"flowStatuses":[{"timeStamp":"Jan 19, 2020 04:00:25 PST","product":"CCB 2.7.0.1","portfolio":"CUSTOMER CARE AND BILLING","release":"UTA","flowExecutionId":"1811","id":1,"flowName":"X1-MDMFlow","portProdXrefId":"100000065","status":"Not Started"}],"flowSetExecutionId":"502","flowSetName":"MyBillingTestSuite","status":"Running"}]
```

The status element in the response contains the current flow set overall run status. The flowStatuses element in the response contains individual flow status for each of the flows that were/are run in the flow set.

Flow Run Analytics

The Flow Run Analytics service provides analytics for the flows run in Oracle Utilities Testing Accelerator by a specific user for a given period.

Endpoint

>/rest/analytics/user

Curl Command for Flow Run Analytics

```
curl -i -k -X GET -k 'https://<hostname>:<port>/rest/analytics/user/<username>/flowExecutionAnalytics/from/<fromDate>/to/<toDate>' -H 'authorization: Bearer <access_token>'
```

Parameters

<hostname>: OUTA application host name

<port>: OUTA application port

<access token>: Authentication token from the step in pre-requisites above

<username>: Name of the user who executed the flow(s)

<fromDate>: The start date to query for analytics - format DD-MON-YY (example: 10-JAN-20)

<toDate>: The end date to query for analytics - format DD-MON-YY (example: 20-JAN-20)

Example

Flow Run Analytics

```
curl -i -k -X GET -k https://my.server.com:8080/rest/analytics/user/admin/flowExecutionAnalytics/from/01-JAN-20/to/15-JAN-20 -H 'authorization: bearer 62593bbd-b057-4f38-8a3e-5915d4ab559f'
```

Once a flow set run is triggered, the service responds with a response as follows.

```
{"totalFlowsRanByUser":50,"totalFlowsRanByUserSuccess":48,"totalFlowsRanByUserFailure":2,"totalFlowsRanByUserRunning":0}
```

Flow Set Run Analytics

The Flow Set Run Analytics service provides analytics for the flow sets run in Oracle Utilities Testing Accelerator by a specific user for a given period.

Endpoint

```
>/rest/analytics/user
```

Curl Command for Flow Run Analytics

```
curl -i -k -X GET -k 'https://<hostname>:<port>/rest/analytics/user/<username>/flowSetExecutionAnalytics/from/<fromDate>/to/<toDate>' -H 'authorization: Bearer <access_token>'
```

Parameters

<hostname>: OUTA application host name

<port>: OUTA application port

<access token>: Authentication token from the step in pre-requisites above

<username>: Name of the user who executed the flow(s)

<fromDate>: The start date to query for analytics - format DD-MON-YY (example: 10-JAN-20)

<toDate>: The end date to query for analytics - format DD-MON-YY (example: 20-JAN-20)

Example

Flow Run Analytics

```
curl -i -k -X GET -k https://my.server.com:8080/rest/analytics/user/admin/flowSetExecutionAnalytics/from/01-JAN-20/to/15-JAN-20 -H 'authorization: bearer 62593bbd-b057-4f38-8a3e-5915d4ab559f'
```

Once a flow set run is triggered, the service responds with a response as follows.

```
{"totalFlowSetsRanByUser":30,"totalFlowSetsRanByUserSuccess":29,"totalFlowSetsRanByUserFailure":1,"totalFlowSetsRanByUserRunning":0}
```

Flow Run Summary

The Flow Run Summary service provides summary file for the flow run in Oracle Utilities Testing Accelerator.

Endpoint

```
>/rest/summary/report
```

Curl Command for Flow Run Summary

```
curl -X POST \
'https://<hostname>:<port>/rest/summary/report?type=flow&flowName=x' \
-H 'accept: application/xml' \
-H 'authorization: Bearer <token>' \
-H 'content-type: application/json' \
-d '{"<flow execution id>"}
```

Parameters

<hostname>: OUTA application host name

<port>: OUTA application port

< token>: Authentication token from the steps mentioned in pre-requisites above

<flow execution id>: Flow execution ID of the flow for which the summary is being requested

Example**Flow Run Summary**

```
curl -X POST \
  'https://my.server.com:8080/rest/summary/report?type=flow&flowName=x' \
  -H 'accept: application/xml' \
  -H 'authorization: Bearer 3fdb6422-b62c-4398-8781-a05fa4d9feccd' \
  -H 'content-type: application/json' \
  -d '{"2507"}'
```

The server responds with a HTML summary file as response upon receiving this request.

Flow Set Run Summary

The Flow Set Run Summary service provides summary file for the flow set run in Oracle Utilities Testing Accelerator.

Endpoint

>/rest/summary/report

Curl Command for Flow Set Run Summary

```
curl -X POST \
  'https://<hostname>:<port>/rest/summary/report?type=flowset&flowSetName=x' \
  -H 'accept: application/xml' \
  -H 'authorization: Bearer <token>' \
  -H 'content-type: application/json' \
  -d '{"<flow execution id>"}'
```

Parameters

<hostname>: OUTA application host name

<port>: OUTA application port

< token>: Authentication token from the steps mentioned in pre-requisites above

<flow execution id>: Flow execution ID of the flow set for which the summary is being requested

Example**Flowset Run Summary**

```
curl -X POST \
  'https://my.server.com:8080/rest/summary/report?type=flowset&flowSetName=x' \
  -H 'accept: application/xml' \
  -H 'authorization: Bearer 3fdb6422-b62c-4398-8781-a05fa4d9feccd' \
  -H 'content-type: application/json' \
```

-d '["2507"]'

The server responds with a HTML summary file of the flow set as response.